University of Crete
School of Sciences and Engineering
Computer Science Department

ONTOLOGY EVOLUTION IN DATA INTEGRATION

Haridimos G.Kondylakis

Doctoral Dissertation

Heraklion, October 2010

University of Crete
Computer Science Department

ONTOLOGY EVOLUTION IN DATA INTEGRATION

Dissertation submitted by
Haridimos G. Kondylakis
in partial fulfillment of the requirements for the
PhD degree in Computer Science

Author:

Supervisory Committee:

Dimitris Plexousakis, Professor, Supervisor

Grigoris Antoniou, Professor, Member

Yannis Tzitzikas, Assistant Professor, Member

Vassilis Christophides, Professor, Member

Anastasia Analyti, Principal Researcher, Member

Yannis Ioannidis, Professor, Member

Manolis Koubarakis, Associate Professor, Member

Approved by:

Panos Trahanias, Professor
Chairman of the Graduate Studies Committee
Heraklion, October 2010

Abstract

Due to the rapid scientific development, ontologies and schemata need to change. When ontologies evolve, the changes should somehow be rendered and used by the pre-existing data integration systems. In most of these systems, when ontologies change their relations with the data sources i.e. the mappings, are recreated manually, a process which is known to be error-prone and time-consuming.

In this dissertation, we provide a solution that allows query answering under evolving ontologies without mapping redefinition. This is achieved by exploiting query rewriting. We elegantly separate the semantics of query rewriting for different ontology versions and for the sources and we present a module that enables ontology evolution over traditional ontology-based data integration systems. That module gets as input the different ontology versions and the user query, and rewrites the query over data integration systems that use different ontology versions. This is performed by the automatic identification of changes among the ontology versions using a high-level language of changes, which are then interpreted as GAV mappings to enable query rewriting among ontology versions.

Although query rewriting always succeeds, several problems may occur due to non information preserving changes among the ontology versions. We identify the problems in such a setting and we provide efficient, intuitive solutions, either by explaining the reasons for the failure or by producing the best "over-approximations".

We prove that our approach imposes only a small overhead over traditional query rewriting algorithms and it is modular and scalable. Finally, we show that it can greatly reduce human effort spent since continuous mapping redefinition on evolving ontologies is no longer necessary.

Supervisor: Dimitris Plexousakis,
Professor

ΠΕΡΙΛΗΨΗ:

Λόγω της ταχείας επιστημονικής ανάπτυξης, οι οντολογίες και τα σχήματα που χρησιμοποιούνται για να μοντελοποιήσουν την επιστημονική γνώση πρέπει να αλλάζουν. Όταν οι οντολογίες εξελίσσονται, οι αλλαγές θα πρέπει με κάποιο τρόπο να ενσωματωθούν και να χρησιμοποιηθούν από τα προ-υπάρχοντα συστήματα ολοκλήρωσης πληροφοριών. Στα περισσότερα από τα συστήματα αυτά, όταν οι οντολογίες αλλάζουν, οι συσχετίσεις τους με τις πηγές των δεδομένων δημιουργούνται από το μηδέν χειρονακτικά, μια διαδικασία η οποία είναι γνωστό ότι είναι επιρρεπής σε λάθη και χρονοβόρα.

Στην εργασία αυτή, προτείνουμε μια λύση που επιτρέπει την απάντηση επερωτήσεων χρησιμοποιώντας οντολογίες που εξελίσσονται χωρίς επαναπροσδιορισμό των συσχετίσεων τους με τις πηγές. Αυτό επιτυγχάνεται με την μετεγγραφή των επερωτήσεων από την μία έκδοση στην άλλη. Έτσι, ξεχωρίζουμε την σημασιολογία της μετεγγραφής επερωτήσεων για διαφορετικές εκδόσεις μιας οντολογίας και για τις πηγές και παρουσιάζουμε ένα σύστημα που επιτρέπει την εξέλιξη των οντολογιών πάνω από παραδοσιακά συστήματα ολοκλήρωσης πληροφοριών. Το σύστημά μας δέχεται σαν είσοδο την επερώτηση του χρήστη και τις διαφορετικές εκδόσεις μιας οντολογίας δίνει απαντήσεις από συστήματα ολοκλήρωσης πληροφοριών που χρησιμοποιούν διαφορετικές εκδόσεις της συγκεκριμένης οντολογίας. Αυτό πραγματοποιείται με την αυτόματη αναγνώριση των αλλαγών ανάμεσα στις διαφορικές εκδόσεις της οντολογίας, που μοντελοποιούνται χρησιμοποιώντας μια γλώσσα αλλαγών υψηλού επιπέδου. Οι αλλαγές αυτές ερμηνεύονται σαν συσχετίσεις «καθολικού σχήματος σαν όψη» και επιτρέπουν την μετεγγραφή των επερωτήσεων από την μία έκδοση στην άλλη.

Αν και η μετεγγραφή επερωτήσεων επιτυγχάνει πάντα, προβλήματα μπορούν να προκύψουν εξ' αιτίας αλλαγών στην οντολογία που δεν διατηρούν την ίδια πληροφορία από την μια έκδοση στην άλλη. Έτσι προχωρούμε στον εντοπισμό των προβλημάτων σε ένα τέτοιο περιβάλλον, και παρέχουμε αποτελεσματικές,

διαισθητικές λύσεις είτε δίνοντας εξηγήσεις για τις αλλαγές αυτές, είτε προτείνοντας εναλλακτικές ερωτήσεις που προσεγγίζουν την ζητούμενη απάντηση.

Αποδεικνύουμε ότι η προσέγγιση μας επιβαρύνει ελάχιστα τους παραδοσιακούς αλγορίθμους για επανεγγραφή επερωτήσεων, είναι επεκτάσιμη και κλιμακούμενη. Τέλος δείχνουμε ότι μειώνει σημαντικά την ανθρώπινη προσπάθεια που δαπανάται μια και ο συνεχής επαναπροσδιορισμός των συσχετίσεων δεν είναι πλέον απαραίτητος.

Επόπτης: Δημήτρης Πλεξουσάκης
Καθηγητής

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης για όλα όσα μου προσέφερε αυτά τα χρόνια και για τις γνώσεις που απέκτησα κατά τις σπουδές μου.

Επιπλέον ένα μεγάλο ευχαριστώ ανήκει και στο Ινστιτούτου Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας μια και χρηματοδότησε εν' μέρει την εργασία μου μέσα από τα Ευρωπαϊκά έργα "ACGT" (FP6-IST-026996), "LOCCANDIA" (FP6-IST-2005-2.5.2)) και "plugIT" (FP7-3ICT-231413)). Περισσότερο, θα ήθελα να ευχαριστήσω ολόκληρη την ομάδα Πληροφοριακών Συστημάτων για την άψογη συνεργασία μας, και το ζεστό ομαδικό κλίμα. Είναι μετά από τόσα χρόνια κάτι σαν οικογένεια για μένα (με τη Μαρία τη Μουτσάκη στο ρόλο της μαμάς).

Ακόμα, θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που πίστεψαν σε μένα και με βοήθησαν να ανακαλύψω την επιστημονική μου ταυτότητα. Ιδιαίτερες ευχαριστίες αξίζουν στον επόπτη μου κ. Δημήτρη Πλεξουσάκη που στάθηκε για μένα δάσκαλος, πατέρας, φίλος. Η ελευθερία της επιλογής που πάντα μου έδινε με έμαθε να στέκομαι επιστημονικά στα πόδια μου και να ζυγίζω την κάθε μου απόφαση. Χωρίς την ουσιαστική του καθοδήγηση, τις επισημάνσεις του, τις ευκαιρίες που μου έδωσε θα ήταν αδύνατο να φτάσω εδώ που είμαι σήμερα.

Θα ήθελα ακόμα να ευχαριστήσω τον κ. Γιάννη Τζίτζικα γιατί ήταν πάντα πρόθυμος να με βοηθήσει οποτεδήποτε ζήτησα την βοήθεια του. Τον ευχαριστώ ιδιαίτερα για τις υποδείξεις του καθώς και για το κριτικό πνεύμα που μου εμφύσησε. Θα ήθελα επίσης να ευχαριστήσω την κ. Αναστασία Αναλυτή για τις πολύτιμες και καίριες διορθώσεις πάνω στο κείμενο της εργασίας μου.

Ακόμα θα ήθελα να ευχαριστήσω τον κ. Γρηγόρη Αντωνίου για την διαφορετική ματιά πάνω στην δουλειά μου που μου προσέφερε και τον κ. Βασίλη Χριστοφίδη για την προθυμία του να συμμετάσχει στην επιτροπή για την αξιολόγηση της εργασίας αυτής καθώς και για τις πολύ χρήσιμες επισημάνσεις του. Τέλος

ευχαριστώ τον κ. Γιάννη Ιωαννίδη και τον κ. Μανόλη Κουμπαράκη για την συμμετοχή τους στην εξεταστική επιτροπή της εργασίας μου και για τα πολύ εποικοδομητικά τους σχόλια.

Τελευταίο αλλά μεγαλύτερο ευχαριστώ ανήκει όμως στην οικογένειά μου και πιο συγκριμένα στους γονείς μου Γιώργο και Μαρία, στην αδερφή μου Χαρά και καθώς και στην κοπέλα μου Μαρία, που ήταν πάντα δίπλα μου να με στηρίζουν υπομονετικά σε όλες τις δυσκολίες. Για το λόγο αυτή η εργασία αυτή είναι αφιερωμένη σ' αυτούς και ελπίζω να αποτελέσει μια μικρή ανταμοιβή για τις θυσίες και τις προσπάθειές τους όλον αυτό τον καιρό.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Everything should be as simple as it is, but not simpler"

- Albert Einstein

Contents

1.1 Motivation

The development of new scientific techniques and the emergence of new high throughput tools have led to a new information revolution. The nature and the amount

of information now available open directions of research that were once in the realm of science fiction. During this information revolution the data gathering capabilities have greatly surpassed the data analysis techniques, making the task to fully analyze the data at the speed at which it is collected a challenge. The amount, diversity, and heterogeneity of that information have led to the adoption of data integration systems in order to manage it and further process it.

The traditional view of database integration is that we have one or more source databases S_i and one wants to issue queries on them as if the queries were issued in a new database T which represents the combined information in S_i. The end user typically knows almost nothing about the source databases and he only sees the global schema which he is using in order to formulate queries. The integration of these data sources is a complex problem and raises several semantic heterogeneity problems.

By accepting an ontology as a point of common reference, naming conflicts are eliminated and semantic conflicts are reduced. Ontologies are used to identify and resolve heterogeneity problems, at schema and data level, as a means for establishing explicit formal vocabulary to share. During the last years, ontologies have been used in database integration (Calvanese, 2009),(Heymans, 2008), obtaining promising results, for example in the fields of biomedicine and bioinformatics (Martin, 2008), (Hartung, 2008).

When using ontologies to integrate data, one is required to produce mappings, to link similar concepts or relationships from the ontology/ies to the sources (or other ontologies) by way of an equivalence, according to some metric. This is the *mapping definition process* (Klein, 2001) and the output of this task is the *mapping*, i.e., a collection of mappings rules. In practice, this process is done manually with the help of graphical user interfaces and it is a time-consuming, labour-intensive and error-prone activity. Defining the mappings between schemata/ontologies is not a goal in itself. The resulting mappings are used for various integration tasks such as data transformation and query answering.

Despite the great amount of work done in ontology-based data integration, an important problem that most of the systems tend to ignore is that ontologies are living artifacts and subject to change (Flouris, 2008). Due to the rapid development of research, ontologies are frequently changed to depict the new knowledge that is acquired. The problem that occurs is the following: when ontologies change, the

mappings may become invalid and should somehow be updated or adapted. This is depicted in the following figure.

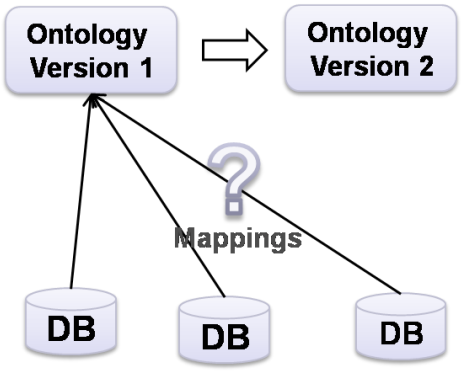


Fig. 1. The problem with the mappings when ontologies evolve

In this document, we address the problem of data integration for evolving RDF/S ontologies. We argue that ontology change should be considered when designing ontology-based data integration systems. A typical solution would be to regenerate the mappings and then regenerate the depending artifacts each time the ontology evolves. However, as ontologies may change too often, the overhead of redefining the mappings each time is significant. The approach to recreate mappings from scratch each time the ontology evolves is recognized to be problematic (Velegrakis, 2004), and instead previously captured information should be reused. However, all current approaches that try to do that suffer from several drawbacks and are inefficient in handling ontology evolution in a state of the art ontology-based data integration system.

1.2 Contributions

The lack of an ideal approach leads us to propose a new mechanism that builds on the latest theoretical advances on the areas of ontology change (Papavassiliou, 2009) and query rewriting (Cali, 2009), (Poggi, 2008) and incorporates and handles ontology evolution efficiently and effectively.

1.2.1 Assessment

To begin with, a comprehensive overview of the works on the area is presented which provides the necessary insights for the practical understanding of the issues involved. The lack of an ideal approach to handle ontology evolution in data integration leads us to establish the requirements for an ideal approach. We highlight what is missing from the current state of the art and outline the requirements for an ideal data integration system that will incorporate and handle ontology evolution efficiently and effectively.

1.2.1 Exploiting Ontology Evolution

To achieve this goal, we firstly capture ontology evolution using high-level changes. As we shall see, a high-level language is beneficial for our problem for two reasons: First, and most important because such a language yields logs that contain a smaller number of individual low-level deletions (which are non-information preserving) and this affects the effectiveness of our rewriting and second because the produced change log has a smaller size.

More specifically we adopt the change operations and the corresponding detection algorithm from (Papavassiliou, 2009) and we show that the proposed language possesses salient properties such as *uniqueness*, *inversibility* and *composability*. We show that the specific language is closed under *composition* and we show how to compute the *composition* of a sequence of changes. Moreover, we define the inverse of a change operation and we show how to compute the inverse of a sequence of changes.

Then we show how to answer queries concerning the evolution of the ontology. In order to do that, we define the concept of a change tree that we use to drive a user's understanding for the evolution of a specific triple and we describe an algorithm for constructing all change trees. Besides computing the change trees for a specific triple we show how to extend the previous algorithm in order to compute the change tree for a specific class/property.

1.2.2 Using high-level changes in data integration

We present the architecture of a data integration system, named *Evolving Data Integration (EDI)* system, that allows the evolution of the ontology used as global schema.

We define the exact semantics of our system and we elegantly separate the semantics of query rewriting for different ontology versions and for the sources. Since query rewriting for the sources has been extensively studied (Cali, 2009), (Poggi, 2008), (Deutsch, 2006) we focus on a layer above and deal only with the query rewriting between ontology versions.

More specifically, we present a *module* that receives a user query specified under the latest ontology version and produces rewritings that will be answered by the underlying data integration systems - that might use different ontology versions. The query processing in this module consists of two steps: a) *query expansion* that considers constraints coming from the ontology, and b) *valid query rewriting* that uses the changes between two ontology versions to produce rewritings among them.

The sequence of changes between the latest and the other ontology versions is produced automatically at setup time and then each one of the change operations identified is translated into a logical GAV mapping. This translation enables query rewriting by unfolding. Then, the inversibility is exploited to rewrite queries from past ontology versions to the current, and vice versa, and composability to avoid the reconstruction of all sequences of changes among the latest and all previous ontology versions.

Despite the fact that query rewriting always terminates, the queries issued to other ontology versions might fail. We show that this problem is not inhibiting in our algorithms but a consequence of information unavailability among ontology versions. For example, no equivalent rewriting will be able to query a deleted class. To tackle this problem, we propose three solutions: either to provide best "over-approximations", namely minimally-containing and minimally-generalized queries, or to provide insights for the failure, thus driving query redefinition only for a specific portion of the affected query. We prove that our method is sound and complete with low complexity.

1.2.3 Implementation & Evaluation

All algorithms were implemented on a novel framework called *Exelixis* which will soon be available to the web. Using our framework we present our experimental analysis based on two real world ontologies. One medium-size ontology (CIDOC-CRM (Doerr, 2007)) from the cultural domain which is rarely changed and one large-size ontology (Gene Ontology (Gene Ontology Consortium, 2004)) from the bioinformatics domain which is heavily updated daily. Using CIDOC-CRM we produced a synthetic set of queries, which we used to evaluate the impact and the scalability of our approach and then we extended our experimentation with real world queries from both CIDOC-CRM and Gene Ontology. The experimentation shows the practical value and the potential impact of our approach.

1.3 Outline of this Dissertation

This thesis is structured as follows. Chapter 2 is an overview of query processing approaches and techniques used to query multi-database systems. Then Chapter 3 reviews existing approaches for handling ontology evolution in data integration and establish the requirements for an idea approach. In Chapter 4 ontology evolution is modelled and algorithms for explaining ontology evolution are presented. Then, in Chapter 5 is shown how to use those changes to rewrite queries among ontology versions. The implementation and the design choices we made are placed in Chapter 6, where also resides our evaluation. Finally, Chapter 7 concludes this dissertation and draws directions for further research work.

A part of Chapter 3 has been published in (Flouris, 2008), while the main part of that chapter was published in (Kondylakis, 2009). Moreover, the techniques for query rewriting based on Chapters 4 and 5 were initially presented in (Kondylakis, 2010a) and (Kondylakis, 2010b) followed by (Kondylakis, 2010c), (Kondylakis, 2010d) and latest submitted to (Kondylakis, 2011a). Moreover, techniques for the detection of invalid queries, constructing the change paths and for identifying the minimal generalized queries presented on Chapter 4 have been submitted to (Kondylakis, 2011b). Finally, a report on these topics will be included in (Zablith, 2011) and our implementation will be submitted in (Kondylakis, 2011c) in the demo session.

Chapter 2

Preliminaries

"Imagination is more important than knowledge."

-Albert Einstein

Contents

2.1 Data integration

Information integration systems aim to provide a uniform query interface to multiple heterogeneous sources. One and useful way of viewing these systems, first proposed in the Information Manifold project (Levy et al. 1996) is to postulate a global schema that provides a unifying data model for all the information sources. A query processor is in charge of accepting queries written in terms of this global schema, translating them to queries on the appropriate sources, and assembling the answers into a global answer.

A question of semantics now arises: what is the meaning of a query? Since a query is expressed in terms of the global schema and the sources implicitly represent an instance of this global schema, it would be natural – at least conceptually – to reconstruct the global database represented by the views and apply the query to this global database. There are at least two issues that must be resolved for this to work.

First the database represented by a set of sources may not be unique; in fact, it may not even exist. Consider the two trivial examples: first suppose we have a single information source, which is defined as the projection on attribute *A* of the global binary relation *R (A, B)*. For any given set of tuples stored at the source, there are many (perhaps an infinite number) of possible global databases. Second, suppose we have not one, but two sources, both storing the projection on *A* as before; one contains the single tuple *<a1>*, and the other the single tuple *<a2>*; then there is no global database whose projection equal this sources. In sum the first issue is: what database or databases are represented by a given set of sources.

Second, suppose we have identified the set of databases that are represented by a given set of sources. Applying the query to each database and producing each possible answer may be impossible (e.g if there is an infinite number of such answers) or undesirable. The second issue is: how to produce a single compact representation of these multiple answers, or an approximation to them if we so desire? Consider for example a schema storing information about the first round of the World Cup Soccer Tournament. Suppose a global relation *Team (Country, Group)* that represents a list of all teams giving the name of the country and the group to which the country has been assigned for first round play.

Suppose first that the only source, S_{Team}, stores a unary relation listing all the countries that are participating in the first round. The corresponding view mapping is given by the conjunctive query:

$$S_{Team}(x) \leftarrow Team(x,y)$$

What global databases are represented by S_{Team}? They are all the relations *Team* such that the view mapping applied to *Team* produces exactly the given instance of S_{Team}, that is:

$$S_{Team}(x) \leftarrow \pi_{country}(Team)$$

In this case, we say that the view is both *sound* and *complete*. On the other hand suppose the only source S_{Qual}, which contains the list of all teams that participated in the qualifying round. This is a strict superset of the teams that will actually be playing in the tournament. Since the global schema says nothing about the qualifying round, the only reasonable view mapping is still

$$S_{Qual}(x) \leftarrow Team(x,y)$$

However, now we understand that this is just an approximation, and that the actual database could be any relation whose projection on Country produces a subset of the source S_{Qual} that is:

$$S_{Qual}(x) \supseteq \pi_{country}(Team)$$

In this case we say the view is complete since it lists every possible team but not sound since it lists teams that are not in the first round. Finally suppose that the only source is S_{Tube} , listing those teams whose games will be televised. Again, the best way to represent the view mapping since there is no information about television in the global schema is by

$$S_{Tube}(x) \leftarrow Team(x, y)$$

In this case, every team listed in STube, corresponds to some tuple in the ideal Team relation, but there are tuples in this ideal relation not represented in S_{Tube} .Thus we take as the set of represented databases all the relations Team that satisfy:

$$S_{Tube}(x) \subseteq \pi_{country}(Team)$$

In this case, we say the view is sound but not complete.

2.1.1 Formal Preliminaries

Before going into the classification we have to formally describe what constitutes a data integration system (Lembo et al. 2002).

Definition 2.1 (Data Integration System)**:** *A data integration system I is a triple <G, S, M> where*

- *G is the global schema expressed in the global language L_g over alphabet A_g. The language L_g determines the expressiveness allowed for specifying the global schema, i.e., the set of constraints that can be defined over it.*
- *S is the set of the local schemas. It is modelled in the source language L_s over the alphabet A_s. A_s in the case of global schema is the language that determines the set of constraints that can be defined over it. Moreover, A_s is disjoint from A_g.*
- *M is the mapping between G and S.*

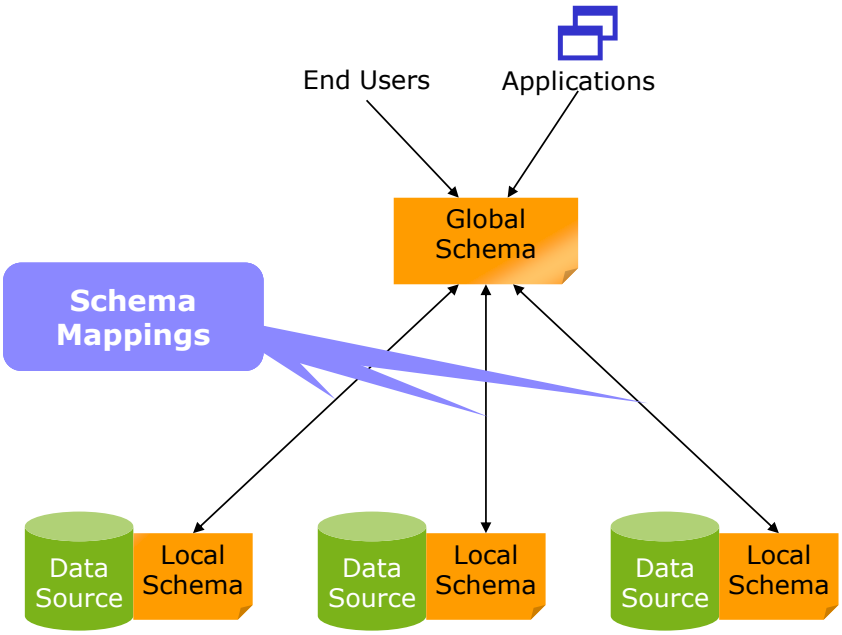


Fig. 2. A Data Integration System

The above definition of data integration system is general enough to capture virtually all approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. For example, the language L_g may be very simple, basically allowing the definition of a set of relations, or may allow for various

forms of integrity constraints to be expressed over the symbols of A_g. Analogously, the type (e.g relational, semi-structured etc.) and the expressive power of L_s vary from one approach to another.

In order to specify the semantics of a data integration system, we start with a set of data at the sources and specify which data satisfies the global schema. We start by considering a source database for *I*, i.e., a database *D* that conforms to the source schema *S* and satisfies all constraints in *S*. Based on *D*, we now specify which is the information content of the global schema *G*. We call *global database* for *I* any database for *G*.

Definition 2.2 (Legal global database) *A global database B for I is said to be legal with respect to D, if:*

- *B is coherent with G, i.e., every constraint in schema G is satisfied by B.*
- *B satisfies the mapping with respect to D, which is its tuples respect the relationships defined between the global and the source schema.*

The notion of *B* satisfying the mapping *M* with respect to *D* depends on how to interpret the assertions in the mapping. Here we simply note that no matter which is the interpretation of the mapping, in general, several global databases exist that are legal for *I* with respect to *D*. This observation motivates the relationship between data integration and databases with incomplete information.

Next, we specify the semantics of queries posed to a data integration system. As we said before, such queries are expressed in terms of the symbols in the global schema of *I*. In general, if *q* is a query of arity *n* and *DB* is a database we denote q^{DB} the set of tuples (of arity *n*) in *DB* that satisfy *q*.

Definition 2.3 (Semantics of a data integration system *I*): *Given a source database D for I, the semantics of I with respect to D, denoted sem(I, D), is defined as:*

$$sem(I, D) = \{ B \mid B \text{ is a legal global database for } I \text{ w.r.t } D \}$$

In order to define the semantics of a query *q* over the global schema *G*, we have to take into account all the legal global databases for *I* with respect to *D*.

Definition 2.4 (Certain Answers): *We call certain answers of a query q of arity n with respect to I and D, the set* $q^{I,D}$ *of n-tuples t such that* $t \in q^{DB}$ *for every database* $DB \in$ *sem(I, D). Certain answers is what we call, answers to a user query.*

The definition above states that the "right" answers are the answers that occur in the intersection of all queries as queries vary over all "possible" databases. Note that from the point of view of logic, finding certain answers is a logical implication of the problem: check whether it logically follows from the information on the sources that t satisfies the query. The dual problem is also of interest: find the so-called possible answers to *q*, i.e checking whether $t \in q^B$ for some global database *B* that is legal for *I* with respect to *D*. Finding possible answers is a consistency problem: check whether assuming that *t* is in the answer set of *q* does not contradict the information on the sources.

Given the above formal definitions, the mapping *M* between the global schema and the sources is provided in terms of a set of assertions of the form <*R, V*>, where *R* is a view over the global schema *G* and *V* is the view over the source schema *S*. Associated to each mapping assertion <*R, V* > we have a specification *as(V)* of which assumption to adopt for the view *V*, i.e., given a source database *D*, how to interpret R^D with respect to the set of tuples in the answer to *V* over a global database *B*, i.e., V^B.

Definition 2.5 (Assumption adopted for a view): *The assumption we adopt for a view V, called as (V), to each mapping assertion < R, V > is defined as follows:*

- *When as(V) = sound, the extension of the associated global view R provides any superset of the tuples satisfying V. In other words, from the fact that a tuple is in* V^D *one can conclude that it satisfies the corresponding global relation R, while from the fact that a tuple is not in* V^D *one cannot conclude that it does not satisfy R. Formally a global database B satisfies the sound view V if* $V^D \subseteq \mu\ R^B$
- *When as(V) = complete, the extension of the associated global view R provides any subset of the tuples satisfying V. In other words, from the fact that a tuple is in* V^D *one cannot conclude that such a tuple satisfies R. On the other hand, from the fact that a tuple is not in* V^D *one can conclude that such a tuple does*

not satisfy R. Formally, a global database B satisfies the complete view V , if $V^D \supseteq R^B$.

- *When a (V) = exact, the extension of the associated global relation R is exactly the set of tuples satisfying V . Formally, a global database B satisfies the exact view V, if* $V^D = R^B$.

Closely related to the query semantic is the global retrieved database.

Definition 2.6 (Retrieved Global Database): *Given a source database C, we call retrieved global database, denoted M(C), the global database obtained by "applying" the queries in the mapping, and "transferring" to the elements of G the corresponding retrieved tuples*

Data integration systems can be classified according to their approach for managing sources, or according to their approach for modelling those.

2.1.2 Classification according to the approach for managing sources

One of the possible classifications may be the one based on whether the queries to the integration system are sent directly to the data sources or whether there are results of the queries that are pre-stored. The virtual view approach corresponds to the former technique, while the materialized view approach uses pre-stored results.

2.1.2.1 Virtual View approach

In the virtual view approach, the data are accessed from the sources on-demand when a user submits a query to the data integration system. The two representative architectures of this approach are federated database systems (FDBS), and mediated systems. Despite of the fact that mediated systems have many similarities with the federated databases, there are some basic differences:

- In mediated systems data sources are not necessarily databases.
- Sources in a mediated system can be added or removed easily.
- Usually, unlike the FDBSs (where access is read/write), in a mediated system access in the sources is read only. This is due to the fact that sources in mediator-based systems are more autonomous.

2.1.2.2 Federated Database Systems

A federated database system (FDBS) consists of some semi-autonomous components that participate in federation to partially share data with each other. Each federated source can also operate independently from the others. These components are not fully autonomous in the sense that they are modified by adding an interface that allows communication with all other databases in the federation. Each component of the federation is either a centralized DBMS, a distributed DBMS or another federated database management system. There are two basic types of FDBS: tightly coupled FDBSs and loosely coupled FDBSs.

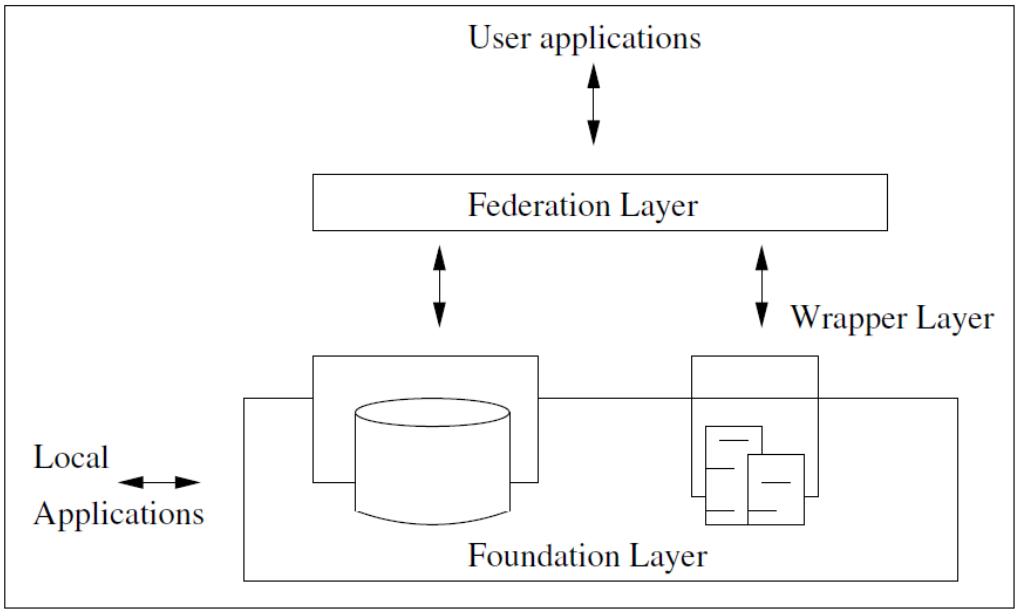


Fig. 3. The architecture of a federated database

A ttightly coupled FDBS has one or more unified schemas which can be produced automatically or manually (by a user). In this type of FDBS, domain experts should undertake the arduous task of integrating all schemas of the federation into a global one. These FDBSs are static and it is very difficult to add or remove components from the integration system.

A loosely coupled FDBS does not have a unified schema, but it provides some unified language for querying sources. In this approach database components are more autonomous and they can decide how they will view all the accessible data in the federation. As there is no global schema, each source can create its own

``federated schema" for its needs. Like the tightly coupled approach, logical heterogeneity should be resolved by domain experts.

The architecture of a federated database system is depicted on Fig. 3. Federated databases is an approach appropriate to use when there is a small number of autonomous sources, and we want to retain their ``independence", allowing user to query them separately and let them collaborate with each other to answer a query.

2.1.2.2 Mediated Systems

Mediated systems are an alternative architecture of data integration systems. They integrate data sources by providing a global virtual view. This global view is called mediated or global schema, and it is employed by the users in order to formulate their queries. The architecture of a mediated system is shown in following Fig. **4**.

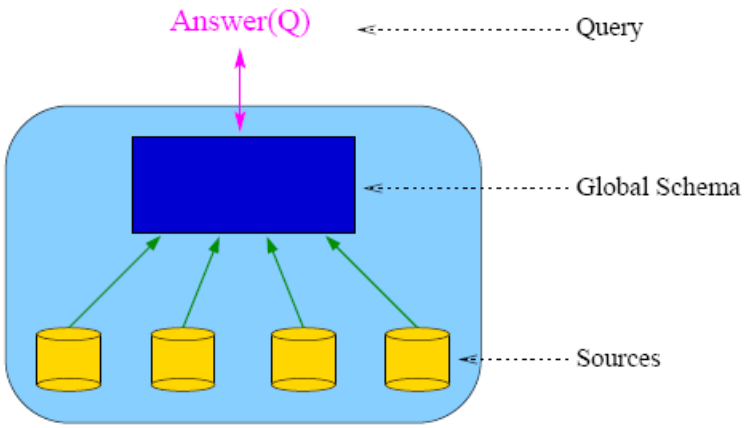


Fig. 4. The architecture of a mediated integration system

There are two basic software components of a mediated system: the mediator and one wrapper per data source. The former offers a common interface to a set of autonomous, independent and possibly heterogeneous data sources. The mediator (a.k.a. integrator) performs the following actions: Firstly it receives a query formulated in terms of the unified schema and decomposes these queries into sub-queries. These queries are addressed to specific data sources. This decomposition is based on source descriptions, which play an important role in sub-queries' execution plan optimization. Finally, the sub-queries are sent to the wrappers of the individual sources, which transform them into queries over the sources. The results of these sub-

queries are sent back to the mediator. At this point the answers are merged into one and returned to the user. Besides the possibility of asking queries, the mediator has no control over the individual sources.

The latter component (wrapper) is responsible for wrapping a data source in such a way that the source can interact with the rest of the integration system. It provides the mediator with data from the source that it is in charge of, as requested by the query execution engine. In consequence, it presents a data source as a convenient database, with the right schema and data, appropriate for being understood and used by the mediator. This presentation schema may be different from the real one, i.e., the internal to the data source. A wrapper hides low-level (protocol) and data model details of the data source from the mediator. It is an important component of both a mediator based architecture and a data warehouse.

A key element in the mediator architecture is the set of source descriptions, i.e., the descriptions of the available sources and their contents, which is achieved by establishing the relationships (mappings) between the global schema and the local schemas. These descriptions can be represented by a set of logical formulas, similar to the way in which views are defined in terms of base tables in the relational data model. The language usually chosen for expressing these mappings is Datalog. There are several fragments of Datalog that are used, based on the existence of recursion, negation and arithmetic comparisons. The most common framework is the one of conjunctive queries (Datalog with relational predicates) with neither recursion nor negation, and arithmetic comparisons limited to equalities. There are different approaches with respect to how mappings are defined, and will be discussed thoroughly later.

2.1.2.2 Materialized View or Data Exchange Approach

In the materialized view approach (a.k.a data warehousing, data exchange) (Fagin et al. 2005a), (Kolaitis et al. 2005) some filtered information from data sources are pre-stored (materialized) in a repository and can be queried later. The single repository in which data are stored is called data warehouse.

There are some important issues that should be taken into account for designing and maintaining a data warehouse. Firstly (designing phase) we need to decide what information from each source is going to be used, what views over these sources is

going to be materialized and what global schema will be employed by the warehouse. Next (maintenance phase) we have to deal with how the warehouse is initially populated by the source data and how it is refreshed when the data in the sources are updated. Finally, there are some query processing, storage and indexing issues that should be taken into consideration. The architecture of the materialized view approach is depicted on Fig. 5.

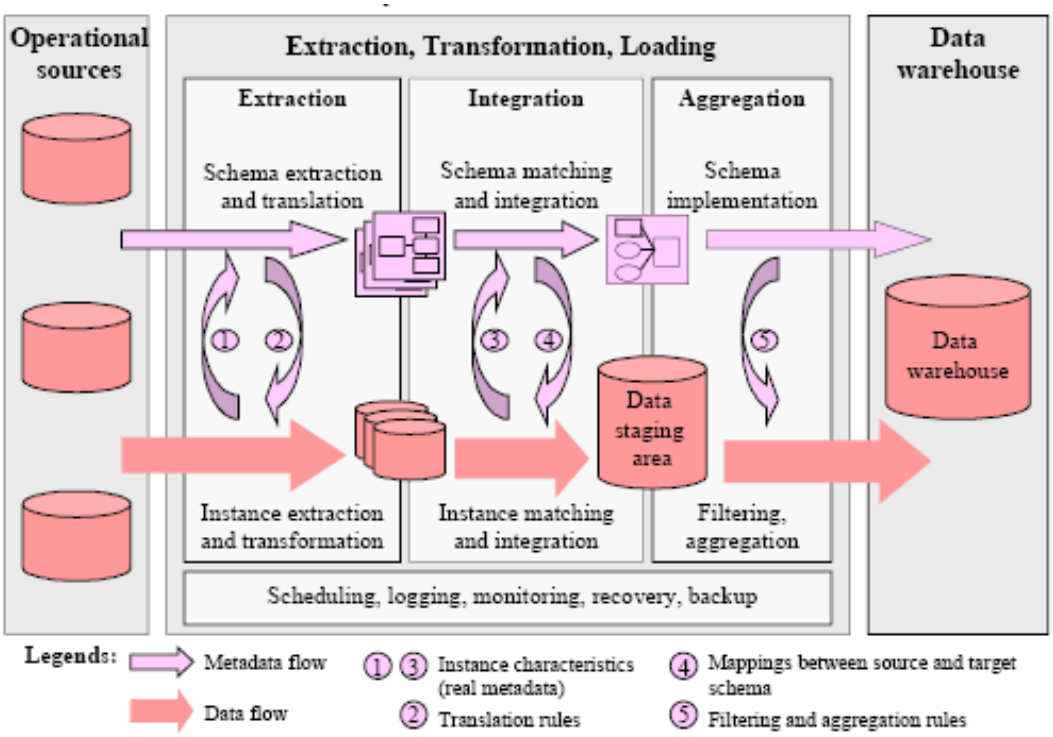


Fig. 5. The architecture of a data warehouse

2.1.2.3 P2P data Integration

Intuitively, data management and data integration tools should be well-suited for exchanging information in a semantically meaningful way. Unfortunately, they suffer from two significant problems: they typically require a comprehensive schema design before they can be used to store or share information, and they are difficult to extend because schema evolution is heavyweight and may break backwards compatibility. As a result, many small-scale data sharing tasks are more easily facilitated by non-database-oriented tools that have little support for semantics.

The goal of the peer data management system (PDMS) is to address this need: So, it is proposed the use of a decentralized, easily extensible data management architecture in which any user can contribute new data, schema information, or even

mappings between other peers' schemas. PDMSs represent a natural step beyond data integration systems, replacing their single logical schema with an interlinked collection of semantic mappings between peers' individual schemas.

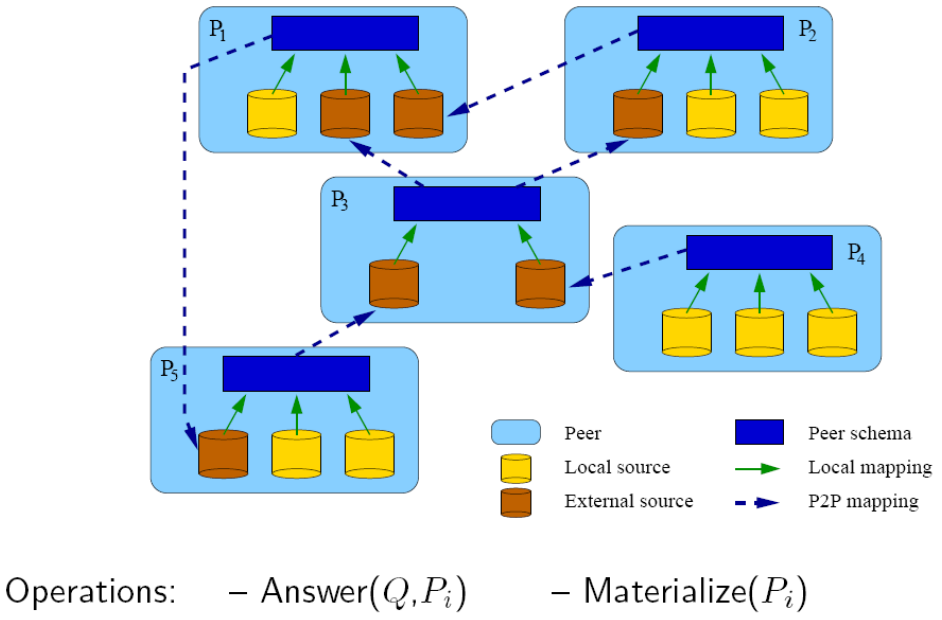


Fig. 6. P2P data integration

Peer data management systems (PDMS), formalized and studied by Halevy et al. (Halevy et al. 2003), constitute a decentralized, extensible architecture in which peers interact with each other in sharing and exchanging data.

According to Halevy, a PDMS *N* with peers *P1*, . . . , *Pn* has the following characteristics.

- Each peer *Pi* has its own schema which is disjoint from those of the other peers, but visible to all other peers.
- The schema of each peer can be a mediated global schema over a set of local sources that are accessible only by that peer (thus each peer can be a data integration system). The relationship between the peer and its local sources is specified using storage descriptions that are containment descriptions $R \subseteq Q$ or equality descriptions $R = Q$, where *R* is one of the relations in the schema of the peer and *Q* is a query over the local sources of the peer.
- The relationship between peers is specified using three types of peer mappings: inclusion mappings, equality mappings, and definitional mappings, where

1. Each inclusion mapping is a containment $Q_1(A_1) \subseteq Q_2(A_2)$ between conjunctive queries $Q_1(A_1)$ and $Q_2(A_2)$, where A_1 andA_2 are subsets of the set of all relations in the schemas of the peers.
2. Each equality mapping is an equality $Q_1(A_1) = Q_2(A_2)$ between conjunctive queries $Q_1(A_1)$ and $Q_2(A_2)$ as above.
3. Each definitional mapping is a Datalog program with rules having single relations from the schemas of the peers in both the head and the body of each rule.

In the terminology of (Halevy et al. 2003), a data instance *D* of a PDMS *N* is an assignment of values to both the local sources of each peer and to the relations of the schema of each peer. A data instance *G* is consistent with *N* and *D* if *G* and *D* satisfy all the specifications given by the storage descriptions and the peer mappings of *N*. This concept captures what it means for a data instance *G* to be a solution for a given data instance *D* in the PDMS *N*.

In (Fuxman et al. 2005) the authors introduce and study a framework, called peer data exchange, for sharing and exchanging data between peers. This framework is a special case of a full-fledged peer data management system and a generalization of data exchange between a source schema and a target schema. The motivation behind peer data exchange is to model authority relationships between peers, where a source peer may contribute data to a target peer, specified using source-to-target constraints, and a target peer may use target-to-source constraints to restrict the data it is willing to receive, but cannot modify the data of the source peer.

A fundamental algorithmic problem in this framework is that of deciding the existence of a solution: given a source instance and a target instance for a fixed peer data exchange setting, can the target instance be augmented in such a way that the source instance and the augmented target instance satisfy all constraints of the setting? They investigate the computational complexity of the problem for peer data exchange settings in which the constraints are given by tuple generating dependencies. They show that this problem is always in NP, and that it can be NP-complete even for "acyclic" peer data exchange settings. They also show that the data complexity of the certain answers of target conjunctive queries is in coNP, and that it can be coNP-complete even for "acyclic" peer data exchange settings.

After this, they explore the boundary between tractability and intractability for the problem of deciding the existence of a solution. To this effect, they identify broad syntactic conditions on the constraints between the peers under which testing for a solution is solvable in polynomial time. These syntactic conditions include the important special case of peer data exchange in which the source-to-target constraints are arbitrary tuple generating dependencies, but the target-to-source constraints are local-as-view dependencies. Finally, they show that the syntactic conditions they identified are tight in the sense that minimal relaxations of them lead to intractability.

2.1.3 Classification according to the approach for modelling sources

Data integration system can be classified according to the way that the mappings *M* are specified between *G* and *S*. There are two main approaches: the Global-As-View approach (GAV) and the Local-As-View approach (LAV). Furthermore, hybrid approaches based on both GAV and BAV have been recently proposed.

