Ontology and Schema Evolution in Data Integration: Review and Assessment

Haridimos Kondylakis[1,2], Giorgos Flouris[1], and Dimitris Plexousakis[1,2]

[1] Information Systems Laboratory, FORTH-ICS
[2] Computer Science Department, University of Crete
Vassilika Vouton P.O Box 1385, GR 71110 Heraklion, Crete
{kondylak,fgeo,dp}@ics.forth.gr

Abstract. The development of new techniques and the emergence of new high-throughput tools have led to a new information revolution. The amount and the diversity of the information that need to be stored and processed have led to the adoption of data integration systems in order to deal with information extraction from disparate sources. The mediation between traditional databases and ontologies has been recognized as a cornerstone issue in bringing in legacy data with formal semantic meaning. However, our knowledge evolves due to the rapid scientific development, so ontologies and schemata need to change in order to capture and accommodate such an evolution. When ontologies change, these changes should somehow be rendered and used by the pre-existing data integration systems, a problem that most of the integration systems seem to ignore. In this paper, we review existing approaches for ontology/schema evolution and examine their applicability in a state-of-the-art, ontology-based data integration setting. Then, we show that changes in schemata differ significantly from changes in ontologies. This strengthens our position that current state of the art systems are not adequate for ontology-based data integration. So, we give the requirements for an ideal data integration system that will enable and exploit ontology evolution.

Keywords: Ontology Evolution, Data Integration, Mappings, Evolution of Mappings.

1 Introduction

The development of new 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. However, the integration of these data sources 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

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 932–947, 2009.

resolve heterogeneity problems, at schema and data level, as a means for establishing explicit formal vocabulary to share. The key in bringing legacy data with formal semantic meaning has been widely recognized to be the inclusion of mediation between traditional databases and ontologies [5, 15]. During the last years, ontologies have been used in database integration, obtaining promising results, for example in the fields of biomedicine and bioinformatics [19].

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* [13] and the output of this task is the *mapping*, i.e., a collection of mappings rules. 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 [5]. 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. A typical solution would be to regenerate the mappings and then regenerate the dependent artifacts. We believe however that the approach to recreate mappings from scratch as the ontology evolves is problematic [32], and instead previously captured information should be reused.

In this paper, we address the problem of data integration for highly dynamic ontologies. We argue that ontology change should be considered when designing ontology-based data integration systems.

We identify solutions proposed in the state of the art 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)* [1] [21] and those that try to evolve the mappings each time a primitive change operation occurs (*mapping adaptation)* [33]. 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. Moreover, the only approach [34] we have seen concerning ontology evolution seems too simple and does not depict reality.

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 paper 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.

This rest of the paper is organized as follows: in Section 2 we give some preliminaries and argue that ontology evolution should be considered when designing ontology-based

data integration systems; then, in Section 3 we review the solutions proposed so far in the literature and show the related problems. This argument is further enhanced in Section 4, by showing that changes in schemata differ significantly from changes in ontologies. All those problems lead to the specification of requirements of an ideal data integration system that will incorporate and handle ontology evolution. Finally in Section 5 we conclude the paper and give directions for future work.

2 Preliminaries

Originally introduced by Aristotle, ontologies are formal models 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 modeling the real world [5]. Ontologies are often large and complex structures, whose development and maintenance give rise to several sturdy and interesting research problems. One of the most important such problems is ontology evolution, which refers to the process of modifying an ontology in response to a certain change in the domain or its conceptualization [13].

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 [28]; 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 [22], or we may discover a design flaw in the original conceptualization of the domain [25]; we may also wish to incorporate additional functionality, according to a change in users' needs [9]. 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 [12]. The importance of this problem is also emphasized by recent studies, which suggest that change, rather than ontologies, should be the central concept of the Semantic Web [31].

In [11] it is shown that most well-known life science ontologies are heavily updated and grow significantly from time to time. There have been several works in the literature addressing the problem of ontology evolution. An indicative list is: [6], [8], [10], [14], [18], [23], [27], [29]; for a more comprehensive and complete survey, see [5].