2.1.3.1 Local Centric Approach

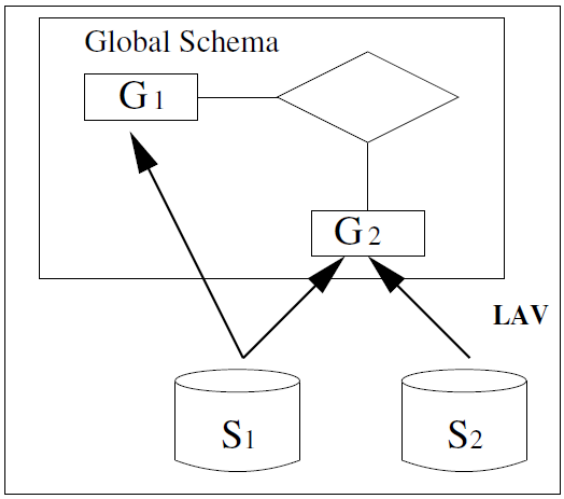


Fig. 7. Local as View approach

In the Local-As-View approach (LAV) (Levy et al. 1995) the global schema is defined independently of the local sources schemas. Each source is described in terms of the global schema relations. That is, the sources are described as materialized view of the global schema. In other words, the query language $L_{M,S,}$, allows only

expressions constituted by one symbol of the alphabet A_S. An overview of the LAV approach can be seen on Fig. **7** where S_1 can be seen as a view over concepts G_1 and G_2, while S_2 can be seen as a view over G_2. An example then is presented below.

Definition 2.7 (LAV mapping)**:** *A LAV mapping is a set of assertions, one for each element s of S of the form:*

$$s \rightarrow q_G$$

From the modelling point of view, the LAV approach is based on the idea that the content of each source s should be characterized in terms of a view q_G over the global schema. A notable case of this type is when the data integration is based on an ontology which will study in the next chapter.

To better characterize each source with respect to the global schema, several authors have proposed more sophisticated assertions (Abiteboul et al. 1998), (Grahne et al. 1999) in the LAV mapping, in particular with the goal of establishing the assumption holding for the various source extensions. Formally, this means that in the LAV mapping, a new specification denoted *as(s)*, is associated to each source element s. The specification *as(s)* determines how accurate is the knowledge on the data satisfying the sources, i.e how accurate is the source with respect to the associated view q_G. Three possibilities have been considered.

Sound views: When a source *s* is sound (denoted with *as(s)=sound*), its extension provides any subset of the tuples satisfying the corresponding view q_G. In other words given a source database *D*, from the fact that a tuple is in s^D one can conclude that it satisfies the associated view over the global schema, while from the fact that a tuple is not in s^D one cannot conclude that it does not satisfy the corresponding view. Formally, when *as(s)=sound*, a database *B* satisfies the assertion $s \rightarrow q_G$ with respect to *D* if:

$$s^D \subseteq q_G^B$$

Note that, from a logical point of view, a sound source s with arity n is modelled through the first order assertion

$$\forall x\, s(x) \rightarrow q_G(x)$$

Where *x* denotes variables $x_1, \ldots, x_n$.

Complete views: When a source *s* is complete (denoted with *as(s)=complete*), its extension provides any superset of the tuples satisfying the corresponding view. In other words, from the fact that a tuple is in s^D one cannot conclude that such a tuple satisfies the corresponding view. On the other hand, from the fact that a tuple is not in s^D one can conclude that such a tuple does not satisfy the view. Formally when *as(s)=complete*, a database *B* satisfies the assertion $s \rightarrow q_G$ with respect to *D* if:

$$s^D \supseteq q_G^B$$

Note that, from a logical point of view, a complete source *s* with arity *n* is modelled through the first order assertion

$$\forall x \; q_G(x) \rightarrow s(x)$$

Where *x* denotes variables $x_1, \ldots, x_n$.

Exact views: When a source *s* is exact (denoted with *as(s)=exact*), its extension is exactly the set of tuples satisfying the corresponding view. Formally when *as(s)=exact*, a database *B* satisfies the assertion $s \rightarrow q_G$ with respect to *D* if:

$$s^D = q_G^B$$

Note that, from a logical point of view, a complete source s with arity *n* is modelled through the first order assertion

$$\forall x \; s(x) \leftrightarrow q_G(x)$$

Where *x* denotes variables $x_1, \ldots, x_n$.

Typically, in the literature, when the specification of *as(s)* is missing, source s is considered sound. This is also the assumption we make in this document.

Due to the fact that the source relations are expressed as views over the global schema, each modification/addition/deletion of sources is costless since the local sources' schemas are the only things that should change. However, query rewriting in this approach is quite complex. The user of the data integration system poses a query in terms of the global schema and this query should be translated in terms of the local ones. An example is shown on Fig. 8 and Fig. 9.

Supposing that we are interested in the movies domain, we can imagine as a running example the following schemata.

Global Schema:

movie (*Title, Year, Director*)

european (*Director*)

review (*Title, Critique*)

Source 1:

r1 (*Title, Year, Director*) since 1960, European directors

r2 (*Title, Critique*) since 1990

Query:

Title and *critique* of movies in 1998

$\exists D.\ movie\ (T, 1998, D) \wedge review\ (T, R)$

$\{(T,R) \mid movie\ (T, 1998, D) \wedge review\ (T, R)\ \}$

Fig. 8. Running example schemata

Global Schema:

movie (*Title, Year, Director*)

european (*Director*)

review (*Title, Critique*)

Here associated to source relations we have views over the global schema

$r1\ (T,Y,D) \rightarrow \{\ (T,Y,D) \mid movie\ (T, Y, D) \wedge european\ (D) \wedge Y>1960\ \}$

$r2\ (T,R) \rightarrow \{\ (T,R) \mid movie\ (T, Y, D) \wedge review\ (T,R) \wedge Y>1990\ \}$

The query $\{(T,R) \mid movie\ (T, 1998, D) \wedge review(\ T, R)\ \}$ is processed by means of an inference mechanism that aims at re-expressing the atoms of the global schema in terms of atoms at the sources. In this case

$\{(T,R) \mid r2\ (T, R) \wedge r1\ (\ T, 1998, D)\ \}$

Fig. 9. Example 2.3

2.1.3.1.1 Query answering in LAV

When we answer a query over the global schema on the basis of a LAV mapping, we know only the extensions of the views associated to the sources, and this provides us with only partial information on the global database. As we already observed, in general, there are several possible global databases that are legal for the data integration system with respect to a given source database. This observation holds even for a setting where only sound views are allowed in the mapping. The problem is even more complicated when sources can be modelled as complete or exact views. In particular, dealing with exact sources essentially means applying the closed world assumption on the corresponding views.

Since in LAV, sources are modelled as views over the global schema, the problem of processing a query is traditionally called *view-based query processing*. Generally speaking, the problem is to compute the answer to a query based on a set of views, rather than on the raw data in the database.

There are two approaches to view-based query processing, called view-based query rewriting and view-based query answering, respectively.

View-based query processing: In the former approach, we are given a query q and a set of view definitions, and the goal is to reformulate the query into an expression of a fixed language L_R that refers only to the views and provides the answer to q. The crucial point is that the language in which we want the rewriting is fixed, and in general coincides with the language used for expressing the original query. In a LAV data integration setting, query rewriting aims at re-formulating, in a way that is independent from the current source database, the original query in terms of a query to the sources. Obviously, it may happen that no rewriting in the target language L_R exists that is equivalent to the original query. In this case, we are interested in computing also-called *maximally contained rewriting*, i.e., an expression that captures the original query in the best way.

View-based query answering: In view-based query answering, besides the query q and the view definitions, we are also given the extensions of the views. The goal is to compute the set of tuples t such that the knowledge on the view extensions logically implies that t is an answer to q, i.e., t is in the answer to q in all the databases that are consistent with the views. It is easy to see that, in a LAV data integration framework,

this is exactly the problem of computing the certain answers to q with respect to a source database.

Notice the difference between the two approaches. In query rewriting, query processing is divided in two steps, where the first one re-expresses the query in terms of a given query language over the alphabet of the view names, and the second one evaluates the rewriting over the view extensions. In query answering, we do not pose any limitations on how queries are processed, and the only goal is to exploit all possible information, in particular the view extensions, to compute the answer to the query.

A large number of results have been reported for both approaches. We first focus on view-based query answering. Query answering has been extensively investigated in the last years (Abiteboul et al. 1998). A comprehensive framework for view-based query answering, as well as several interesting results, is presented in (Grahne et al. 1999). The framework considers various assumptions for interpreting the view extensions with respect to the corresponding definitions (*closed*, *open*, and *exact* view assumptions).

Sound	CQ	$CQ^{\neq}$	PQ	Datalog	FOL
CQ	PTIME	coNP	PTIME	PTIME	undec
$CQ^{\neq}$	PTIME	coNP	PTIME	PTIME	undec
PQ	coNP	coNP	coNP	coNP	undec
Datalog	coNP	undec	coNP	undec	undec
FOL	undec.	undec	undec	undec	undec
Exact	CQ	$CQ^{\neq}$	PQ	Datalog	FOL
CQ	coNP	coNP	coNP	coNP	undec
$CQ^{\neq}$	coNP	coNP	coNP	coNP	undec
PQ	coNP	coNP	coNP	coNP	undec
Datalog	undec	undec	undec	undec	undec
FOL	undec	undec	undec	undec	undec

Fig. 10. Complexity of view based query answering

In (Abiteboul et a. 1998) , an analysis of the complexity of the problem under the different assumptions is carried out for the case where the views and the queries are expressed in terms of various languages (conjunctive queries without and with

inequalities, positive queries, Datalog, and first-order queries). The complexity is measured with respect to the size of the view extensions (data complexity). Fig. 10 summarizes the results presented in (Abiteboul et a. 1998). Note that, for the query languages considered in that paper, the exact view assumption complicates the problem. For example, the data complexity of query answering for the case of conjunctive queries is PTIME under the sound view assumption, and coNP-complete for exact views. This can be explained by noticing that the exact view assumption introduces a form of negation, and therefore it may force to reason by cases on the objects stored in the views.

Considering view-based query rewriting, several papers investigate the rewriting question for different classes of queries. The problem is investigated for the case of conjunctive queries (with or without arithmetic comparisons) in (Levy et al. 1995), for disjunctive views in (Afrati et al. 1999), for queries with aggregates in (Cohen et al. 1999), (Grumback et al. 1999), for recursive queries and non-recursive views in (Duschka et al. 1997b), for queries expressed in Description Logics in (Beeri et al. 1997), for regular-path queries and their extensions in (Calvanese et al. 1998), (Calvanese et al. 2000e), (Calvanese et al. 2000f) and in the presence of integrity constraints in (Duschka et al. 1997b).

We already noted that view-based query rewriting and view-based query answering are different problems. Unfortunately, their similarity sometimes gives raise to a sort of confusion between the two notions. Part of the problem comes from the fact that when the query and the views are conjunctive queries, the best possible rewriting is expressible as union of conjunctive queries, which is basically the same language as the one of the original query and views. However, for other query languages this is not the case. Abstracting from the language used to express the rewriting, we can define a rewriting of a query with respect to a set of views as a function that, given the extensions of the views, returns a set of tuples that is contained in the answer set of the query in every database consistent with the views. We call the rewriting that returns precisely such set the perfect rewriting of the query with respect to the views. Observe that, by evaluating the perfect rewriting over given view extensions, one obtains the same set of tuples provided by view-based query answering. i.e., in data integration terminology, the set of certain answers to the query with respect to the view extension. Hence, the perfect rewriting is the best rewriting

one can obtain, given the available information on both the definitions and the extensions of the views.

An immediate consequence of the relationship between perfect rewriting and query answering is that the data complexity of evaluating the perfect rewriting over the view extensions is the same as the data complexity of answering queries using views. Typically, one is interested in queries that can be evaluated in PTIME (i.e., are PTIME functions in data complexity), and hence we would like rewritings to be PTIME as well. For queries and views that are conjunctive queries (without union), the perfect rewriting is a union of conjunctive queries and hence is PTIME. However, already for very simple query languages containing union the perfect rewriting is not PTIME in general. Hence, for such languages it would be interesting to characterize which instances of query rewriting admit a perfect rewriting that is PTIME. By establishing a tight connection between view-based query answering and constraint-satisfaction problems, it is argued in (Calvanese et al. 2000e) that this is a difficult task.

2.1.3.1.2 Query containment in LAV

Recent work addresses the problem of reasoning on queries in data integration systems. The basic form of reasoning on queries is checking containment, i.e., verifying whether one query returns a subset of the result computed by the other query in all databases.

Definition 2.8 (Query containment): *A query Q_1 is said to be contained in a query Q_2, denoted by $Q_1 \subseteq Q_2$, if for all database instances D, the set of tuples computed for Q_1 is a subset of those computed for Q_2. The two queries are said to be equivalent if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.*

Besides the usual notion of containment, several other notions have been introduced related to the idea of comparing queries in a data integration setting, especially in the context of the LAV approach.

In (Millstein et al. 2000), a query is said to be contained in another query relative to a set of sources modelled as views, if, for each extension of the views, the certain answers to the former query are a subset of the certain answers to the latter. Note that this reasoning problem is different from the usual containment checking:

here we are comparing the two queries with respect to the certain answers computable on the basis of the views available. The difference becomes evident if one considers a counterexample to relative containment: Q_1 is not contained in Q_2 relative to views V if there is a tuple t and an extension E of V, such that for each database DB consistent with E (i.e., a database DB such that, the result V^{DB} of evaluating the views over DB is exactly E), t is an answer of Q_1 to DB, but there is a database DB' consistent with E such that t is not an answer of Q_2 to DB'. In other words, Q_1 is not contained in Q_2 relative to views V if there are two databases DB and DB' such that $V^{DB} = V^{DB'}$ and $Q_1^{DB} = Q_2^{DB'}$

In (Millstein et al. 2000) it is shown that the problem of checking relative containment is Π_2^P -complete in the case of conjunctive queries and views. In (Li et al. 2001), the authors introduce the notion of "p-containment", where *p* stands for power.

Definition 2.9 (p-containment)**:** *A view set V is said to be p-contained in another view set W, i.e W has at least the answering power of V, if W can answer all queries that can be answered using V.*

One of the ideas underlying the above mentioned papers is the one of losslessness: a set of views is lossless with respect to a query, if, no matter what the database is, we can answer the query by solely relying on the content of the views. This question is relevant for example in mobile computing, where we may be interested in checking whether a set of cached data allows us to derive the requested information without accessing the network, or in data warehouse design, in particular for the view selection problem, where we have to measure the quality of the choice of the views to materialize in the data warehouse. In data integration, losslessness may help in the design of the data integration system, in particular, by selecting a minimal subset of sources to access without losing query-answering power. The definition of losslessness relies on that of certain answers:

Definition 2.10 (Lossless views): *A set of views is lossless with respect to a query, if for every database, we can answer the query over that database by computing the certain answers based on the view extensions.*

It follows that there are at least two versions of losslessness, namely, losslessness under the sound view assumption, and losslessness under the exact view assumption. The first version is obviously weaker than the second one. If views *V* are lossless with respect to a query *Q* under the sound view assumption, then we know that, from the intensional point of views, *V* contain enough information to completely answer *Q*, even though the possible incompleteness of the view extensions may prevent us from obtaining all the answers that *Q* would get from the database. On the other hand, if *V* are lossless with respect to a query *Q* under the exact view assumption, then we know that they contain enough information to completely answer *Q*, both from the intensional and from the extensional point of view.

2.1.3.2 Global Centric Approach

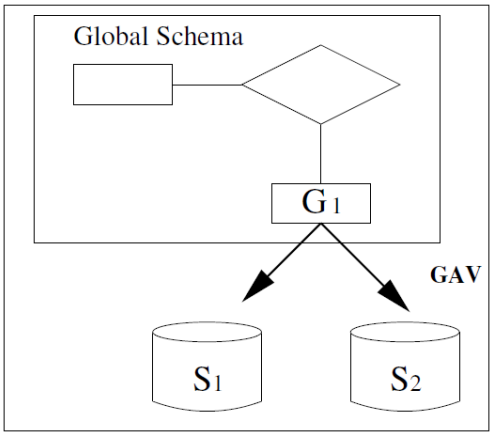


Fig. 11. The global centric approach

In the Global-As-View approach (GAV) (Ullman et al 2000), the global schema is expressed in terms of the local data sources. That is, the global schema is defined as a view over the local sources' schemas. An overview of the approach is illustrated in Fig. 11, where concept G_1 of the global schema is expressed in terms of the relations S_1 and S_2 of the local database sources. If both these schemas are relational, then one can write a rule-based conjunctive query over the source relations. This query specifies how to obtain the tuples for the global schema relations. Each query specifies that in order to compute the tuples in the relation in the head of the rule, one has to go to the body of the rule and compute whatever is specified there. The attributes appearing in the head indicate that they are the attributes of interest, thus the others (in the body) can be projected out at the end.

In the GAV approach, the mapping *M* associates to each element *g* in *G* a query q_S over *S*. In other words, the query language $L_{M,G}$ allows only expressions constituted by one symbol of the alphabet A_G. Therefore, a GAV mapping is a set of assertions, one for each element *g* of *G*, of the form:

$$g \rightarrow q_S$$

From the modelling point of view, the GAV approach is based on the idea that the content of each element *g* of the global schema should be characterized in terms of a view q_S over the sources. In some sense, the mapping explicitly tells the system how to retrieve the data when one wants to evaluate the various elements of the global schema. This idea is effective whenever the data integration system is based on a set of sources that is stable. Note that, in principle, the GAV approach favours the system in carrying out query processing, because it tells the system how to use the sources to retrieve data. However, extending the system with a new source is now a problem: the new source may indeed have an impact on the definition of various elements of the global schema, whose associated views need to be redefined.

To better characterize each element of the global schema with respect to the sources, more sophisticated assertions in the GAV mapping can be used, in the same spirit as we saw for LAV. Formally, this means that in the GAV mapping, a new specification, denoted *as(g)* (either *sound*, *complete*, or *exact*) is associated to each element *g* of the global schema. When *as(g) = sound* (resp., *complete*, *exact*), a database *B* satisfies the assertion $g \rightarrow q_S$ with respect to a source database *D* if:

$$q_S^D \subseteq g^B \qquad (\text{resp. } q_S^D \supseteq g^B,\ q_S^D = g^B)$$

The logical characterization of sound views and complete views in GAV is therefore through the first order assertions:

$$\forall x\ q_S(x) \rightarrow g_S(x)$$

respectively.

It is interesting to observe that the implicit assumption in many GAV proposals is the one of exact views. Indeed, in a setting where all the views are exact, there are no constraints in the global schema, and a first order query language is used as $L_{M,S}$, a GAV data integration system enjoys what we can call the "single database property", i.e., it is characterized by a single database, namely the global database that is obtained by associating to each element the set of tuples computed by the

corresponding view over the sources. This motivates the widely shared intuition that query processing in GAV is easier than in LAV. However, it should be pointed out that the single database property only holds in such a restricted setting.

In particular, the possibility of specifying constraints in *G* greatly enhances the modelling power of GAV systems, especially in those situations where the global schema is intended to be expressed in terms of a conceptual data model, or in terms of an ontology. In these cases, the language L_G is in fact sufficiently powerful to allow for specifying, either implicitly or explicitly, various forms of integrity constraints on the global database.

In general, the views associated to the elements of the global schema are considered sound, i.e. all the data provided by a view satisfies the corresponding element of the global schema, but there may be additional data satisfying the element not provided by the view.

It is an implicit assumption (Lenzerini et al. 2002), in many GAV proposals, that the assertions above are exact. This assumption is true when in the global schema there are no additional constraints. Under these circumstances, the query rewriting in GAV approach is quite easy (it is illustrated in the Fig. **12**). However, the possibility of specifying constraints in the global schema enhances the expressing power of GAV systems.

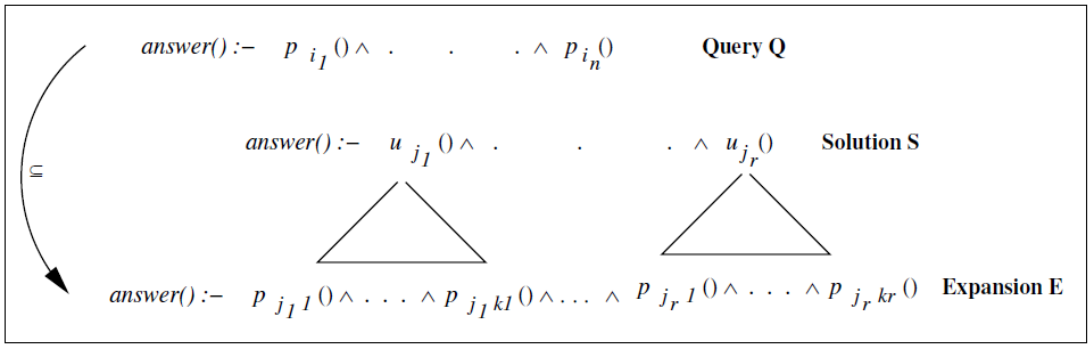


Fig. 12. Query Unfolding

As depicted in Fig. 12 the Global-As-View approach has the advantage of simple query rewriting. Due to the fact that the global schema is expressed in terms of the local schemas query rewriting consists of replacing each atom of the query with its definition. In this way the query is finally expressed in terms of the local sources. This substitution is called *query unfolding*.

2.1.3.2.1 Query answering in GAV

Most GAV data integration systems do not allow integrity constraints in the global schema, and assume that views are exact. It is easy to see that, under these assumptions, query processing can be based on a simple unfolding strategy. When we have a query q over the alphabet A_G of the global schema, every element of A_G is substituted with the corresponding query over the sources, and the resulting query is then evaluated at the sources. As we said before, such a strategy suffices mainly because the data integration system enjoys the single database property. Notably, the same strategy applies also in the case of sound views.

However, when the language L_G used for expressing the global schema allows for integrity constraints, and the views are sound, then query processing in GAV systems becomes more complex. Indeed, in this case, integrity constraints can in principle be used in order to overcome incompleteness of data at the sources.

The assumption of sound views asserts that the tuples retrieved for a relation r are a subset of the tuples that the system assigns to r; therefore, we may think of completing the retrieved global database by suitably adding tuples in order to satisfy foreign key constraints, while still conforming to the mapping. When a foreign key constraint is violated, there are several ways of adding tuples to the retrieved global database to satisfy such a constraint. In other words, in the presence of foreign key constraints in the global schema, the semantics of a data integration system must be formulated in terms of a set of databases, instead of a single one. Since we are interested in the certain answers $q^{I;D}$ to a query q, i.e., the tuples that satisfy q in all global databases that are legal for I with respect to D, the existence of several such databases complicates the task of query answering. In (Cali et al. 2002), a system called IBIS is presented, that takes into account key and foreign key constraints over the global relational schema. The system uses the foreign key constraints in order to retrieve data that could not be obtained in traditional data integration systems. The language for expressing both the user query and the queries in the mapping is the one of union of conjunctive queries. To process a query q, IBIS expands q by taking into account the foreign key constraints on the global relations appearing in the atoms. Such an expansion is performed by viewing each foreign key constraint $r1[X] \subseteq r2[Y]$, where X and Y are sets of h attributes and Y is a key for r_2, as a logic programming rule

$$r_2'(\vec{X}, f_{h+1}(\vec{X}), \ldots, f_n(\vec{X})) \leftarrow r_1'(\vec{X}, X_{h+1} \ldots, X_m)$$

where each f_i is a Skolem function, $\vec{X}$ is a vector of *h* variables, and we have assumed for simplicity that the attributes involved in the foreign key are the first *h* ones.

Global Schema:

movie (*Title, Year, Director*)

european (*Director*)

review (*Title, Critique*)

Here associated to relations in the global schema we have views over the sources

movie (T,Y,D) → *{ (T,Y,D) | r1 (T,Y,D) }*

european (D) → *{ (D) | r1 (T,Y,D)}*

review (T,R) → *{ (T,R) | r2(T,R)}*

Global schema containing constraints:

m*ovie* (*Title, Year, Director*)

e*uropean* (*Director*)

review (*Title, Critique*)

european_movie_60s (*Title, Year, Director*)

$\forall$ *T,Y,D. european_movies_60s (T,Y,D)* $\supset$ *movie (T,Y,D)*

$\exists$ *T, Y, D. european_movies_60s (T,Y,D)* $\supset$ *European (D)*

GAV mappings :

european_movies_60s (T,Y,D) → *{ (T,Y,D) | r1 (T,Y,D)}*

european (D) → *{ (D) | r1(T,Y,D)}*

review(T,R) → *{ (T,R) | r2 (T,R)}*

Query processing :

The query {(*T*,*R*) | *movie* (*T*, 1998, *D*) $\wedge$ *review* (*T*, *R*) } is processed by means of unfolding. In this case it becomes: *r1*(*T*,1998,*D*) $\wedge$ *r2*(*T*,*R*)

Fig. 13. GAV example

Each r_i' is a predicate, corresponding to the global relation r_i, defined by the above rules for foreign key constraints, together with the rule

$$r_i'(X_1, \ldots, X_n) \leftarrow r_i(X_1, \ldots, X_n)$$

An example is shown on Fig. 13, where the global and the local schemata are shown and an example of query unfolding as well.

Once such a logic program Π_G has been defined, it can be used to generate the expanded query *expand*(*q)* associated to the original query *q*. This is done by performing a partial evaluation with respect to Π_G of the body of *q′*, which is the query obtained by substituting in *q* each predicate r_i with r_i'. In the partial evaluation tree, a node is not expanded anymore either when no atom in the node unifies with a head of a rule, or when the node is subsumed by (i.e., is more specific than) one of its predecessors. In the latter case, the node gets an empty node as a child; intuitively this is because such a node cannot provide any answer that is not already provided by its more general predecessor.

These conditions guarantee that the construction of the partial evaluation tree for a query always terminates. Then, the expansion *expand(q)* of *q* is a union of conjunctive queries whose body is constituted by the disjunction of all nonempty leaves of the partial evaluation tree. It is possible to show that, by unfolding *expand(q)* according to the mapping, and evaluating the resulting query over the sources, one obtains exactly the set of certain answers of *q* to *I* with respect to *D*.

The major drawback of this approach is its lack of flexibility with respect to the addition/deletion of the sources to the data integration system, or the modification of the sources schemas. This is due to the fact that each modification of a local source schema results in modification of global schema.

2.1.3.3 Combining Global and Local Approach

As discussed earlier, both GAV and LAV have some drawbacks that should be overcome. Thus, an approach has been proposed that combines global and local approach. It is called GLAV (Friedman et al.1999). In this approach we are able to express a local source in terms of the global schema (LAV), a global source in terms of the local sources (GAV) and additionally a whole view (``part'') of the global schema in terms of the local sources. An overview of this approach can be seen in Fig.

14 above where the whole ``relation" of G_1 and G_2 is described by the local sources relations, S_1 and S_2.

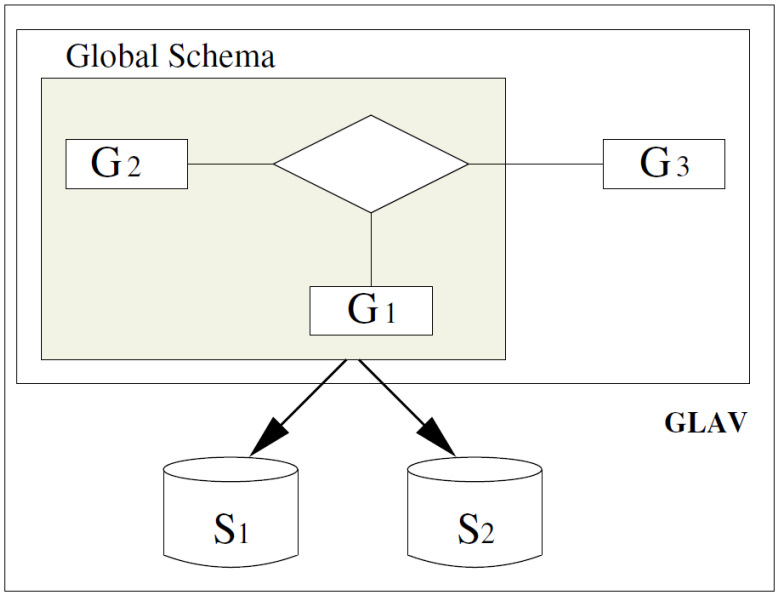


Fig. 14. GLAV approach

Definition 2.11 (A GLAV mapping): *The mappings M, in GLAV approach, are in the form:*

$$G_1(X_1, Z_1), G_2(X_2, Z_2), \ldots, G_j(X_j, Z_j) \longleftarrow V(X, Y)$$

where $X = \bigcup_i X_i$, $(\bigcup_i Z_i) \bigcap Y = 0$, G_i *are global relations and* $V(X, Y)$ *is a conjunction of source relations.*

An example scenario is shown in Fig. 15 where obviously the mappings combine a query over the sources on the left-hand side, with a query over the global schema on the right-hand side.

Definition 2.12: *The mapping M between the global schema and the sources, constitutes of the following assertions:*

$$R^B \supseteq V^D \text{ (sound source)}$$

or

$$R^B \subseteq V^D \text{ (complete source)}$$

or

$$R^B \equiv V^D \text{ (exact source)}$$

where R is a view (query) of the global schema, and V is a view (query) over the local sources.

Global Schema:

works (Person, Project)

area (Project, Field)

Source 1:

hasJob (Person, Field)

Source 2:

teach (Professor, Course)

in (Course, Field)

Source 3:

get (Researcher, Grant)

for (Grant, Project)

GLAV mapping:

$\{ (r,f) \mid hasJob(r,f)\} \rightarrow \{ (r,f) \mid works(r,p) \wedge area(p,f)\}$

$\{ (r,f) \mid teach(r,c) \wedge in(c,f)\} \rightarrow \{ (r,f) \mid works(r,p) \wedge area(p,f)\}$

$\{ (r,p) \mid get(r,g) \wedge for(g,p)\} \rightarrow \{ (r,p) \mid works(r,p)\}$

Fig. 15. GLAV example

The GLAV approach combines the expressive power of GAV and LAV. Additionally, it reaches the limits of the expressive power of a data source description language. This is because slight additions to the expressive power of GLAV would make query answering co-NP-hard in the size of the data in the sources. Query rewriting in this approach is shown to be no harder than it is for the LAV approach. It should be noted that GLAV is also of interest for data integration independent of data sources, because of the flexibility it provides in integrating heterogeneous sources.

2.1.3.4 Another hybrid approach:BAV

BAV is another data integration approach. BAV transformation sequences are partially derived from LAV or GAV view definitions. BAV is a rich integration framework, which is based on the use of reversible sequences of primitive schema transformations, called transformation pathways. It is an expressive data integration language, since it allows the expression of mappings in both directions. Another major advantage of using the BAV approach is that it supports the evolution of both

global and local schemas, in contrast to taking either a GAV or LAV approach. With the BAV approach it becomes possible to extract a definition of the global schema as a view over the local schemas and vice versa. BAV combines the benefits of LAV and GAV in the sense that any reasoning or processing which is possible with the view definitions of GAV or LAV will also be possible with the BAV definition. However, BAV is likely to be more costly to reason with and process than the corresponding LAV, GAV or GLAV view definitions would be. The most representative system that implements this approach is AutoMed (Boyd et al. 2004)

2.1.3.5 Comparison of the approaches

In conclusion, there are some interesting facts that should be noted about the approaches above. It is true that both LAV and GAV, which are the first approaches proposed, have advantages and disadvantages. LAV is really flexible in addition/deletion of the local sources that participate in the integration system; this is the main drawback of the GAV approach, since every addition/deletion leads to a new rewriting of the global schema description. Exactly the same occurs when it is necessary to add some more complicated constraints on sources, since LAV demands only the addition of the necessary changes to the source, while GAV demands the rewriting of the global schema description. These constraints usually concern the availability of the data during querying the sources. However, query answering is quite simple in GAV, whereas it is harder in LAV. In the GAV approach, query rewriting can be achieved by unfolding the source descriptions of the global ``parts'' of the query. In the LAV approach, this is not feasible since such descriptions are not available.

A first attempt to analyze the similarities and differences between GAV and LAV approach can be found in (Cali et al. 2001), (Cali et al. 2002), where the authors address the problem of checking whether a LAV system can be transformed into a GAV one, and vice-versa. They deal with transformations that are equivalent with respect to query answering, i.e., that enjoy the property that queries posed to the original system have the same answers when posed to the target system. Results on query reducibility from LAV to GAV systems may be useful, for example, to derive a procedural specification from a declarative one. Conversely, results on query reducibility from GAV to LAV may be useful to derive a declarative characterization

of the content of the sources starting from a procedural specification. We briefly discuss the notions of query-preserving transformation, and of query-reducibility between classes of data integration systems.

Definition 2.13 (query preserving data integration system)*: Given two integration systems $I = < G, S, M>$ and $I' = < G', S, M'>$ over the same source schema S and such that all elements of G are also elements of G', I' is said to be query-preserving with respect to I, if for every query q to I and for every source database D, we have that:*

$$q^{I,D} = q^{I',D}$$

In other words, *I'* is query-preserving with respect to *I* if, for each query over the global schema of *I* and each source database, the certain answers to the query with respect to the source database that we get from the two integration systems are identical.

Definition 2.14: *A class C_1 of integration systems is query-reducible to a class C_2 of integration systems if there exist a function $f : C_1 \rightarrow C_2$ such that, for each $I_1 \in C_1$ we have that $f(I_1)$ is query-preserving with respect to I_1.*

With the two notions in place, the question of query reducibility between LAV and GAV is studied in (Cali et al. 2002) within a setting where views are considered sound, the global schema is expressed in the relational model, and the queries used in the integration systems (both the queries on the global schema, and the queries in the mapping) are expressed in the language of conjunctive queries. The results show that in such a setting none of the two transformations is possible. On the contrary, if one extends the framework, allowing for integrity constraints in the global schema, then reducibility holds in both directions. In particular, inclusion dependencies and a simple form of equality-generating dependencies suffice for a query-preserving transformation from a LAV system into a GAV one, whereas single head full dependencies are sufficient for the other direction. Both transformations result in a query-preserving system whose size is linearly related to the size of the original one.

In the GLAV approach, the source evolution and addition/removal is easier, since, in fact, both directions are implemented (LAV and GAV) and in this case it is

appropriate to use the LAV approach. Additionally, query answering is not harder than in LAV. However, GLAV gives the ability of more expressive mappings.

An interesting observation, as stated in (Cali et al. 2002b), is that, under certain circumstances, the existence of constraints (e.g., keys or foreign keys) in the global schema can turn the GLAV mappings into GAV ones, and thus take benefit of the query reformulation algorithm proposed for the GAV approach. Unfortunately, despite their expressiveness, GLAV mappings introduce new challenges. Further, works (Madhavan et al. 2003) has shown that the composition of GLAV mappings may be undecidable in certain cases (e.g., the composition of GLAV rules, which map non CQ_k (CQ_k queries is the class of conjunctive queries in which every nested expression has at most k variables) queries over a source schema to non-CQ_k queries over a target schema, may result in an infinite set of mappings). Additionally, as illustrated in (Fagin et al. 2005b) the composition of two GLAV rules does not always imply a new GLAV rule that maps a conjunctive query to another conjunctive query. There are cases (e.g., when we compose two finite sets of non full[1] source-to-target dependencies used for the interpretation of the mappings) where the composition is definable only with the use of existential second-order formulas. In these formulas, new function symbols that guarantee the presence of the existentially quantified variables appearing in the dependencies, are introduced.

BAV on the other hand, combines the benefits of LAV and GAV in the sense that any reasoning or processing which is possible with the view definitions of GAV or LAV will also be possible with the BAV definition. However, BAV is likely to be more costly to reason with and process than the corresponding LAV, GAV or GLAV view definitions would be.

2.2 Ontology based Data Integration

2.2.1 What is an ontology?

Ontologies can play a key role in the task of knowledge exchange acting as Enterprise models. Originally introduced by Aristotle, ontologies are formal models

[1] A dependency is *full* if no existentially quantified variables occur in it.

about how we perceive a domain of interest and provide a precise, logical account of the intended meaning of terms, data structures and other elements modelling the real world. As such, they are often viewed as the key means through which the SemanticWeb vision (Berners-Lee et al. 2001) can be realized and have already found several applications in the area of Knowledge Representation (KR) and in the Semantic Web. Ontologies are so important in the Semantic Web because they provide a means to formally define the basic terms and relations that comprise the vocabulary of a certain domain of interest (Lambrix & Edberg 2003), enabling machines to process information provided by human agents. As a result, ontologies can help in the representation of the content of a web page in a formal manner, so as to be suitable for use by an automated computer agent, crawler, search engine or other web service. The importance of ontologies in current Artificial Intelligence (AI) research is also emphasized by the interest shown by both the research and the enterprise community to various problems related to ontologies and ontology manipulation (McGuiness et al. 2000).

The term ontology has come to refer to a wide range of formal representations, including taxonomies, hierarchical terminology vocabularies or detailed logical theories describing a domain (Noy & Klein 2004). For this reason, a precise definition of the term is rather difficult and different definitions have appeared in the literature (see, for example, (Gruber 1993a), (Guarino 1998)). One commonly used definition is based on the original use of the term in philosophy, where an ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented (Gruber 1993b); therefore, an ontology in the AI context is a structure that specifies a conceptualization, or, more accurately, a formal specification of a shared conceptualization of a domain (Gruber 1993a).

A more formal, algebraic, approach, identifies an ontology as a pair $< S, A >$, where S is the vocabulary (or signature) of the ontology (being modelled by some mathematical structure, such as a poset, a lattice or an unstructured set) and A is the set of ontological axioms, which specify the intended interpretation of the vocabulary in a given domain of discourse (Kalfoglou & Schorlemmer 2003). A similar definition is given in (De Bruijn et al. 2004), where the signature S is broken down in three (not necessarily disjoint) sets, the set of concepts (C), the set of relations (R) and the set of instances (I); thus, an ontology is defined as a 4-tuple $< C, R, I, A >$.

Ontologies are best used in applications where the core problem is the use and management of common representations. Many applications have been developed for instance in bio-informatics, or for knowledge management purposes inside organizations. Local data models (a.k.a *contexts* (Bouquet et al. 2004)), instead, are best used in those applications where the problem is the use and management of local and autonomous representations with a need for a limited and controlled form of globalization (or, using the terminology used in the context literature, maintaining locality still guaranteeing semantic compatibility among representations). Contexts and ontologies have both strengths and weaknesses. It can be argued that the strengths of the ontologies are the weaknesses of contexts and vice versa. On the one hand, the use of ontologies enables the parties to communicate and exchange information. Shared ontologies define a common understanding of specific terms, and thus make it possible to communicate between systems on a semantic level. On the weak side, ontologies can be used only as long as consensus about their contents is reached. Furthermore, building and maintaining them may become arbitrary hard, in particular in a very dynamic, open and distributed domain like the web. On the other hand, contexts encode not shared interpretation schemas of individuals or groups of individuals. Contexts are easier to define and to maintain. They can be constructed with no consensus with the other parties, or only with the limited consensus which makes it possible to achieve the desired level of communication and only with the relevant parties. On the weak side, since contexts are local to parties, communication can be achieved only by constructing explicit mapping among the elements of the contexts of the involved parties; and extending the communication to new topics and/or new parties requires the explicit definition of new mappings.

2.3.1 Ontologies as Enterprise Models

As Calvanese et al. (Calvanese et al. 1998b) propose (see Fig. 16), in the conceptual layer of an information integration problem we distinguish the Enterprise or Target Model and multiple Source Models. The Enterprise or Target Model is a conceptual representation of the global concepts and relationships that are of interest to the application. The Source Model of an information source is a conceptual representation of the data residing in underlying information sources, or at least of the portion of data currently taken into account.

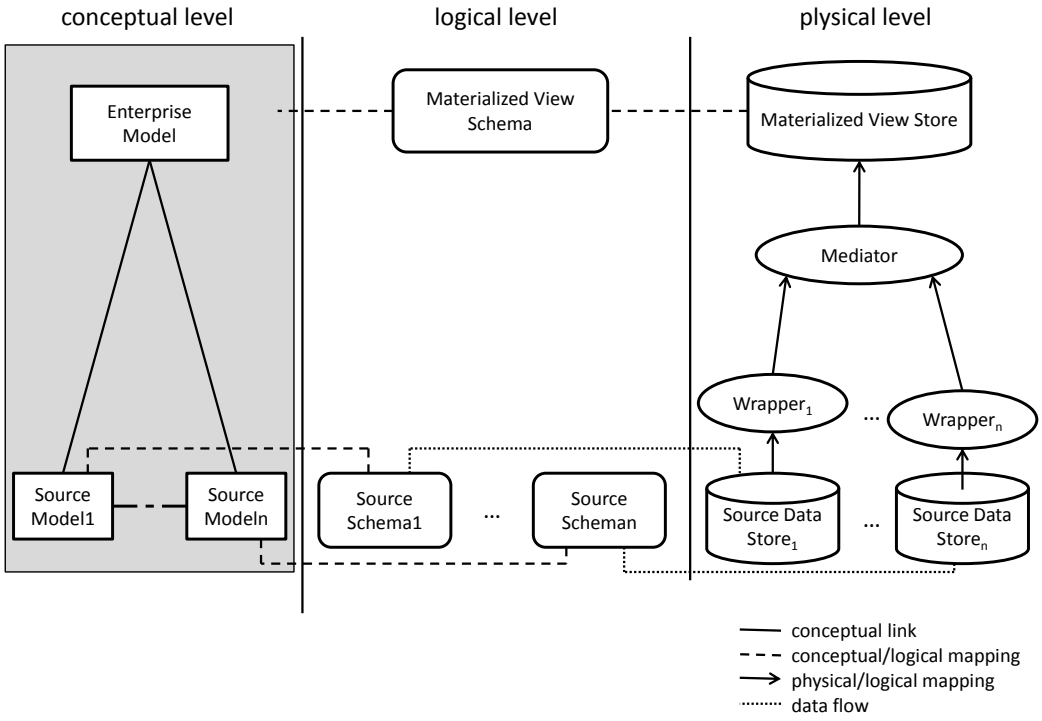


Fig. 16. Architecture for Data Integration

In order to achieve logical transparency using this model the global schema should provide a conceptual view, as shown on Fig. 17, that is independent from the sources, that is described with a semantically rich formalism. In this context, the need for explicit models of semantic information in order to support information exchange has been widely acknowledged by the research community.

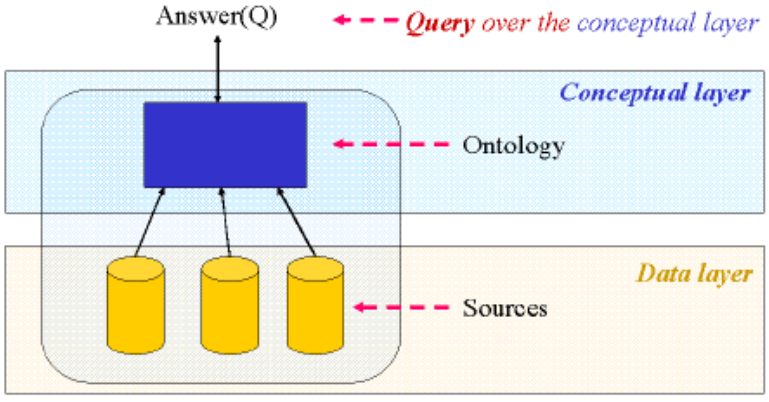


Fig. 17. Data Integration though an ontology

Now a data integration system I is a triple (G, S, M) where

- *G* is the global schema – now is an ontology. The global schema is a logical theory over an alphabet A_g.
- *S* is the source schema. The source schema is constituted simply by an alphabet A_s disjoint from A_g.
- *M* is the mapping between *S* and *G*.

The proposed language for expressing ontologies is OWL, which is also advocated by the Semantic Web community. Description Logic languages such as the OWL are considered the fundamental formal tool for expressing ontologies. Typical reasoning tasks in DLs are classification, subsumption, instance checking, all based on logical inference.

Now the question is whether view based query answering (of conjunctive queries) is decidable or not if we use an expressive description logics such OWL to express the ontology. The answer is that this can be done in 2EXPTIME in combined complexity.

We consider query answering in the following setting:

- Data (i.e ABox *A)* are incomplete and assumed to be large (their size dominates the size of schema)
- Schema (i.e TBox *T*) constraints the possible models
- Query *q* is a complex expression (conjunctive query)

Here the task is to compute $cert(q, T, A) = \{ c \mid T \cup A \models q(c) \}$ as shown also on Fig. 18

Fig. 18. Query answering

2.3.2 Single, Multiple & Hybrid Ontology Approaches

Ontologies can be used as the global schema and it seems that database integration is currently evolving towards this direction. By accepting an ontology as a

point of common reference, naming conflicts are eliminated and semantic conflicts are reduced. Below, we review a few recent ontology-based integration projects.

However, there are different ways of how to employ the ontologies. In general, three different directions can be identified: single ontology approaches, multiple ontology approaches and hybrid approaches.

The integration based on a single ontology seems to be the simplest approach because it can be simulated by the other approaches. Single ontology approaches use one global ontology providing a shared vocabulary for the specification of semantics and all information sources are related to that ontology. Single ontology approaches can be applied to integration problems where all information sources to be integrated provide nearly the same view on a domain. But if one information source has a different view on the domain, e.g. by providing another level of granularity, finding the minimal ontology commitment (Gruber, 1993b) becomes a difficult task. Also single ontology approaches are susceptible to changes in the information sources which can affect the conceptualization of the domain represented in the ontology. Depending on the nature of the changes in one information source it can imply changes in the global ontology and in the mappings to other information sources. These disadvantages led to the development of multiple ontology approaches.

In multiple ontology approaches, each information source is described by its own ontology. In principle, the "source ontology" can be a combination of several other ontologies but it cannot be assumed that the different "source ontologies" share the same vocabulary. At first, the advantage of multiple ontology approaches seems to be that no common and minimal ontology commitment about the global ontology is needed. Each source ontology could be developed without respect to other sources or their ontologies – no common ontology with the agreement of all sources are needed. This ontology architecture can simplify the change, i.e modifications in one information source or the addition/removal of sources. However, in reality the lack of a common vocabulary makes it extremely difficult to compare different source ontologies. To overcome this problem, an additional representation formalism defining the inter-ontology mapping should be provided. The inter-ontology mapping identifies semantically corresponding terms of different source ontologies. However, the mapping also has to consider different views on a domain, e.g. different

aggregation and granularity on the ontology concepts and it is really difficult to be defined in practice.

To overcome the drawbacks of both single and multiple ontology approaches, hybrid approaches were developed. Similar to multiple ontology approaches the semantics of each source is described by its own ontology. But in order to make the source ontologies comparable to each other they are built upon one global shared vocabulary. The shared vocabulary contains basic terms of a domain and in order to build complex terms of a source ontology, the primitives are combined by some operators. Since each term of a source ontology is based on the primitives, the terms become easier to compare than in multiple ontology approaches. Usually, the shared vocabulary is also an ontology. The advantage of a hybrid approach is that new sources can be easily added without the need of modification in the mappings or the shared vocabulary. It also supports the acquisition and evolution of ontologies. The use of a shared vocabulary makes the source ontologies comparable and avoids the disadvantages of multiple ontology approaches. However, existing ontologies cannot be reused easily, but have to be re-developed from scratch. Representative system of this category is MECOTA (Wache et al. 1999).

2.3.3 Representative Ontology based Data Integration Systems

In BACIIS (Ben Miled et al. 2005) and TAMBIS (Stevens et al 2000) , a single conceptualization is provided trying to capture the information from the system data sources. User queries are built and results are returned in terms of this global conceptual schema. However, any change in the sources may require the modification of the global domain conceptualization. Specifically, in TAMBIS, the integration process is restricted to combine data from sources that contain different types of information for the same semantic entity, since it does not take into account the potential overlapping aspect of sources or the probable incompleteness of some of them. Moreover, BACIIS only integrates Web Databases and the mappings are based on text parsing from web pages.

2.3.3.1 D2R Map

DR2 Map (Bizer, 2003) is a declarative and XML-based language. It allows describing mappings between relational database schemata and OWL/RDFS

ontologies. With DR2, users can create flexible mappings of complex relational structures without having to change the existing database schema, which is achieved by applying SQL statements directly on the mapping rules. The DR2 processor is responsible for the mapping process, which is performed in four logical steps:

- A record set is selected from the database, based on class similarity.
- The record set is grouped according to the groupBy columns.
- Class instances are created.
- The record set data is mapped to instance properties.

DR2 MAP is kept as simple as possible, expressing mappings with just three elements. Fig. 19 shows the mapping process used in DR2 MAP.

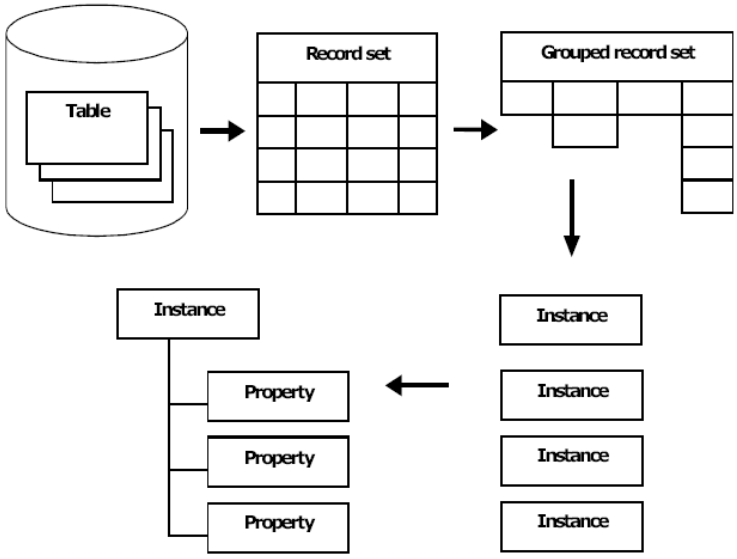


Fig. 19. The D2R Mapping Process

2.3.3.2 KAON

KAON (Volz, 2003) is an open source Tool suite that provides a multitude of software modules specially designed for the semantic web. It includes a persistent RDF store, an ontology store, ontology editors, etc. It has been developed as a result of a joint effort by the institute AIFB (University of Karlsruhe) and the Research Center of Information Technologies (FZI).

KAON offers an ontology management infrastructure, mainly targeted at business applications. It allows creating and managing ontologies easily and provides a framework aimed at building ontology-based applications.

KAON Reverse tool offers the possibility of mapping relational databases to ontologies, enabling two tasks: updating databases contents and performing queries

through the conceptualization of a database. One drawback of this tool is that changes cannot be applied to the structure of the database with respect to the ontology, since the whole process should be repeated. This work is not reusable.

The kernel of this suite is the KAON SERVER, which brings all the software modules together. KAON SERVER is implemented with the Java programming language. The Java Management Extensions (JMX) are used to manage and monitor all the resources KAON handles. Fig. 20 shows KAON architecture.

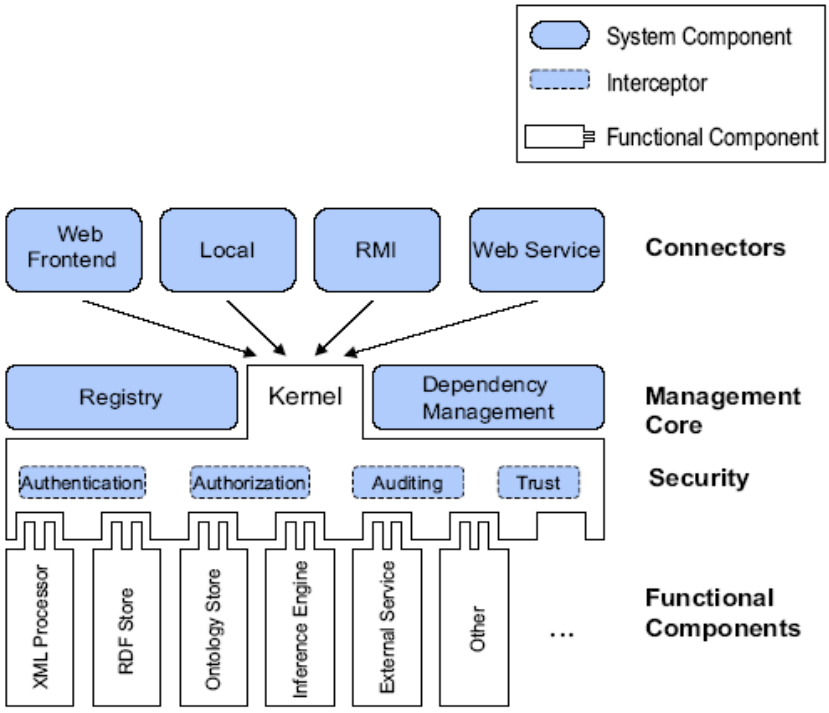


Fig. 20 Kaon server architecture

2.3.3.3 Ontofusion & similar projects

In ONTOFUSION (Perez-Rey et al. 2006), separate conceptual schemas are used to describe the semantics of each data source. Every concept in a physical database is mapped to a virtual schema. Virtual schemas are ontologies representing the structure of the database at a conceptual level. Then, the various virtual schemas corresponding to the distinct databases are merged into new, unified virtual schemas that can be accessed by the users in order to form their queries. This approach adds more complexity to the whole task, but is otherwise promising. Similar projects are PICSEL (Goasdoui et al. 2000), MECOTA (Wache et al. 1999), and SEMEDA (Kohler et al. 2003).

.3.3.4 MASTRO-I

A latest attempt worth mentioning is the MASTRO-I system (Calvanese, 2008). The global schema in this system is specified in terms of an ontology, specifically in terms of a TBOX expressed in a tractable Description Logics, namely DL-Lite$_A$. So, their approach conforms to the view that the global schema of a data integration system can be profitably represented by an ontology, so the clients can rely on a shared conceptualization when accessing the services provided by the system. Moreover, the source schema is the schema of a relational database. Such a schema may result from the federation of a set of heterogeneous, possibly non-relational data sources. This can be realized by means of a data federation tool, which presents without materializing them, physical data sources to MASTRO-I as they were a single relational database, obtained by simple transforming each source into a set of virtual relational views and taking their union. The mapping language they use allows for expressing GAV sound mappings between the sources and the global schema. A GAV sound mapping specifies that the extension of a source view provides a subset of the tuples satisfying the corresponding element of the global schema. Moreover, the mapping language has specific mechanisms for addressing the so-called *impedance mismatch* problem. The problem arises from the fact that, while data sources store values, the instances of concepts in the ontology are objects, each one denoted by an identifier not to be confused with any data item. The system is able to answer union of conjunctive queries posed to the global schema according to a method which is sound and complete with respect to the semantics of ontology. The careful design of various languages used in the system, result in a very efficient technique (LOGSPACE w.r.t. Data complexity), which reduces query answering to standard SQL query evaluation over the sources.

Chapter 3

Ontology change in Data Integration

"An expert is a man who has made all the mistakes which can be made, in a narrow field."

- Niels Borh

Contents

In this Chapter, we identify solutions proposed in the state of the art that try to tackle the problem of ontology evolution in data integration. Most specifically we focus on the approaches which try to reuse previously captured information. Since most of the approaches today concern database schema evolution, we examine them first and check if they can be applied in an ontology-based data integration scenario. We classify them into two general categories. Those that try to compose successive schema mappings (*mapping composition)* and those that try to evolve the mappings each time a primitive change operation occurs (*mapping adaptation).* Although, those approaches deal with closely related issues, their applicability in a dynamic ontology has not yet been examined. We demonstrate some drawbacks of both approaches by means of simple examples and prove that they are inefficient in a state of the art ontology-based data integration setting. This belief is further enhanced by showing that changes in database schemata differ greatly from changes in ontologies.

The lack of an ideal approach to handle ontology evolution in data integration leads us to propose requirements for a new approach. We highlight what is missing from the current state of the art and outline the requirements for an ideal data integration system that will incorporate and handle ontology evolution efficiently and effectively.

The overall goal of this Chapter is not only to give readers a comprehensive overview of the works in the area, but also to provide necessary insights for the practical understanding of the issues involved.

3.1 Why ontologies change?

Ontology change refers to the generic process of changing an ontology in response to a certain need. Several reasons for changing an ontology have been identified in the literature. An ontology, just like any structure holding information regarding a domain of interest, may need to change simply because the domain of interest has changed (Stojanovic et al. 2003); but even if we assume a static world (domain), which is a rather unrealistic assumption for most applications, we may need to change the perspective under which the domain is viewed (Noy & Klein 2004), or we may discover a design flaw in the original conceptualization of the domain (Plessers & de Troyer 2005); we may also wish to incorporate additional

functionality, according to a change in users' needs (Haase & Stojanovic 2005). Furthermore, new information, which was previously unknown, classified or otherwise unavailable may become available or different features of the domain may become known and/or important (Heflin et al. 1999). Moreover, ontology development is becoming more and more a collaborative and parallelized process, whose sub-products (parts of the ontology) need to be combined to produce the final ontology (Klein & Noy 2003), (Noy et al. 2006); this process would require changes in each sub-ontology to reach a consistent final state; but even then, the so-called final state is rarely final, as ontology development is usually an ongoing process (Heflin et al. 1999).

There are also reasons related to the distributed nature of the Semantic Web: ontologies are usually depending on other ontologies, over which the knowledge engineer may have no control; if the remote ontology is changed for any of the above reasons, the dependent ontology might also need to be modified to reflect possible changes in terminology or representation (Heflin et al. 1999). In other cases, a certain agent, service or application may need to use an ontology whose terminology or representation is different from the one it can understand (Euzenat et al. 2004); in such cases, some kind of translation (change) needs to be performed in the imported ontology to be of use. Last but not least, we may need to combine information from two or more ontologies in order to produce a more appropriate one for a certain application (Pinto et al. 1999).

The problem of ontology change is far from trivial. Several philosophical issues related to the general problem of adaptation of knowledge to new information have been identified in the research area of belief change, also known as belief revision (Gardenfors 1992a), (Gardenfors 1992b), (Katsuno & Mendelzon 1990); most of them are also applicable to knowledge represented in ontologies (Flouris & Plexousakis 2005), (Flouris & Plexousakis 2006). The large size of modern day ontologies complicates this problem even further (McGuiness et al. 2000). But it's not just that: the Semantic Web is characterized by decentralization, heterogeneity and lack of central control or authority. This is both a blessing and a curse; these features have greatly contributed to the success of the WWW (and constitute key features of the Semantic Web) but they have also introduced several new, challenging and interesting problems, which don't exist in traditional AI.

As far as ontology change is concerned, one such problem is the lack of control on who uses a certain ontology once it has been published. Subtle changes in an ontology may have unforeseeable effects in dependent applications, services, data and ontologies (Stojanovic et al. 2002); ontology designers cannot know who uses which part of their ontology and for what purpose, so they cannot predict the effects that a given change on their ontology would have upon dependent elements. The same holds in the opposite direction: if an ontology is depending on other ontologies, there is no way for the ontology designer to control when and how these ontologies will change. These facts raise the need to support and maintain different interoperable versions of the same ontology (Heflin et al. 1999), (Huang & Stuckenschmidt 2005), (Klein et al. 2002), a problem greatly interwoven with ontology change (Klein & Fensel 2001). On the other hand, heterogeneity leads to the absence of a standard terminology for any given domain which may cause problems when an agent, service or application uses information from two different ontologies (Euzenat et al. 2004). As ontologies often cover overlapping domains from different viewpoints and with different terminology, some kind of translation may be necessary in many practical applications.

All these arguments indicate the importance of the problem of ontology change and motivate us to use the term in order to cover all aspects of ontology dynamics, as well as the problems that are indirectly related to the change operation such as the maintenance of different versions of an ontology or the translation of ontological information in a common terminology. More specifically, we will use the term ontology change to refer to the problem of deciding the modifications to perform upon an ontology in response to a certain need for change as well as the implementation of these modifications and the management of their effects in depending data, services, applications, agents or other elements.

Notice that the decision on the modifications to perform may be made automatically, semi-automatically or manually; the implementation of the chosen modifications may (but need not) involve keeping a copy of the original ontology (versioning). The need to change the ontology may take several different forms, including, but not limited to, the discovery of new information (which could be some instance data, another ontology, a new observation or other), a change in the focus or the viewpoint of the conceptualization, information received by some external source, a change in the domain (i.e., a dynamic change in the modeled world), communication

needs between heterogeneous sources of information, the fusion of information from different ontologies and so on.

3.1.1 Ontology Change Subfields

Our definition of ontology change covers several related research fields which are studied separately in the literature. These fields are greatly interlinked and several papers and systems deal with more than one of these. In other cases, the same term is used in different papers to describe different research areas. This situation can easily lead to misunderstandings, confusion and unnecessary waste of effort, especially for a newcomer. In the remainder of this Section we will attempt to precisely define the boundaries of each ontology change subarea and uncover their relations and differences. This attempt will hopefully draw a fine line between these areas, allowing the clarification of the meaning of each term and making the differences and similarities between them explicit. The provided definitions will not be arbitrary, but will be based on the most common uses of each term in the literature and on similar previous attempts, like (Kalfoglou & Schorlemmer 2003), (Pinto et al. 1999), (Flouris et al. 2008).

In particular, we will identify nine subfields of ontology change, namely ontology mapping, morphism, matching, articulation, translation, evolution, versioning, integration and merging; in addition, we will clarify the meaning of the term ontology alignment, which is closely related to ontology matching. Each of these areas deals with a certain facet of the problem of change from a different view or perspective, covering different application needs, change scenarios or "needs for change". In this subsection, we provide a very short description of each of these fields; for more details, the reader is referred to the (Flouris et al. 2008), where the properties of each field are discussed in detail.

The first five fields in the above list (ontology mapping, morphism, matching, articulation and translation), as well as ontology alignment deal with heterogeneity resolution, i.e., how to resolve differences in terminology, language or syntax between ontologies. We have to note that the terms are closely related, so mapping for example produces declarative relations (functions) whereas the matching is more liberal and only identifies binary links between the two ontologies. Usually, this problem is solved by providing a set of "translation rules" that identify similar ontology

elements. The distinguishing difference between these fields is the methodology followed and the expected type of output (translation rules). These fields may look unrelated to ontology change, as there is no obvious "change" performed in the involved ontologies; translation rules do not seem to constitute change themselves. However, heterogeneity resolution falls under the definition of ontology change, in the wide sense of the term that we use in this paper.

Indeed, consider two agents with heterogeneous ontologies that need to communicate and a set of translation rules that allows this communication. In this particular example, the driving force (need) behind the process is the need for communication. The translation rules produced do not directly modify any ontology; however, they allow each agent to change the other agent's ontology locally to fit his own terminology, language and syntax. So the change in this case is made on-the-fly by each agent during each message exchange and it is trivial, given the translation rules. In this sense, heterogeneity resolution can be considered a type of ontology change that provides us with a method to change an ontology (but does not perform the change directly).

Furthermore, it is important to note that heterogeneity resolution constitutes a prerequisite for any type of successful ontology change, as it makes no sense to try to change an ontology in response to new information unless both the ontology and the new information are formulated using the same terminology, language and syntax. So, it makes practical sense to study these fields along with the problem of ontology change; this is also apparent in the relevant literature, where many research efforts, systems or algorithms that deal with some specific aspect (subfield) of ontology change also deal with the problem of heterogeneity resolution (e.g., (De Bruijn et al. 2004b), (Chalupsky 2000), (McGuiness et al. 2000), (Noy & Musen 1999a), (Noy & Musen 1999b), (Noy & Musen 2000)).

Ontology evolution and versioning are often used in confusing ways in the literature. Ontology evolution deals with the problem of incorporating new information in an existing ontology, so it deals with the changes themselves. Ontology versioning manages different versions of a changing ontology, trying to minimize any adverse effects that a change could have upon related (dependent) ontologies, agents, applications or other elements.

This is done by providing transparent access to either the current or some older version of the ontology, depending on the accessing element. This ability allows the accessing (dependent) elements, to upgrade to the new version at their own pace (if at all), which is considered a very useful feature, given the distributed and decentralized nature of the Semantic Web (Heflin et al. 1999), (Heflin & Pan 2004).

Ontology Mapping	**Purpose:** **Input:** **Output:** **Properties:**	Heterogeneity resolution, interoperability of ontologies Two (heterogeneous) ontologies A mapping between the ontologies' vocabularies The output identifies related vocabulary entities
Ontology Morphism	**Purpose:** **Input:** **Output:** **Properties:**	Heterogeneity resolution, interoperability of ontologies Two (heterogeneous) ontologies Mappings between the ontologies' vocabularies and axioms The output identifies related vocabulary entities and axioms
Ontology Matching (its output is called Ontology Alignment)	**Purpose:** **Input:** **Output:** **Properties:**	Heterogeneity resolution, interoperability of ontologies Two (heterogeneous) ontologies A relation between the ontologies' vocabularies The output identifies related vocabulary entities
Ontology Articulation	**Purpose:** **Input:** **Output:** **Properties:**	Heterogeneity resolution, interoperability of ontologies Two (heterogeneous) ontologies An intermediate ontology and mappings between the vocabularies of the intermediate ontology and each source The output is equivalent to a relation and identifies related vocabulary entities (like ontology matching)
Ontology Translation (first reading)	**Purpose:** **Input:** **Output:** **Properties:**	Translation to a different ontology representation language An ontology and a target ontology representation language An ontology expressed in the target language Should produce an equivalent ontology, if possible
Ontology Translation (second reading)	**Purpose:** **Input:** **Output:** **Properties:**	Implementation of a vocabulary mapping An ontology and a mapping An ontology Implements the vocabulary change to the source ontology as specified by the input mapping
Ontology Evolution	**Purpose:** **Input:** **Output:** **Properties:**	Respond to a change in the domain or its conceptualization An ontology and a (set of) change operation(s) An ontology Implements a (set of) change(s) to the source ontology
Ontology Versioning	**Purpose:** **Input:** **Output:** **Properties:**	Transparent access to different versions of an ontology Different versions of an ontology A versioning system Uses version ids to identify versions; provides transparent access to the correct version; determines compatibility
Ontology Integration	**Purpose:** **Input:** **Output:** **Properties:**	Fuse knowledge from ontologies covering similar domains Two ontologies (covering similar domains) An ontology Fuses knowledge to cover a broader domain
Ontology Merging	**Purpose:** **Input:** **Output:** **Properties:**	Fuse knowledge from ontologies covering identical domains Two ontologies (covering identical domains) An ontology Fuses knowledge to describe the domain more accurately

Table 1 Ontology change subfields

Ontology integration and merging both deal with the fusion of knowledge from two or more source ontologies. There is a subtle difference between them, related to the domain covered by the source ontologies.

Table 1 provides a compact description of the ontology change subfields. In Table 1, we summarize the need for change that motivates each field (purpose), the expected input of an algorithm that deals with the problem (input), its expected output (output), as well as certain comments on its desired properties (properties).

3.2 A review of the State of the Art

So far, we described the reasons that lead to ontology evolution and we‘ve made a list with the subfields of ontology change. Trying to tackle the problem of ontology change in data integration systems, a typical solution would be to regenerate the mappings and then the dependent artifacts. This method is called the “blank-sheet approach” (Yu, 2005). However, even with the help of mapping generation tools, this process can be costly in terms of human effort and expertise since it still requires extensive input from human experts. As large, complicated schemata become more prevalent, and as data is reused in more applications, manually maintaining mappings is becoming impractical. Moreover, there is no guarantee that the regenerated mappings preserve the semantics of the original mappings since they are not considered during the regeneration. We believe that the effort required to recreate mappings from scratch as the ontology evolves is problematic and costly (Velegrakis, 2004), and instead previously captured information should be reused. It is really important that domain experts specify the necessary mappings only once and then they can retrieve data disregarding the changes in the ontology. The rest of this section aims to provide a comprehensive overview of the approaches that try to reuse previously captured information in order to cope with schema/ontology evolution.

3.2.1 Earlier Works

Work in the area of database schema evolution started to emerge in the early 90’s where mappings were considered as view definitions. Gupta et al. (Gupta, 1996) and Mohania and Dong (Mohania, 1996) addressed the problem of maintaining a materialized view after user redefinition, while (Ra, 1997) explored how to use view technology to handle schema changes transparently.

Lee et al. (Lee, 2002) were the first to address the problem of defining view definitions when the schemata of base relations change. They identified the view adaptation problem for view evolution in the context of information systems schema changes, which they called view synchronization. They proposed E-SQL, an extended version of SQL for defining views that incorporated user preferences in order to change the semantics of the view and with which the view definer could direct the view evolution process. They proposed a view rewriting process that finds a view redefinition that meets all view preservation constraints specified by the E-SQL view definition. Such a solution prevented manual interaction. However, the supported changes were limited and evolution could only appear at the source side.

3.2.2 Approaches for similar problems

Besides those earlier approaches, several others have been proposed so far to tackle similar problems. For example, for XML databases there have been several approaches that try to preserve mapping information under changes (Barbosa, 2005) or propose guidelines for XML schema evolution in order to maintain the mapping information (Moro, 2007). Moreover, augmented schemata were introduced in (Rizzi, 2007) to enable query answering over multiple schemata in a data warehouse, whereas other approaches change the underlying database systems to store versioning and temporal information such as (Bounif, 2006), (Edelweiss, 2005), (Moon, 2010). Moreover, MORE (Huang, 2005) proposed a framework for reasoning with multi-version ontology, using temporal logic in order to detect ontology changes and their consequences.

However, our goals differ from all the above approaches and the most relevant approaches that could be employed for resolving the problem of data integration with evolving ontologies is *mapping composition* and *mapping adaptation*.

3.2.3 Mapping Composition

Despite the fact that mapping composition is not primarily focused on ontology evolution it could be employed in order to handle ontology evolution. The approach would be to describe ontology evolution itself as mappings and to employ mapping composition to derive the adapted mappings. Madhavan and Halevy (Madhavan,

2003) in 2003 were the first to address the problem of composing semantic mappings. Specifically, given mappings between data sources *S* and *T* and between *T* and *T′*, is it possible to generate a direct mapping *M′* between *S* and *T′* that is equivalent to the original mappings (see Fig. 21). Equivalence means that for any query in a given class of queries *Q*, and for any instance of the data sources, using the direct mapping yields exactly the same answer that would be obtained by the two original mappings.

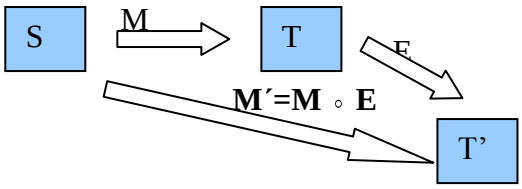


Fig. 21. Composing Schema Mappings

The semantics of the composition operator proposed by Madhavan and Halevy was a significant first step, but it suffered from certain drawbacks caused by the fact that this semantics was given relative to a class of queries. The set of formulas specifying a composition *M′* of *M* and *E* relative to a class *Q* of queries need not be unique up to logical equivalence, even when the class *Q* of queries is fixed. Moreover, this semantics is rather fragile because a schema mapping *M′* may be a composition of *M* and *E* when *Q* is the class of conjunctive queries (the class *Q* that Madhavan and Halevy focused on), but fail to be a composition of these two schema mappings when *Q* is the class of conjunctive queries with inequalities. In addition, they showed that the result of composition may be an infinite set of formulas even when the query language is that of conjunctive queries.

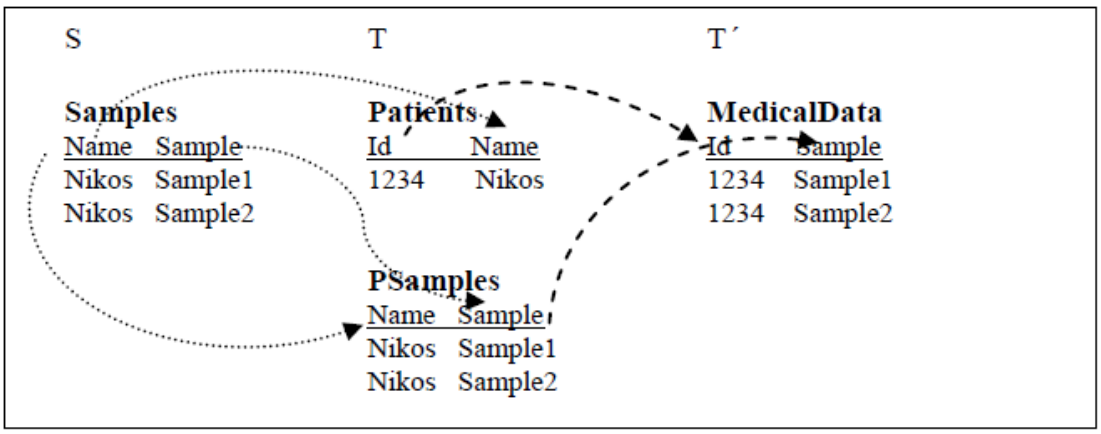


Fig. 22. The example schemata

Consider for example the three schemata *S*, *T* and *T′* shown in Fig. 22. We use a trivial example just to show our key points. Schema *S* consists of a single binary

relation symbol *Samples* that associates patient *names* with their medical *samples*. Schema *T* consists of a similar relation *PSamples* that is intended to provide a copy of *Samples*, and provides an additional relation *Patients*, that associates each patient *name* with a patient *id*. Schema *T′* consists of the relation *MedicalData* that associates patiend *ids* with their *samples*.

Consider now the schema mappings Σ_{12} between *S* and *T* and Σ_{23} between *T* and *T′* where:

$$\Sigma_{12} = \{ \forall n \ \forall s \ ((Samples(n, s) \rightarrow PSamples(n, s)),$$
$$\forall n \ \forall s \ ((Samples(n, s) \rightarrow \exists i \ Patients(i, n))\}$$
$$\Sigma_{23} = \{ \forall n \ \forall i \ \forall s \ (Patients(n, i) \wedge PSamples(n, s) \rightarrow MedicalData(i, s))\}$$

The three formulas in Σ_{12} and Σ_{23} are source-to-target tuple generating dependencies (s-t tgds) that have been extensively used to formalize data exchange (Fagin, 2005). A s-t tgd has the form $\forall x \varphi(x) \rightarrow \exists y \psi(x, y)$, where $\varphi(x)$ is a conjunction of atomic formulae over *S* and $\psi(x, y)$ is a conjunction of atomic formulae over *T*. A tuple-generating dependency specifies an inclusion of two conjunctive queries, $Q1 \subseteq Q2$. It is called source-to-target when $Q1$ refers only to symbols from the source schema and $Q2$ refers only to symbols from the target schema. The first mapping requires that "copies" of the tuples in *Samples* must exist in *PSamples* relation and moreover, that each patient name *n* must be associated with some patient *id i* in *Patients*. The second mapping requires that pairs of patient *id* and *sample* must exist in the relation *MedicalData*, provided that they are associated with the same patient *name*.

Moreover, let *Samples*={(*Nikos*, *Sample1*), (*Nikos*, *Sample2*)} be instances I_1 of *S*, *PSamples*=*Samples* and *Patients*={(*1234*, *Nikos*)} the instances I_2 of *T*, and *MedicalData*={(*1234*, *Sample1*), (*1234*, *Sample2*)} the instances I_3 of *T′*. It is easy to verify that the instances satisfy the mappings Σ_{12} and Σ_{23} that is $\{I_1, I_2\} \in Inst(M)$ and $\{I_2, I_3\} \in Inst(E)$. Now we are looking for a composition of *M* and *E* such that an instance $\{I_1, I_3\}$ is in $Inst(M) \circ Inst(E)$ if and only if it satisfies Σ_{13}. A first guess for Σ_{13} could be:

$$\Sigma_{13} = \{ \forall n \ \forall s \ (Samples(n, s) \rightarrow \exists i \ MedicalData(i, s))\}$$

However, here the patient id *i* depends on both the patient name *n* and the sample id *s*. So *(i, s)* must be a tuple in the *MedicalData* relation for every sample *s* where *(n, s)* is in the *Samples* relation. This is clearly incorrect. Consider, for each $k \geq 1$, the following source-to-target tgd:

$$\varphi\kappa = \{ \forall n \ \forall s1 \ldots \forall sk \ (Samples(n,s1) \wedge \ldots \wedge Samples(n, sk) \rightarrow \exists i \ MedicalData(i, s1) \wedge \ldots \wedge MedicalData(i, sk))\}$$

It is easy to verify that the composition Σ_{13} *is the infinite set* $\{ \varphi_{1,\ldots,} \varphi_{\kappa,} \ldots \}$ *of source to target tgds.* Fagin et al. (Fagin, 2005) identified that problem and showed that the compositions of certain kinds of first-order mappings may not be expressible in any first-order language, even by an infinite set of constraints. That is, that language is not closed under composition. In order to face that problem they introduced second-order s-t tgds, a mapping language that is closed under composition. Using second-order tgds, the composition of the previous example becomes:

$$\Sigma_{13} = \{ \forall n \ \exists i \ \forall s \ (Samples(n,s) \rightarrow MedicalData(i,s)),$$
$$\exists f \ (\forall n \ \forall s \ (Samples(n,s)) \rightarrow MedicalData(f(n),s)))\}$$

Where *f* is a function symbol that associates each patient name *n* with a patient id *f(n).* The second-order language they propose uses existentially quantified function symbols, which essentially can be thought of as Skolem functions. Fagin et al. presented a composition algorithm for this language and showed that it can have practical value for some data management problems, such as data exchange.

Yu and Popa (Yu, 2005) considered mapping composition for second order source-to-target constraints over nested relational schemata in support of schema evolution. Despite the close relation, all the previous approaches did not specifically consider schema evolution. They presented a composition algorithm similar to the one in (Fagin, 2005), with extensions to handle nesting and with significant attention to minimizing the size of the result. They reported a set of experiments using mappings on both synthetic and real-life schemata, to demonstrate that their algorithm is fast and is effective at minimizing the size of the result.

Nash et al. (Nash, 2007) tried to extend the work of Fagin et al. They studied constraints that need not be source-to-target and they concentrated on obtaining first-order embedded dependencies. They considered dependencies that could express key constraints and inclusions of conjunctive queries $Q1 \subseteq Q2$, where *Q1* and *Q2* may reference symbols from both the source and target schema. They do not allow existential quantifiers over function symbols. The closure of composition of constraints in this language does not hold and determining whether a composition result exists is undecidable. One important contribution of this article is an algorithm for composing the mappings given by embedded dependencies. Upon a successful execution, the algorithm produces a mapping that is also given by embedded dependencies. The algorithm however, has some inherent limitations since it may fail to produce a result, even if a set of embedded dependencies that expresses the composition mapping exists. Moreover, it may generate a set of dependencies that is exponentially larger than the input. They show that these difficulties are intrinsic and not an artifact of the algorithm. They address them in part by providing sufficient conditions on the input mappings which guarantee that the algorithm will succeed. Furthermore, they devote significant attention to the novel and most challenging component of their algorithm, which performs "de-Skolemization" to obtain first-order constraints from second-order constraints. Very roughly speaking, the main two challenges that they face are involved recursion and de-Skolemization.

The latest work on mapping composition is that of Bernstein et al. (Bernstein, 2008) in 2008 that propose a new composition algorithm that targets practical applications. Like (Nash, 2007), they explore the mapping composition problem for constraints that are not restricted to being source-to-target. If the input is a set of source-to-target embedded dependencies their algorithm behaves similarly to that of (Fagin, 2005), except that as in (Nash, 2007), they also attempt to express the results as embedded dependencies through a de-Skolemization step. Their algorithm for composing these types of algebraic mappings gives a partial solution when it is unable to find a complete one. The heart of their algorithm is a procedure to eliminate relation symbols from the intermediate signature. Such elimination can be done one symbol at a time. It makes a best effort to eliminate as many relation symbols from the intermediate schema as possible, even if it cannot eliminate all of them.

Despite the great work that has been done in mapping composition we are not aware of an attempt trying to implement it in the context of ontology evolution. All the approaches deal with relational or nested relational schemata and usually have to do with some particular classes of mappings under consideration each time. Hence, mapping composition does not always address the problem in a satisfactory manner.

This belief is further enhanced by the fact that first-order mappings are not closed under composition and second-order ones are too difficult to handle using current DBMS. We doubt that second-order constraints will be supported by the DBMS in the near future as well. Moreover, given a source and a target database, deciding whether they satisfy a mapping given by second-order tgds may in general require exponential time in the size of input databases as proved in (Fagin, 2005).

Furthermore, in mapping composition someone has to produce several sets of mappings (between S and T and between T and T΄). This would impose a large overhead whenever a new version of the ontology is produced -which can be quite often for dynamic ontologies. Schema evolution is rarely represented as mapping in practice (Yu, 2005). Instead, it is either represented as a list of changes or, more often, implicitly embedded in the new version of the schema.

Moreover, each constraint should be created or at least confirmed by a domain expert. A database system may be implemented by an IT expert but only the appropriate domain expert can understand the specific semantics of the system and s/he is the only one who can ultimately verify the results of the whole mapping process. We argue that second-order constraints are too difficult for domain experts to grasp and understand.

Finally, mapping composition poses increased scalability challenges when compared to usual query rewriting approaches. This is due to the fact that mappings between schemata must often cover the entire schema, while queries usually access only parts of a schema and typically produce simple output.

PRISM (Curino, 2009) is one of the latest approaches that try to build on mapping composition and inversibility. PRISM seeks to develop the methods and tools that turn the difficult schema evolution process into one that is controllable, predictable and avoids down-time. To do so, they try to predict the effect of schema changes on current applications and to translate old queries to work on the new schema version. However, it requires the repeated manual mapping among the schema

versions and different mapping sets may have the same result. So disambiguation is usually needed in several places without offering strong guarantees. Finally, the authors do not consider the constraints coming from the schemata used.

3.2.3 Mapping Adaptation

In parallel with the previous approaches that considered mapping composition, Velegrakis et al. (Velegrakis, 2005) focused on incrementally adapting mappings on schema change.

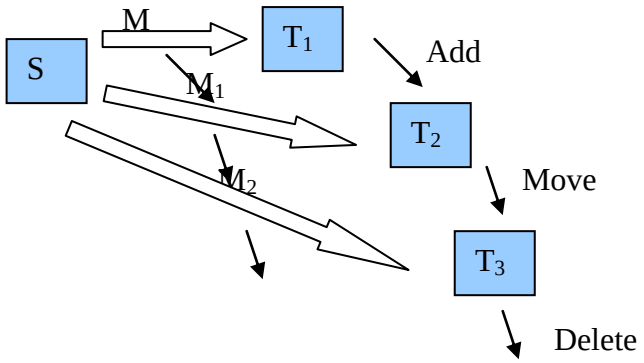
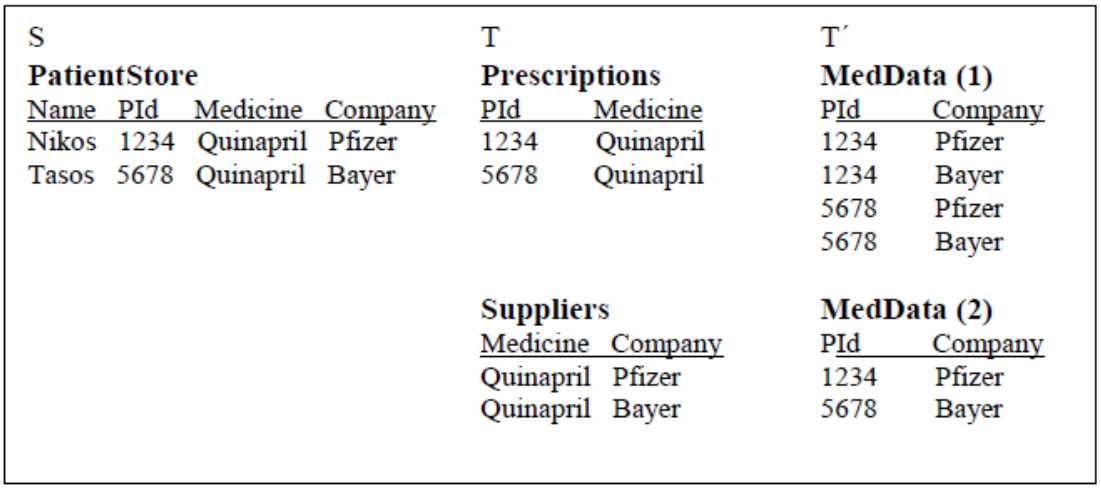


Fig. 23. Adapting Schema Mappings

Their approach is to use a mapping adaptation tool in which a designer can change and evolve schemata. The tool detects mappings that are made inconsistent by a schema change and incrementally modifies the mappings in response. The term incrementally means that only the mappings and, more specifically, the parts of the mappings that are affected by a schema change, are modified while the rest remain unchanged. This approach has the advantage that it can track the semantic decisions made by a designer either in creating the mapping or in earlier modification decisions. These semantic decisions are needed because schemata are often ambiguous (or semantically impoverished) and may not contain sufficient information to make all mapping choices. Those decisions can be reused when appropriate.



S

PatientStore

Name	PId	Medicine	Company
Nikos	1234	Quinapril	Pfizer
Tasos	5678	Quinapril	Bayer

T

Prescriptions

PId	Medicine
1234	Quinapril
5678	Quinapril

Suppliers

Medicine	Company
Quinapril	Pfizer
Quinapril	Bayer

T´

MedData (1)

PId	Company
1234	Pfizer
1234	Bayer
5678	Pfizer
5678	Bayer

MedData (2)

PId	Company
1234	Pfizer
5678	Bayer

Fig. 24. Identifying mapping adaptation problems.

Consider for example the schemata *T* and *T´* shown in Fig. 24. Schema *T* describes patients and the medicines they are administered, along with the suppliers of those medicines. Schema *T´* provides statistical data for the patients that use medicines of a specific company. The mapping between *T* and *T´* is:

$$\Sigma_{TT'} = \{ \forall p \ \forall m \ \forall c \ (Prescriptions(p, m) \wedge Suppliers(m, c) \rightarrow MedData(p, c))\}$$

Assume now that raw data arrive from a new source in the form of tuples *(n, p, m, c)* relating a *name* and an *id* of a patient to a *medicine* and the *supplier* of that medicine. Rather than splitting and inserting the data into the two relations *Prescriptions* and *Suppliers*, a decision is made by the application to store the incoming tuples as they are in the *PatientStore* relation which becomes the new schema S. The mapping $\Sigma_{TT'}$ that depends on the schema *T* and *T´* must now be changed.

So the following operations are issued in *T* in order to become the *S* and according to the mapping adaptation policy the mapping will be updated as well.

- **Move** *Suppliers/Company* to *Prescriptions/ Company*. After this operation the mapping will be updated as well to become:

$$\Sigma' = \{ \forall p \ \forall m \ \forall c \ (Prescriptions(p, m, c) \wedge Suppliers(m) \rightarrow MedData(p, c))\}$$

- **Delete** *Suppliers/Medicine* and then **Delete** the relation *Suppliers*. The mapping now becomes:

$$\Sigma'' = \{ \forall p \ \forall m \forall c \ (Prescriptions\ (p, m, c) \rightarrow MedData\ (p, c))\}$$

- **Rename** *Prescriptions* relation to *PatientStore* and **Add** the field *Name*. The new mapping now becomes

$$\Sigma''' = \{ \forall n \ \forall p \ \forall m \forall c \ (PatientStore\ (n, p, m, c) \rightarrow MedData\ (p, c))\}$$

Their approach considers not only local changes to schema, but also changes that may affect and transform many components of a schema. They consider a comprehensive class of mappings for relational and XML schemata with choice types and constraints that may or may not be nested. Their algorithm detects mappings affected by a structural or constraint change and generates all the rewritings that are consistent with the semantics of the mapped schemata. Their approach explicitly models mapping choices made by a user and maintains these choices, whenever possible, as the schemata and mappings evolve.

The main idea here is that schemata often evolve in small, primitive steps; after each step the schema mapping can be incrementally adapted by applying local modifications. Despite the fact that the specific implementation is system dependent, the idea to incrementally change the mappings each time a primitive change occurs in the source or target schemata has more drawbacks.

When drastic schema evolution occurs (significant restructuring in one of the original schemata) and the new schema version is directly given, it is unclear how feasible it is to extract the list of primitive changes that can describe the evolution. Such scenarios often occur in practice, especially in scientific fields (HL7[2], mzXML[3] standards etc.). The list of changes may not be given and may need to be discovered (Zeginis, 2007), but even then there may be multiple lists of changes with the same effect of evolving the old schema into a new one and we have to be sure that the

[2] http://www.hl7.org/

[3] http://sashimi.sourceforge.net/software_glossolalia.html

resulting mapping is independent of which list of changes is considered. Moreover, the set of primitive changes is not expressive enough to capture complex evolution. Furthermore, even when such a list of changes can be obtained, applying the incremental algorithm for each change in this potentially very long list will be highly inefficient. There is also, no guarantee that after repeatedly applying the algorithm, the semantics of the resulting mappings will be the desired ones.

In order to prove that, consider the example we just discussed. Surprisingly, the semantics of the above mapping may not be the expected one. The instance under *S* consists of two patients that are prescribed with one medicine which is consistent with *T′*. The relation *MedData(1)* under *T* includes all pairs of *Pid* and *Company* that the original mapping requires to exist in *MedData*, based on *T* data. In contrast, the relation *MedData(2)* contains the pairs that the incrementally adapted mapping Σ'''' requires to exist in *MedData*, based on *S* data. Notably, the Σ'''' loses the fact that the patient with id *1234* is also related with *Bayer*.

Thus, Σ'''' does not quite capture the intention of the original mapping, given the new format of the incoming data. Part of the reason this happens is that the new source data does not necessarily satisfy a join dependency that is explicitly encoded in the original mapping $\Sigma_{TT'}$. There are other examples where the incremental approach falls short in terms of preserving the semantics. Furthermore, the same goes for the blank-sheet approach. Indeed, on the previous example, if we just match the common attributes of *S* and *T′*, and regenerate the mapping based on this matching, we would obtain the same mapping *M′* as in the incremental approach. A systematic approach, with stronger semantic guarantees, is clearly needed.

3.2.5 Floating Model

Xuan et al. (Xuan, 2006) propose an approach and a model to deal with the asynchronous versioning problem in the context of a materialized integration system.

Their system is based on the following assumptions: a) each data source participating in the integration process has its own ontology; b) each local source references a shared ontology by subsumption relationships “as much as possible” (each local class must reference its smallest subsuming class in the shared ontology); and c) a local ontology may restrict and extend the shared ontology as much as needed.

However, the authors of (Xuan, 2006) are focused mostly on instances and they add semantics on them using implicit storage. So, they add semantic keys on instances, they use universal identifiers for properties and consider a validation period for each instance.