An interesting classification of changes that is of interest for the problem of data integration appears in [4]. In this work, changes are classified under three broad categories. The first level (*logic-level changes*), which is difficult and not supported by current approaches, corresponds to changes in the logical formalism which is used to represent the ontology, rather than the ontology itself. The second (*language-level changes*) and third (*KB-level changes*) levels are more interesting and supported by ontology evolution approaches. Language-level changes correspond to changes in the objects of the ontology (e.g., classes, properties etc); examples of language-level changes are the addition of a new concept or the deletion of a property. KB-level changes correspond to changes in the information about the existing objects, i.e., structural changes in the ontology; for example, changes in the class or property hierarchies, or changes in some constraints related to a particular class are KB-level changes.

This categorization is relevant to the data integration problem due to the different effects that a change in each of the levels would have in the underlying matching. For example, KB-level changes affect the structure of the ontology, and, consequently, the possible models of it, as well as the intended meaning of the used terms. Such effects should be reflected in the mapping in order to avoid inconsistent mappings. On the other hand, language-level changes correspond to additions and deletions of objects from the ontology, therefore their effects on the mapping, if any, can be trivially computed. Logic-level changes are irrelevant in our setting for two reasons: first, because such changes are not supported by ontology evolution approaches, and, second, because a critical assumption in our work is that the underlying logical formalism is the same in all mapped ontologies.

3 A Review of the State the Art

A typical solution to the problem of data integration with evolving ontologies would be to regenerate the mappings and then the dependent artifacts. This method is called the "blank-sheet approach" [35]. 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 [32], 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.

3.1 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.[7] and Mohania and Dong [20] addressed the problem of maintaining a materialized view after user redefinition, while [26] explored how to use view technology to handle schema changes transparently.

Lee et al. [16] 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 human interaction. However, the supported changes were limited and evolution could only appear at the source side.

3.2 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 [17] 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. 1). 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.

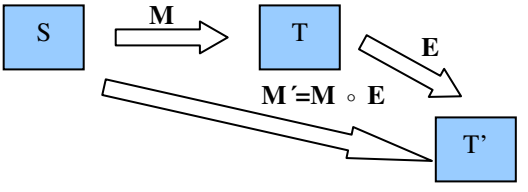

Fig. 1. Composing Schema Mappings

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.

Consider for example the three schemata *S, T* and *T′* shown in Fig. 2. 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

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

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

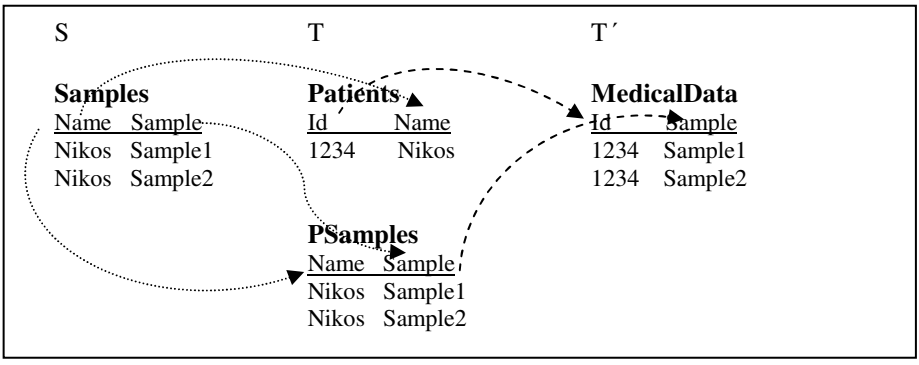

Fig. 2. The example schemata

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 [2]. 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, $Q_1 \subseteq Q_2$. It is called source-to-target when Q_1 refers only to symbols from the source schema and Q_2 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={(Nikos, 1234)}* 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≥1 the following source-to-target tgd:

$$\varphi_\kappa = \{ \forall n \ \forall s_1 \ldots \forall s_k \ (Samples (n,s_1) \wedge \ldots \wedge Samples (n, s_k) \rightarrow \exists i \ MedicalData (i, s_1) \wedge \ldots \wedge MedicalData (i, s_k))\}$$

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. [3] 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. [3] 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 [35] 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 [3], with extensions to handle nesting and with significant attention to minimizing the size of the result. They reported on 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. [21] tried to extend the work of Fagin et al. [3]. 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 $Q_1 \subseteq Q_2$ where Q_1 and Q_2 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. [1] in 2008 that propose a new composition algorithm that targets practical applications. Like [21], 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 [3], except that as in [21], 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 [3].