To support ontology changes they propose the *principle of ontology continuity* which supposes that an evolution of an ontology should not falsify axioms that were previously true. This principle allows the management of each old instance using the new version of the ontology. With this assumption, they propose an approach which they call the floating version model in order to fully automate the whole integration process. This paper deals more with temporal databases than ontology evolution and they support only "ontology deeping" as they named it. That is, they only allow addition of information and not deletion, since they rely on the persistence of classes, properties and subsumption relationships (principle *of ontology continuity).* Despite the simplicity of the approach, in practice the deletion of a class/property is a common operation in ontology evolution (Hartung, 2008). Therefore, we argue that this approach is not useful in real-world scenarios and does not adequately reflect reality. Furthermore the paper only describes abstractly the ideas without formal definitions and algorithms.

3.3 Why Traditional Techniques are not Enough?

As shown in the previous sections the solutions proposed so far have several drawbacks and cannot constitute a generic solution. Almost all the approaches deal with relational or nested relational schemata and the single approach we have seen considering ontology change is too simple and is not useful in real-world scenarios. Schema composition is too difficult and mapping adaptation lacks a precise criterion under which the adapted mapping is indeed the "right" result. But even if we tried to neglect those problems we have to face the fact that data integration in ontologies is a problem that is inherently different from the data integration problem for databases (Noy, 2004). We argue that this is true due to the different nature of the two formalisms, and essentially boils down to a number of differences, discussed below.

The first, very important difference is related to the semantics of databases as opposed to the semantics of logical formalisms that are used in ontologies. Ontology

representation formalisms involve the notion of validity, meaning that certain combinations of ontology axioms are not valid. This is not true for databases, in which any set of tuples that corresponds to the schema is valid (barring the use of integrity constraints, which are, in essence, logical formulas). The notion of validity also affects the change process, forcing us to introduce adequate side-effects in each change operation, in a way that would allow us to maintain validity in the face of such changes (see, e.g., (Konstantinidis, 2007), (Magiridou, 2005)). Therefore, maintaining the correct mappings is more difficult in ontologies (where side-effects must also be considered) than in databases.

For similar reasons, the notion of inference, which exists in ontological formalisms but not in relational databases, affects the process of maintaining the mappings. This issue has two facets: one is related to the different semantics (foundational or coherence (Flouris, 2008)) that could be employed during change and its effects on the update results, and, consequently, on the mappings; the second is related to the fact that inferred knowledge could also give rise to inferred mappings, which should similarly be maintained.

One could claim that relational approaches to maintaining the mappings could be used because of the fact that many ontology manipulation systems use a relational database as a backend for storing the information (Theoharis, 2005). This claim however is problematic because the transformation of ontological knowledge into a relational schema is often a complicated process. In (Theoharis, 2005), several different approaches are considered and compared. Under the simplest ones, a single change in an ontological axiom corresponds to a single change in one tuple in the underlying representation; this is not true in the more sophisticated methods (which are also the most efficient, according to (Theoharis, 2005)), where a single change may correspond to a complicated set of changes in various tuples of the database. Therefore, the corresponding mapping changes may be difficult to figure out, especially given the fact that it is difficult to understand the semantics of an ontology change by just looking at the changed tuples.

As a result, we need to consider *the changes directly on the ontology level*, rather than the database level, which is the first requirement for an ideal ontology-based data integration system. Using such an approach, we could also exploit the fact

that schema/ontology evolution is rarely represented as mappings and is usually presented as a list of changes (Yu, 2005).

The second requirement is to be able to *query information concerning not only source data but ontology evolution as well*. Efficient version management and queries concerning evolution are useful in order to understand how our knowledge advances over time since ontologies depict how we perceive a domain of interest. Moreover, we would like to know the modeling choices we have made in the past. On the other hand, the mapping definition process remains a very difficult problem. In practice, it is done manually with the help of graphical user interfaces and it is a labor-intensive and error prone activity for humans. So in an ideal system the domain expert *should be able to provide, or at least verify, the mapping* between the ontologies and the data sources. The domain experts need a simple mapping language, yet expressive enough to handle the heterogeneity between the ontology and the DBMS. Moreover, the whole *mapping process should be performed only once*, and the generated mappings should not be changed or translated in order to be verified and refined whenever requested in the future.

Finally we need *precise criteria under which the answer produced is the right one*. It is obvious that an answer to a question may not be possible or meaningful, and we need to know under which conditions we can actually retrieve such an answer.

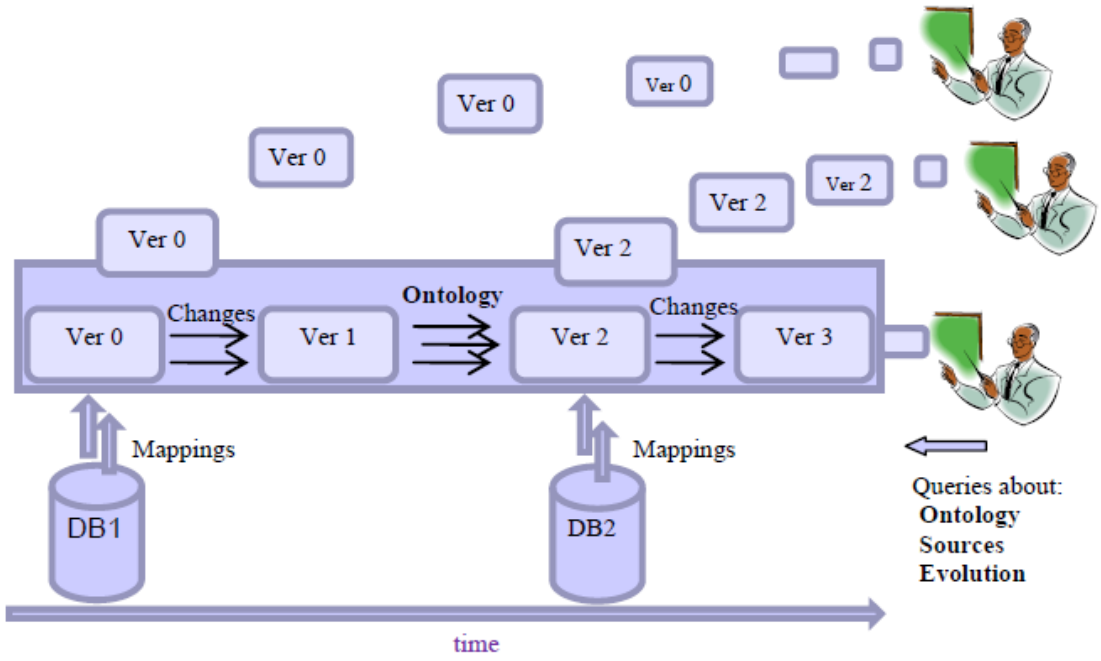


Fig. 25. An ideal solution

In an ideal system, several databases would be mapped to the ontology as the ontology evolves. For example, as shown in Fig. 25, DB1 is mapped using ontology version 0, then the ontology evolves through time, and a second database is mapped when the ontology has reached version 2. Having all those databases mapped using different ontology versions, we would like to answer queries formulated under any ontology version. We would like to support queries that have been formulated using even version 0 since in many systems queries are stored and we wouldn't like to change them every time the ontology changes.

To conclude, an ideal solution should try to exploit the initial mappings, the changes of the ontology and the query expressed using a specific version of the ontology to try to get answers from all databases mapped.

Chapter 4

"Everything that exists, it is only change."

-Heraclitus 535 BCE

Modelling Ontology Change

Contents

In this Chapter we will focus on RDF/S ontologies (Bouquet, 2004). This is because most of the Semantic Web Schemas (85,45%) are expressed in RDF/S (Theoharis, 2007). For those ontologies we will show how to model ontology evolution using a language of high-level changes and how to provide to the users more specific information for a specific changed part of the ontology.

The representation of knowledge in RDF (Bouquet, 2004) is based on triples of the form *predicate (subject, object)*. Assuming two disjoint and infinite sets U, L, denoting the URIs and literals respectively, $T = U \times U \times (U \cup L)$ is the set of all triples. An RDF Graph V is defined as a set of triples, i.e., $V \subseteq T$. RDFS (Brickley, 2004) introduces some built-in classes (class, property) which are used to determine the *type* of each resource. The typing mechanism allows us to concentrate on nodes of RDF graphs, rather than triples, which is closer to ontology curators' perception and useful for defining intuitive high-level changes. RDFS provides also *inference semantics*, which is of two types, namely *structural inference* (provided mainly by the transitivity of subsumption relations) and *type inference* (provided by the typing system, e.g., if *p* is a property, the triple (*p*, *type*, *property*) can be inferred).

Moreover, we assume that the ontology versions we consider are *valid*. The notion of validity has been described in various fragments of the RDFS language. The validity constraints that we consider in this work concern the *type uniqueness*, i.e., that each resource has a unique type, the *acyclicity* of the *subClassOf* and *subPropertyOf* relations and that the subject and object of the instance of some property should be correctly classified under the domain and range of the property, respectively. For a full list of the validity constraints see (Serfiotis, 2005). Those (strict) constraints on the ontology are enforced in order to be enable unique and non-ambiguous detection of the changes among the ontology versions.

A valid RDF Graph containing all triples that are either explicit or can be inferred from explicit triples in an RDF Graph V (using *both* types of inference), is called the *closure*[4] of V and is denoted by $Cl(V)$. An *RDF/S Knowledge Base (RDF/S KB) B* is an RDF Graph which is closed with respect to *type inference*, i.e., it contains all the triples that can be inferred from *B* using type inference.

4.1 Motivating Example

Assume for example the ontology version O_0 shown on the left of Fig. 26 describing persons and their contact points. At some point in time, the ontology evolves and we get O_1 by adding the class "*Cont.Point*" (contact point) and the

[4] http://www.w3.org/TR/rdf-mt/

property "*has_cont_point*" between the class "*Actor*" and the class "*Cont.Point*". Moreover, literal "*town"* is renamed to the literal "*city",* and then the domain of the literals "*street"* and "*city*" is changed to the class "*Cont.Point*".

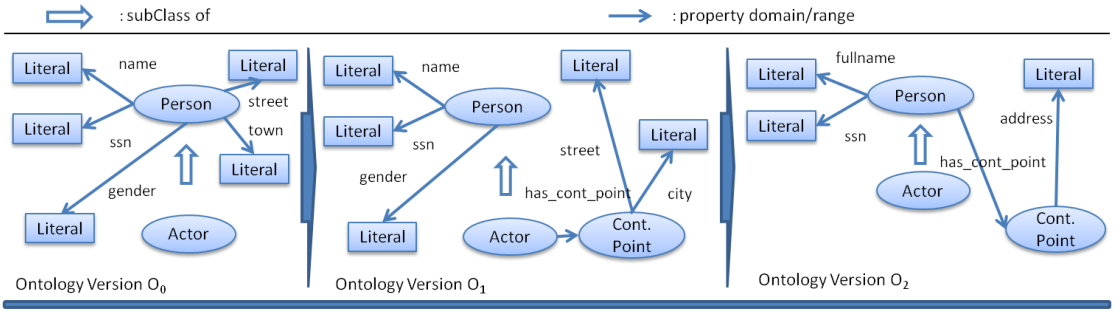


Fig. 26. Example ontology evolution

Then, the ontology designer decides to move the domain of the "*has_cont_point*" property from the class "*Actor*" to the class "*Person*", and to delete the literal "*gender*". Moreover, the "*street*" and the "*city*" properties are merged to the "*address*" property. The resulted ontology O_2 can be seen on the right of Fig. 26.

Now, we would like to be able to express exactly how the ontology has been evolved using a language of changes.

4.1 Using High-level Changes to Model Evolution

For modelling ontology evolution we use a language of changes that describes how an ontology version was derived from another ontology version. In its simplest form, a language of changes consists of only two *low-level* operations, *Add(x)* and *Delete(x),* which determine individual constructs (e.g., triples) that were added or deleted (Volkel, 2005), (Zeginis, 2007). However, a significant number of recent works (Noy, 2006), (Plessers, 2005), (Rogozan, 2005), (Zeginis, 2007), (Papavassiliou, 2009) imply that *high-level* change operations should be employed instead, which describe more complex updates, as for instance the insertion of an entire subsumption hierarchy. A high-level language is preferable than a low-level one, as it is more *intuitive*, *concise*, *closer to the intentions* of the ontology editors and *captures more accurately* the semantics of change (Stojanovic, 2004). As we shall see later on, a high-level language is beneficial for our problem for three reasons: First, because the produced change log has a smaller size, second because the explanations for the ontology change are more concise and more importantly because such a

language yields logs that contain a smaller number of individual low-level deletions (which are non-information preserving as we shall see next) and this affects the effectiveness of our rewriting as we shall see at the next chapter. Moreover properties like *composability* and *inversibility* can be exploited for improving efficiency as we shall see on the sequel. In our work, a change operation is defined as follows:

Definition 4.1 (Change Operation). *A change operation u over O, is any tuple* (δ_a, δ_d) *where* $\delta_a \cap O = \emptyset$ *and* $\delta_d \subseteq O$. *A change operation u from* O_1 *to* O_2 *is a change operation over* O_1 *such that* $\delta_a \subseteq O_2 \backslash O_1$ *and* $\delta_d \subseteq O_1 \backslash O_2$.

Obviously, δ_α and δ_d are sets of triples end especially the triples in δ_d are triples coming from the ontology O (also interpreted as a set of triples as already mentioned). For simplicity, we will denote $\delta_a(u)$ $(\delta_d(u))$ the *added (deleted)* triples of a change *u*. From the definition, it follows that $\delta_a(u) \cap \delta_d(u) = \emptyset$ since $\delta_a \subseteq O_2 \backslash O_1$ *and* $\delta_d \subseteq O_1 \backslash O_2$ if $O_1 \neq O_2$ and we are interested for and $\delta_a(u) \cup \delta_d(u) \neq \emptyset$, i.e. that they either insert or delete something from the ontology.

For the language $\mathcal{L}$ of change operations proposed in (Papavassiliou, 2009) and the corresponding detection algorithm, it has been proved that the sequence of changes between two ontology versions is *unique.* Moreover, it is shown that for any two changes u_1, u_2 in such a sequence it holds that $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$ and $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$. The language $\mathcal{L}$ is proved to satisfy several intuitive properties such as completeness, non-ambiguity and reversibility. Moreover, the detection algorithm was shown to be quite efficient (quadratic worst-case complexity, linear average-case complexity). These are the reasons that led us to adopt that specific language for describing changes among ontologies. Note, that the existence of other languages satisfying these properties is not ruled out. In fact the result of query rewriting described in the next chapter is irrelevant of the specific language used as long as the properties of completeness, non-ambiguity and uniqueness are preserved.

Hereafter, whenever we refer to a change operation, we mean a change operation from those proposed in (Papavassiliou, 2009). Using such high-level change operations we need to define their application semantics.

Definition 4.2 (Application semantics of a high-level change). *The application of a change u over an ontology version O, denoted by u(O), is defined as*

$$u(O) = (O \cup \delta_a(u)) \setminus \delta_d(u).$$

In this point we have two key observations to make: The first is that the application of out change operations is not conditioned by the current state of the ontology (similarly with the approach followed on STRIPS (Russell, 2003) and the second is that we don't handle inconsistency, i.e., $(O \cup \delta_a(u)) \setminus \delta_d(u)$ is always assumed to be valid. Moreover, our approach cannot be directly used with OWL or DL-Lite ontologies.

In our example the change log between O_2 and O_1, denoted by the E^{O_2,O_1}, consists of the following change operations:

u*$_1$:Rename_Property(fullname, name)*

u*$_2$:Split_Property(address, {street, city})*

u*$_3$:Specialize_Domain(has_cont_point, Person, Actor)*

u*$_4$:Add_Property(gender, ø, ø ,ø ,ø, Person, xsd:String, ø, ø)*

Moreover, the change log between O_1 and O_0, denoted by the E^{O_1,O_0}, consists of the following change operations:

u*$_5$: Rename_Property(city, town)*

u*$_6$: Change_Domain(town, Cont.Poing, Person)*

u*$_7$: Change_Domain(street, Cont.Poing, Person)*

u*$_8$: Delete_Property(has_cont_point, ø, ø ,ø ,ø, Actor, Cont.Point, ø, ø)*

u*$_9$: Delete_Class(Cont.Point, ø, ø, ø, ø, ø,)*

Change	**Generalize_Domain (a,b,c)**	**Rename_Property(a, b)**	**Split_Property(a,B)**
Intuition	Change the domain of property *a* from *b* to a superclass *c*	Rename property *a* to *b*	Split property *a* into properties contained in *B*
δ_a	[(*a*, *domain*, *c*)]	[(*b*, *type*, *property*)]	$\forall b_i \in B : [(b_i, type, property)]$ $(1 \leq i \leq n)$
δ_d	[(*a*, *domain*, *y*)]	*[(a, type, property)]*	*[(a, type, property)]*

Fig. 27. The definition of some change operations

The definition of some change operations that are used in this chapter can be seen on Fig. 27, whereas the full list of the considered change operations can be found on the Appendix. It is obvious, that applying those change operations on O_2, results

O_0. Now it is time to define the composition of the change operations. By proving that the change operations are composable, we will be able to use the intermediate evolution logs between ontology versions instead of constructing all change logs between the latest ontology version and all past ontology versions.

Definition 4.3 (Composition of change operations). *A change operation u_{comp} is the composition of u_1 and u_2 (computed over O_1 and O_2), if the result of applying u_{comp} on O_1 is the same with the result of applying u_1 and then u_2 in any order on O_1.*

$$u_{comp}\,(O_1) = u_2(u_1(O_1)) = u_1(u_2(O_1))$$

Now we will show that the change operations as detected in (Papavassiliou, 2009) compose indeed.

Proposition 1: *Let u_1, u_2 two change operations from O_1 to O_2. Then u_{comp} = ($\delta_a(u_1) \cup \delta_a(u_2)$, $\delta_d(u_1) \cup \delta_d(u_2)$).*

Proof: First we have to show that u_{comp} is a change operation from O_1 to O_2, i.e. that $\delta_a(u_{comp}) \subseteq O_2 \backslash O_1$ and that $\delta_d\,(u_{comp}) \subseteq O_2 \backslash O_1$. Indeed $\delta_a(u_{comp}) = (\delta_a(u_1) \cup \delta_a(u_2)) \subseteq O_2 \backslash O_1$ and $\delta_d\,(u_{comp}) = \delta_d(u_1) \cup \delta_d(u_2) \subseteq O_2 \backslash O_1$. Now we will show that $u_{comp}\,(O_1) = u_2(u_1(O_1)) = u_1(u_2(O_1))$ which is also sketched in Fig. 28.

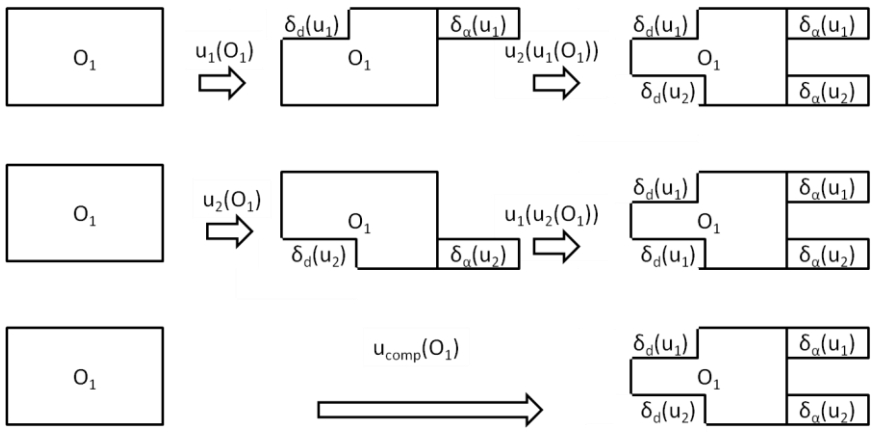


Fig. 28. $u_{comp}\,(O_1) = u_2(u_1(O_1)) = u_1(u_2(O_1))$

Indeed $u_{comp}\,(O_1) = (O_1 \cup \delta_a(u_{comp})) \setminus \delta_d(u_{comp}) = (O_1 \cup \delta_a(u_1) \cup \delta_a(u_2)) \setminus (\delta_d(u_1) \cup \delta_d(u_2)) = ((O_1 \cup \delta_a(u_1)) \setminus \delta_d(u_1) \cup \delta_a(u_2)) \setminus \delta_d(u_2) = u_2\,((O_1 \cup \delta_a(u_1)) \setminus \delta_d(u_1) = u_2(u_1(O_1))$ *and* $u_{comp}\,(O_1) = (O_1 \cup \delta_a(u_{comp})) \setminus \delta_d(u_{comp}) = (O_1 \cup \delta_a(u_1) \cup \delta_a(u_2)) \setminus$

$(\delta_d(u_1) \cup \delta_d(u_2)) = u_1((O_1 \cup \delta_a(u_2)) \setminus \delta_d(u_2) = u_1(u_2(O_1))$ since $\delta_a(u_1) \cap \delta_a(u_2) = \varnothing$ and that $\delta_d(u_1) \cap \delta_d(u_2) = \varnothing$▪

Finally, since a change operation is actually a mapping function that maps one ontology version O_1 to another ontology version O_2, a question is whether there exists the inverse function, the inverse change operation that maps the O_2 ontology version to the O_1 ontology version. By automatically constructing the inverse of a sequence of change operations (from O_1 to O_2), we will be able to rewrite queries expressed using O_2 to O_1 and vice versa.

Definition 4.4 (Inverse of a change operation*). Let u be a change operation from O_1 to O_2. A change operation u_{inv} from O_2 to O_1 is the inverse of u if:*

$$u_{inv}(u(O_1)) \equiv O_1$$

Now we will show how to compute the inverse of a change operation. The inverses of the change operations used in this paper can be found on the Appendix as well.

Proposition 2: *The inverse of a change operation u (denoted by inv(u)) from O_1 to O_2 is:*

$$inv(u)=(\delta_d(u), \delta_a(u))$$

Proof: First we have to show that inv(u) is a change operation, defined over O_2, O_1 i.e. $\delta_a(inv(u)) \subseteq O_1 \setminus O_2$ and that $\delta_d(inv(u)) \subseteq O_2 \setminus O_1$. Indeed $\delta_a(inv(u))=\delta_d(u) \subseteq O_1 \setminus O_2$ and $\delta_d(inv(u))= \delta_a(u) \subseteq O_2 \setminus O_1$.

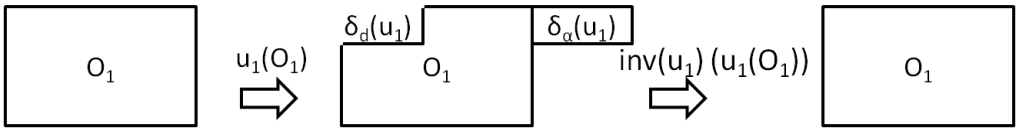


Fig. 29. inv(u)(u(O)) = O

Now we have to prove that $inv(u)(u(O)) = O$ which is also sketched in Fig. 29. Remember that from Definition 4.2. $u(O)=(O \cup \delta_a(u)) \setminus \delta_d(u)$. From the definition, $inv(u)(u(O)) = inv(u)((O \cup \delta_a(u)) \setminus \delta_d(u)) = ((((O \cup \delta_a(u)) \setminus \delta_d(u)) \cup \delta_a(inv(u))) \setminus \delta_d(inv(u)) = (((O \cup \delta_a(u)) \setminus \delta_d(u)) \cup \delta_d(u)) \setminus \delta_a(u) = O$ since $\delta_a(u) \cup \delta_d(u) \neq \varnothing$ *and* $\delta_a(u) \cap \delta_d(u) = \varnothing$ ▪

Based on Propositions 1 and 2 we can conclude that

Corollary 1: *The inverse of a sequence of change operations* $E^{O_1,O_2} = [u_1, \ldots, u_n]$ *constructed from* O_1 *to* O_2, *is* $E^{O_1,O_2}_{inv} = [inv(u_n), \ldots, inv(u_1)]$.

Proof: We must show that $E^{O_1,O_2}_{inv}(E^{O_1,O_2}(O_1)) \equiv O_1$. Indeed $E^{O_1,O_2}_{inv}(E^{O_1,O_2}(O_1)) = E^{O_1,O_2}_{inv}[u_n(u_{n-1}(\ldots(u_1(O_1))))] = inv(u_n)(\ldots inv(u_1)([u_n(u_{n-1}(\ldots(u_1(O_1))))])) = inv(u_n)(u_n(\ldots(inv(u_1)(u_1(O_1))))) = O_1$ since is has already been proved that they can be composed▪

The inverse of the sequence of change operations (i.e the E^{O_0,O_2}) for our running example is:

inv(u9)*: Add_Class(Cont.Point, ø, ø, ø, ø, ø,)*
inv(u8)*: Add_Property(has_cont_point, ø, ø ,ø ,ø, Actor, Cont.Point, ø, ø)*
inv(u7)*: Change_Domain(town, Person, Cont.Point))*
inv(u6)*: Change_Domain(street, Person, Cont.Point)*
inv(u5)*: Rename_Property(town, city)*

inv(u4):*Delete_Property(gender, ø, ø ,ø ,ø, Person, xsd:String, ø, ø)*
inv(u3):*Generalize_Domain(has_cont_point, Actor, Person)*
inv(u2):*Merge_Properties({street, city},address)*
inv(u1):*Rename_Property(name, fullname)*

4.2 Constructing Ontology Versions from Logs

An important question is whether from the sequence of change operations we can compute the current (or a past) version of the ontology efficiently. The cost of constructing an ontology version O_i when operation u_i was issued ($1 \leq i \leq n$), is $O(i)$, so for constructing the latest version of the ontology the cost is $O(n)$. Clearly, if O_i is already constructed and $j>i$, then the cost for constructing O_j is $O(j-i)$.

Moreover, it is also obvious that having only the latest ontology version and the evolution log we can trivially produce an older version using Corollary 1. The cost for constructing an ontology version O_i when operation u_i was issued ($1 \leq i \leq n$) is $O(n\text{-}i)$, since we have to apply the change operations $[inv(u_n),..., inv(u_i)]$.

4.3 Debugging Ontology Evolution with Change Trees

It is clear, that in order to understand the impact of ontology evolution, we should provide to the users an overview of the changes applied to a particular ontology (Plessers, 2007). The simplest way to achieve this is by providing a list of all change operations that were explicitly used by an ontology engineer to change the ontology. However, this approach has a number of serious drawbacks which relate to the different level of granularity for the different change operations, the different viewpoints and implications of those changes. Recent research results have demonstrated that only providing a list of change operations between two ontologies is not sufficient (Plessers, 2007) and new mechanisms need to be provided. So, instead of providing the whole list of changes that have taken place, our idea is to present the history of the creation of individual triples.

.

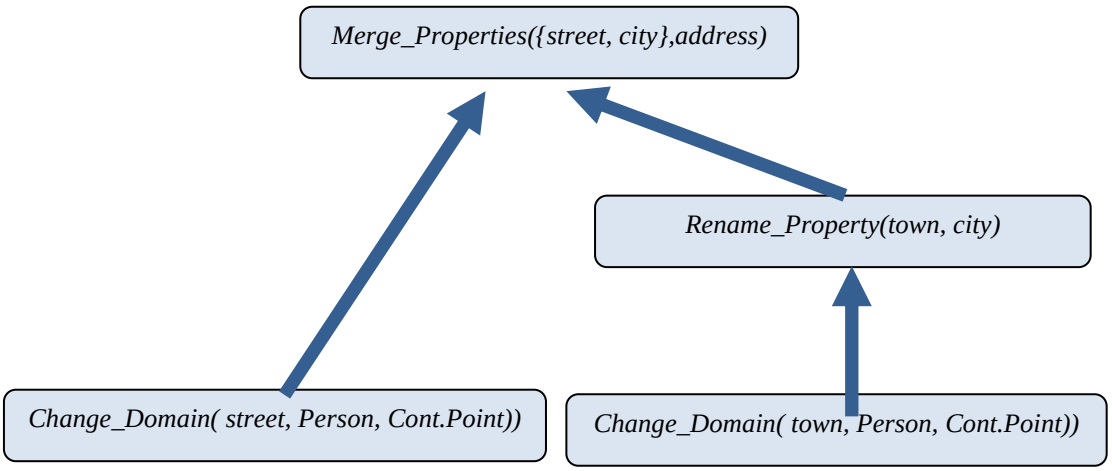


Fig. 30.The change tree for the triple *domain(Cont.Point, address)*

Imagine for example, that we have reached ontology version O_2 and we would like to know how a specific triple has been produced, and what modelling choices have been made to the past concerning that triple. Having only the ontology O_2 is impossible to answer such a question. However, if the change log is available we can

easily answer such a query. For example, consider that we would like to retrieve the past modelling choices for the triple "*address(Cont.Point, xsd:String)*". By checking the entire change log E^{O_0,O_2} presented at the end of Section 4.1, we can identify that is has been produced by executing the changes shown on Fig. 30. We can observe that the tree shown describes exactly the part of the ontology that evolved each time and finally produced the requested triple. Moreover, if even one of those change operations was missing then the triple "*address(Cont.Point, xsd:String)*" would not be able to be inserted and there is not another sequence of change operations producing the specific triple

Such a tree is produced from changes that appear in E^{O_0,O_2}. We name such a tree a *change tree.* Such a tree is only used for visualization purposes and can be easily produced from a *change path* us_{path}.

Definition 4.5 (Change path for a triple *t*). *Let* E^{O_1,O_2} *be the sequence of change operations from* O_1 *to* O_2*. A change path* $us_{path} \subseteq E^{O_1,O_2}$ *for the triple* $t \in O_2$ *is the minimal sequence of change operations such that* $us_{path}(O_1) \equiv O_1'$, $t \in O_1'$.

O_1' is actually an intermediate ontology version between O_1 and O_2. A change path is minimal in the sense that one cannot remove any of the change operations and still be able to produce *t.* For example, the *change path* for the triple *t=address(Cont.Point, xsd:String)* that corresponds to the *change tree* of Fig. 30 is us_{path} = ***[inv(u_7), inv(u_6), inv(u_5), inv(u_2)]*** and obviously $us_{path}(O_0) \equiv O_0'$, and $t \in O_0'$.

Algorithm 4.1: *ComputeChangePathTriple(E^{O_1,O_2}, t_{input})*

Input: A sequence $E^{O_1,O_2} = [u_1, \ldots, u_n]$ and one triple t_{input}

Output: a sequence of change operations *us′*

1. *us′* := ∅
2. For *i=n* to *1*
3. If $t_{input} \in \delta_a(u_i)$
4. $us' := us' \cup u_i$
5. else if $\exists t \in \delta_a(u_i)$ such that $t \in \delta_d(us')$
6. $us' := us' \cup u_i$
7. Return *us′*

Fig. 31. An algorithm for computing the change path for a given triple

Proposition 3 (Uniqueness)***: The change path $us_{path}(t)$ over E^{O_1,O_2} for the triple t is unique.*

Proof: Assume $us_{path}(t)$ is not unique. This would mean that we have two change paths us_{path1} and $us_{path2.}$ Since they are both change paths it should hold that size(us_{path1})=size(us_{path2}) since they both have to be minimal. Now let $us_{path1} = [u_{k1}, \ldots, u_{kn}]$ and $us_{path1} = [u_{m1}, \ldots, u_{mn}]$. Since by using both u_{kn} ,u_{mn} we can reach triple *t* and by the fact that for two change operation *u1*, *u2* over E^{O_1,O_2} it holds that $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$ and $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ it means that $u_{kn} = u_{mn.}$ So, in order for $us_{path1} \neq us_{path2}$ to hold there should be an *i* such that $u_{ki} \neq u_{mi}$ and $\delta_a(u_{ki}) \cap \delta_a(u_{mi}) \neq \emptyset$ (they should add the same triple but they should be different change operations) which is impossible since $\delta_a(u1) \cap \delta_a(u2) = \emptyset$ for our change operations▪

Now we will present an algorithm that, given a change log, produces the change path for a triple t_{input}. The algorithm is shown in Fig. 31. The idea is the following: initially the algorithm searches for the change operation that adds the triple t_{input} possibly by deleting other triples. Then we search for the changes that led to those other triples and so on. After the execution of the algorithm the change path for t_{input} will be stored in *us′*.

Theorem 4.1: *The algorithm ComputeChangePathTriple computes the change path for a given triple t_{input} using a change log E^{O_1,O_2} .*

Proof: In order to prove that ComputeChangePathTriple computes the change path for triple t using a change log E^{O_1,O_2} we have to prove that (a) $t_{input} \in O_1'$ where $us'(O_1) \equiv O_1'$ and that (b) *us′* is minimal for the *us′* that is produced using Algorithm 4.1.

(a) Let $us' = [u_{k1}, u_{k2}, \ldots, u_{km.}]$. Since $u_{km} \in us'$ that means that $t_{input} \in \delta_a(u_{km})$ (lines 2-3 of the algorithm) which means that indeed t_{input} is added to the ontology version resulting after applying u_{km}.

(b) Now we prove minimality. Let's assume that *us′* is not minimal. Then we can assume that there is us_{path} with size(us_{path})< size(*us′*). This would mean that there exist $u_i \in us'$ such that $u_i \notin us_{path.}$ Of course this would mean from lines 5 and 6 that there exist t_i such that $t_i \in \delta_a(u_i)$ such that $t_i \in \delta_d(us')$. This means that we can reach t_{input}

through t_i. However, since $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$ and $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ for two change operation u_1, u_2, the only way to get t_{input} is by using t_i, so t_i should be contained in us_{path}, which contradicts our initial statement. So us' is minimal as well▪

The time complexity of the algorithm is $O(N*M*S)$, where N is the number of change operations, M the maximum size of triples in a change operation u ($\delta_a(u) \cup \delta_d(u)$) and S is the number of triples in $\delta_d(us')$. However, as we will see later on Section 6, in our experiments the number of triples in a change operation typically does not exceed 5 and typically a change path consists of at most 7 change operations. So the time complexity mainly depends mostly on the number of change operations in the evolution log.

Moreover, it is easy to change Algorithm 4.1 in order to retrieve the *change path for a given resource* (class or property).

Definition 4.8 (Change path for a resource *r*). *A change path us_{path} over E^{O_1,O_2} for the resource $r \in O_2$ is $us_{path}(r) \equiv \cup us_{path}(t)$, $r \in t$.*

The idea is that we would like to retrieve all triples that are changed and contain the resource *r*. So, we need to search all triples in order to identify if they contain *r* and then we should construct the change path for each one of them. The corresponding algorithm is shown in Fig. 32.

Algorithm 4.1: *ComputeChangePathResource(E^{O_1,O_2}, r)*

Input: A sequence $E^{O_1,O_2} = [u_1, \ldots, u_n]$ and one resource *r*

Output: a sequence of change operations us'

1. $us' := \emptyset$
2. For $i=n$ to *1*
3. If $\exists t \in \delta_a(u_i)$ such that $r \in t$
4. $us' := us' \cup ComputeChangePathTriple(us, t)$
5. Return us'

Fig. 32. An algorithm for computing the change path for a given resource

Theorem 4.2: *The algorithm ComputeChangePathResource computes the change path for a given resource r over E^{O_1,O_2}.*

Proof: In order to prove that *ComputeChangePathResource* computes the change path for a resource r using a change log E^{O_1,O_2} we have to prove that $us_{path}(r) \equiv \cup us_{path}(t_i)$, $r \in t_i$. From lines 3-4, $us_{path}(r) = \cup us_{path}(t_i)$, $r \in t_i$. By construction, this proves the claim▪

Since for each t_i such that $r \in t_i$ we need to construct the corresponding change path, the time complexity of the algorithm is $O(T*N*M*S)$, where T is the number of triples t_i for which $r \in t_i$, N is the number of change operations, M is the maximum size of triples in a change operation u, i.e. in $\delta_a(u) \cup \delta_d(u)$, and S the number of triples in $\delta_d(us')$. Again, typically in our experiments we have at most three triples for which $r \in t_i$ and the time for the execution of the algorithm mainly depends on the size of the change log.

We have to note that our algorithms are not sensitive to the particular language of changes used, as long as the language maintains the completeness, non-ambiguity and uniqueness properties. Since each triple is inserted or deleted by only one change operation per log we can always identify a change path which is unique.

Chapter 5

Enabling Ontology Evolution in DI

"I only ask for Information"

-Charles Dickens

Contents

In this Chapter, we address the problem of data integration for evolving ontologies. The lack of an ideal approach, shown on Chapter 2, leads us to propose a new mechanism that builds on the latest theoretical advances on the areas of ontology change (Papavassiliou, 2009) and query rewriting (Cali, 2009), (Poggi, 2008) and incorporates and handles ontology evolution efficiently and effectively.

More specifically:

- We present the architecture of a data integration system, named Evolving Data Integration (EDI) system, that allows the evolution of the ontology used as global schema.
- We define the exact semantics of our system and we elegantly separate the semantics of query rewriting for different ontology versions and for the sources. Since query rewriting for the sources has been extensively studied (Cali, 2009), (Poggi, 2008), (Lenzerini, 2002), (Cali, 2003), (Deutsch, 2006), we focus on a layer above and deal only with the query rewriting between ontology versions.
- More specifically, we present a module that receives a user query specified under the latest ontology version and produces rewritings that will be answered by the underlying data integration systems - that might use different ontology versions. The query processing in this module consists of two steps: a) query expansion that considers constraints coming from the ontology, and b) valid query rewriting that uses the changes between two ontology versions to produce rewritings among them.
- In order to identify the changes between the ontology versions we adopt the high-level language of changes described on Chapter 4. The sequence of changes between the latest and the other ontology versions is produced automatically at setup time and then each one of the change operations identified is translated into a logical GAV mapping. This translation enables query rewriting by unfolding. Then, the inversibility is exploited to rewrite queries from past ontology versions to the current, and vice versa, and composability to avoid the reconstruction of all sequences of changes among the latest and all previous ontology versions.

- Despite the fact that query rewriting always terminates in our case, the queries issued to the past ontology versions might fail. We show that this problem is not inhibiting in our algorithms but a consequence of information unavailability among ontology versions. To tackle this problem, we propose three solutions. The first solution is to provide insights for the failure, thus driving query redefinition only for a specific portion of the affected query. Besides driving query redefinition we can provide answers to minimally-containing or minimally-generalized queries instead that are the best over-approximations of input queries.
- Finally we prove that our method is sound and complete with low complexity.

Such a mechanism, that provides rewritings among data integration systems that use different ontology versions, is flexible, modular and scalable. It can be used on top of any data integration system – independently of the family of the mappings they use (GAV, LAV, GLAV, etc. (Lenzerini, 2002)). New mappings or ontology versions can be easily and independently introduced without affecting other mappings or other ontology versions. Our engine takes the responsibility of assembling a coherent view of the world out of each specific setting.

5.1 Motivating Example

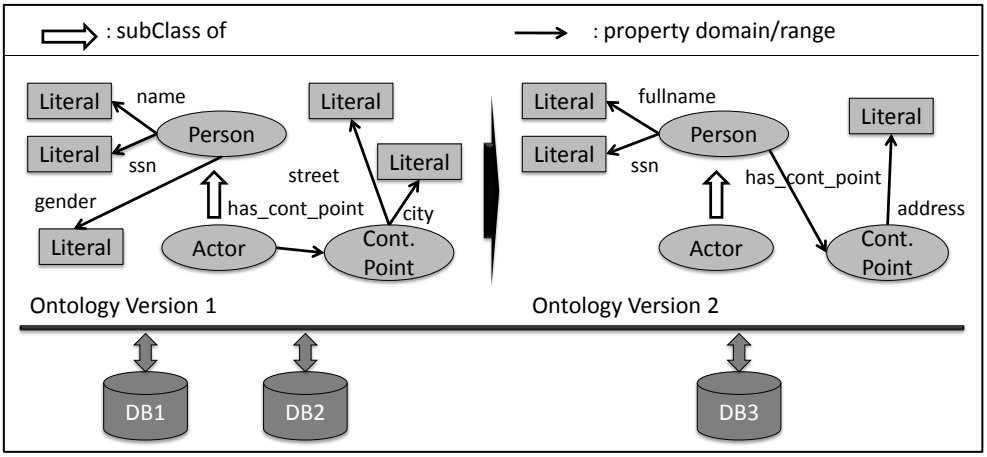


Fig. 33. The motivating example of an evolving ontology.

Consider a part of the example ontology described at Chapter 4 shown also at the left of Fig. 33. This ontology is used as a point of common reference, describing

persons and their contact points. We also have two relational databases DB1 and DB2 mapped to that version of the ontology. Assume now that the ontology designer decides to move the domain of the "*has_cont_point*" property from the class "*Actor*" to the class "*Person*", and to delete the literal "*gender*". Moreover, the "*street*" and the "*city*" properties are merged to the "*address*" property as shown at the right of Fig. 33. Then, one new database DB3 is mapped to the new version of the ontology leading to two data integration systems that work independently. In such a setting we would like to issue queries formulated using any ontology version available. Moreover, we would like to retrieve answers from all underlying databases.

5.2 Evolving Data Integration

We conceive an *Evolving Data Integration* (EDI) system as a collection of data integration systems, each using a different ontology version as global schema. Therefore, we extend the traditional formalism from (Lembo, 2002) and define an EDI as:

Definition 5.1 (Evolving Data Integration System). *An EDI system I is a tuple of the form* $((O_1, S_1, M_1), ..., (O_m, S_m, M_m))$ *where*

- O_i *is a version of the ontology* $(1 \leq i \leq m)$.
- S_i *is a set of local sources* $(1 \leq i \leq m)$.
- M_i *is the mapping between* S_i *and* O_i $(1 \leq i \leq m)$.

Next we discuss how the specific components are specialized in the context of an EDI.

5.2.1 Global & Local Schemata

Considering O_i we restrict ourselves to valid *RDF/S knowledge bases* as already described at Chapter 4. This is due to the fact that most of the Semantic Web Schemas (85,45%) are expressed in RDF/S (Theoharis, 2007).

Moreover, we consider relational databases as source schemata. We choose to use relational databases since the majority of information currently available is still stored on relational databases (Cali, 2010).

5.2.2 Semantics of an EDI

Now we will define semantics for an EDI system *I.* Fig. 34 sketches the proposed approach.

We start by considering a *local database* for each (O_i, S_i, M_i), i.e., a database D_i that conforms to the local sources of S_i. Based on D_i, we shall specify which is the information content of the global schema O_i (recall that a global database is any database for O_i from Chapter 2).

Definition 5.2 (Legal global database): *A global database G_i for (O_i, S_i, M_i) is said to be legal with respect to D_i, if*

- *G_i is legal with respect to O_i, i.e., G_i satisfies all the constraints of O_i.*
- *G_i satisfies the mapping M_i with respect to D_i.*

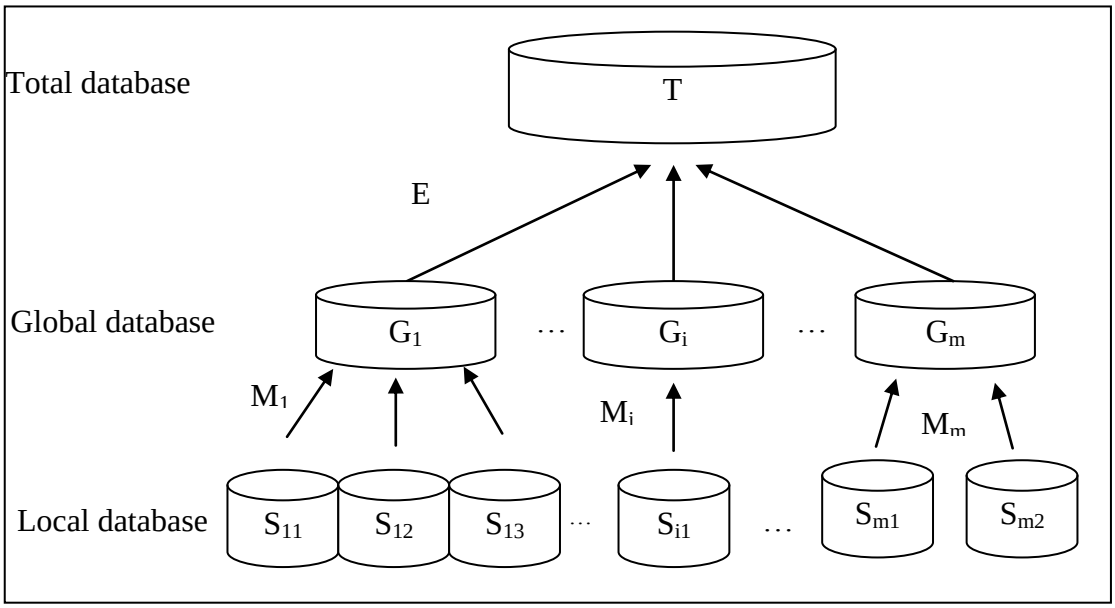


Fig. 34. The semantics of an EDI

The notion of G_i satisfying the mapping M_i, with respect to D_i, is defined as it is commonly done in traditional data integration systems (see (Lenzerini, 2002) for more details). It depends on the different assumptions that can be adopted for

interpreting the tuples that *D* assigns to relations in local sources with respect to tuples that actually satisfy (O_i, S_i, M_i). Since such systems have been extensively studied in the literature we abstract from the internal details and focus on the fact that for each (O_i, *Si*, M_i) of our system we can obtain a global database G_i.

Now, we can repeat the same process, i.e., to consider the global databases as sources and a database $\mathcal{D}$ which we will simply call the *global database*, the database that conforms to them. Now we can define the *legal total database.* Obviously a total database is a database for the latest ontology version O_m. We use the term "*total*" only to differentiate it from a *global* database, since we will extensively use it from now on.

Definition 5.3 (Legal total database): *A total database T for EDI I is said to be legal with respect to $\mathcal{D}$, if*

- *T is legal with respect to O_m, i.e., T satisfies all the constraints of the latest ontology version O_m.*
- *T satisfies E with respect to $\mathcal{D}$ where* $E=\bigcup_1^{m-1} E^{O_m, O_i}$.

The constraints of an RDF/S ontology concern the transitivity of the *subClass* and *subProperty* relations. Moreover, we have to note that the different ontology versions are considered to be valid. Now we specify the notion of *T* satisfying *E* ($E=\bigcup_1^{m-1} E^{O_m, O_i}$) with respect to $\mathcal{D}$. In order to exploit the strength of the logical languages towards query reformulation, we convert our change operations to GAV mappings. So when we refer to the notion of *T* satisfying *E*, we mean *T* satisfying the GAV mappings produced from *E*. The GAV mappings for all change operations used in this paper can be found on the Appendix. A GAV mapping associates to each element *g* in *T* a query q_G over $G_{1, \ldots,} G_m$.

$$g \rightarrow q_G$$

Definition 5.4 *A database T satisfies the mappings $g \rightarrow q_G$ with respect to $\mathcal{D}$ if*

$$g^{T} \supseteq q_G^{\mathcal{D}}$$

where $q_G^{\mathcal{D}}$ is the result of evaluating the query q_G over $\mathcal{D}$.

For example, the sequence of the GAV mappings that corresponds to our sequence of changes is:

$\boldsymbol{mu_1}$: $\forall x, y, fullname(x, y) \rightarrow name(x , y)$

$\boldsymbol{mu_2}$:$\forall x, y, address(x,y) \rightarrow \exists\ a, b, street(x, a) \wedge city(x, b) \wedge concat(y, a, b)$

$\boldsymbol{mu_3}$:$\forall x, has\_cont\_point(Person,x) \rightarrow has\_cont\_point(Actor, x)$

Recall that $E^{O_2,O_1} = [u_1, u_2, u_3, u_4]$ from Chapter 4 where :

$\boldsymbol{u_1}$:*Rename_Property(fullname, name)*

$\boldsymbol{u_2}$:*Split_Property(address, {street, city})*

$\boldsymbol{u_3}$:*Specialize_Domain(has_cont_point, Person, Actor)*

$\boldsymbol{u_4}$:*Delete_Property(gender, ø, ø ,ø ,ø, Person, xsd:String, ø, ø)*

Notice that for u_4 there is no GAV mapping constructed since we do not know where to map the deleted element. Now it becomes obvious that the lower the level of the language of changes used the more change operations won't have corresponding GAV mappings (since more low-level individual additions and deletions will appear). Moreover, note the function "*concat*" in mu_2 which will latter require specific heuristics on the query answering phase, since the u_3 change operations has been constructed using various heuristic-based techniques for identifying elements with different names that correspond to the same real world entity.

By the careful separation between *the legal total database T* and *the legal global databases* G_i we have achieved the modular design of our EDI system and the separation between the traditional data integration semantics and the additions we have imposed in order to enable ontology evolution. Thus, our approach can be applied on top of any existing data integration system to enable ontology evolution.

5.2.3 Query Processing

Queries to *I* are posed in terms of the global schema O_m. For querying, we adopt the language SPARQL (Prud'hommeaux, 2008). We chose SPARQL since it is currently the standard query language for the semantic web and has become an official W3C recommendation. Essentially, SPARQL is a graph-matching language.

Given a data source, a query consists of a pattern which is matched against, and the values obtained from this matching are processed to give the answer. A SPARQL query consists of three parts. The *pattern matching part*, which includes several features of pattern matching of graphs, like optional parts, union of patterns, nesting, filtering (or restricting) values of possible matchings. The *solution modifiers*, which once the output of the pattern has been computed (in the form of a table of values of variables), allows to modify these values applying classical operators like projection, distinct, order, limit, and offset. Finally, the *output* of a SPARQL query can be of different types: yes/no answers, selections of values of the variables which match the patterns, construction of new triples from these values, and descriptions of resources.
In order to avoid ambiguities in parsing, we present the syntax of SPARQL graph patterns in a more traditional algebraic way, using the binary operators *UNION AND* and *OPT*, and *FILTER* according to (Perez, 2009). Assuming the existence of an infinite set of variables *Var* disjoint from *U, L,* a SPARQL graph pattern expression is defined recursively as follows:

- A tuple from $(U \cup L \cup Var) \times (L \cup Var) \times (U \cup L \cup Var)$ is a graph pattern *(a triple pattern).*
- If P_1 and P_2 are graph patterns, then expressions $(P_1\ AND\ P_2)$, $(P_1\ OPT\ P_2)$ and $(P_1\ UNION\ P_2)$ are graph patterns.
- If *P* is a graph pattern and *R* is a SPARQL *built-in condition*, then the expression (*P FILTER R*) is a graph pattern.

A SPARQL built-in condition is constructed using elements of the set $(U \cup L \cup Var)$ and constants, logical connectives, inequality symbols, the equality symbol etc. (see (Prud'hommeaux, 2008) for a complete list). In this paper, we do not consider *OPT* and *FILTER* operators since we leave it for future work. The remaining SPARQL fragment we consider here corresponds to union of conjunctive queries (Perez, 2009). Moreover, the application of the *solution modifiers* and the *output is* done after the evaluation of the query, and is not of interest.

Continuing our example, assume that we would like to know the "*ssn*" and "*fullname*" of all persons stored on our DBs and their corresponding address. The SPARQL query, formulated using the latest version of our example ontology is:

***q*$_1$:** *select ?SSN ?NAME ?ADDRESS where {*

?X type Person.

?X ssn ?SSN.

?X fullname ?NAME.

?X has_cont_point ?Y.

?Y type Cont.Point.

?Y address ?ADDRESS}

Using the semantics from (Perez, 2009) the algebraic representation of q_1 is equivalent to:

q$_1$: $\pi_{?SSN,?NAME,?ADDRESS}$ *(*

(?X, type, Person) AND

(?X, ssn, ?SSN) AND

(?X, fullname, ?NAME) AND

(?X, has_cont_point, ?Y) AND

(?Y, type, Cont.Point) AND

(?Y, address, ?ADDRESS))

Now we define what constitutes an answer to a query over O_m. We will adopt the notion of *certain answers* (Lenzerini, 2002), (Cali, 2009).

Definition 5.5 (Certain answers): *Given a global database $\mathcal{D}$ for I, the answer $q^{I,\mathcal{D}}$ to a query q with respect to I and $\mathcal{D}$, is the set of tuples t such that $t \in q^T$ for every total database T that is legal for I with respect to $\mathcal{D}$, i.e. such that t is an answer to q over every database T that is legal for I with respect to $\mathcal{D}$. The set $q^{I,\mathcal{D}}$ is called the set of certain answers to q with respect to I and $\mathcal{D}$.*

Note that, from a logical point of view, finding certain answers is a logical implication problem: check whether it logically follows from the information in the global databases G_i that *t* satisfies the query.

It has been shown (Calì, 2006), (Cali, 2010) that computing certain answers to union of conjunctive queries over a total database with constraints, corresponds to

evaluating the query over a special database called *canonical* which represents all possible total databases legal for the data integration system and which may be infinite in general. However, instead of trying to construct the canonical database and then evaluate the query, another approach is to transform the original query q into a new query $exp_{O_m}(q)$ over the O_m, (which is called the *expansion* of *q* w.r.t. O_m) such that the answer to $exp_{O_m}(q)$ over the *retrieved total database* is equal to the answer to *q* over the *canonical database* (Calì, 2006).

Definition 5.6 (Retrieved total database): *If $\mathcal{D}$ is a global database for the EDI-system I, then the retrieved total database ret(I, $\mathcal{D}$) is the total database obtained by computing and evaluating, for every element of O_m the query associated to it by our GAV mappings over the global database $\mathcal{D}$.*

Definition 5.7 (Canonical total database): *If $\mathcal{D}$ is a global database for the EDI-system I, then the canonical total database can(I, $\mathcal{D}$) is the retrieved total databases ret(I, $\mathcal{D}$) that do not violate any constraint in O_m.*

Recall that since we have GAV mappings, for each element in O_m, we have a query over the global database $\mathcal{D}$. This is a common approach in data integration under constraints, and we also adopt it here.

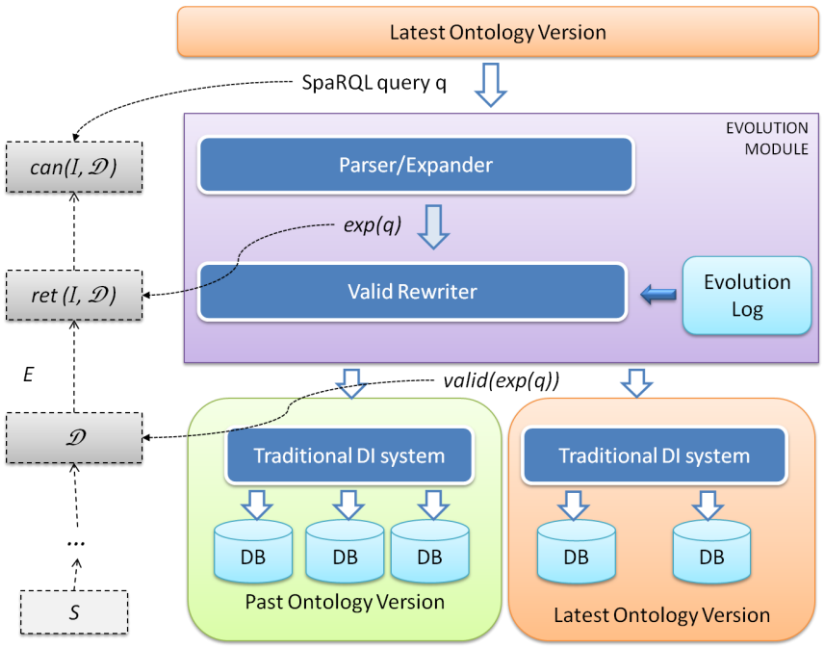


Fig. 35. Query processing

This step is performed by the "*Parser/Expander*" component shown on Fig. 35. Now, in order to avoid building the retrieved total database we do not evaluate $exp_{O_m}(q)$ on the retrieved total database. Instead, we transform $exp_{O_m}(q)$ to a new query $valid_E(exp_{O_m}(q))$ over the global relations on the basis of *E* and we use that query to access the underlying data integration systems. This is performed by the "*Valid Rewriter*" component which is also shown on Fig. 35. Below we describe the implementation of the aforementioned steps.

5.2.3.1 Query expansion.

In this step, the query is expanded to take into account the constraints coming from the ontology. Query expansion amounts to rewriting the query *q* posed to the ontology version O_m into a new query q', so that all the knowledge about the constraints in ontology has been "compiled" into q'. Recall that we consider an ontology as a schema with constraints. This is performed by constructing the *perfect rewriting* of *q*.

Definition 5.8 (Perfect rewriting): *Let I an EDI system and let q be a query over* O_m*. Then* q_p *is called a perfect rewriting of q w.r.t. I if, for every global database* $\mathfrak{D}$, $q^{I,\mathfrak{D}} = q_p{}^{ret(I,\mathfrak{D})}$.

Algorithms for computing the perfect rewriting of a query *q* w.r.t to a schema, have been presented in (Cali, 2010), (Cali, 2009), (Cali, 2003), (Poggi, 2008) and mainly use chase/backchase algorithms (Deutsch, 2006). In our work, we use the QuOnto system (Poggi, 2008) in order to produce the perfect rewriting of our initial query. Perfect rewriting is in our case PTIME in the size of ontology and NP in the size of query. For more general classes of logic it is complete for PSPACE and 2EXPTIME as proved in (Cali, 2009).

Continuing our example if we expand q_1 we get q_2*:*

q2: $\pi_{?SSN,?NAME,?ADDRESS}$ *(*

(?X, type, Person) AND

(?X, ssn, ?SSN) AND

(?X, fullname, ?NAME) AND

(?X, has_cont_point, ?Y) AND

(?Y, type, Cont.Point) AND

(?Y, address, ?ADDRESS))

UNION

$\pi_{?SSN,?NAME,?ADDRESS}$ *(*

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

(?X, fullname, ?NAME) AND

(?X, has_cont_point, ?Y) AND

(?Y, type, Cont.Point) AND

(?Y, address, ?ADDRESS))

This is produced by considering the transitive constraint of the *subClass* relation among the classes "*Person"* and *"Actor"*.

5.2.3.2 Computing Valid Rewritings

Now instead of evaluating $exp_{O_m}(q)$ on the retrieved total database, we transform it to a new query called *valid rewriting*, i.e. $valid_E(exp_{O_m}(q))$. This is done as already discussed in order to avoid the construction of the retrieved total database.

Definition 5.9 (Valid Rewriting): *Let I an EDI system and let q be a query over ret(I, $\mathcal{D}$) . Then $valid_E(q)$ is called a valid rewriting of q w.r.t. ret(I, $\mathcal{D}$) if, for every global database $\mathcal{D}$, $q^{ret(I,\mathcal{D})} = [valid_E(q)]^{\mathcal{D}}$.*

When the retrieved total database is produced by GAV mappings as in our case, query rewriting is simply performed using *unfolding* (Poggi, 2008). This is a standard step in data integration (Lenzerini, 2002) which trivially terminates and it is proved that it preserves soundness and completeness (Cali, 2006).

Theorem 5.1 (Soundness and Completeness of unfolding (Cali, 2003)): *Let I be an EDI system, q a query posed to I, $\mathcal{D}$ a global database for I such that I is consistent w.r.t. $\mathcal{D}$, and t a tuple of constants of the same arity as q. Then $t \in q^{ret(I,\mathcal{D})}$ if and only if $t \in [valid_E(q)]^{\mathcal{D}}$.*

Moreover, due to the disjointness of the input and the output alphabet assumed, each GAV mapping acts in isolation on its input to produce its output. So we only need to scan the GAV mappings once in order to unfold the query and the time complexity of this step *O(N*M)* where *N* is the number of change operations in the evolution log and *M* is the number of sub-goals in the query.

Now, we can state the main result of this section.

Theorem 5.2 (Soundness and Completeness): *Let I be an EDI system, q a query posed to I, $\mathcal{D}$ a global database for I such that I is consistent w.r.t. $\mathcal{D}$, and t a tuple of constants of the same arity as q. Then $t \in q^{I,\mathcal{D}}$ if and only if $t \in [valid_E(exp(q))]^{\mathcal{D}}$.*

Proof: By soundness and completeness of unfolding $t \in [valid_E(exp(q))]^{\mathcal{D}}$ if and only if $t \in exp_{O_m}(q)^{ret(I,\mathcal{D})}$. Now by the soundness of the perfect rewriting step we have that $t \in exp_{O_m}(q)^{ret(I,\mathcal{D})}$ if and only if $t \in q^{can(I,\mathcal{D})}$. By the canonical database $t \in q^{can(I,\mathcal{D})}$ if and only $t \in q^{I,\mathcal{D}}$. This proves the claim▪

Continuing our example we will show how the valid rewriting of q_2 is constructed using unfolding steps. Each one of those steps uses one GAV mapping to replace a subgoal in the query with its definition in the mapping. So, initially the mapping mu_1 is used. Recall that mu_1 is produced from the u_1 change operation (*Rename_Property(fullname, name)*) that replaces the property "*fullname*" with the property "*name*". So, the following query is produced by renaming also the "*fullname"* property *on the query* with the "*name*" property.

$\mathbf{q_3}$:$\pi_{?SSN,?NAME,?ADDRESS}$ (
(?X, type, Person) AND
(?X, ssn, ?SSN) AND
*(?X, **name**, ?NAME) AND*
(?X, has_cont_point,?Y) AND
(?Y, type, Cont.Point) AND
(?Y, address, ?ADDRESS))
UNION
$\pi_{?SSN,?NAME,?ADDRESS}$ (

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

*(?X, **name**, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

(?Y, address, ?ADDRESS))

Then the mappings mu_2 is used for replacing the "*address*" property with the "*city*" and the "*street*" literals. So, the following query is produced.

q4: $\pi_{?SSN,?NAME,?ADDRESS}$ *(*

(?X, type, Person) AND

(?X, ssn, ?SSN) AND

*(?X, name, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

*(?Y, **street**, ?ADDRESS1) AND*

*(?Y, **city**, ?ADDRESS2) AND*

concat(?ADDRESS, ?ADDRESS1, ?ADDRESS2))

UNION

$\pi_{?SSN,?NAME,?ADDRESS}$ *(*

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

*(?X, name, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

*(?Y, **street**, ?ADDRESS1) AND*

*(?Y, **city**, ?ADDRESS2)*

concat(?ADDRESS, ?ADDRESS1, ?ADDRESS2))

Then mu_3 is used. Recall that this is produced from the u_3 change operation (*Specialize_Domain(has_cont_point, Person, Actor))* that specializes the domain of the "*has_cont_point*" property to the class "*Actor*". So, the query q_5 is generated.

q_5: $\pi_{?SSN,?NAME,?ADDRESS}$ (

*(?X, type, **Actor**) AND*

(?X, ssn, ?SSN) AND

*(?X, name, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

(?Y, street, ?ADDRESS1) AND

(?Y, city, ?ADDRESS2)

concat(?ADDRESS, ?ADDRESS1, ?ADDRESS2))

UNION

$\pi_{?SSN,?NAME,?ADDRESS}$ (

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

*(?X, name, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

(?Y, street, ?ADDRESS1) AND

(?Y, city, ?ADDRESS2)

concat(?ADDRESS, ?ADDRESS1, ?ADDRESS2))

Since this is actually the union of a query with itself, the query that will be generated for O_1 is q_6.

q_6: $\pi_{?SSN,?NAME,?ADDRESS}$ (

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

*(?X, name, ?NAME**)** AND*

*(?X, has_cont_point, ?Y**)** AND*

*(?Y, type, Cont.Point**)** AND*

(?Y, street, ?ADDRESS1) AND

(?Y, city, ?ADDRESS2)

concat(?ADDRESS, ?ADDRESS1, ?ADDRESS2))

Finally our initial query will be rewritten to the union of q_6 (issued to the data integration system that uses O_1) and q_2 (issued to the data integration system that uses O_2).

Note however, that q_6 sent to the data integration system that uses O_1 has encoded a function (*concat*) to concatenate the two literals "*streets*" and "*city*" to the literal "*address"*. This function should be executed in order to be able to unify the returned results with the results from q_2. However, this query cannot be sent as is to the data integration system that uses O_1 since SPARQL cannot handle functions and we don't know how the SPARQL query is executed by the underlying data integration system. That is why q_6 is rewritten to q_7 before it is sent to the data integration system that uses O_1.

q7: $\pi_{?SSN,?NAME,?ADDRESS1,\ ?ADDRESS2}$ *(*

(?X, type, Actor) AND

(?X, ssn, ?SSN) AND

(?X, name, ?NAME) AND

(?X, has_cont_point, ?Y) AND

(?Y, type, Cont.Point) AND

(?Y, street, ?ADDRESS1) AND

(?Y, city, ?ADDRESS2))

When the results are returned, the concatenation function is executed in our system and the final results are unified. Similar strategy is followed for all GAV mappings that encode a function and is the result of encoding heuristics when detecting the change operations among ontology versions.

5.3 Discussion

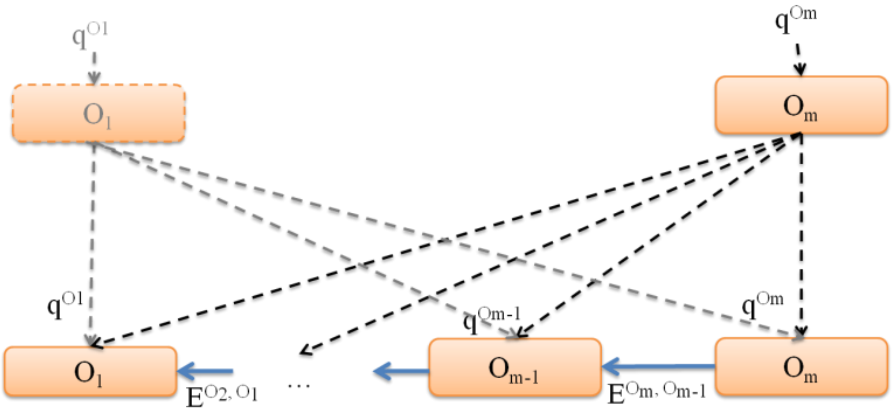


Fig. 36. Exploiting composition & inversion

5.3.1 Exploiting Composition.

So far we have described the scenario where we construct the change logs E^{O_m, O_i} between O_m and all O_i ($1 \leq i < m$) using the algorithm from (Papavassiliou, 2009). Then, we formulate a query q using the ontology version O_m, and we use the corresponding GAV mappings to produce and evaluate $valid_E(exp_{O_m}(q))$.

However, based on the composition property (Proposition 1), we could avoid the computation of all those change logs from scratch each time. Instead, of constructing E^{O_m, O_i}, for all i ($1 \leq i < m$), we could only construct all $E^{O_j, O_{j-1}}$ $(2 \leq j \leq m)$ between the subsequent ontology versions as shown in Fig. 36, thus minimizing the total construction cost[5] - since the compared ontologies now have more common elements. However, we have to keep in mind that the time of constructing a sequence of changes is spent only once during system setup.

Corollary 2: $E^{O_m, O_i} = \bigcup_i^{m-1} E^{O_{j+1}, O_j}$.

Proof: The proof directly follows from the fact that the change operations we consider compose (Proposition 1) ▪

[5] The complexity of the algorithm for input O_1, O_2 is $O(max(N_1, N_2, N^2))$ (Papavassiliou, 2009) where N_i is the size in triples of O_i, and N is the size of their set difference between O_1 and O_2.

Moreover, whenever a new ontology version occurs, we can construct the change log between the new ontology version and the previous ontology version - and not all change logs from scratch. Of course, this will lead to larger sequences of change logs, but will allow the uninterrupted introduction of new ontology versions to the system.

5.3.2 Exploiting Inversion

Ideally, we would also like to accept queries formulated using ontology version O_1 and to rewrite it to the newer ontology versions. This would be really useful since in many systems queries might be stored and we wouldn't like to change them every time the ontology evolves. However, in order to achieve this we would have to use the inverse GAV mappings for query rewritings which are not always possible to produce. Our approach deals with the inversibility on the level of change operations and not at the logical level of the produced GAV mappings. So, instead of trying to produce the inverse of the initial GAV mappings, we invert the sequence of changes (which is always possible according to Corollary 1) and then use the inverted sequence of changes to produce the GAV mappings that will be used for query rewriting to the current ontology version. This is also shown in Fig. 36 and enhances the impact of our approach.

Actually, it now becomes obvious that it is straight forward to accept a query formulated in any ontology version O_i ($1 \leq i \leq m$) and to get the rewritings for all ontology versions using the inverted list of changes for the O_j that $j > i$.

5.3.3 Non-information preserving changes.

Although in its basic form our query rewriting strategy produces equivalent rewritings, it turns out that problems may occur due to *non-information changes* between ontology versions. Consider as an example the query q_8 that asks for the "*gender*" and the "*name"* of an "*Actor*" using ontology version O_1.

q_8: $\pi_{?NAME,?GENDER}$ (

(?X, type, Actor) AND

(?X, name, ?NAME) AND
(?X, gender, ?GENDER))

Trying to rewrite the query q_8 to the ontology version O_2 our system will first expand it. Then it will consider the GAV mappings produced from the inverted sequence of changes (as they have been presented at the end of the sub-section 4.1). So, the following query will be produced by unfolding using the mapping $\forall x, y, name(x, y) \rightarrow fullname(x , y)$ - produced from $inv(u_1)$.

$\pi_{?NAME,?GENDER}$ (
(?X, type, Actor) AND
(?X, ***fullname****, ?NAME) AND*
(?X, gender, ?GENDER))

However, it is obvious that the query produced will not provide any answers when issued to the data integration system that uses O_2, since the "g*ender"* literal no longer exists in O_2. This happens because the $inv(u_4)$ change operation is not an *information preserving change* among the ontology versions. It deletes information from the ontology version O_1 without providing the knowledge that this information is transferred on another part of the ontology. This is also the reason that low-level change operations (simple triple addition or deletion) are not enough to dictate query rewriting and a high-level language of changes is preferable.

Although, this might be considered as a problem, actually it is not, since if we miss the literal "*gender*" in version O_2, this would mean that we have no data in the underlying local databases for that literal. However the query still will fail and we need a mechanism to a) notify the user for the failure and b) provide best approximations.

A question that arises is whether we could identify failures on the issued queries before the expansion phase. This would allow us to identify really fast the impact that the evolution has on the aforementioned queries. Although, we would identify the direct failures, the indirect ones (coming from the expansion of the queries) would not be identified. The case that such a mechanism would be useful would be when

mappings are considered to be exact between the ontology versions, or when ontologies are interpreted as global schemata without constraints.

Another question that arises is what happens if cycles occur on class/property hierarchies between the ontology versions. For example, imagine the scenario where we formulate a query asking the instances of a class A using ontology version 1. In that ontology version for class A has as a subclass the class B. However, after some changes we reach ontology version 2 where now the class A is a subclass of B. This is not a problem since the cycles affect only the algorithm for detecting the changes between ontology versions and the expansion phase. However, in both phases we enforce no cycles due to validity constraints, and our approach is not affected by cycles occurring between ontology versions.

5.3.3.1 Reasoning on queries.

The first option is to notify the user that some underlying data integration systems were not able to answer their queries and present the reasons for that. For our example, our system will report that the data integration system that uses O_2 was not able to answer the initial query since the literal "*gender*" does not exist in that ontology version. To identify the change operations that lead to such a result we define the notion of *affecting change operations*.

Definition 5.10 (Affecting change operation): *A change operation* $u \in E^{O_1,O_2}$ *affects the query q expressed using terms from* O_1*, denoted by* $u \Diamond q$*, iff*

I. $\delta_a(u)=\emptyset$

II. *there exists triple pattern* $t \in q$ *that can be unified with a triple of* $\delta_d(u)$.

The first condition ensures that the operation deletes information from the ontology without replacing it with other information, thus the specific change operation is not information preserving. However, we are not interested in general for the change operations that are not information preserving. We specifically target those change operations that change the ontology part which corresponds to our query (condition II).

Unification is a standard operation in logic programming. For more information see (Lloyd, 1987). The algorithm for identifying affecting change operations is shown in Fig. 37 and checks directly the change operations for the conditions described

above. The time complexity of the algorithm is *O(N*M*T)*, where *N* is the number of change operations in E^{O_1,O_2}, *M* is the number of triple patterns in q and *T* is the maximum number of triples in the $\delta_d(u)$ that $u \in E^{O_1,O_2}$.

Algorithm 5.1: *IdentifyAffectingOperations(q, E^{O_1,O_2})*

Input: The query q formulated using ontology version O_1 and the the evolution log E^{O_1,O_2}.

Output: The set of affecting change operations

1. $S := \emptyset$
2. For each $u \in E^{O_1,O_2}$
3. if $\delta_a(u) = \emptyset$ and $\exists\, t \in q,\ t' \in \delta_d(u)$ such that t unifies t'
4. $S := S \cup u$
5. Return S

Fig. 37. The algorithm for identifying affecting change operations for a query q

Proposition 4 (Correctness): *The algorithm IdentifyAffectingOperations identifies the affecting change operations for a given query q, over* E^{O_1,O_2}.

Proof: In line 2 the algorithm searches all change operations. For each one of those change operations, the algorithm checks the conditions in line 3. This immediately proves the claim▪

Having defined the notion of affecting change operation we will prove the following:

Proposition 5: *Let* $q=\bigcup_1^m q_i$. *If for all* q_i, *there exists* $u \in E^{O_1,O_2}$ *such that* $u \Diamond q_i$, *then* $valid_E(q)$ *returns no answers.*

Proof: The proof follows from the fact that if for a conjunctive query q, there exists $u \in E^{O_1,O_2}$ such that $u \Diamond q$ then according to the Definition 5.9 the change operation will delete a part from the next version of the ontology that q still queries. Since the part of the schema that q will query would not be available in O_2, this means that the query will not return any answers. And since for all q_i, there exists $u \in E^{O_1,O_2}$ such that $u \Diamond q_i$ this means that no subquery will return any answer▪

Users then can use this knowledge *a-priori* in order to re-specify their queries if desired.

5.3.3.2 Minimally-containing rewriting

The first approximation that we propose to the user is the minimally-containing rewriting (Halevy, 2001). A containing rewriting of *q* is a conjunctive query against the views which contain *q*. A minimally-containing rewriting is a containing rewriting which is contained in any other containing rewriting of *q*. It is thus the best "over-approximation" of *q* and it is dual to the "maximally-contained rewriting" which is the best "under-approximation of *q* as shown on Fig. 38.

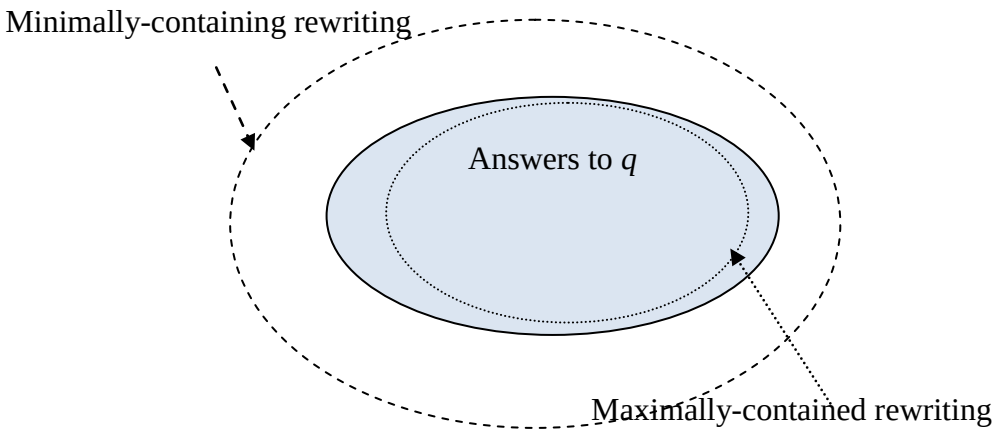


Fig. 38. Minimally-containing rewriting vs. Maximally-contained rewriting

Definition 5.11 (Minimally-Containing Rewriting (Afrati, 2005)): *A query q′ is a minimally-containing rewriting of a conjunctive query q using a set of mappings (views) M if and only if (1) q′ is a containing rewriting of q (q⊆ q′) and (1) there exists no containing rewriting q″ of q using M, such that the expansion of q″ properly contains the expansion of q′.*

Now we will present an algorithm shown on Fig. 39 and we will prove that the query *q′* that is computed by Algorithm 5.2 is indeed a minimally-containing rewriting of *q*, and thus it can be used in order to compute the minimally-containing rewriting of *exp(q).*

Theorem 5.3 (Correctness): *MinimallyContainingRewriting(q, E) is a minimally-containing rewriting of a conjunctive query q using E.*

Proof: In order to prove that *MinimallyContainingRewriting(q, E)* is a minimally-containing rewriting of *q* we will show that the Algorithm 5.2 is equivalent to the simplified version of the Chase/Backchase algorithm that has been proved (Deutsch, 2006) to output such a rewriting. Recall that the simplified version of Chase/Backchase for a query *q* is the following:

1. Chase *q* and obtain the universal plan *U*.
2. Restrict the body of *U* only to the vocabulary of views obtaining a query *M*.
3. If *M* is safe (i.e. head variables appear in the body) output *M*, otherwise output "no containing rewriting of *q* exists".

The first step of the algorithm consists of a number of chase steps. In each chase step a constraint is applied to the query. Each chase step is actually one unfolding step with the difference that the head of one constraint is not replaced by the body, but it is added to the query as well. Then in the second step, according to the simplified version of Chase/Backchase, the body of *U* is restricted to the vocabulary of views obtaining a query *M*. This step is actually the same as replacing the head of the mappings with their body. So the first two steps of the simplified chase algorithm behave exactly like the unfolding steps in our algorithm. The only difference is that in our case, several conjuncts might not be deleted in the unfolding step. However, according to the Definition 5.9 and Proposition 4, those conjuncts are discovered using the algorithm for identifying the affected change operations and the deletion of these conjuncts is actually performed on line 4 of our algorithm. Finally, the third step of the algorithm is the same in our case as well. So our algorithm is equivalent to the simplified chase/backchase and returns the minimally-containing rewriting of the initial query with respect to *E*▪

A query is safe if all variables in the head of the query appear in the body as well. Concerning the time complexity, the algorithm first needs to unfold the query (*O(N*M)* where *N* is the number of change operations in the evolution log and *M* is the number of sub-goals in the query) according to line 1 and then to detect the affecting change operations for the unfolded query (*O(N*UM*T)* where *UM* is the number of sub-goals for the unfolded query and *T* is the maximum number of triples in the $\delta_d(u)$ that $u \in E^{O_1,O_2}$). Finally the algorithms should search all subgoals of the unfolded query to identify the triples that unify with the affected changes and to delete

them (*O(UM*T*A)* where *A* is the number of the affected change operations). So the total time complexity is $O(N*M)+ O(N*UM*T)+$ O(UM*T*A)$\leq O(N*M)+ O(N*UM*T)+$ O(UM*T*N) $\leq O(N*M)+2\ O(N*M*T*T) \leq O(N*M)+ O(N*M*T^2) \leq O(N*M*T^2)$.

Algorithm 5.2: *MinimallyContainingRewriting(q, E^{O_1,O_2})*

Input: The conjunctive query query *q* formulated using ontology version O_1 and E^{O_1,O_2} the sequence of change operation from O_1 to O_2.

Output: The minimally-containing rewriting of *q* or *false*

1. q':= $valid_E(q)$
2. *A*:=*IdentifyAffectingOperations*(q', E^{O_1,O_2})
3. For each $a \in A$
4. Let $t \in q$, $t' \in \delta_d(a)$ such that *t* unifies t'
5. q':= q' - {*t*}
6. *A*:=*A* –{*a*}
7. If q' is safe
8. Return q'
9. else
10. Return *false*

Fig. 39. An alternative algorithm for computing minimally-containing rewritings

For example consider an alternative of the q_8 query, asking for the name of the male actors:

$\pi_{?NAME}$(
(?X, type, Actor) AND
(?X, name, ?NAME) AND (?X, gender, "Male"))

Obviously, the expander phase will not produce a new query and the valid rewriter will return the following query after considering the mapping $\forall x, y, name(x, y) \rightarrow fullname(x , y)$.

$\pi_{?NAME}$(
(?X, type, Actor) AND
(?X, fullname, ?NAME) AND

(?X, gender, "Male"))

However, the previous query will not return any answers when issued to the data integration system using O_2 since the property "*gender"* no longer exists. So, we have to search for minimally-containing rewritings, which will be produced by removing the conjuct asking for that property. So, the minimally-containing query that will be produced is:

$\pi_{?NAME}$ (

(?X, type, Actor) AND (?X, fullname, ?NAME))

And although, the minimally-containing rewriting is always unique (as proved in (Deutsch, 2007)), it is now always possible to produce it (when the query safety is not maintained). That's why we produce minimally-generalized queries, as well, presented bellow.

5.3.3.4 Generalized Queries

Besides providing an explanation for the failure of a sub-query, we can also produce *more general answers* for the data integration sub-systems that cannot answer input queries. Our solution here is that when a change operation *affects* a query rewriting, we can check if there is another triple t' (in the previous ontology version) which is the "*parent*" of the deleted triple *t*. The "*parent*" means that *domain(t)* is subclass of *domain(t')*, that *range(t)* is subclass of *range(t')* and that property *t* is subproperty of *t'*. If such a triple exists in the next ontology version we can ask for that triple instead, thus providing *a generalized query*.

Definition 5.12 (Generalized query): *Let q a conjunctive query expressed using O_1. We call q_{GEN} a generalized query of q over E^{O_1,O_2} iff:*

I. *q is contained in q_{GEN} ($q \subseteq q_{GEN}$)*
II. *It does not exist $u \in E^{O_1,O_2}$ such that* u ◊ $q_{GEN.}$

Now we will define the notion of minimally-generalized query.

Definition 5.13 (Minimally-Generalized query): *A generalized query q_{GEN} of q over E^{O_1,O_2} is called minimal if there is not q_{GEN}' such that $q \subseteq q_{GEN}'$ and $q_{GEN}' \subseteq q_{GEN.}$*

The idea of minimally-generalized query is that it is a query that can be answered on the evolved ontology version after applying the minimum number of *"repairs"* on the query in order to achieve that. The algorithm for producing a minimally-generalized query over E^{O_1,O_2} for a given query q is shown in Fig. 40.

The algorithm *getParent* is implemented by just querying a reasoner (*Pellet*[6] for example) and returns the first direct "*parent"* triple of t' if many exists (in lexicographic order). Moreover, it always terminates since the affecting change operations are finite. Our algorithm runs in *O(A*N*M*T),* where A is the maximum number of affecting change operations, N is the number of change operations in E^{O_1,O_2}, M is the number of triple patterns in q and T is the maximum number of triples in the $\delta_d(u)$ that $u \in E^{O_1,O_2}$. Now we will prove the correctness of our algorithm.

Algorithm 5.3: *MinimallyGeneralizedQuery*(q, O_1, E^{O_1,O_2})

Input: The query q formulated using ontology version O_1 and E^{O_1,O_2} the sequence of change operations from O_1 to O_2.

Output: A minimally-generalized query of q or *false*

```
1.  q':=q
2.  A:=IdentifyAffectingOperations(q', E^{O1,O2} )
3.  While(A≠ ø)
4.      Let a ∈ A
5.      Let t ∈ δa(a)∪ δd(a) such that t unifies with t' ∈ q'
6.      parent:= getParent(t)
7.      If parent ≠ ø then
8.          Replace t' with parent in q'
9.      else
10.         q':= false
11.         break
12.     A:=IdentifyAffectingOperations(q', E^{O1,O2} )
13. Return q'
```

Fig. 40. The algorithm for identifying a minimally-generalized query

[6] http://clarkparsia.com/pellet/

Theorem 5.2: *The algorithm MinimallyGeneralizedQuery produces a minimally-generalized query of q over* E^{O_1,O_2} .

Proof: First we have to show that (a) the query produced is actually a generalized query and then that (b) the generalized query produced is minimal. Before proceeding in the proof recall that if q and q' are two queries (of the same arity) for a schema *S*, we say that q in contained in q' with respect to *S*, denoted by $q \subseteq q'$, if $q^{MO} \subseteq q'^{MO}$, i.e. the result of evaluating q is a subset of the results of q' evaluation for every model *MO* of *S*.

(a) Now in order to show that q' produced from Algorithm 5.3 is a generalized query we have to show that i) $\nexists u \in E^{O_1,O_2}$ such that $u \Diamond q'$ and ii) that $q \subseteq q'$. Indeed from line (line 3) we remove each time one affecting change operation until $A = \emptyset$. So if the algorithm finishes (and $q' \neq$false) there would not be any change operations affecting q'. Moreover, since in one iteration a triple pattern $t' \in q_1'$ is replaced with its parent to produce q_2' the answers to q_1' would be contained in the answers to q_2', thus $q' \subseteq q''$. By repeating the same operation $q_1' \subseteq q_2', \ldots, q_m' \subseteq q_{m+1}'$, and thus $q_1' \subseteq q_{m+1}' = q'$ by transitivity.

(b) Now we have to show that the generalized query produced q' is minimal. Let's suppose that it is not minimal. This would allow the existence of a minimal generalized q_{min} such that $q \subseteq q_{min}$ *and* $q_{min} \subseteq q'$. By $q_{min} \subseteq q'$ this would mean that $\exists t' \in q'$ such that t' is parent of $t \in q_{min.}$ But in order to construct q' we only use a parent triple pattern if a change operation affects that triple. This means that t is affected by a triple patter. Thus, q_{min} is not a generalized query which is not true▪

Although, the generalized query produced from the previous algorithm is always minimal, however, it might not be unique. It might be the case that several other minimally-generalized rewritings may exist as well, since we might have multiple super-properties of a deleted property, and all might be minimal wrt. the initial query, since they add different set of answers to the answer of q. An algorithm that identifies all minimally-generalized rewritings for a query q, is shown on Fig. 41. The algorithm behaves exactly like the algorithm for identifying a minimally-generalized rewriting but instead of limiting the options for replacing a deleted triple with the "parent" triple it considers all different combinations. The complexity of the

algorithm obviously is $O(S^M*N*T*M^2)$ where S is the maximum number of super-properties of a property, N is the number of change operations in E^{O_1,O_2}, M is the number of triple patterns in q and T is the maximum number of triples in the $\delta_d(u)$ that $u \in E^{O_1,O_2}$.

Algorithm 5.4: *MinimallyGeneralizedQueries*(q, O_1, E^{O_1,O_2})

Input: The query q formulated using ontology version O_1 and E^{O_1,O_2} the sequence of change operation from O_1 to O_2.

Output: The set of minimally-generalized rewritings of q

```
1.  queries:={q}
2.  For each query ∈ queries
3.      A:=IdentifyAffectingOperations(query, E^{O1,O2} )
4.      While(A≠ ø)
5.          Let a ∈ A
6.          Let t ∈ a such that t unifies with t' ∈ query
7.          parents:= getParents(t)
8.          tempquery:= query
9.          For i=1 to size(parents)
10.             If i=0 then
11.                 query:=Replace t' with parents[i] in query
12.             Else
13.                 queries:= queries ∪ Replace t' with parents[i] in tempquery
14.         if size(parents)=0 then
15.             queries:= queries - query
16.             break
17.         A:=IdentifyAffectingOperations(q', E^{O1,O2} )
18.  Return queries
```

Fig. 41. The algorithm for identifying all minimally-generalized query

Theorem 5.3: *The algorithm MinimallyGeneralizedQueries produces all minimally-generalized query of q over* E^{O_1,O_2}.

Proof: First we have to show that (a) the query produced is actually a generalized query and then that (b) the generalized query produced is minimal. Before proceeding

in the proof recall that if q and q' are two queries (of the same arity) for a schema S, we say that q in contained in q' with respect to S, denoted by $q \subseteq q'$, if $q^{MO} \subseteq q'^{MO}$, i.e. the result of evaluating q is a subset of the results of q' evaluation for every model MO of S.

(a) Now in order to show that q' produced from Algorithm 5.4 is a generalized query we have to show that i) $\nexists u \in E^{O_1,O_2}$ such that $u \Diamond q'$ and ii) that $q \subseteq q'$. Indeed from line (line 4) we remove each time one affecting change operation until $A = \emptyset$. So if the algorithm finishes (and *queries*$\neq\emptyset$) there would not be any change operations affecting q'. Moreover, since in one iteration a triple pattern $t' \in q_1'$ is replaced with its parent to produce q_2' the answers to q_1' would be contained in the answers to q_2', thus $q' \subseteq q''$. By repeating the same operation $q_1' \subseteq q_2', \ldots, q_m' \subseteq q_{m+1}'$, and thus $q_1' \subseteq q_{m+1}' = q'$ by transitivity for each one of the subqueries in *queries*.

(b) Now we have to show that each the generalized query produced q' is minimal. Let's suppose that it is not minimal. This would allow the existence of a minimal generalized q_{min} such that $q \subseteq q_{min}$ *and* $q_{min} \subseteq q'$. By $q_{min} \subseteq q'$ this would mean that $\exists t' \in q'$ such that t' is parent of $t \in q_{min}$. But in order to construct q' we only use a parent triple pattern if a change operation affects that triple. This means that t is affected by a triple patter. Thus, q_{min} is not a generalized query which is not true▪

Assume for example, an alternative ontology version O_1, where the *"personal_info"* property is a super-property of the "*gender*" property. Assume also the same sequence of changes from O_1 to O_2 (the list of inverted changes presented in Chapter 4). Then, if query q_7 previously described is issued, we will be able to identify that the triple "*Actor, gender, xsd:String*" has been deleted and to look for a more general query. The query that our system produces, and that provides a more general answer to user query is:

$\boldsymbol{q_8}$: $\pi_{?NAME,?GENDER}$ (
(?X, type, Actor) AND
(?X, fullname, ?NAME) AND
(?X, personal_info, ?GENDER)

5.4 A real example from CIDOC-CRM

Now we will present a real example from the CIDOC-CRM using a simple template query from (Theodoridou, 2010). Assume for example that the user would like to get all objects used to capture an image. The corresponding SPARQL query formulated using the CIDOC-CRM version 4.2 is:

```
SELECT $x WHERE {
$a rdf:type "E38.Image";
    :P108B.was_produced_by $y.
 $y rdf:type "E5.Creation ";
    :P8F.took_place_on_or_within $x.
$x  rdf:type "E22.Man-Made_Object".
 }
```

The query is issued to the system and initially it is expanded using the QuOnto engine. The engine will identify the subclasses and the sub-properties of the used classes/properties and it will produce the following query:

$\pi_{?x}$((?a, type, E38.Image) AND
(?a, P108B.was_produced_by, ?y) AND
(?y, type, E65.Creation) AND
(?y, P8F.took_place_on_or_within, ?x) AND
(?x, type, E22.Man-Made_Object))
UNION
...
UNION
$\pi_{?x}$((?a, type, E38.Image) AND
(?a, P108B.was_produced_by, ?y) AND
(?y, type, E65.Creation) AND
(?y, P8F.took_place_on_or_within, ?x) AND
(?x, type, E84.Information_Carrier))

Assuming that we have databases mapped to the ontology version 3.2.1, the "*Valid Rewriter"* will check the constructed evolution log and will identify the mappings that should be used for unfolding. The only mappings that will be used are the ones occurring from the following change operation:

Rename_Class(E65.Creation, E65.Creation_event)

So, the query that it is issued on the data integration system that uses the version 3.2.1 is:

$\pi_{?x}$*((?a, type, E38.Image) AND*
(?a, P108B.was_produced_by, ?y) AND
(?y, type, ***E65.Creation_event****) AND*
(?y, P8F.took_place_on_or_within, ?x) AND
(?x, type, E22.Man-Made_Object))
UNION
...
UNION
$\pi_{?x}$*((?a, type, E38.Image) AND*
(?a, P108B.was_produced_by, ?y) AND
(?y, type, ***E65.Creation_event****) AND*
(?y, P8F.took_place_on_or_within, ?x) AND
(?x, type, E84.Information_Carrier))

However, the class "*E84.Information_Carrier"* is not available to the ontology version 3.2.1 since it was added later to the ontology. So, no equivalent rewriting can be produced and we have to go for minimally-containing rewritings. So the following query is produced which is a minimally-containing rewriting of the initial query.