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 [35]. 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.

3.3 Mapping Adaptation

In parallel with the previous approaches that considered mapping composition, Velegrakis et al. [33] focused on incrementally adapting mappings on schema change.

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.

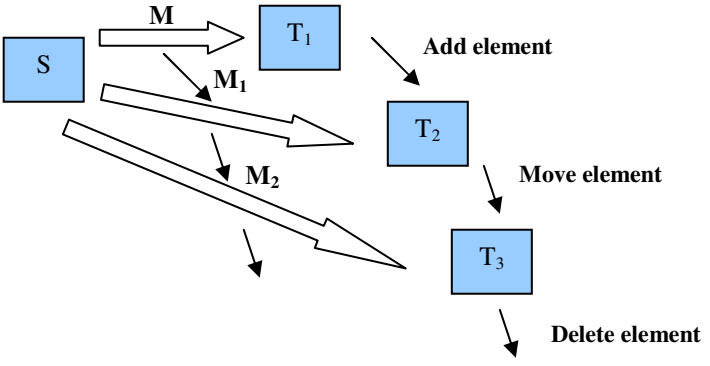

Fig. 3. Adapting Schema Mappings

Consider for example the schemata T and T' shown in Fig. 4. 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.

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. 4. Identifying mapping adaptation problems

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.

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

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

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

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

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

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

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[1], mzXML[2] standards etc.). The list of changes may not be given and may need to be discovered [36], 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 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.

[1] http://www.hl7.org/
[2] http://sashimi.sourceforge.net/software_glossolalia.html

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 one patient that is 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.

3.4 Floating Model

Xuan et al. [34] 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 [34] 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 [11]. 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.

4 Discussion

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 [24]. 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., [14], [18]). 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 [5]) 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 [30]. This claim however is problematic because the transformation of ontological knowledge into a relational schema is often a complicated process. In [30], 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 [30]), 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[35].

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.

In an ideal system, several databases would be mapped to the ontology as the ontology evolves. For example, as shown in Fig 5, 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.

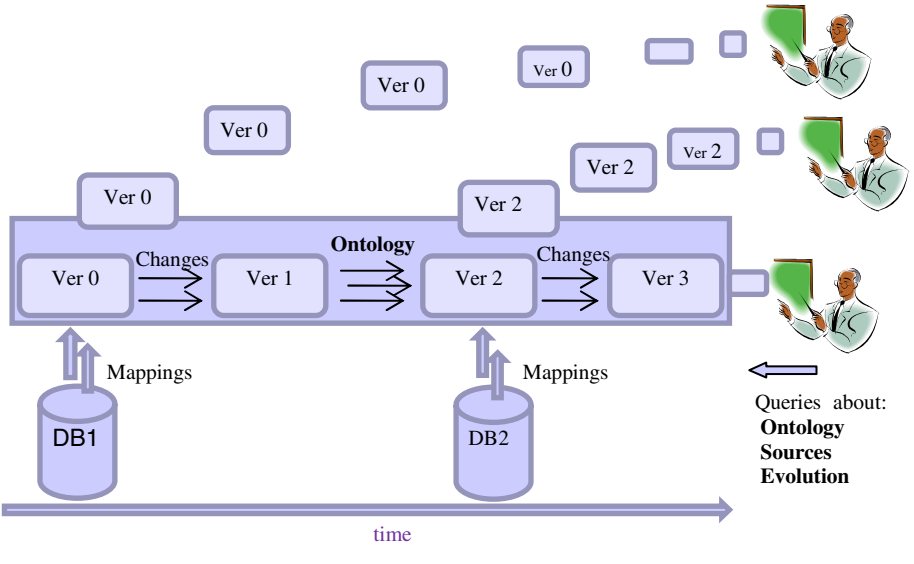

Fig. 5. An ideal solution

To conclude, an ideal solution should to 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.