$\pi_{?x}$*((?a, type, E38.Image) AND*
(?a, P108B.was_produced_by, ?y) AND
(?y, type, E65.Creation_event) AND
(?y, P8F.took_place_on_or_within, ?x) AND
(?x, type, E22.Man-Made_Object))
UNION

...
UNION
$\pi_{?x}$((?a, type, E38.Image) AND
(?a, P108B.was_produced_by, ?y) AND
(?y, type, **E65.Creation_event**) AND
(?y, P8F.took_place_on_or_within, ?x)

This example, is minimalistic and its purpose is only to show how our approach is applied on the evaluation scenarios.

5.5 Conclusions

5.5.1 Language of changes independent approach

The first question that naturally arises from our approach is whether the language of changes we adopt is the only language that could be used. The answer to that question obviously is no. In fact, any high-level language that guarantees uniqueness, non-ambiguity and completeness could be used as well. However, our choice among the possible languages would have to be based on the following properties:

a) Individual *add*/*del*: First and most important a high-level language of changes is better than another if the sequence of changes among two ontology versions yields a smaller number of change operations *u* such that $\delta_a(u)=\emptyset$ or $\delta_d(u)=\emptyset$.
b) Now if we assume that the languages of changes under consideration return the same number of individual *add*/*del* a language of changes is better than another the more fine-grained change operations it has. The more fine-grained are the change operations, the better the specification of logical mappings among the ontology versions. This is due to the fact that we have to resort to heuristics as the change operations become more coarse-grained.
c) Finally, desirable but not required properties would be composition and inversion, in order to be able to compose and invert the sequence of changes instead of trying to compute them each time.

However, is we assume that the heuristic change operations of a specific language can be correctly translated to a number of fine-grained change operations

without affecting the number of individual add/deletes, then our approach becomes independent of the language of changes used. In fact, we could even allow the experienced users to specify manually the logical mappings they desire among the ontology versions and our results will be exactly the same.

5.5.2 More generic than mapping composition.

Another question that arises is how our approach can be compared with mapping rewriting/composition. In fact since the change operations are interpreted as GAV mappings, they can always be composed with the initial mappings of the data sources as shown in (Fagin, 2011). (Of course this would require SO dependencies that would complicate the mappings making them difficult to understand for the domain experts). This would lead to a setting where the users can issue queries formulated using the past ontology version and retrieve information from data sources mapped with the current ontology version as well (and all intermediate versions). However in order to use the current ontology version to formulate queries that will be answered by the past ontology versions difficulties can occur, due to the requirement of the inversion of schema mappings. Although partial solutions exist, such as quasi-inverse (Fagin, 2008), chase-inverse (Fagin, 2011) and maximum-recoveries (Arenas, 2008) it still remains a difficult and open problem.

In our approach however, we have not such limitations and as a result the approach to rewrite the mappings can be seen as a specific use case of our solution. We provide a more general solution, allowing the users to formulate queries using all ontology versions. This is due to the fact that we consider inversion (composition) on a layer on top where always can find the inverse (composition) of any sequence of changes efficiently.

Chapter 6

Implementation & Evaluation

"Not everything that counts can be counted and not everything that can be counted counts."

- Albert Einstein

Contents

6.1 Implementation

In order to show the feasibility of our approach we implemented the *"exelixis"* platform. It is a web-based application that is developed in a three-tier architecture. HTML and JQuery are used for the presentation layer, the business logic is implemented on an Apache Tomcat[7] server using Java, and PostgreSQL[8] stores the parameters of our system. Moreover, our system uses the Pellet[9] reasoner for producing minimally-generalized queries and the QuOnto reasoner for expansion, and interacts with underlying data integration systems in order to enable query answering. The platform can be accessed online and the architecture of the system is shown on Fig. 42.

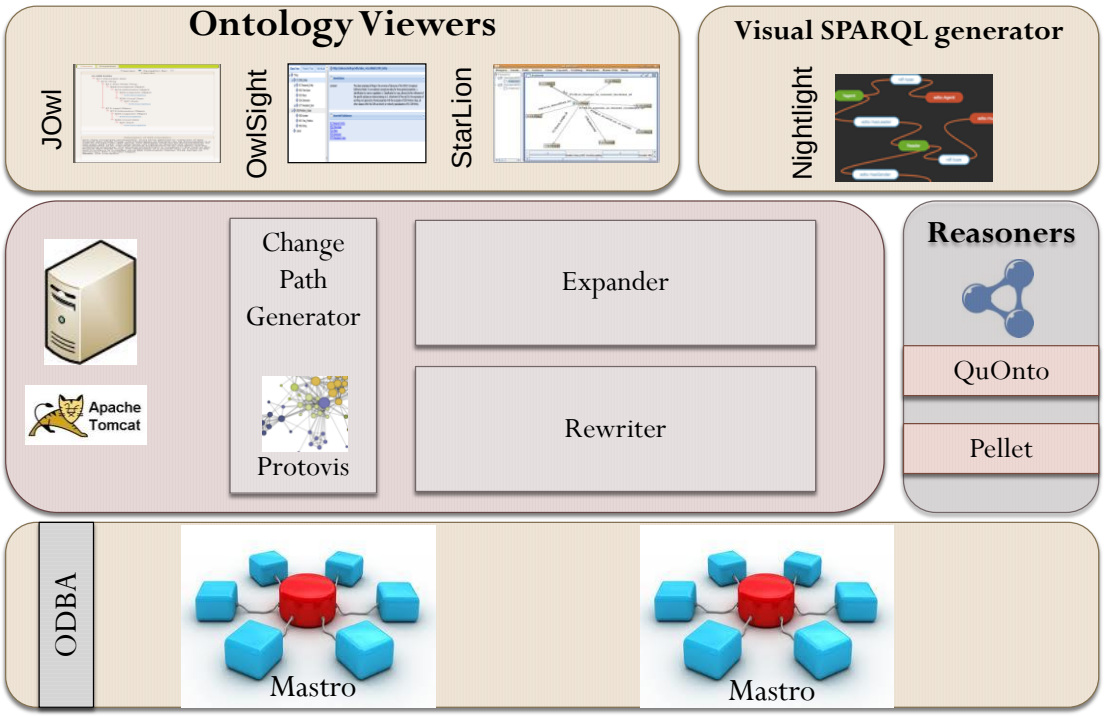


Fig. 42. System Architecture

The initial page that appears when visiting our home page is shown on Fig. 43. The user is able to see the ontology currently in use and to formulate queries for either the ontology or the underlying data sources by selecting the appropriate button from

[7] http://tomcat.apache.org/

[8] http://www.postgresql.org/

[9] http://clarkparsia.com/pellet/

the menu placed on the top of the page. Moreover, the user can select the parameters button in order to define the parameters of our system. The four options appearing on the top menu shown on Fig. 43 are:

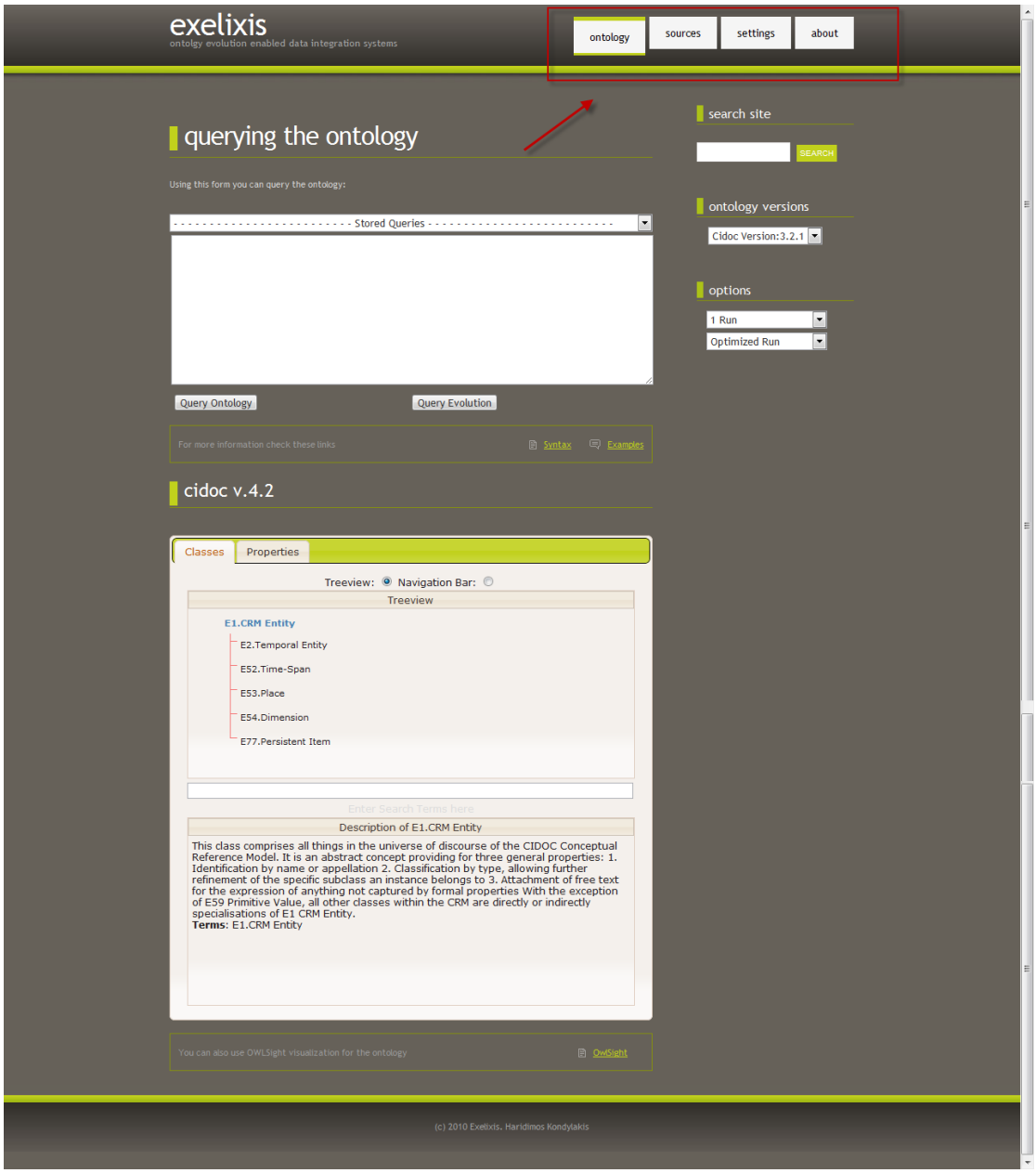


Fig. 43. The initial screen of our platform

1. **Ontology**: By selecting this option the user is able to visualize and query a selected ontology.
2. **Sources**: By selecting this option the user can issue SPARQL queries and get the rewriting among ontology versions. The queries are forwarded to the underlying data integration systems to be answered.

3. **Settings**: By selecting this option the user is able to define the parameters of our system.
4. **About**: By selecting this option, information about the system and the corresponding publications and terms of use appear.

The different functionalities will be described in detail in the following sub-chapters.

6.1.1 Setting the parameters

Before using the system, the user has to set the parameters for our system. This is performed by selecting the "*settings*" button on the initial web page. Then, the web page shown on Fig. 44 is presented to the user where he is able define the ontologies, the change logs, and the data integration systems that will be used.

Fig. 44. Defining the settings of our platform

On the top of the page are presented the ontologies and their different versions. The user is able to download, delete or visualize the selected ontology by selecting the corresponding link. Moreover, the user can upload a new ontology file by defining also the name and the version of the uploaded ontology.

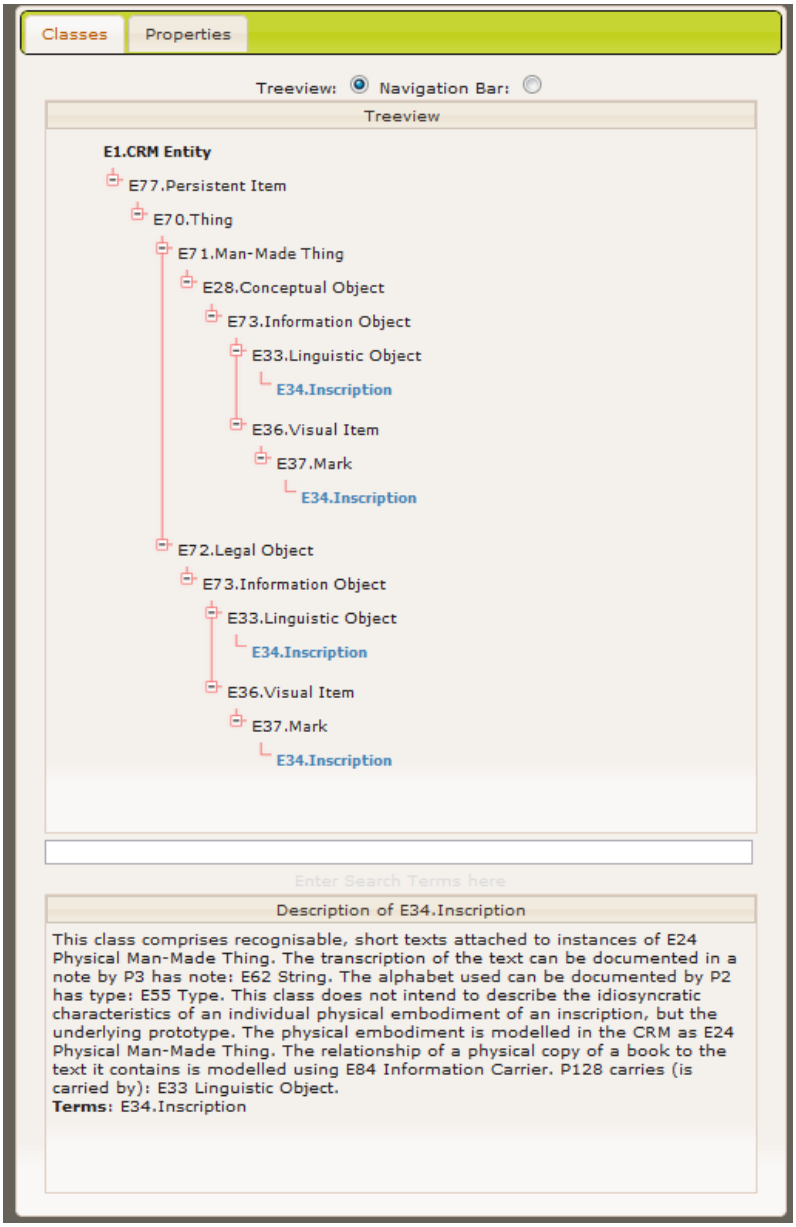


Fig. 45. Visualization using jOWL Api

Having defined the ontology versions in use, the system also presents the change logs between the different ontology versions. The user is able to delete, download and visualize the change log of two selected ontologies. Moreover, the user is able to visualize the inverse of a change log and to compose two or more change logs. Additionally, the user is able to upload a new change log by selecting the appropriate ontology versions.

Finally, the user is able to specify the url of the underlying data integration systems in use, and to select the ontology and the versions that each specific data integration system uses.

6.1.2 Visualizing Ontologies

In order to issue queries using an ontology, the first task is to present it to the user. The user is able to select the ontology and the version he wishes, which is subsequently visualized. Three options are provided for visualizing an ontology.

The first one is by embedding the ontology terms/properties directly on our web page. This is achieved using the jOWL[10] API and it is shown on Fig. 45. That specific API provides functions for querying the ontology, as well, which we also use to provide answers for queries on the ontology level. So, the user is able to search for a class or a property, to visualize the corresponding description and to explore the hierarchy either as a tree or as a navigation bar.

The second visualization option we provide for our users is to use the OWLSight[11] plug-in offered by the Pellet reasoner. By selecting this option the user is able to see the interface shown on Fig. 46. The interface is closest to the current state-of-the-art ontology editors and it is more intuitive than the jOWL approach. However, it cannot be embedded on our web page (it opens in another web-page), and is not managed by us.

Fig. 46. Using OwlSight plug-in for ontology visualization

[10] http://jowl.ontologyonline.org/

[11] http://pellet.owldl.com/ontology-browser/

The last visualization option we provide to the users is the StarLion (Zampetakis, 2010) tool as shown on Fig. 47. It can be loaded using Java Web Start directly from our homepage and it loads ontologies expressed in .rdfs files. Its distinctive characteristics are:

1. provision of Top-k diagrams for aiding the process of understanding large in size ontologies,
2. configurable force-directed layout algorithms (appropriate for semantic networks)
3. support of a semi-automatic layout process (where the user can change node positions, nail down nodes, apply layout algorithms, etc),
4. star graph-based (with variable radius) exploration mode.

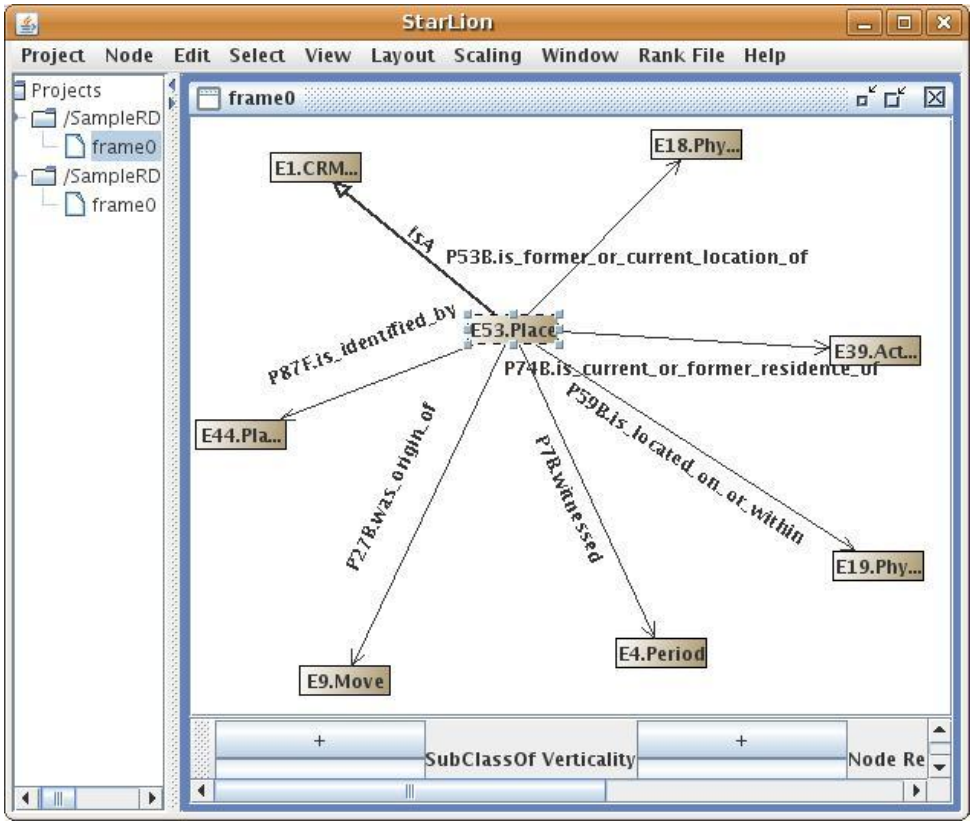


Fig. 47. The interface of the StarLion system.

Those three options provide a complete solution for the visualization of the ontologies used.

6.1.3 Querying Ontologies & Evolution

Fig. 48. Querying the Ontology

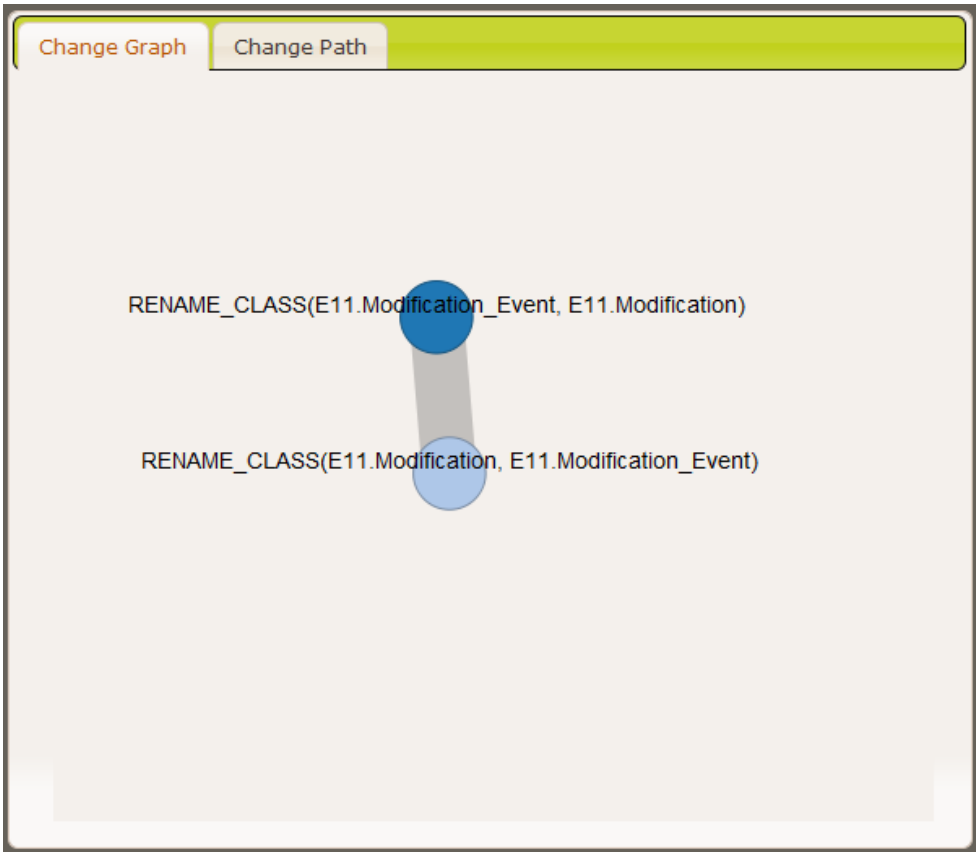


Fig. 49. Example query about the evolution of the ontology

After visualizing the corresponding ontology, the user is able to issue queries concerning either the ontology or the evolution of the corresponding ontology. This is performed by entering an appropriate query and pressing the corresponding button. All queries should be placed in the textbox in the middle of the page as shown on Fig. 48.

The syntax and example queries are shown below the text-area and appear after selecting the corresponding links. The user is able to query about classes, properties, subproperties or change paths in ontology evolution. Moreover, the user is able to select how many times the same query will be executed (for testing purposes) and if he would like debugging messages to be presented as well. Those options can be found at the right of the page.

Fig. 50. The change path in detail

By entering a query and pressing the appropriate button the results are shown to the user. For example in Fig. 49 the results of the query *"how(E11.Modification,,)"* are shown. The query asks for the evolution of ontology concerning the class *"E11.Modification"*. Our algorithm is executed and it reports that the previous name of the class was "*E11.Modification_Event*" and that before that the class was also name "*E11.Modification"*. This simple example shows the great value of change paths for describing evolution and can be used from ontology developers to identify the modelling choices of the past. Besides the graph of the change path, the users are

able to check the comments added as well to the ontology concerning those changes. This is shown in Fig. 50.

6.1.2 Querying data sources

If the user chooses to query the underlying data sources and to rewrite queries among ontology versions, the following page is presented.

Fig. 51. Querying the sources

The user interface consists of 3 main areas as shown on Fig. 51. In the centre of the web page the user can formulate the SPARQL query that will be submit to the system.

On the right of the text area for the query formulation there are several running options. Initially the user can choose the ontology versions that will be used as shown

on Fig. 52. Then, he can choose the running options that concern the performance of the system. If we choose the "*Optimized Run*" option the user will be able to see only the answers produced at the end of the whole process. However, if the user chooses the *"Debug Run"* he is able to check all the steps performed in order to produce the final results, but with degraded performance. Moreover, the user can select the number of runs for each experiment.

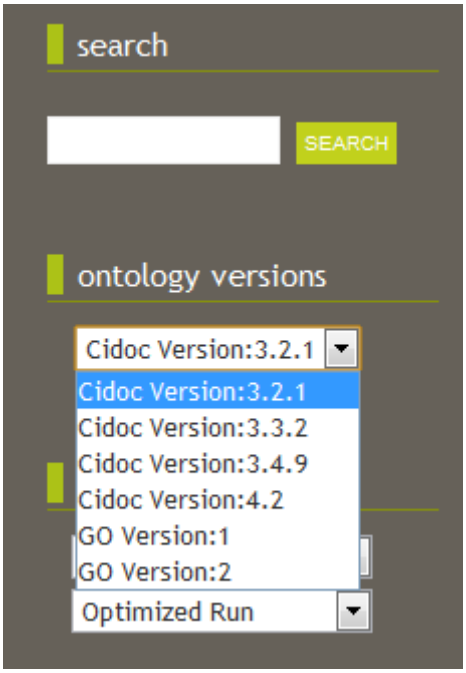


Fig. 52. Selecting ontology versions and the running options

Besides changing the different parameters of our system the user is also able to graphically formulate the SPARQL query using a modified version of the NiteLight (Russel, 2008) plug-in. NITELIGHT uses a Visual Query Language (VQL), called vSPARQL, which provides graphical formalisms for SPARQL query specification. NITELIGHT is a highly reusable Web-based component, and that is embedded in our platform. This is shown on Fig. 53.

The users can drag-n-drop variables, classes and properties from the selected ontology and to graphically design the graph pattern of the corresponding SPARQL query. The SPARQL query is generated on-the-fly and presented at the bottom of the web page.

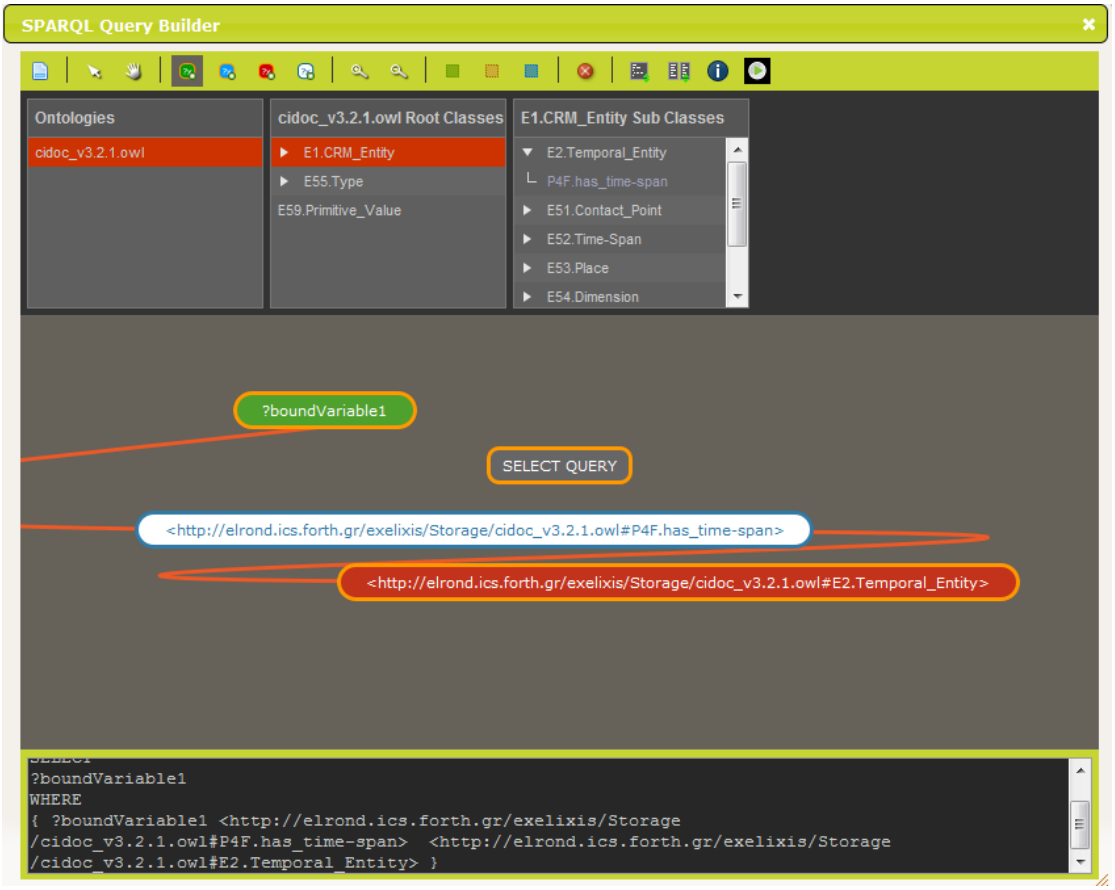


Fig. 53. Visual Query Builder

Having selected all running options and by pressing the "*Submit*" button the query is issued to the system. After a while at the bottom of the page, we can see 4 more tabs with the results of the execution of our algorithms.

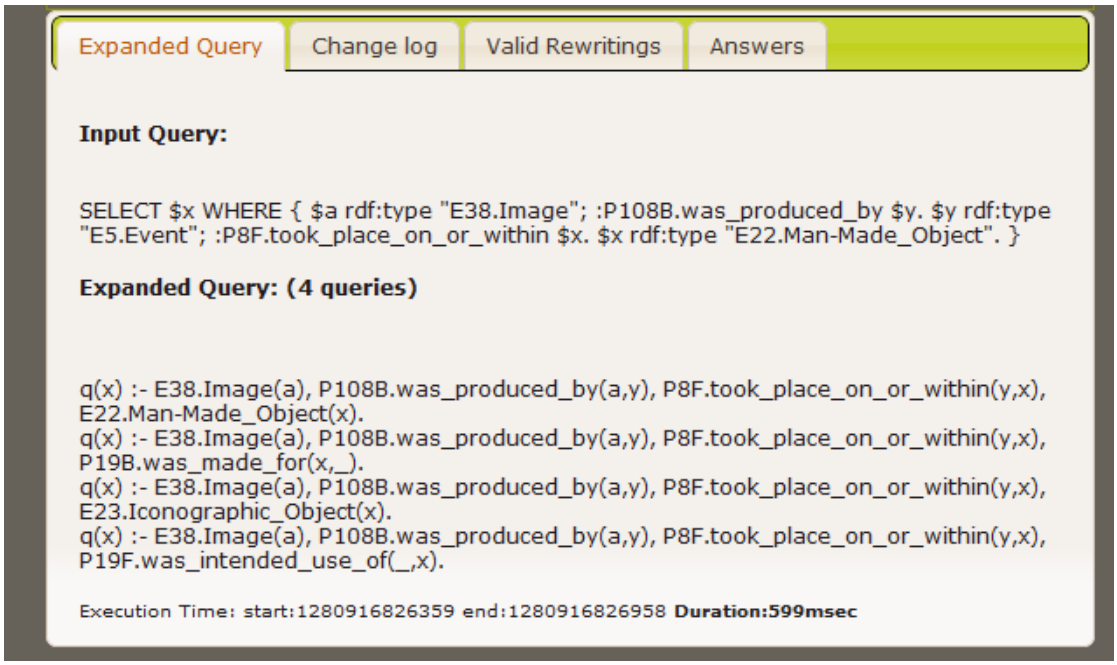


Fig. 54. The query converted to Datalog

On the first tab we can see the expanded query produced from the QuOnto reasoner as shown on Fig. 54. At the bottom of each tab, we can see the time spent of the specific part of our algorithm execution.

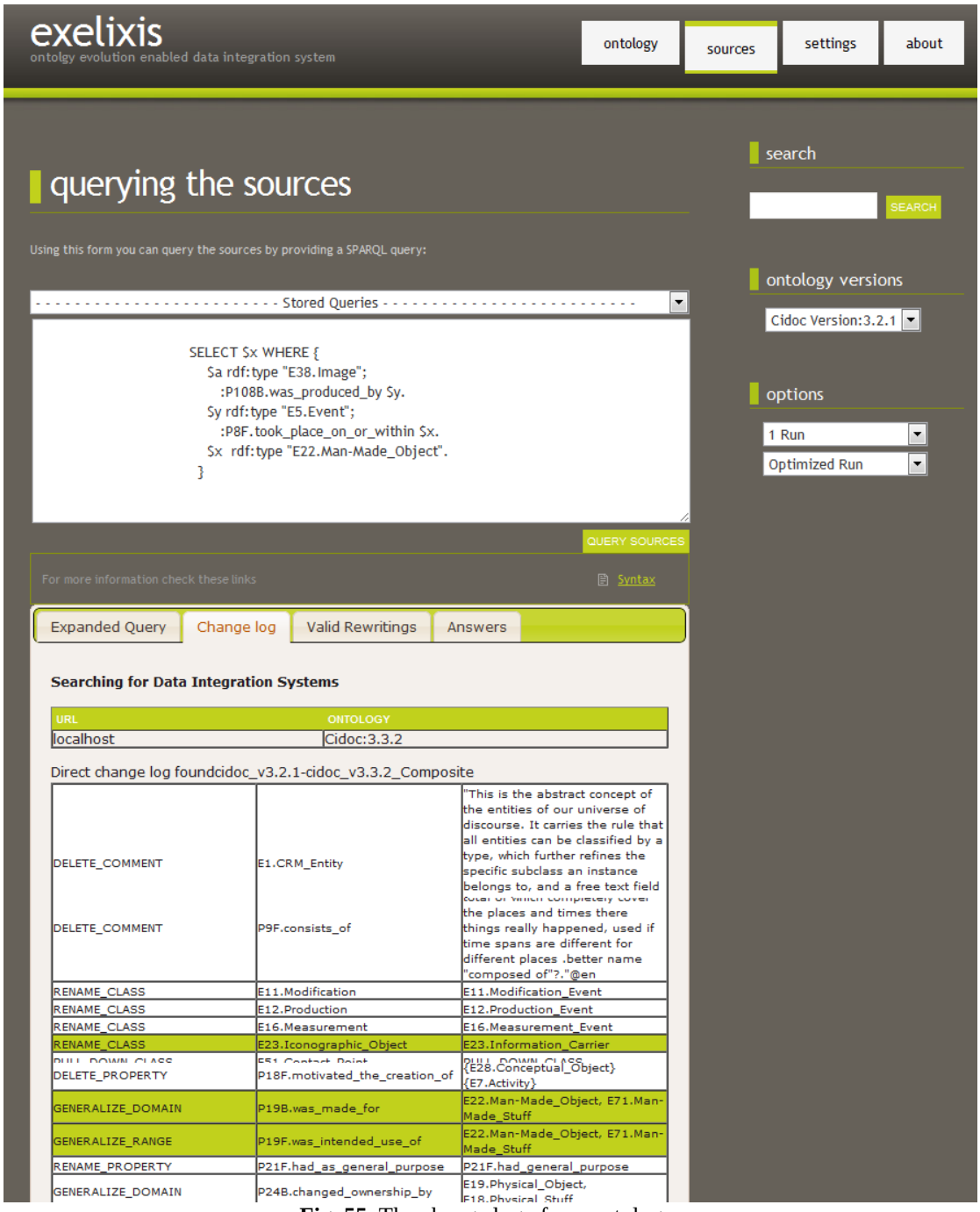


Fig. 55. The change log of our ontology

Going on the second tab, our system is able to identify the underlying data integration systems that use the same ontology and to search for the corresponding change logs. If the "*Debug Mode*" is selected, we are able to see the change operations that change the expanded query and that are used in order to produce the valid rewriting of the expanded query. Those change operations are colored in green

whereas if there are change operations that affect the expanded query they are colored in orange. This is show on Fig. 55.

On the third tab, the user can check the valid rewritings of the expanded query to the other ontology versions. This is shown on Fig. 56 where we can identify in our example the query that is going to be forwarded to our underlying data integration system.

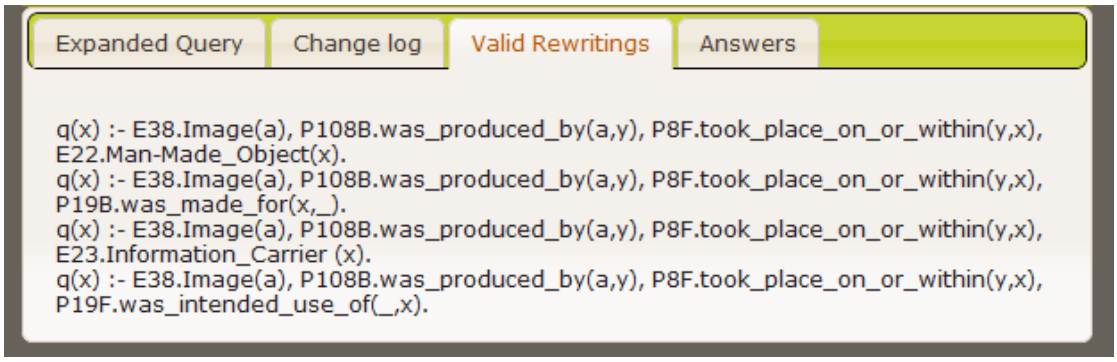


Fig. 56. The ontology rewritten queries

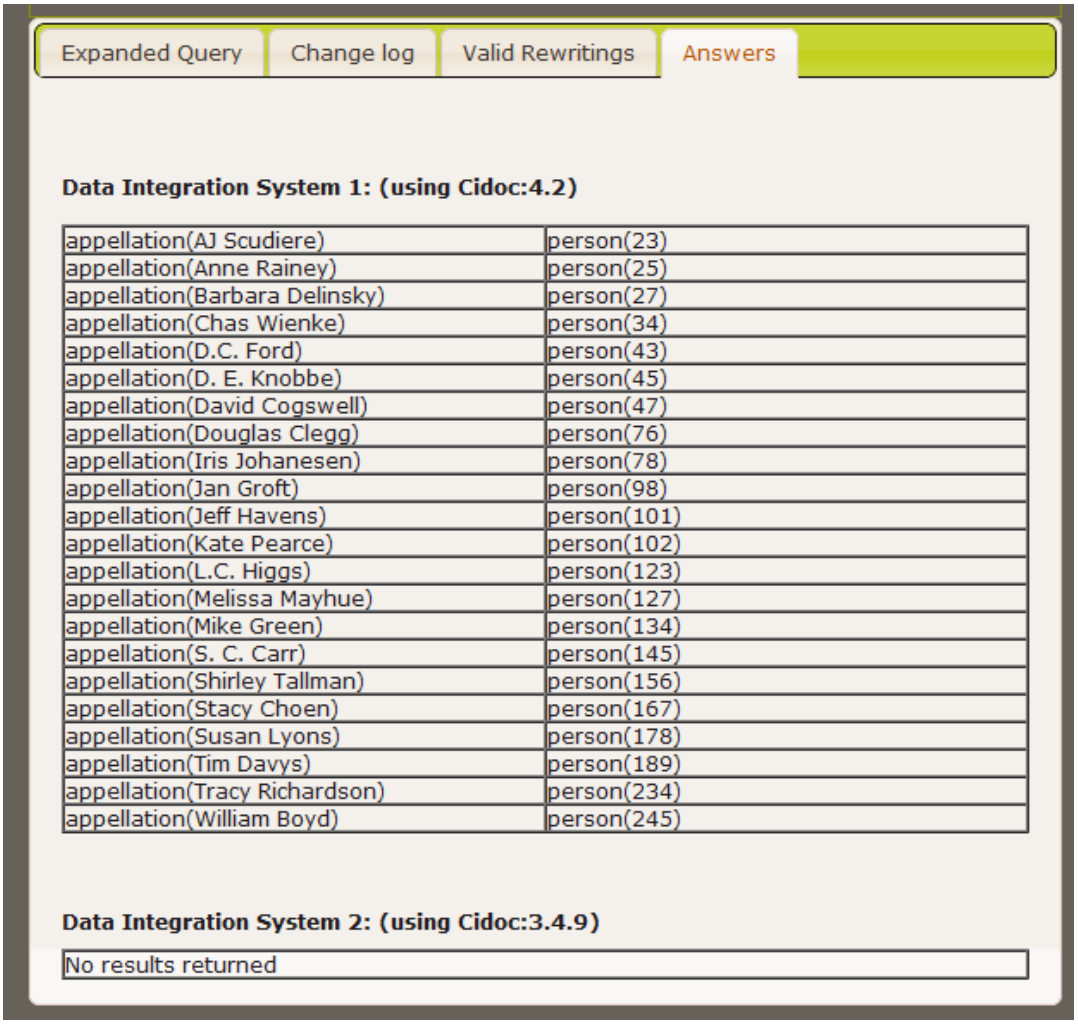


Fig. 57. The results tab

Finally on the last tab we are to see the answers from the underlying data integration systems, as shown on Fig. 57.

6.2 Evaluation

In order to evaluate our system we used a machine running Windows 7 with an Intel Core 2 Duo processor at 3.0Ghz, and 3GB memory. Moreover, to test our system we used two ontologies. One medium-sized ontology (CIDOC-CRM (Martin, 2007)) from the cultural domain which is rarely changed and one large-sized ontology (Gene Ontology (Gene Ontology Consortium, 2004)) from the bioinformatics domain which is heavily updated daily.

CIDOC-CRM is an ISO standard which consists of nearly 80 classes and 250 properties, but no instances. For our experiments we used versions dated from 02.2002 (v3.2.1) to 06.2005 (v4.2) encoded in RDF/S. The detected change log that was produced identified 711 total changes.

Gene Ontology (GO) (Gene Ontology Consortium, 2004) on the other hand, is composed of about 28000 classes and 1350 property instances. We have to note that the file containing the Gene ontology is over 100MB and most of the ontology editors fail to load the entire file. GO is updated on a daily basis and for our experiments we used the change log from 16.12.08 to 26.05.09. The change log that was produced contained 3482 changes.

6.2.1 Computing Change Paths

In order to check the running time of our system we exhaustively queried for the change tree for all classes and properties appearing in a change operation in those two ontologies. In our first experiment we allowed all 3482 changes from GO and 711 changes from CIDOC to be processed in order to identify the correlation between the size of the change path identified and the time to identify it.

GO v.26.05.09-v.16.12.08 (3482 changes)		
Change Path Size	No of such paths	Avg. Time (msec)
0	750	0,2921
1	1910	0,2922
2	456	0,2927
3	151	0,2930
4	95	0,2952
5	42	0,2992
6	26	0,2998
7	8	0,3020
8	14	0,3329

Table 2 . The correlation between the size of the change path and the average time spent for identifying it using GO

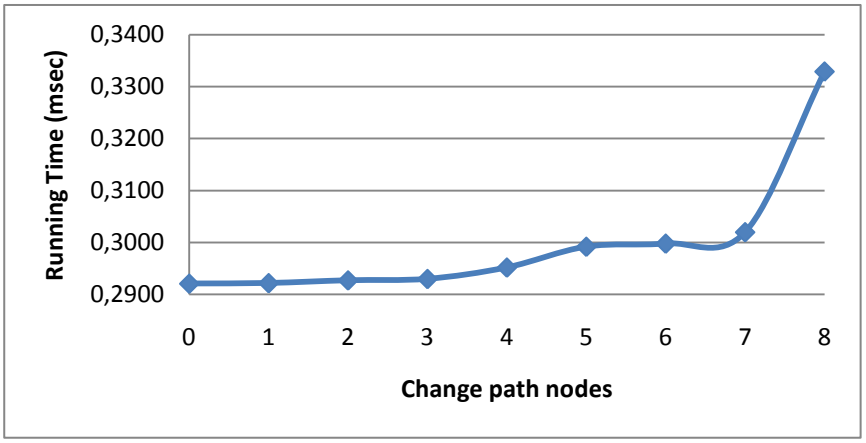


Fig. 58. The graph for visualizing the change path size related to the average time spent for identifying it using GO

Our experiments concerning GO are shown on Fig. 58 and Table 2. We can see that the maximum change path has only 8 nodes, which can be easily processed by humans. Moreover, we can see that even for computing a change path with 8 nodes over 3482 changes we need only 0,3329 msec. Finally, we can see that the time spent becomes larger, the more nodes in a change path we have to compute which is reasonable since we have to search the remaining change log for the new nodes that are added to the change path as well. The same experiments for CIDOC-CRM showed the following statistics shown on Table 3 and Fig. 59.

CIDOC v4.2-v.3.2.1 (711 changes)		
Change Path Size	No of such paths	Avg. Time (msec)
0	79	0,0163
1	49	0,0164
2	218	0,0177
3	74	0,0182
4	20	0,0188
5	2	0,0193

Table 3 The correlation between the size of the change path and the average time spent for identifying it using CIDOC

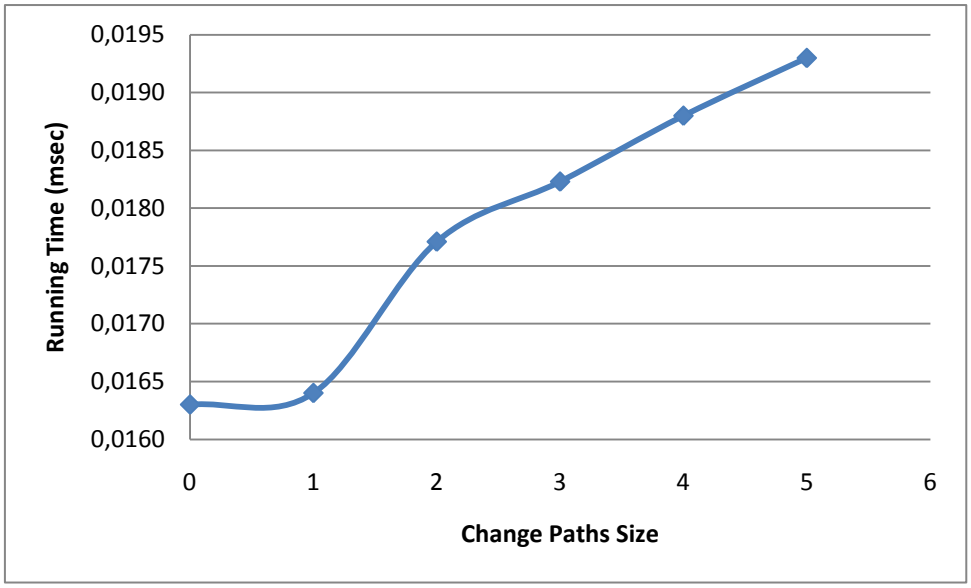


Fig. 59. The graph for visualizing the change path size related to the average time spent for identifying it using CIDOC

We can see that the maximum change path for CIDOC-CRM has only 5 nodes since the ontology is smaller and more stable than GO. This is obvious from the number of change operations as well – 711 for CIDOC vs. 3482 for GO. That's why the average time for identifying such change paths is smaller (0,02 msec for computing a change path with 5 nodes). Moreover, the graph shows that the time spent becomes larger, the more change paths we have to compute here as well (almost linear).

Then we tried to identify how the time for computing the change path is affected by the size of the change log. From the diagrams in Fig. 60 and Fig. 61 below we can

see that the more changes are there to process of, the more time is spent on processing those, and in fact their relationship is linear.

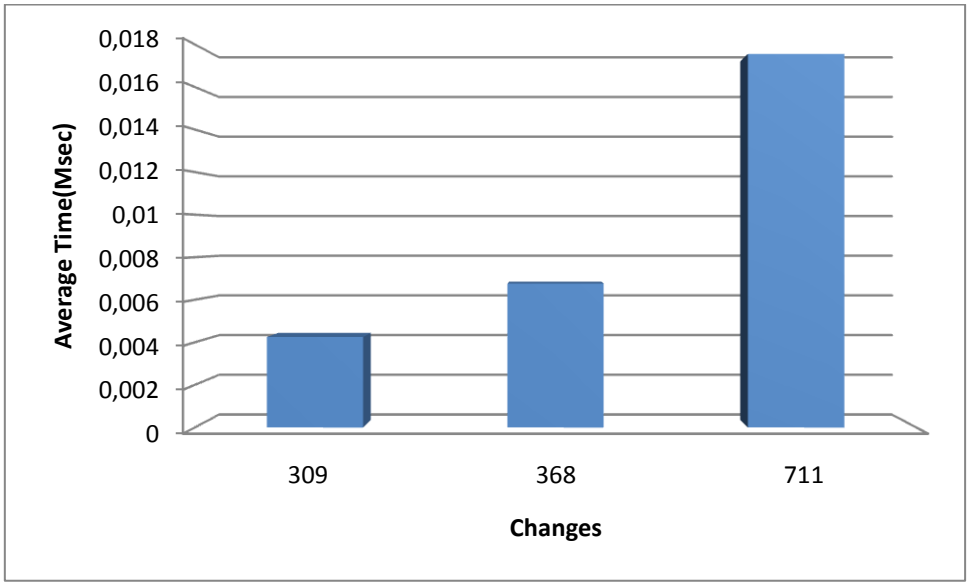


Fig. 60. The number of changes that should be processed and the average running time for CIDOC-CRM Ontology

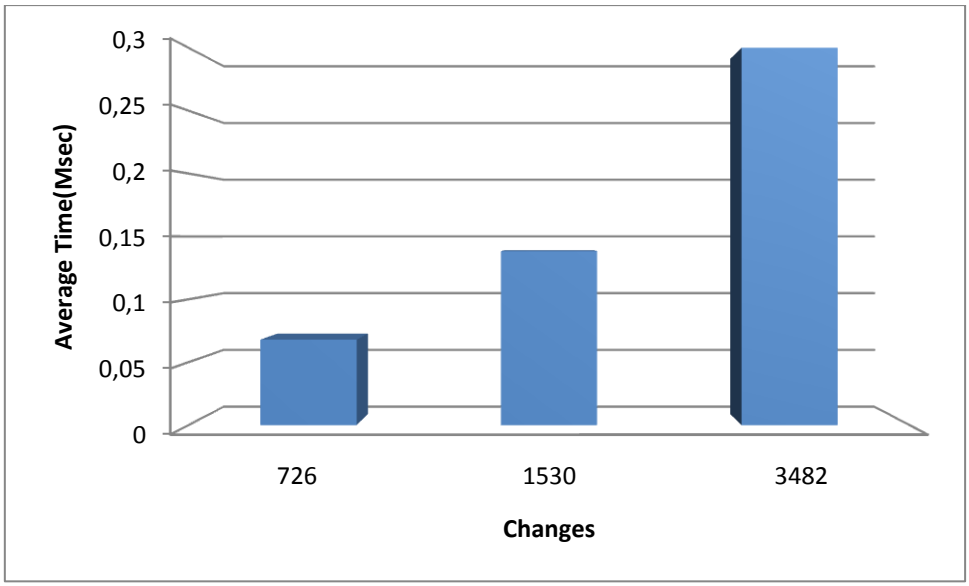


Fig. 61. The number of changes that should be processed and the average running time for Gene Ontology

6.2.2 Query Rewriting

To illustrate the scalability and the impact of our system we performed an extensive evaluation based on two scenarios. One scenario with synthetic queries and one scenario with real queries captured from related projects.

6.2.2.1 Synthetic Evaluation

In the synthetic scenario we automatically generated random queries using CIDOC-CRM v.4.2. We created 20 queries for each one of the following categories: queries with 1, 3, 7 and 20 triple patterns. The synthetic evaluation was performed only for queries formulated using CIDOC-CRM ontology since the queries used for the GO ontology ask for instances of only one GO-term (GO ontology is mostly a taxonomy) and rich queries including several properties cannot be produced.

Scalability

Since query rewriting is depending on the query size and the number of changes among ontology versions (assuming fixed number of ontological constraints which is usually the case in bibliography) at first we fix the query size and we present the time for query expansion and valid rewriting as the number of changes increases. The results are shown on Fig. 62 for queries with 1 triple pattern and in Fig. 63 for queries with 20 triple patterns.

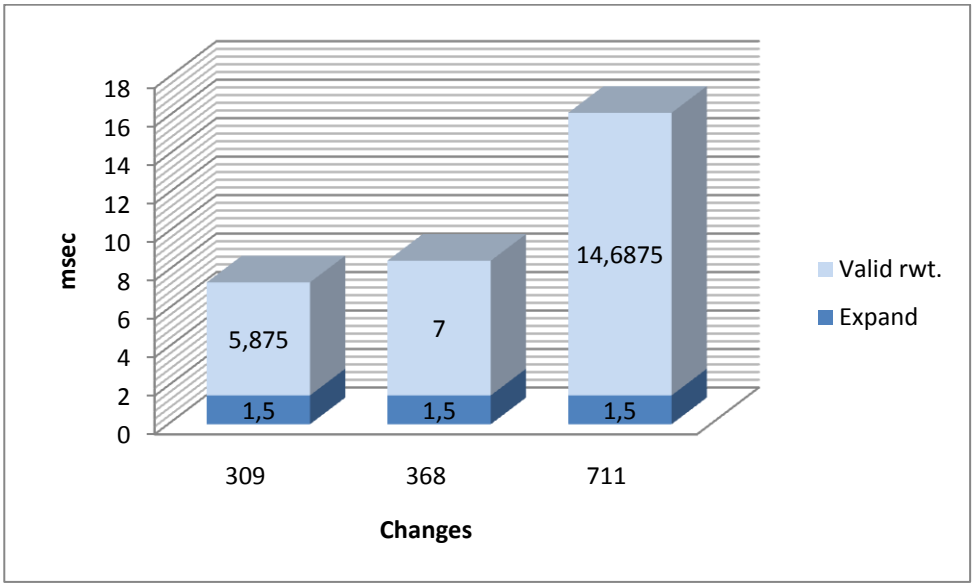


Fig. 62. Query rewriting for queries with 1 triple pattern

Obviously, in both cases, the total execution time increases as the number of change operations increase as well. This is not however, due to the expansion phase since the expansion time is only affected by the number of dependencies which is fixed. We can see that in both cases when the size of the change log doubles it happens the same for the time required for valid rewriting.

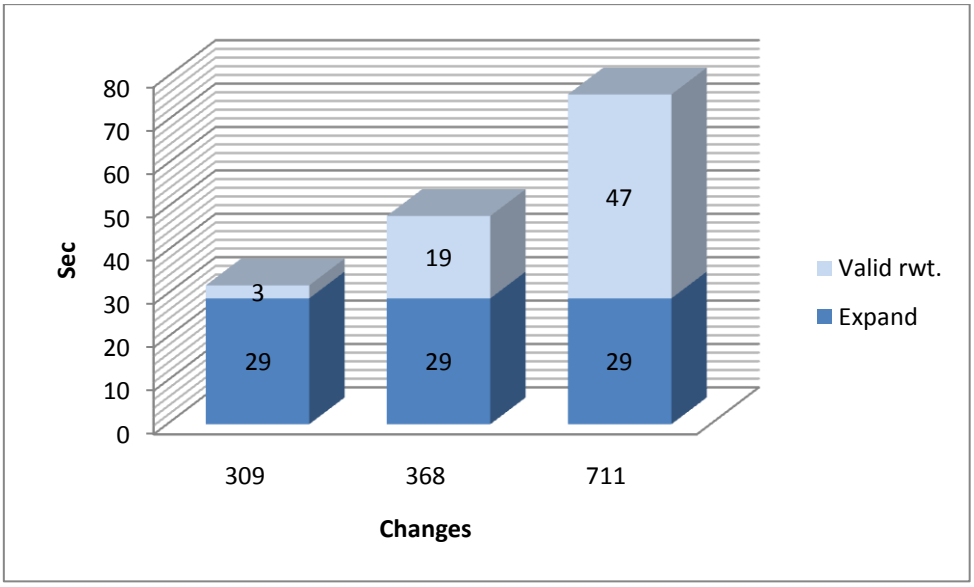


Fig. 63. Query rewriting for queries with 20 triple patterns

Then, we fix the number of change operations and we present the total execution time as the triple pattern's of the queries increase. The results are shown on Fig. 64 and Fig. 65. On Fig. 64 we have all 711 change operations whereas on Fig. 65 we only have 309 change operations. Obviously, the total execution time becomes bigger as the number of triple patterns in the queries increase. Moreover, the size of the query affects both the expansion phase and the valid rewriting phase as well.

Finally, we can identify that for smaller number of changes the dominant time is the time required for expansion whereas for more changes the time for valid rewriting increases as well. Those experiments confirm the theoretical expectations of our query rewriting algorithm.

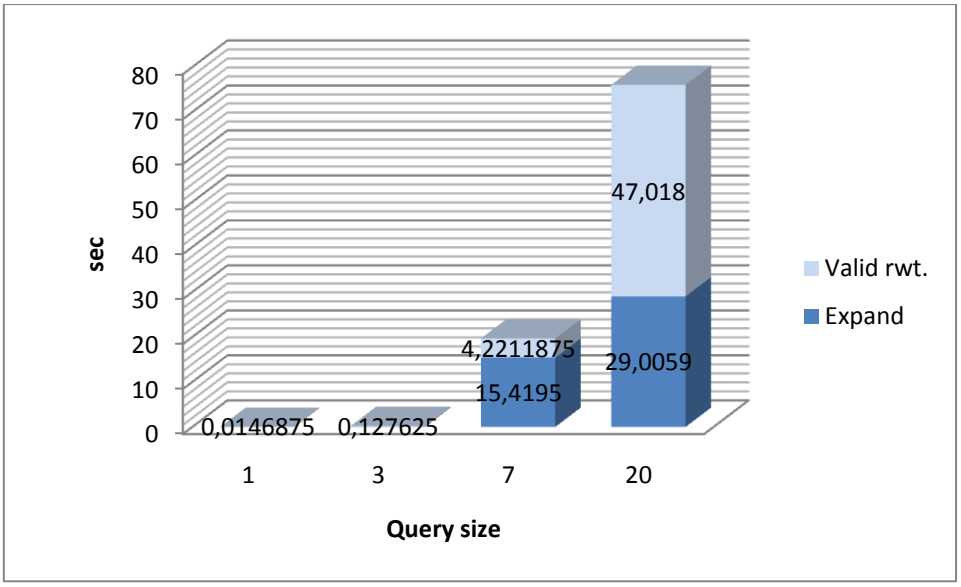


Fig. 64. Execution time as the triple patterns in the query increase (711 change operations)

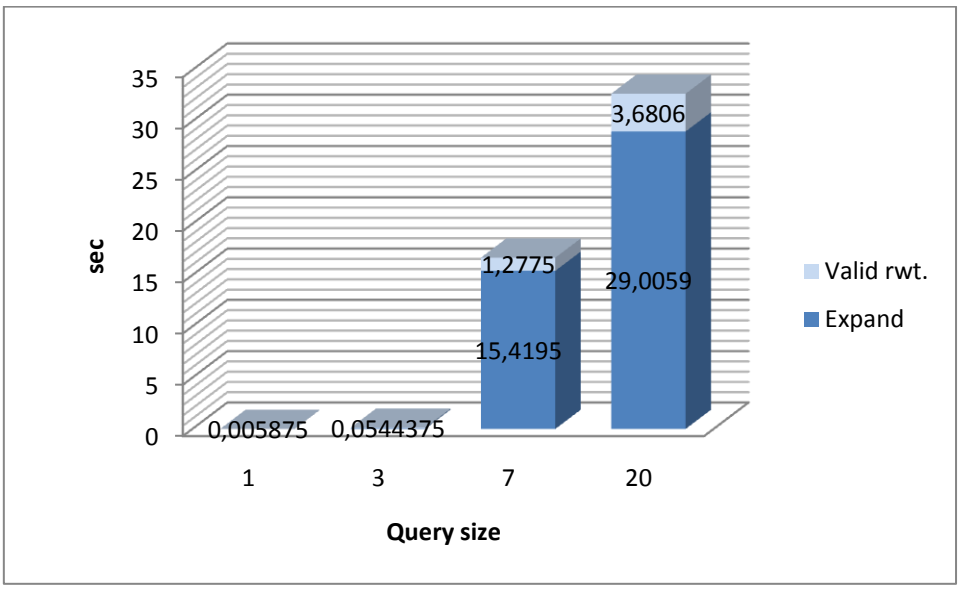


Fig. 65. Execution time as the triple patterns in the query increase (309 change operations)

Impact

In this subsection we will present experiments for identifying the impact of our approach. Initially, we fix the number of the triple patterns of the queries and we present the percentage of equivalent, minimally-containing and minimally-generalized rewritings that we were able to produce. Moreover, we present the percentage of the queries that could be answered "as-is" without any changes when they were issued to the data integration systems that used previous ontology versions.

The results are shown on Fig. 66 and Fig. 67. In each case the first bar shows the percentage of equivalent and the minimally-containing queries, the second bar the percentage of equivalent and the minimally-generalized queries, and the third bar the queries that could be answered "as-is". From the charts we can see that the number of queries that can be answered "as-is" from the previous ontology versions decreases as the number of change operations increases. Moreover, the bigger the size of the query, the bigger is the probability not to be able to answer the query "as-is" to a past ontology version.

For example, looking at Fig. 66, using queries with 20 triple patterns after 343 change operations the queries that could be answered "as-is" were only the 40% of the total queries, whereas we could produce the equivalent rewritings to the past ontology version for all of them. Moreover, we can see that as the number of change operations increase the number of equivalent rewritings drops and we have to go for minimally-containing or minimally-generalized rewritings. Even after 711 change operations, none of them could be answered "as-is" using the past ontology version. However, our system, even then, could produce the minimally-containing rewritings for the 40% of input queries.

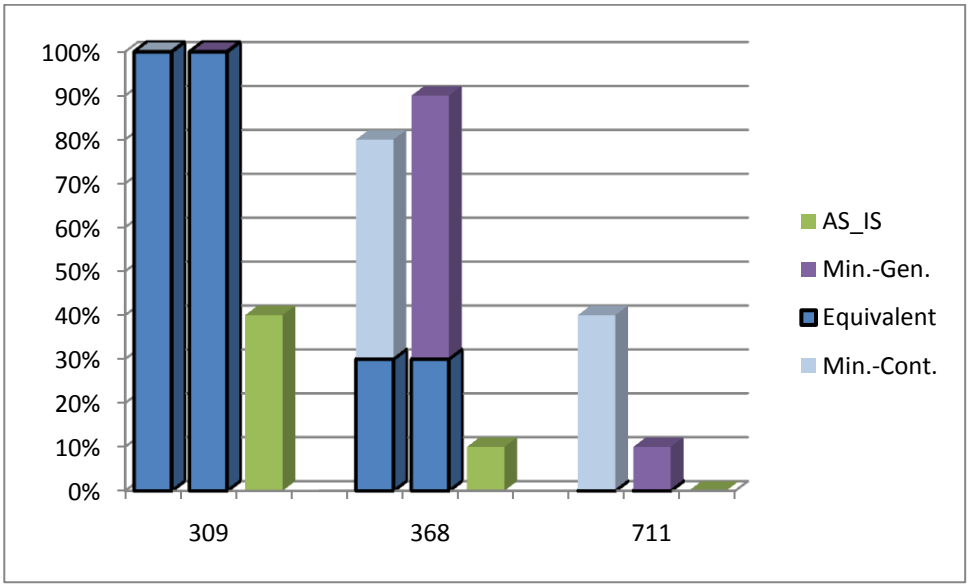


Fig. 66. Rewriting queries with 20 triple patterns

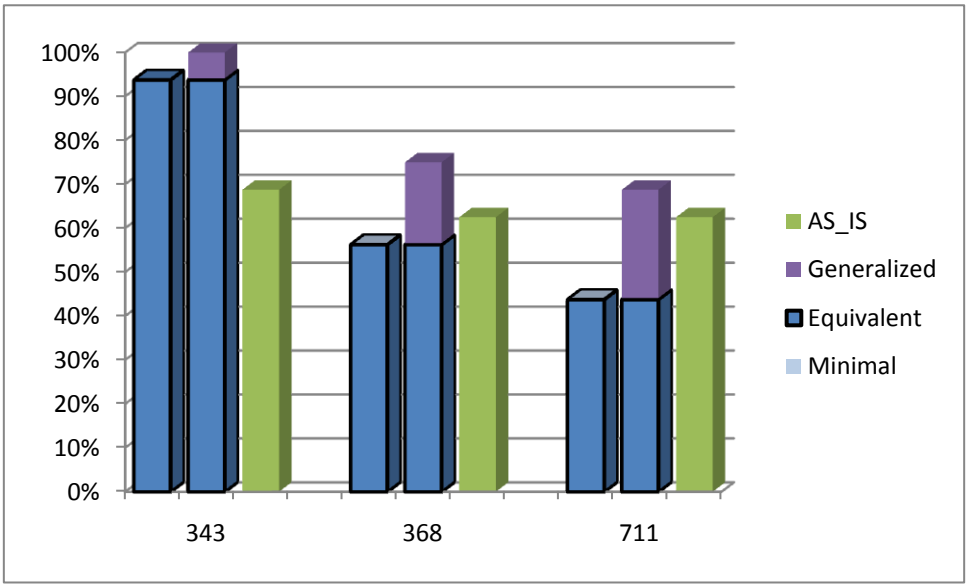


Fig. 67. Rewriting queries with 1 triple pattern

On the other hand, looking at Fig. 67, we can identify that we can produce more equivalent rewritings at all cases. However, we can notice that after 368 or 711 change operations the percentage of equivalent rewritings that we could produce was smaller than the number of queries that could be answered "as-is". This is because changes occurred on the subclasses that those queries asked for and our equivalent rewritings had to consider those changes as well (in contrast to "as-is" queries which ignore changes below the class hierarchy they query). Moreover, on all cases we could not produce a minimally-containing rewriting since when a class was deleted the resulted query was not safe anymore. However, on those cases we could produce minimally-generalized queries since in most of the cases the deleted classes had a parent class that we could query instead.

Then, we fixed the number of change operations and we present the results of query rewriting as the number of triple patterns increases. The results are shown on Fig. 68 and Fig. 69. We can observe that with 711 change operations we could not produce an equivalent rewriting and we had to go for minimally-containing rewritings. With 711 change operations the number of the queries that could be answered "as-is" decreased as well as the number of triple patterns in the queries increase.

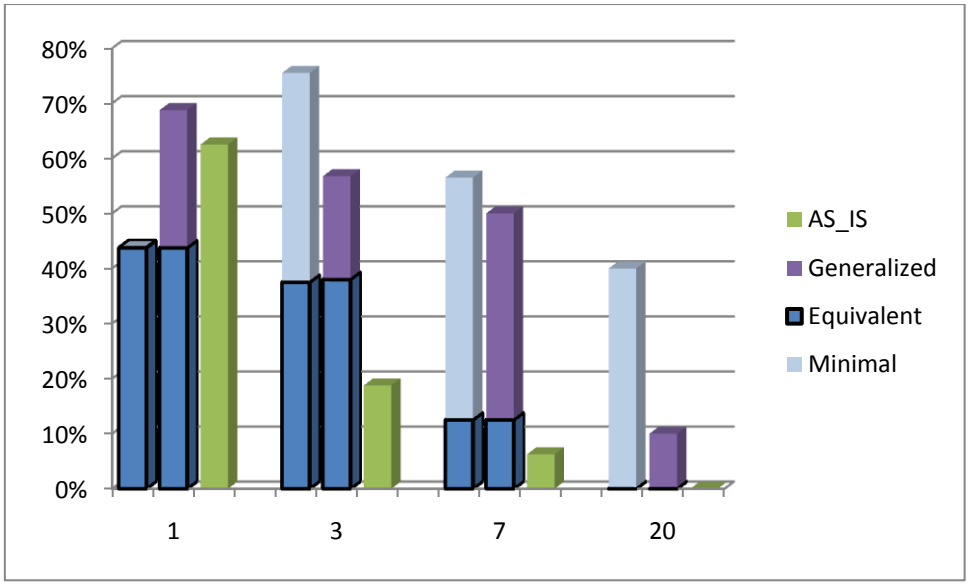


Fig. 68. Query rewriting using 711 change operations

Finally, with only 309 change operations we could get equivalent rewritings for most of queries whereas we could not get always answers to the queries "as-is". Moreover, on most of the cases with 309 change operations we could not produce generalized answers because properties or classes that were deleted they had no a superproperty or a superclass.

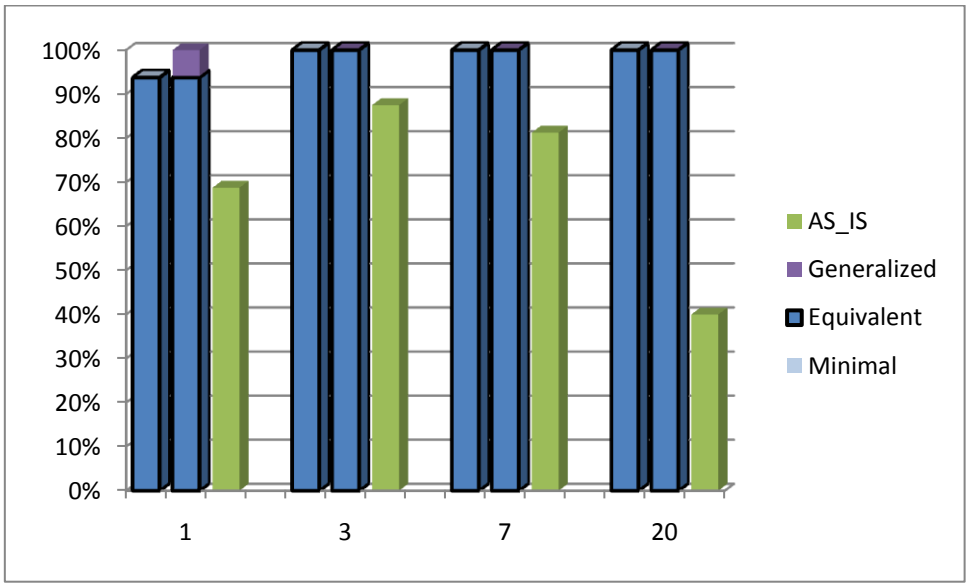


Fig. 69. Query rewriting using 309 change operations

6.2.2.2 Pragmatic Evaluation

To check the effectiveness of our system on real cases we used two sets of queries: 21 template queries for CIDOC-CRM coming from hundreds of user queries (9 query templates from (Theodoridou, 2010) and 12 query templates from project 3d-

COFORM[12]) and the 38 most popular queries as they have been identified and provided from the AmiGO[13] search engine.

Scalability

Firstly, we tried to identify the average execution time for rewriting the evaluation queries using CIDOC-CRM. The results are shown on Fig. 70. Obviously the average time to produce a rewriting even after 711 change operations is less than 5sec which shows the scalability of our approach. Notice that the time for performing query expansion is greater than the time to perform v also, and when the number of change operations doubles the same happens to the time for valid rewriting according to the complexity of our algorithm.

The results for queries formulated using GO are presented on Fig. 71. The average execution time for GO is 16sec and is justified from the large size of the ontology. Most of the time is spent calculating the expansion of the queries since our system has to consider the inclusion dependencies of 28000 classes and only 2,16sec is spent (at the worst case) for valid rewriting.

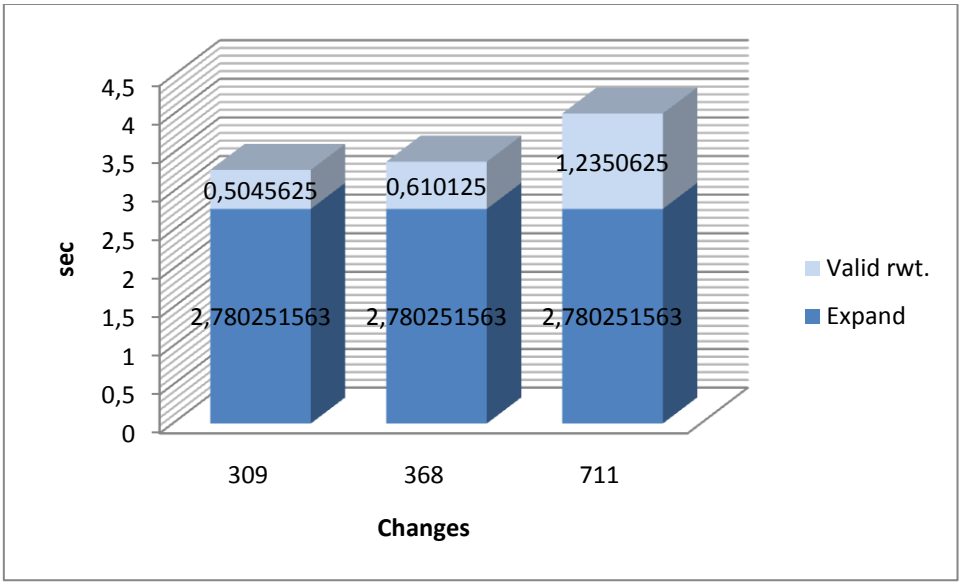


Fig. 70. Average Execution time for CIDOC-CRM queries

[12] http://www.3d-coform.eu/

[13] http://amigo.geneontology.org/cgi-bin/amigo/go.cgi

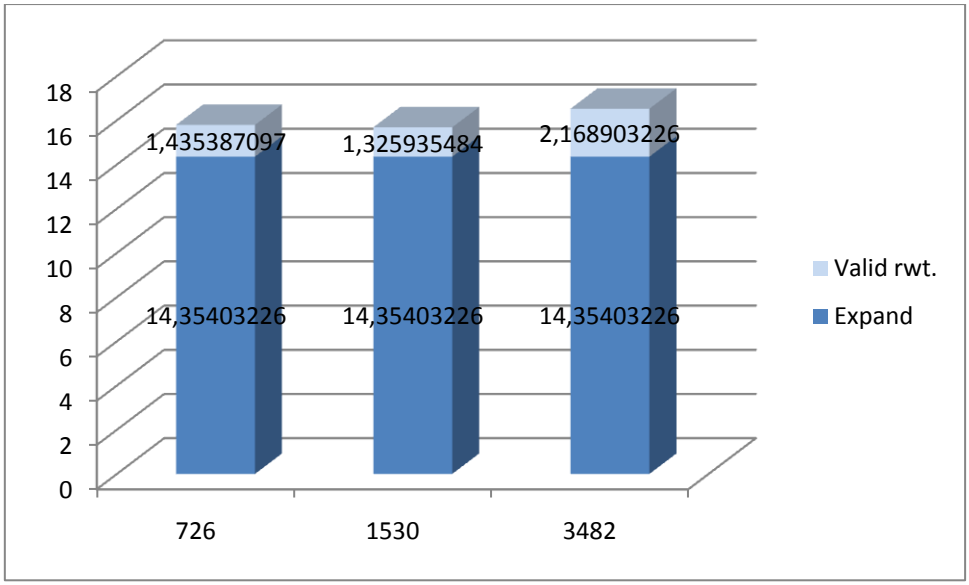


Fig. 71. Average Execution time for GO queries

Impact

To illustrate the impact of our approach we present the percentage of equivalent, minimally-containing and generalized queries that our system could produce for the two set of queries. The results are shown on Fig. 72 and Fig. 73.

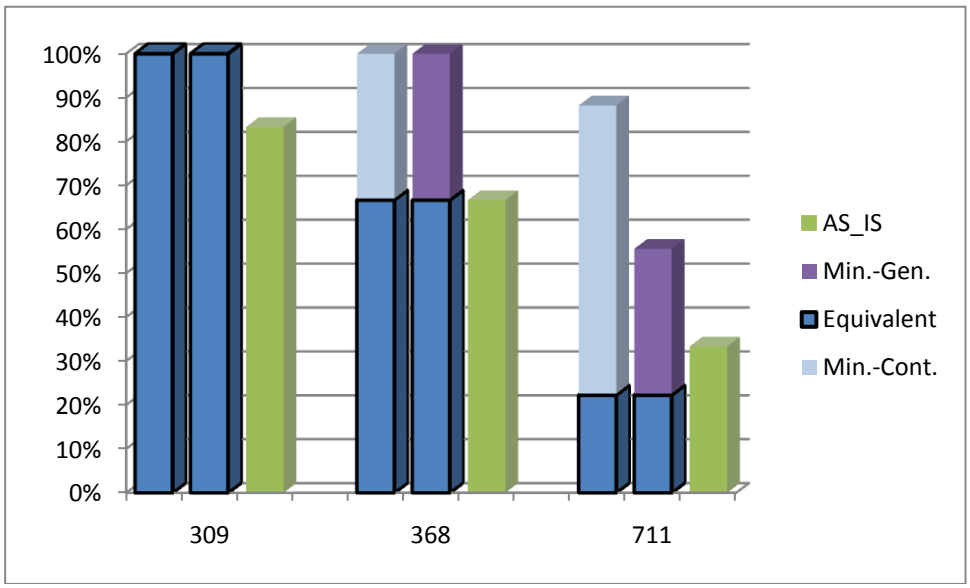


Fig. 72. Query rewriting for real CIDOC-CRM queries

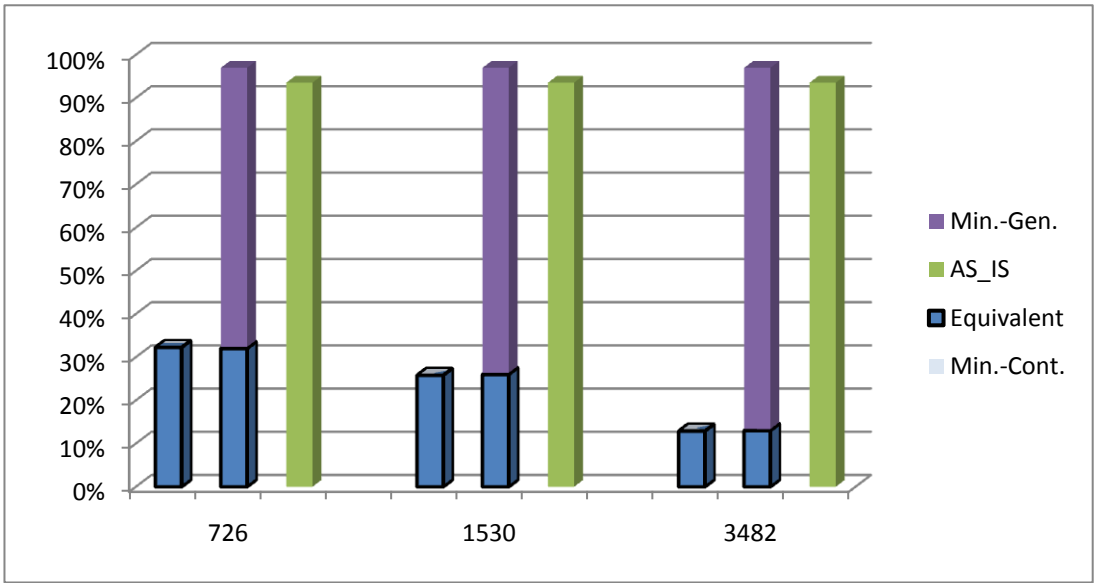


Fig. 73. Query rewriting for GO most popular queries

For CIDOC-CRM we observe that as the number of changes increase, the percentage of the queries that can be answered "as-is" drops to 33% whereas in GO as the number of changes increases the percentage of queries that can be answered as is remains the same 94%. This may seem peculiar because of the higher number of change operations that we have in the case of the Gene Ontology. However, if we carefully examine the corresponding change operations in each case we can easily identify that they change only a small percentage of the GO ontology (10% of the entire ontology was changed by the 3482 changes), whereas for the CIDOC-CRM the 711 change operations changed 54% of the entire ontology. This is shown on Fig. 74.

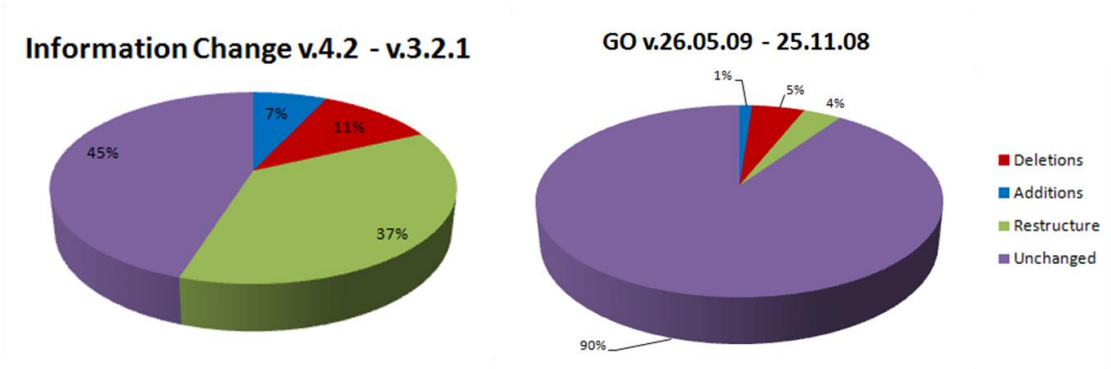


Fig. 74. The total information change for CIDOC-CRM and GO

Moreover, we can identify that the number of equivalent rewritings we can produce drops as the number of changes increases in both cases. However, in GO we can produce a smaller percentage of equivalent rewritings compared to the CIDOC-CRM ontology. This is due to the fact that the GO ontology usually evolves by adding GO terms (which are translated in delete change operations when trying to produce

rewritings to the previous ontology versions). And since the queries using GO ontology involve only one GO term, when this term is deleted we cannot produce minimally-containing rewritings since the query produced is not safe any more. However, in most of the cases the deleted term had a superclass that could be queried instead. That's why we could get minimally-generalized queries instead.

These two test cases show the flexibility of our solution in different kind of ontologies, and the great practical value of our approach.

Conclusions on Pragmatic Scenario and Further Analysis

For the queries using CIDOC-CRM we have to mention that they had in average 10 triple patterns, whereas the queries involved in average 3,7 different classes and 3,5 different properties.

Moreover, if we carefully try to identify the different types of the queries according to (Schmidt, 2008) where queries are distinguished as *Long Path Chain queries* (nodes linked to each other via a long path) and *Bushy patterns queries* (single nodes linked to several other nodes) we will notice that we can find equivalent rewritings for all *Bushy patterns queries*. So, our approach has better results for *bushy pattern queries* (which can be seen as star queries).

Moreover, another conclusion is that the higher the level in the hierarchy of the queries classes and properties, the more probable is not to be able to produce an equivalent rewriting, since the expansion of the query will use more terms of the ontology.

Chapter 7

Conclusions & Future Work

"Give me a place to stand on, and I will move the Earth"

- Archimedes 212BC

In this thesis we argue that ontology evolution is reality and data integration systems should be aware and ready to deal with that. To that direction, we presented a novel approach that allows query answering under evolving ontologies without mapping redefinition.

Our architecture is based on a module that can be placed on top of any traditional ontology-based data integration system, enabling ontology evolution. It does so by using high-level changes to model ontology evolution, which are then interpreted as GAV mappings. Those GAV mappings are then used in order to rewrite not the mappings but the query itself among ontology versions.

The process of query rewriting proceeds in two steps, namely *query expansion* and *valid rewriting*. *Query expansion* is used in order to consider constraints coming from the ontology and then *valid rewriting* is used in order to produce query rewritings among ontology versions using the GAV mappings produced form the high-level sequence of changes among ontology versions. The query rewriting approach we use is proved to be effective, scalable and efficient.

Even for the cases where no equivalent rewriting can be produced we offer three alternatives: a) we offer assistance to the users to redefine the affected queries, b) we provide minimally-containing and c) minimally-generalized rewritings for the cases

that equivalent rewritings cannot be produced that offer two best over-approximations.

The potential impact of our approach is witnessed by being able to successfully provide rewritings on the worst case for the 88% of the CIDOC-CRM queries (after 711 change operations) and for the 97% of the GO queries (after 3482 change operations) among ontology versions. On the other hand if our system were not used, only a small percentage of the initial queries would be successful. For most of the queries, query rewriting is achieved within 5sec using a simple workstation, which also shows the usability and the scalability of our approach.

The great benefit of our approach is that its simplicity, modularity and the short deployment time it requires. It is only a matter of providing a new ontology version to our system to be able to use it to formulate queries that will be answered by data integration systems independent of the ontology version used.

To the best of our knowledge, no other system today is capable of automatically answering queries over multiple ontology versions.

7.1 Future Work

As future work, several challenging issues need to be further investigated which we will describe in order of importance and difficulty.

At first, a really interesting topic would be to extend our approach to OWL ontologies or to consider that ontologies used as schema are not consistent. Both cases would require handling inconsistencies among ontology versions which complicates even more the problem. At first, another mechanism would be required to describe changes among ontologies. To that direction (Plessers, 2007) could be used as starting point, and techniques for repairing inconsistent databases should be also used (Afrati, 2009). However, it still remains a really interesting open problem.

Another direction would be to extend our approach to handle the full expressiveness of SPARQL language. Full-SPARQL queries, no longer correspond to union of conjunctive queries, and the traditional algorithms for expanding those queries (perfect reformulation and chase) cannot directly be applied. Query processing would require more sophisticated techniques, with bad complexity (Schmidt, 2010) and heuristic solutions would have to be adopted.

Another direction for future work would be also to apply our solution in traditional schema evolution, and to consider the evolution of data sources as well. Another language would be required in that case to describe changes among schemata and new algorithms for query rewriting would be required as well. This is mainly due to the richer constraints we can have in such a setting. The first step towards that direction can be found on (Curino, 2008) and (Fagin, 2011).

Finally, our system could be easily extended to become a fully fledged peer-to-peer system that is based on different versions of ontologies that are evolved independently. This could be achieved by integrating a layer that would handle peer discovery, registration and negotiation (probably a DHT).

It becomes obvious that ontology evolution in data integration is an important topic and several challenging issues remain to be investigated in near future.

Bibliography

[1] Abiteboul, S. Duschka, O.: Complexity of Answering Queries Using Materialized Views. *PODS*, 1998: 254-263.

[2] Afrati, F., Chandrachud, M.: Information about queries Obtained by a set of views, *TR-2005-14*.

[3] Afrati, F., Gergatsoulis, M., Kavalieros, T.: Answering queries using materialized views with disjunction. *ICDT*, 1999: 435–452.

[4] Afrati, F., Kolaitis, P.: Repair checking in inconsistent databases: algorithms and complexity. *ICDT*, 2009:31-41.

[5] Arenas, M., Perez, J., Riveros, C.: The Recovery of a Schema Mapping: Bringing Exhcanged Data Back, *PODS*, 2008:13-22

[6] Barbosa, D., Freire, J. and Mendelzon, A. O.: Designing information-preserving mapping schemes for XML. *VLDB*, 2005.

[7] Beeri, C., Levy, A., Rousset, M.: Rewriting Queries Using Views in Description Logics. *PODS*, 1997: 99-108.

[8] Ben Miled, Z., Li, N., and Bukhres, O.: BACIIS: Biological and Chemical Information Integration Systems. *Journal of Database Management*, 16(3), 2005: 73-85.

[9] Berners-Lee, T., Hendler, J. & Lassila, O.: The Semantic Web, *Scientific American*, 284 (5), 2001:34-43.

[10] Bernstein, P. A., Green, T. J., Melnik, S. and Nash, A.: Implementing mapping composition. *The VLDB Journal*, 17, 2 2008:333-353.

[11] Bizer, C.: D2R MAP - A Database to RDF Mapping Language. *WWW* (Posters) 2003.

[12] Bouquet, P., Giunchiglia, F., Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing Ontologies, *Journal of Semantics*, 2004:325-343.

[13] Bounif, H. Schema Repository for Database Schema Evolution. *DEXA*, 2006.

[14] Boyd, M., Lazanitis, C., Kittivoraviktul, S., Mc Brien, P., Rizopoulos, N.: An Overview of The AutoMed Repository. *Technical report, Department of Computing,* Imperial College, London SW7 2AZ, 2004.

[15] Brickley, D., Guha, R.: {RDF Vocabulary Description Language 1.0: RDF Schema}. *W3C Recommendation,* 2004.

[16] de Bruijn, J., Polleres, A.: Towards an ontology mapping specification language for the semantic web. *Technical Report DERI-2004-06-30,* DERI, 2004.

[17] de Bruijn, J., Martin-Recuerda, F., Manov, D., Ehrig, M.: D4.2.1: State of the Art Survey on Ontology Merging and Aligning, available on the Web, *SEKT project deliverable,* 2004b.

[18] Calì, A., Calvanese, D., Giacomo, G. D. and Lenzerini, M.: Data Integration under Integrity Constraints. *CAiSE,* 2006.

[19] Calı, A., Calvanese, D., Giacomo, G. Lenzerini, M.: On the expressive power of data integration systems. *ER,* 2002: 338-350.

[20] Cali, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data Integration under Integrity Constraints. *CAiSE,* 2002b.

[21] Cali, A., Gottlob, G. and Pieris, A. Advanced Processing for Ontological Queries. *PVLDB,* 3(1), 2010:554-565.

[22] Cali, A., Gottlob, G. and Lukasiewicz, T. Datalog+-: a unified approach to ontologies and integrity constraints. *ICDT,* 2009.

[23] Calı, A., Giacomo, G., Lenzerini, M.: Models of information integration: Turning local-as-view into global-as-view. *In Foundations of Models for Information Integration*. On-line proceedings, http://www.fmldo.org/FMII-2001, 2001.

[24] Cali, A., Lembo, D. and Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. *PODS (San Diego, California),* 2003.

[25] Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and Databases: The DL-Lite Approach. *Reasoning Web,* 2009:255-356.

[26] Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R., Ruzzi, M.: Data Integration through DL-Lite Ontologies. *SDKB,* 2008:26-47.

[27] Cali, A. and Martinenghi, D.: Querying the deep web. *EDBT (Lausanne, Switzerland),* 2010.

[28] Calvanese, D., Giacomo, G., Lenzerini, M.: On the Decidability of Query Containment under Constraints, *PODS,* 1998:149-158.

[29] Calvanese, D., Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Description Logic Framework for Information Integration. *KR*, 1998b:2-13

[30] Calvanese, D., Giacomo, G., Lenzerini, M., Vardi, M.: Containment of conjunctive regular path queries with inverse. *KR,* 2000e:176–185.

[31] Calvanese, D., Giacomo, G., Lenzerini, M., Vardi, M.: Query processing using views for regular path queries with inverse. *PODS,* 2000f:58–66.

[32] Chalupsky, H.: OntoMorph: A Translation System for Symbolic Knowledge, *KR*, 2000.

[33] Cohen, S.,Nutt, W., Serebrenik, A.: Rewriting aggregate queries using views. *PODS,* 1999:155–166.

[34] Curino, C. A., Moon, H. J., Ham, M. and Zaniolo, C.: The PRISM Workwench: Database Schema Evolution without Tears. *ICDE,* 2009.

[35] Deutsch, A., Ludascher, B., Nash, A.: Rewriting queries using views with access patterns under integrity constraints, *Theoretical Computer Science,* Vol 371(3), Database Theory, 1 March 2007:200-226.

[36] Deutsch, A., Popa, L. and Tannen, V.: Query reformulation with constraints. *SIGMOD Rec.,* 35, 1 , 2006:65-73.

[37] Doerr, M., Ore, C.-E., Stead, S.: The CIDOC conceptual reference model: a new standard for knowledge sharing. *Tutorials, posters, panels and industrial contributions at ER*, 2007:51-56.

[38] Duschka, O, Levy, A.: Recursive plans for information gathering. *IJCAI,* 1997b:778–784.

[39] Edelweiss, N. and Moreira, A. F.: Temporal and versioning model for schema evolution in object-oriented databases. *Data Knowl. Eng.,* 53, 2 2005:99-128.

[40] Euzenat, J., Le Bach, T., Barrasa, J., Bouquet, P., de Bo, J., Dieng, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckeschmidt, H., Shvaiko, P., Tessaris, S., van Acker, S., Zaihrayeu, I: D2.2.3: State of the Art on Ontology Alignment, *KWEB project deliverable,* 2004.

[41] Fagin, R., Kolaitis, P., Popa, L., Tan, W. C.: Schema mapping evolution through Composition and Inversion. *Schema Matching and Mapping*, Springer, 2011

[42] Fagin, R., Kolaitis, P., Popa, L., Tan, W. C.: Quasi-Inverses of Schema Mappings. *ACM Transactions on Database Systems (TODS)*, 33(2), 2008.

[43] Fagin, R., Kolaitis, P., Popa, L.: Data exchange: getting to the core. *ACM Trans. Database Syst. 30(1)*, 2005a:174-210.

[44] Fagin, R., Kolaitis, P., Popa, L., Chiew Tan, W.: Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst*, 30(4), 2005b: 994-1055.

[45] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: Classification and survey. *Knowl. Eng. Rev*. 23, 2008:117-152.

[46] Flouris, G., Plexousakis, D.: Handling Ontology Change: Survey and Proposal for a Future Research Direction, *Technical Report FORTH-ICS/TR-362*, 2005.

[47] Flouris, G., Plexousakis, D.: Bridging Ontology Evolution and Belief Change. *SETN*, 2006.

[48] Friedman, M., Levy, A., Millstein, T.: Navigational Plans for Data Integration. *AAAI/IAAI 1999:67-73*, 1999:67-73.

[49] Fuxman, A. Kolaitis, P., Miller, R., Chiew Tan, W.: Peer data exchange. *ACM Trans. Database Syst.* 31(4), 2006:1454-1498.

[50] Gardenfors, P.: Belief Revision: An Introduction, *Cambridge University Press*. 1992a:1-20.

[51] Gardenfors, P.: The Dynamics of Belief Systems: Foundations Versus Coherence Theories. Revue Internationale de Philosophie, 44, 1992b:24-46.

[52] Gene Ontology Consortium: The Gene Ontology (GO) database and informatics resource. *Nucl. Acids Res*. 32 (2004) D258-261.

[53] Goasdoui, F., Lattes, V., and Rousset, M. C.: The use of CARIN language and algorithms for information integration: the PICSEL project. *International Journal of Cooperative Information Systems.* 9(4), 2000:383-401.

[54] Grahne, G., Mendelzon, A., O.: Tableau Techniques for Querying Information Sources through Global Schemas. *ICDT*, 1999: 332-347.

[55] Spezzano, F., Greco, S.: Chase Termination: A Constraints Rewriting Approach. *PVLDB*, 3(1), 2010:93-104.

[56] Gruber, T.R..: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5 (2), 1993a:199-220.

[57] Gruber, T.R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *Formal Ontology in Conceptual Analysis and Knowledge Representation, also available as Technical Report KSL-93-04*, Knowledge Systems Laboratory, Stanford University, 1993b.

[58] Grumbach, S., Rafanelli, M., Tininini, L.: Querying aggregate data. *PODS*, 1999:174–184.

[59] Guarino, N.: Formal Ontology and Information Systems, *FOIS*, 1998:3-15.

[60] Gupta, A., Jagadish, H. V. and Mumick, I. S. Data Integration using Self-Maintainable Views. *EDBT*, 1996.

[61] Halevy, A.: Answering queries using views: A survey. *VLDB J.* 10(4), 2001:270-294.

[62] Halevy, A., Ives, Z., Suciu, D. Tatarinov, I.: Schema Mediation in Peer Data Management Systems. *ICDE*, 2003: 505.

[63] Haase, P., Stojanovic, L.: Consistent Evolution of OWL Ontologies. *ESWC*, 2005.

[64] Hartung, M., Kirsten, T., Rahm, E.: Analyzing the Evolution of Life Science Ontologies and Mappings. *DILS*, 2008:11-27.

[65] Heflin, J., Hendler, J., Luke, S.: Coping with Changing Ontologies in a Distributed Environment. *AAAI, WS-99-13*, AAAI Press, 1999:74-79.

[66] Heflin, J. & Pan, Z.: A Model Theoretic Semantics for Ontology Versioning, *ISWC*, LNCS 3298 Springer, 2004:62-76.

[67] Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Feier, C., Hench, G., Wetzstein, B., Keller, U.: Ontology Reasoning with Large Data Repositories. *Ontology Management*. 2008:89-128.

[68] Huang, Z., Stuckenschmidt, H.: Reasoning with Multi-version Ontologies: A Temporal Logic Approach, *ISWC*, 2005:398-412.

[69] Kalfoglou, Y., Schorlemmer, M.:Ontology Mapping: the State of the Art, *Knowledge Engineering Review*, 18 (1), 2003:1-31.

[70] Katsuno, H., Mendelzon, A. O.: On the Difference Between Updating a Knowledge Base and Revising It. *Technical Report on Knowledge Representation and Reasoning*, University of Toronto, Canada, KRR-TR-90-6, 1990.

[71] Klein, M.: Combining and relating ontologies: an analysis of problems and solutions. Workshop on Ontologies and Information Sharing, *IJCAI*, 2001.

[72] Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D.: Ontology Versioning and Change Detection on the Web, *EKAW*, 2002.

[73] Klein, M., Fensel, D.: Ontology Versioning on the SemanticWeb, *SWWS*, 2001:75-91.

[74] Klein, M., Noy, N.: A Component-Based Framework for Ontology Evolution, *IJCAI Workshop on Ontologies and Distributed Systems*, CEUR-WS, vol. 71., 2003.

[75] Koffina, I., Serfiotis, G., Christophides, V., Tannen, V.: Mediating RDF/S Queries to Relational and XML Sources, *International Journal on semantic Web & Information System*, 2(4), 1006:68-91.

[76] Kohler, J., Philippi, S., and Lange, M.: SEMEDA: ontology based semantic integration of biological databases. *Bioinformatics* 19(18), 2003:2420–2427.

[77] Kolaitis, P.: Schema mappings, data exchange, and metadata management. *PODS*, 2005: 61-75.

[78] Kondylakis, H., Flouris, G., Plexousakis, D.: Ontology and Schema Evolution in Data Integration: Review and Assessment, *OTM Conferences (CoopIS/DOA/IS/ODBASE)*, 2009.

[79] Kondylakis, H., Plexousakis, D.: Enabling ontology evolution in data integration, *EDBT/ICDT Workshops*, 2010a.

[80] Kondylakis, H., Plexousakis D., Tzitzikas, Y.: Ontology Evolution in Data Integration, *In Poster session of ESWC*, 2010b.

[81] Kondylakis, H., Plexousakis D., Tzitzikas, Y.: Ontology Evolution in Data Integration, *HDMS*, 2010c.

[82] Kondylakis, H., Plexousakis D., Tzitzikas, Y.: Enabling Ontology Evolution in Data Integration, *IWOD*, 2010d.

[83] Kondylakis, H., Plexousakis D., Tzitzikas, Y.: Enabling Ontology Evolution in Data Integration, *ICDE*, 2011a (submitted).

[84] Kondylakis, H., Plexousakis D.: Ontology Evolution without Tears, *EDBT/ICDT, 2011b* (submitted).

[85] Kondylakis, H., Plexousakis D.: Exelixis: An Evolving Ontology-based Data Integration System, *Demo Session of SIGMOD*, 2011(to be submitted).

[86] Konstantinidis, G., Flouris, G., Antoniou, G. and Christophides, V. Ontology Evolution: A Framework and its Application to RDF. *ODBIS & SWDB Workshop on Semantic Web, Ontologies, Databases (SWDB-ODBIS-07)*, 2007.

[87] Lambrix, P. & Edberg, A.: Evaluation of Ontology Merging Tools in Bioinformatics, *Pacific Symposium on Biocomputing*, 2003:589-600.

[88] Lee, A. J., Nica, A. and Rundensteiner, E. A. The EVE Approach: View Synchronization in Dynamic Distributed Environments. *IEEE Trans. on Knowl. and Data Eng.*, 14, 5, 2002:931-954.

[89] Lembo, D., Lenzerini, M., Rosati, R.: Source inconsistency and incompleteness in data integration. *KRDB*, 2002.

[90] Levy, A., Mendelzon, A., Sagiv, Y., Srivastava, D.: Answering Queries Using Views. *PODS*, 1995: 95-104.

[91] Levy, A, Rajaraman, A., Ordille, J.: Querying Heterogeneous Information Sources Using Source Descriptions. *VLDB* 1996: 251-262.

[92] Lenzerini M.: Data Integration: A Theoretical Perspective. *PODS*, Madison, Wisconsin, USA, 3-6 June, 2002.

[93] Li, C., Bawa, M., Ullman, J.: Minimizing view sets without loosing query-answering power. *ICDT*,, 2001:99–103.

[94] Lloyd, J. W.: Foundations of logic programming; (2nd extended ed.). Springer-Verlag New York, Inc., 1987.

[95] Martin, L., Anguita, A., Maojo, V., Bonsma, E., Bucur, A.I.D., Vrijnsen, J., Brochhausen, M., Cocos, C., Stenzhorn, H., Tsiknakis, M., Doerr, M., Kondylakis, H.: Ontology Based Integration of Distributed and Heterogeneous Data Sources in ACGT. *HEALTHINF*, Funchal, Madeira, Portugal, 2008:301-306.

[96] Madhavan, J.,Halevy, A.: Composing Mappings Among Data Sources. *VLDB* 2003: 572-583.

[97] Magiridou, M., Sahtouris, S., Christophides, V. and Koubarakis, M. RUL: A Declarative Update Language for RDF. *ISWC*, 2005.

[98] McGuiness, D., Fikes, R., Rice, J. & Wilder, S.:An Environment for Merging and Testing Large Ontologies, *KR, also available as Technical Report KSL-00-16,* Knowledge Systems Laboratory, Stanford University, 2000.

[99] Millstein, T., Levy, A.,Friedman, M.: Query containment for data integration systems. *PODS,* 2000:67–75.

[100] Mohania, M. and Dong, G.: Algorithms for Adapting Materialised Views in Data Warehouses. *CODAS*, 1996.

[101] Moon, H. J., Curino, C. A. and Zaniolo, C.: Scalable architecture and query optimization for transaction-time DBs with evolving schemas. *SIGMOD*, Indianapolis, Indiana, USA, 2010.

[102] Moro, M. M., Malaika, S. and Lim, L.: Preserving XML queries during schema evolution. *WWW*, Banff, Alberta, Canada, 2007.

[103] Nash, A., Bernstein, P. A. and Melnik, S.: Composition of mappings given by embedded dependencies. *ACM Trans. Database Syst*., 32, 1, 2007, 4.

[104] Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A Framework for Ontology Evolution in Collaborative Environments, *ISWC,* 2006:544-558.

[105] Noy, N., Klein, M.: Ontology Evolution: Not the Same as Schema Evolution. *KAIS, 6(4), Available as SMI technical report SMI-2002-0926*, 2004:428-440.

[106] Noy, N. & Musen, M.: An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support, *Workshop on Ontology Management at AAAI, also available as SMI technical report SMI-1999-0799*, 1999a.

[107] Noy, N. & Musen, M.. : SMART: Automated Support for Ontology Merging and Alignment, *Workshop on Knowledge Acquisition, Modelling and Management, also available as SMI technical report SMI-1999-0813*, 1999b.

[108] Noy, N. & Musen, M.: Algorithm and Tool for Automated Ontology Merging and Alignment, *AAAI*, also available as SMI technical report SMI-2000-0831. 2000.

[109] Noy, N.F., Musen, M. A.: Promptdiff: a fixed-point algorithm for comparing ontology versions. *AAAI*, Edmonton, Alberta, Canada, 2002.

[110] Oliver, D.E., Shahar, Y., Shortliffe, E.H., Musen, M. A.: Representation of change in controlled medical terminologies. *Artificial Intelligence in Medicine,* 15, 1999:53-76.

[111] Ognyanov, D., Kiryakov, A.: Tracking Changes in RDF(S) Repositories. *EKAW*, 2002:373-378.

[112] Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V.: On Detecting High-Level Changes in RDF/S KBs. *ISWC*, 2009:473 – 488.

[113] Papavassiliou, V. : Detecting Deterministically High-level Changes for RDF/S Knowledge Bases, *Master's Thesis*, University of Crete, Computer Science Department, 2010.

[114] Perez-Rey, D., Maojo, V., Garcia-Remesal, M., Alonso-Calvo, R., Billhardt, H., Martin-Sanchez, F., and Sousa, A.: ONTOFUSION: Ontology-based integration of genomic and clinical databases. *Computers in Biology and Medicine*, 36(7-8), pp. 712-730, 2006.

[115] Perez, J., Arenas, M. and Gutierrez, C. Semantics and complexity of SPARQL. *ACM Trans. Database Syst*., 34, 3 2009:1-45.

[116] Pinto, S. H., Gomez-Perez, A. and Martins, J. P. Some Issues on Ontology Integration. *IJCAI Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends.* Stockholm, Sweden, 1999.

[117] Plessers, P., Troyer, O.D.: Ontology Change Detection Using a Version Log. *ISWC*, 2005:578-592.

[118] Plessers, P., Troyer, O. D. and Casteleyn, S.: Understanding ontology evolution: A change detection approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5, 1 2007:39-49.

[119] Poggi, A., Lembo, D., Calvanese, D., Giacomo, G. D., Lenzerini, M. and Rosati, R. Linking data to ontologies. *Journal on data semantics X*, 2008:133-173.

[120] Pottinger, R., Halevy, A.: MiniCon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10, 2001:182-198.

[121] Prud'hommeaux, E., Seaborne, A.: (2008, SPARQL Query Language for RDF. Available: http://www.w3.org/TR/rdf-sparql-query/.

[122] Ra, Y.-G., Rundensteiner, E. A.: A Transparent Schema-Evolution System Based on Object-Oriented View Technology. *Trans. on Knowl. and Data Eng.*, 9, 4 1997:600-624.

[123] Rizzi, S. and Golfarelli, M.: X-Time: Schema Versioning and Cross-Version Querying in Data Warehouses. *ICDE*, 2007:15-20.

[124] Rogozan, D., Paquette, G.: Managing Ontology Changes on the Semantic Web. *Web Intelligence*. IEEE Computer Society, 2005:430-433.

[125] Russell, A. and Smart, P.: NITELIGHT: A Graphical Editor for SPARQL Queries. *ISWC*, 2008.

[126] Russel, S., Norvig, P.: Artificial Intelligence a modern approach. Prentice Hall, 2003.

[127] Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. *ICDE*, 2009:222-233.

[128] Serfiotis, G., Koffina, I., Christophides, V., Tannen, V.: Containment and Minimization of RDF/S Query Patterns. *ISWC*, 2005:607-623.

[129] Stevens, R., Baker, P., G., Bechhofer, S., Ng, G., Jacoby, A., Paton, N., W., Goble, C., A., Brass, A.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics*, 16(2), 2000:184-186.

[130] Stojanovic, L., Maedche, A., Motik, B. and Stojanovic, N.: User-Driven Ontology Evolution Management, *EKAW*, 2002.

[131] Stojanovic, L.: Methods and Tools for Ontology Evolution. Vol. Phd. Univ. of Karlsruhe, 2004.

[132] Stojanovic, L., Maedche, A., Stojanovic, N. and Studer, R.: Ontology evolution as reconfiguration-design problem solving. *K-CAP*, Sanibel Island, FL, USA, 2003.

[133] Theodoridou, M., Tzitzikas, Y., Doerr, M., Marketakis, Y. and Melessanakis, V.: Modeling and querying provenance by extending CIDOC CRM. *Distrib. Parallel Databases*, 27, 2 2010:169-210.

[134] Theoharis, Y., Christophides, V. and Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. *ISWC*, 2005.

[135] Theoharis, Y.: On Graph Features of Semantic Web Schemas. *IEEE Transactions on Knowledge and Data Engineering*, 2007:692-702.

[136] Ullman, J.: Information Integration Using Logical Views. *Theoretical Computer Science*, 239(2), 2000:189-210.

[137] Velegrakis, Y., Miller, J., Popa, L.: Preserving mapping consistency under schema changes. *The VLDB Journal*, 13, 2004:274-293.

[138] Velegrakis, Y., Miller, R. J. and Mylopoulos, J. Representing and Querying Data Transformations. *ICDE*, 2005.

[139] Volkel, M., Winkler, W., Sure, Y., Kruk, S.R., Synak, M.: Semversion: A versioning system for rdf and ontologies. *ESWC*, 2005.

[140] Volz, R., Oberle, D., Staab, S., Motik, B.: KAON SERVER - A Semantic Web Management System. *WWW* (Alternate Paper Tracks), 2003.

[141] Wache, H., Scholz, T., Stieghanh, H., and Konig-Ries, B.: An integration method for the specification of rule-oriented mediators. *DANTE*, 1999:109-112.

[142] Xuan, D. N., Bellatreche, L. and Pierra, G.: A Versioning Management Model for Ontology-Based Data Warehouses. *DaWaK*, Krakow, Poland, 2006.

[143] Yu, C. and Popa, L.: Semantic adaptation of schema mappings when schemas evolve. *VLDB*, 2005.

[144] Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. *ICDT*, 2010:4-33.

[145] Stamatis Zampetakis, Yannis Tzitzikas, Asterios Leonidis, Dimitris Kotzinos StarLion: Auto-Configurable Layouts for Exploring Ontologies, *ESWC (Demo Track)*, Heraklion, Greece, June 2010.

[146] Zablith, F.,Antoniou, G., d'Aquin, M., Aussenac-Gilles, N., Flouris, G., Kondylakis, H., Laublet, P., Motta, E., Pan, J., Plexousakis, D., Sabou, M.: Ontology Evolution: A process Centric Survey, *Semantic Web Journal*, IOS Press (to be submitted).

[147] Zeginis, D., Tzitzikas, Y., Christophides, V.: On the Foundations of Computing Deltas Between RDF Models. *ISWC/ASWC*, 2007:637-651.

Appendix

A. Change Operations

The language of changes we consider, the inverse of each change operation and the corresponding GAV mapping are presented bellow.

Basic changes

The basic changes we adopt here are extensively presented and defined in (Papavassiliou, 2010) and consider the individual addition and the deletion of classes, properties, metaproperties, metaclasses, individuals. For those change operations cannot be produced GAV mappings.

Composite changes

We have to note that the change operations reclassifying classes and properties do not have a GAV mapping since the reclassifications are handled on the ontology expansion level.

Change	***Add_Class(a,P1,P2,P3,P4,P5,P6)***	***Delete_Class(a,P1,P2,P3,P4,P5,P6)***
Intuition	Add class *a* with its neighborhood links	Delete class *a* with its neighborhood links
Arguments	*P1* = set of new parent classes of *a*, *P2* = set of classes that have as parent *a*, *P3* = set of new metaclasses of *a*, *P4* = set of new individuals that are type of *a*, *P5* = set of new comments of *a*, *P6* = set of new labels of *a*	*P1* = set of old parent classes of *a*, *P2* = set of classes that had as parent *a*, *P3* = set of old metaclasses of *a*, *P4* = set of individuals that were type of *a*, *P5* = set of old comments of *a*, *P6* = set of old labels of *a*
δ_a	$\forall p \in P1 : (a, subClassOf , p)$, $\forall p \in P2 : (p, subClassOf , a)$, $\forall p \in P3 : (a, type, p)$, $\forall p \in P4 : (a, type, p)$, $\forall p \in P5 : (a, comment, p)$, $\forall p \in P6 : (a, label, p)$, *(a, type, class),* *(a, subClassOf , resource)*	ø
δ_d	ø	$\forall p \in P1 : (a, subClassOf , p)$, $\forall p \in P2 : (p, subClassOf , a)$, $\forall p \in P3 : (a, type, p)$, $\forall p \in P4 : (a, type, p)$, $\forall p \in P5 : (a, comment, p)$, $\forall p \in P6 : (a, label, p)$, *(a, type, class),* *(a, subClassOf , resource)*
Inverse	*Delete_Class(a,P1,P2,P3,P4,P5,P6)*	*Add_Class(a,P1,P2,P3,P4,P5,P6)*
GAV Mappings	-	-

The changes *Add_Metaclass* and *Add_Metaproperty* are defined similarly with the exception of (*a*, *subClassOf* , *class*) and (*a*, *subClassOf* , *property*) being in δa respectively instead of (*a*, *subClassOf* , *resource*). The changes Delete_Metaclass and *Delete_Metaproperty* are defined similarly with the exception of (*a*, *subClassOf* , *class*) and (*a*, *subClassOf* , *property*) being in δ_d respectively instead of (*a*, *subClassOf* , *resource*).

Change	***Group_Classes(A,b)***	***Ungroup_Classes(A,b)***
Intuition	Group classes in *A* under *b*	Ungroup classes in *A*
Arguments	*A* = set of classes that have as new parent *b*, *b* = new parent class *b*	*A* = set of classes that had as parent *b*, *b* = old parent class *b*
δ_a	$\forall a \in A : (a, subClassOf , b)$	ø
δ_d	ø	$\forall a \in A$: *Delete_Superclass*(*a*,*b*)
Inverse	*Ungroup_Classes*(*A*,*b*)	*Group_Classes*(*A*,*b*)
GAV Mappings	-	-

The changes Group_*Metalasses* and *Group_Metaproperties* are defined similarly. We have to note that the change operations reclassifying classes and properties do not have a GAV mapping since the reclassifications are handled on the ontology expansion level.

Change	***Pull_up_Class(a,B,C)***	***Pull_down_Class(a,B,C)***
Intuition	Move class *a* to a higher position in the subsumption hierarchy	Move *a* class to a lower position in the subsumption hierarchy
Arguments	*B* = set of old parents of *a*, *C* = set of new parents of *a*	*B* = set of old parents of *a*, *C* = set of new parents of *a*
δ_a	$\forall c_i \in C : (a, subClassOf , c_i) (1 \leq i \leq n)$	$\forall c_i \in C : (a, subClassOf , c_i) (1 \leq i \leq n)$
δ_d	$\forall b_i \in B : (a, subClassOf , b_i) (1 \leq i \leq n)$	$\forall b_i \in B : (a, subClassOf , b_i) (1 \leq i \leq n)$
Inverse	*Pull_down_Class(a,C,B)*	*Pull_up_Class(a,C,B)*
GAV Mappings	-	-

The changes *Pull_up_Metaclass* and *Pull_up_Metaproperty* are defined similarly and the changes *Pull_down_Metaclass* and *Pull_down_Metaproperty* are defined similarly as well.

Change	***Move_Class(a,B,C)***	***Change_Superclasses(a,B,C)***
Intuition	Move *a* class to a different subsumption hierarchy	Change the parents of class *a*
Arguments	*B* = set of old parents of *a*, *C* = set of new parents of *a*	*B* = set of old parents of *a*, *C* = set of new parents of *a*
δ_a	$\forall c_i \in C : (a, subClassOf , c_i) (1 \leq i \leq n)$	$\forall c_i \in C : (a, subClassOf , c_i) (1 \leq i \leq n)$
δ_d	$\forall b_i \in B : (a, subClassOf , b_i) (1 \leq i \leq n)$	$\forall b_i \in B : (a, subClassOf , b_i) (1 \leq i \leq n)$
Inverse	*Move_Class(a,C,B)*	*Change_Superclasses(a,C,B)*
GAV Mappings	–	–

Change	**Reclassify_Class_Higher(a,B,C)**	**Reclassify_Class_lower(a,B,C)**
Intuition	Move *a* class to a higher position in the subsumption hierarchy	Reclassify *a* class to a lower position in the subsumption hierarchy
Arguments	*B* = set of old types of *a*, *C* = set of new types of *a*	*B* = set of old types of *a*, *C* = set of new types of *a*
δ_a	$\forall c \in C : (a, type, c)$	$\forall c \in C : (a, type, c)$
δ_d	$\forall b \in B : (a, type, b)$	$\forall b \in B : (a, type, b)$
Inverse	Reclassify_Class_lower(a,C,B)	Reclassify_Class_higher(a,C,B)
GAV Mappings	–	–

The changes *Reclassify_Metaclass_Higher* and *Reclassify_Metaproperty_Higher* are defined similarly. The changes *Reclassify_Metaclass_Lower* and *Reclassify_Metaproperty_Lower* are defined similarly as well.

Change	***Add_Property(a,P1,P2,P3,P4,p5,p6,P7,P8)***	***Delete_Property(a,P1,P2,P3,P4,p5,p6,P7,P8)***
Intuition	Add property *a* with its neighborhood links	Delete property *a* with its neighborhood links
Arguments	*P1* = set of new parent properties of *a*, *P2* = set of properties that have as parent *a*, *P3* = set of new metaproperties of *a*, *P4* = set of new property instances of *a*, *p5* = the new domain of *a*, *p6* = the new range of *a*, *P7* = set of new comments of *a*, *P8* = set of new labels of *a*	*P1* = set of old parent properties of *a*, *P2* = set of properties that had as parent *a*, *P3* = set of old metaproperties of *a*, *P4* = set of old property instances of *a*, *p5* = the old domain of *a*, *p6* = the old range of *a*, *P7* = set of old comments of *a*, *P8* = set of old labels of *a*
δ_a	$\forall p \in P1 : (a, subPropertyOf , p)$, $\forall p \in P2 : (p, subPropertyOf , a)$, $\forall p \in P3 : (a, type, p)$, $\forall p1, p2 \in P4 : (p1, a, p2)$, *(a, domain, p5),* *(a, range, p6),* $\forall p \in P7 : (a, comment, p)$, $\forall p \in P8 : (a, label, p)$, *(a, type, property)*	ø
δ_d	ø	$\forall p \in P1 : (a, subPropertyOf , p)$, $\forall p \in P2 : (p, subPropertyOf , a)$, $\forall p \in P3 : (a, type, p)$, $\forall p1, p2 \in P4 : (p1, a, p2)$, *(a, domain, p5),* *(a, range, p6),* $\forall p \in P7 : (a, comment, p)$, $\forall p \in P8 : (a, label, p)$, *(a, type, property)*
Inverse	*Delete_Property(a,P1,P2,P3,P4,p5,p6,P7,P8)*	*Add_Property(a,P1,P2,P3,P4,p5,p6,P7,P8)*
GAV Mappings	-	-

Change	***Reclassify_Class(a,B,C)***
Intuition	Reclassify a class
Arguments	*B* = set of old types of of *a*, *C* = set of new types of of *a*
δ_a	$\forall c \in C : (a, type, c)$
δ_d	$\forall b \in B : (a, type, b)$
Inverse	*Reclassify_Class(a,C,B)*
GAV Mappings	–

The changes *Reclassify_Metaclass* and *Reclassify_Metaproperty* are defined similarly.

Change	***Ungroup_Properties_Under(A,b)***	***Ungroup_Properties_Under(A,b)***
Intuition	Group properties in *A* under *b*	Ungroup properties in *A* under *b*
Arguments	*A* = set of properties that have as new parent *b*, *b* = new parent property *b*	*A* = set of properties that had as parent *b*, *b* = the old parent property *b*
δ_a	$\forall a \in A : (a, subPropertyOf , b)$	ø
δ_d	ø	$\forall a \in A : (a, subPropertyOf , b)$
Inverse	*Ungroup_Properties_Under(A,b)*	*Group_Properties_UnderA,b)*
GAV Mappings	-	-

Change	***Pull_up_Property(a,B,C)***	***Pull_down_Property(a,B,C)***
Intuition	Move property *a* to a higher position in the subsumption hierarchy	Move property *a* to a lower position in the subsumption hierarchy
Arguments	*B* = set of old parents of *a*, *C* = set of new parents of *a*	*B* = set of old parents of *a*, *C* = set of new parents of *a*
δ_a	$\forall c \in C : (a, subPropertyOf , c)$	$\forall c \in C : (a, subPropertyOf , c)$
δ_d	$\forall b \in B : (a, subPropertyOf , b)$	$\forall b \in B : (a, subPropertyOf , b)$
Inverse	Pull_down_Property(a,C,B)	Pull_up_Property(a,C,B)
GAV Mappings	–	–

Change	***Move_Property(a,B,C)***	***Change_Superproperties(a,B,C)***
Intuition	Move property *a* to a different subsumption hierarchy	Change the parents of property *a*
Arguments	*B* = set of old parents of *a*, *C* = set of new parents of *a*	*B* = set of old parents of *a*, *C* = set of new parents of *a*
δ_a	$\forall c \in C$: (a, subClassOf , c)	$\forall c \in C$: (a, subClassOf , c)
δ_d	$\forall b \in B$: (a, subClassOf , b)	$\forall b \in B$: (a, subClassOf , b)
Inverse	*Move_Property(a,C,B)*	*Change_Superproperties(a,C,B)*
GAV Mappings	–	–

Change	***Reclassify_Property_higher(a,B,C)***	***Reclassify_Property_lower(a,B,C)***
Intuition	Reclassify property *a* to a higher position in the subsumption hierarchy	Reclassify property *a* to a lower position in the subsumption hierarchy
Arguments	*B* = set of old types of *a*, *C* = set of new types of *a*	*B* = set of old types of *a*, *C* = set of new types of *a*
δ_a	$\forall c \in C$: (a, type, c)	$\forall c \in C$: (a, type, c)
δ_d	$\forall b \in B$: (a, type, b)	$\forall b \in B$: (a, type, b)
Inverse	*Reclassify_Property_lower(a,C,B)*	*Reclassify_Property_higher(a,C,B)*
GAV Mappings	–	–

Change	***Reclassify_Property(a,B,C)***
Intuition	Reclassify property *a*
Arguments	*B* = set of old types of *a*, *C* = set of new types of of *a*
δ_a	$\forall c \in C$: (a, type, c)
δ_d	$\forall b \in B$: (a, type, b)
Inverse	*Reclassify_Property(a,C,B)*
GAV Mappings	–

Change	***Change_To_Datatype_Property(a,b,c)***	***Change_To_Object_Property(a,b,c)***
Intuition	Change the range of property *a* to a datatype	Change the range of property *a* to an object
Arguments	*b* = old range of *a*, *c* = new range of *a*	*b* = old range of *a*, *c* = new range of *a*
δ_a	$\forall c \in C$: (a, range, c)	$\forall c \in C$: (a, range, c)
δ_d	$\forall b \in B$: (a, range, b)	$\forall b \in B$: (a, range, b)
Inverse	*Change_To_Object_Property(a,c,b)*	*Change_To_Datatype_Property(a,c,b)*
GAV Mappings	$range(a, b) \rightarrow range(a, c)$ *(high-level)* $\forall x, a(x, b) \rightarrow a(x, c)$ *(low-level)*	$range(a, b) \rightarrow range(a, c)$ *(high-level)* $\forall x, a(x, b) \rightarrow a(x, c)$ *(low-level)*

Change	***Specialize_Range(a,b,c)***	***Generalize_Range(a,b,c)***
Intuition	Change the range of property a to a subclass of it	Change the range of property a to a superClass of it
Arguments	*b* = old range of *a*, *c* = new range of *a*	*b* = old range of *a*, *c* = new range of *a*
δ_a	$\forall c \in C : (a, range, c)$	$\forall c \in C : (a, range, c)$
δ_d	$\forall b \in B : (a, range, b)$	$\forall b \in B : (a, range, b)$
Inverse	*Generalize_Range(a,c,b)*	*Specialize_Range(a,c,b)*
GAV Mappings	$range(a, b) \rightarrow range(a, c)$ *(high-level)* $\forall x, a(x, b) \rightarrow a(x, c)$ *(low-level)*	$range(a, b) \rightarrow range(a, c)$ *(high-level)* $\forall x, a(x, b) \rightarrow a(x, c)$ *(low-level)*

Change	***Change_Range(a,b,c)***
Intuition	Change the range of property a.
Arguments	b = old range of a, c = new range of a
δ_a	$\forall c \in C : (a, range, c)$
δ_d	$\forall b \in B : (a, range, b)$
Inverse	*Change_Range(a,c,b)*
GAV Mappings	$range(a, b) \rightarrow range(a, c)$ *(high-level)* $\forall x, a(x, b) \rightarrow a(x, c)$ *(low-level)*

Change	***Specialize_Domain(a,b,c)***	***Generalize_Domain(a,b,c)***
Intuition	Change the domain of property *a* to a subClass of it	Change the domain of property *a* to a super-Class of it
Arguments	*b* = old domain of *a*, *c* = new domain of *a*	*b* = old domain of *a*, *c* = new domain of *a*
δ_a	$\forall c \in C : (a, domain, c)$	$\forall c \in C : (a, domain, c)$
δ_d	$\forall b \in B : (a, domain, b)$	$\forall b \in B : (a, domain, b)$
Inverse	*Generalize_Range(a,c,b)*	*Generalize_Domain(a,c,b)*
GAV Mappings	$domain(a, b) \rightarrow domain(a, c)$ *(high-level)* $\forall x, a(b, x) \rightarrow a(c, x)$ *(low-level)*	$domain(a, b) \rightarrow domain(a, c)$ *(high-level)* $\forall x, a(b, x) \rightarrow a(c, x)$ *(low-level)*

Change	***Change_Domain(a,b,c)***
Intuition	Change the domain of property *a*.
Arguments	*b* = old domain of *a*, *c* = new domain of *a*
δ_a	$\forall c \in C : (a, domain, c)$
δ_d	$\forall b \in B : (a, domain, b)$
Inverse	*Change Domain(a,c,b)*
GAV Mappings	$domain(a, b) \rightarrow domain(a, c)$ *(high-level)* $\forall x, a(b, x) \rightarrow a(c, x)$ *(low-level)*

Heuristics

Change	***Rename_Class(a,b)***
Intuition	Rename class a to b
Arguments	*a* = the old name of the class, *b* = the new name of the class
δ_a	*(b, type, class), (b, subClassOf , resource)*
δ_d	*(a, type, class), (a, subClassOf , resource)*
Inverse	*Rename_Class(b,a)*
GAV Mappings	*type(b, class)* $\rightarrow$ *type(b, class)* *(high-level)* $\forall x$, *a(x)* $\rightarrow$ *b(x)* *(low-level)*

The change *Rename_Metaclass* is defined similarly with the exception of (*a*, *subClassOf*, *class*) being in δ_a and δ_d respectively instead of (*a*, *subClassOf* , *resource*).

Change	***Merge_Classes(A,b)***	***Split_Class(a,B)***
Intuition	Merge classes contained in *A* into *b*	Split class *a* into classes contained in *B*
Arguments	*A* = the set of old names of the classes, *b* = the new name of the class	*a* = the old name of the class, *B* = the set of new names of the classes
δ_a	*(b, type, class),* *(b, subClassOf , resource)*	$\forall b_i \in B$ *: (b, type, class), (1≤i≤n)* *(b, subClassOf , resource)*
δ_d	$\forall a_i \in A$ *: (a, type, class), (1≤i≤n)* *(a, subClassOf , resource)*	*(a, type, class),* *(a, subClassOf , resource)*
Inverse	*Split_Class(b,A)*	*Merge_Classes(A,b)*
GAV Mappings	*(high-level)* $type(a_1, class) \rightarrow type(b, class) \wedge split(a_1, b, A)$ ... $type(a_n, class) \rightarrow type(b, class) \wedge split(a_n, b, A)$ *(low-level)* $\forall x, a_1(x) \rightarrow \exists x_1, b(x_1) \wedge split(x, x_1, A)$ ... $\forall x, a_n(x) \rightarrow \exists x_n, b(x_n) \wedge split(x, x_n, A)$	*(high-level)* $type(a, class) \rightarrow type(b_1, class) \wedge ... \wedge type(b_n, class) \wedge concat(a, \{b_1,...,b_n\})$ *(low-level)* $\forall x, a(x) \rightarrow \exists x_1, ..., x_n, b_1(x_1) \wedge ... \wedge b_n(x_n) \wedge concat(x, \{x_1,...,x_n\})$

The changes *Merge_Metaclasses*, *Merge_Metaproperties* and *Split_Metaclass*, *Split_Metaproperty* are defined similarly with the exception of (*a*, *subClassOf* , *class*)

and (*a*, *subClassOf* , *property*) being in δ_a and δ_d respectively instead of (*a*, *subClassOf* , *resource*).

Change	***Merge_Classes_Into_Existing(A,b)***	***Split_Class_Into_Existing(a,B)***
Intuition	Merge classes contained in *A* into *b*	Split class *a* into classes contained in *B*
Arguments	*A* = the set of old names of the classes, *b* = the new name of the class	*a* = the old name of the class, *B* = the set of new names of the classes
δ_a	-	$\forall b_i \in B \setminus \{a\} : (b_i, type, class), (1 \le i \le n)$ $(b_i, subClassOf , resource)$
δ_d	$\forall a_i \in A \setminus \{b\} : (a_i, type, class), (1 \le i \le n)$ $(a_i, subClassOf , resource)$	-
Inverse	Split_Class_Into_Existing(b,A)	Merge_Classes_Into_Existing(A,b)
GAV Mappings	*(high-level)* $type(a_1, class) \rightarrow type(b, class) \wedge split(a_1, b, A)$ ... $type(a_n, class) \rightarrow type(b, class) \wedge split(a_n, b, A)$ *(low-level)* $\forall x, a_1(x) \rightarrow \exists x_1, b(x_1) \wedge split(x, x_1, A)$ ... $\forall x, a_n(x) \rightarrow \exists x_n, b(x_n) \wedge split(x, x_n, A)$	*(high-level)* $type(a, class) \rightarrow type(b_1, class) \wedge ... \wedge type(b_n, class) \wedge concat(a, \{b_1,...,b_n\})$ *(low-level)* $\forall x, a(x) \rightarrow \exists x_1, ..., x_n, b_1(x_1) \wedge ... \wedge b_n(x_n) \wedge concat(x, \{x_1,...,x_n\})$

The changes *Merge_Metaclasses_Into_Existing*, and *Split_Metaclass_Into_Existing* are defined similarly with the exception of (*a*, *subClassOf*, *class*) being in δ_a and δ_d respectively instead of (*a*, *subClassOf*, *resource*).

Change	***Rename_Property(a,b)***
Intuition	Rename property a to *b*
Arguments	*a* = the old name of the property, *b* = the new name of the property
δ_a	*(b, type, property)*
δ_d	*(a, type, property)*
Inverse	*Rename_Property(b,a)*
GAV Mappings	$type(a, property) \rightarrow type(b, property)$ *(high-level)* $\forall x,y, a(x, y) \rightarrow b(x, y)$ *(low-level)*

The change *Rename_Metaproperty* is defined similarly with the exception of (*a*, *subClassOf*, *property*) being in δ_a (δ_d) instead of (*a*, *subClassOf* , *resource*).

Change	***Merge_Properties(A,b)***	***Split_Property(a,B)***
Intuition	Merge properties contained in *A* into *b*	Split property a into properties contained in *B*
Arguments	*A* = the set of old names of the properties, *b* = the new name of the property	*a* = the old name of the property, *B* = the set of new names of the properties
δ_a	*(b, type, property)*	*$\forall\, b_i \in B$: (b, type, property) (1≤i≤n)*
δ_d	*$\forall\, a_i \in A$: (a, type, property) (1≤i≤n)*	*(a, type, property)*
Inverse	*Split_Property(b,A)*	*Merge_Properties(A,b)*
GAV Mappings	*(high-level)* *$type(a_1, property) \rightarrow type(b, property) \wedge split(a_1, b, A)$* ... *$type(a_n, property) \rightarrow type(b, property) \wedge split(a_n, b, A)$* *(low-level)* *$\forall x, y,\ a_1(x\ ,\ y) \rightarrow \exists\, y_1,\ b(x\ ,\ y_1) \wedge split(y, y_1, A)$* ... *$\forall x, y,\ a_n(x\ ,\ y) \rightarrow \exists\, y_n,\ b(x\ ,\ y_1) \wedge split(y, y_n, A)$*	*(high-level)* *$type(a, property) \rightarrow type(b_1, property) \wedge \ldots \wedge type(b_n, property) \wedge concat(a, \{b_1,\ldots,b_n\})$* *(low-level)* *$\forall x,y,\ a(x\ ,\ y) \rightarrow \exists x_1,\ \ldots,\ x_n,\ b_1(x\ ,\ x_1) \wedge \ldots \wedge b_n(x, x_n) \wedge concat(y, \{x_1,\ldots,x_n\})$*

The changes *Merge_Metaproperties_Into_Existing* and *Split_Metaproperty_Into_Existing* are defined similarly with the exception of (*a*, *subClassOf*, *property*) being in δ_a and δ_d respectively instead of (*a*, *subClassOf*, *resource*).

Change	***Merge_Properties_Into_Existing(A,b)***	***Split_Property_Into_Existing(a,B)***
Intuition	Merge properties contained in A into b	Split property a into properties contained in B
Arguments	*A* = the set of old names of the properties, *b* = the new name of the property	*a* = the old name of the property, *B* = the set of new names of the properties
δ_a	-	$\forall\, b_i \in B \setminus \{a\}$: *(b, type, property)* $(1 \le i \le n)$
δ_d	$\forall a_i \in A \setminus \{b\}$: *(a, type, property)* $(1 \le i \le n)$	-
Inverse	*Split_Property_Into_Existing(b,A)*	*Merge_Properties_Into_Existing(A,b)*
GAV Mappings	*(high-level)* $type(a_1, property) \rightarrow type(b, property) \wedge split(a_1, b, A)$ ... $type(a_n, property) \rightarrow type(b, property) \wedge split(a_n, b, A)$ *(low-level)* $\forall x, y,\ a_1(x\ ,\ y) \rightarrow \exists\, y_1,\ b(x\ ,\ y_1) \wedge split(y, y_1, A)$ ... $\forall x, y,\ a_n(x\ ,\ y) \rightarrow \exists\, y_n,\ b(x\ ,\ y_1) \wedge split(y, y_n, A)$	*(high-level)* $type(a, property) \rightarrow type(b1, property) \wedge \ldots \wedge type(bn, property) \wedge concat(a, \{b1,\ldots,bn\})$ *(low-level)* $\forall x,y,\ a(x\ ,\ y) \rightarrow \exists x1,\ \ldots,\ xn,\ b1(x\ ,\ x1) \wedge \ldots \wedge bn(x, xn) \wedge concat(y, \{x1,\ldots,xn\})$

Change	***Change_Comment(u,a,b)***	***Change_Label(a,b)***
Intuition	Change comment of resource *u* from *a* to *b*	Change label of resource *u* from *a* to *b*
Arguments	*a* = the old comment, *b* = the new comment	*a* = the old label, *B* = the new label
δ_a	*(u, comment, b)*	*(u, label, b)*
δ_d	*(u, comment, a)*	*(u, label, a)*
Inverse	*Change_Comment(u,b,a)*	*Change_Label(u,b,a)*
GAV Mappings	$\forall u$, *rdfs:comment (u, a)* $\rightarrow$ *rdfs:comment (u, b)*	$\forall u$, *rdfs:label (u, a)* $\rightarrow$ *rdfs:label (u, b)*