5 Conclusion

In this paper we showed that dynamic ontologies are very common, so data integration systems should be aware and ready to deal with that. We reviewed existing approaches for handling schema and ontology evolution and assessed their applicability in an ontology-based data integration system. We identified their drawbacks and concluded that they cannot be used "as-is" in a general solution. Moreover, we showed that data integration in ontologies is a problem that is inherently different from the data integration problem for databases.

Then, we tried to highlight the requirements for an ideal system. In such a system:

1. the changes should be considered directly on the ontology level
2. queries should concern ontology evolution as well
3. the whole mapping process should be performed only once
4. the domain experts should be able to provide, or at least verify, the mapping between the ontologies and the data sources
5. precise criteria need to ensure that the produced answer is the right one

A query, formulated using one ontology version, should be able to retrieve answers from all databases, even if they are mapped with a different ontology version.

To the best of our knowledge, no system today is capable of fulfilling all the requirements specified and further research is required. Several challenges need to be resolved as it might not be possible to extract information mapped to a class, using a later ontology version in which the specific class is deleted or moved. Even more, it might not be meaningful to do so. Moreover, whenever an answer from a specific mapped database is not possible we might want to check the most relevant answer to our question. Even worse, local schemata may evolve, and the structured DBMS data might be replaced with semi-structured or unstructured data. It is obvious that ontology evolution in data integration is an important topic and several challenging issues remain to be investigated in the near future.

Acknowledgments. This work was partially supported by the EU project PlugIT (ICT-231430). The authors thank the reviewers for their useful comments.

References

1. Bernstein, P.A., Green, T.J., Melnik, S., Nash, A.: Implementing mapping composition. The VLDB Journal 17, 333–353 (2008)
2. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoretical Computer Science 336, 89–124 (2005)
3. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.-C.: Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst. 30, 994–1055 (2005)
4. Flouris, G.: On the Evolution of Ontological Signatures. In: Proceedings of the Workshop on Ontology Evolution, pp. 67–72 (2007)
5. Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: Classification and survey. Knowl. Eng. Rev. 23, 117–152 (2008)
6. Flouris, G., Plexousakis, D., Antoniou, G.: On Applying the AGM Theory to DLs and OWL. In: Proc. of Int. Semantic Web Conf., pp. 216–231 (2005)

7. Gupta, A., Jagadish, H.V., Mumick, I.S.: Data Integration using Self-Maintainable Views. In: Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology. Springer, Heidelberg (1996)
8. Haase, P., Harmelen, F.v., Huang, Z., Stuckenschmidt, H., Sure, Y.: A Framework for Handling Inconsistency in Changing Ontologies. In: International Semantic Web Conference, pp. 353-367 (2005)
9. Haase, P., Stojanovic, L.: Consistent Evolution of OWL Ontologies. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 182–197. Springer, Heidelberg (2005)
10. Halaschek-Wiener, C., Katz, Y.: Belief Base Revision for Expressive Description Logics. In: OWLED (2006)
11. Hartung, M., Kirsten, T., Rahm, E.: Analyzing the Evolution of Life Science Ontologies and Mappings. In: Proceedings of the 5th international workshop on Data Integration in the Life Sciences, Springer, Evry (2008)
12. Heflin, J., Hendler, J., Luke, S.: Coping with Changing Ontologies in a Distributed Environment. In: Proceedings of AAAI, Workshop on Ontology Management, pp. 74-79. Press (1999)
13. Klein, M.: Combining and relating ontologies:an analysis of problems and solutions. In: IJCAI (2001)
14. Konstantinidis, G., Flouris, G., Antoniou, G., Christophides, V.: Ontology Evolution: A Framework and its Application to RDF. In: Proceedings of the Joint ODBIS & SWDB Work-shop on Semantic Web, Ontologies, Databases, SWDB-ODBIS 2007 (2007)
15. Konstantinou, N., Spanos, D.-E., Mitrou, N.: Ontology and database mapping: A survey of current implementations and future directions. Journal of Web Engineering 7, 1–24 (2008)
16. Lee, A.J., Nica, A., Rundensteiner, E.A.: The EVE Approach: View Synchronization in Dynamic Distributed Environments. IEEE Trans. on Knowl. and Data Eng. 14, 931–954 (2002)
17. Madhavan, J., Halevy, A.Y.: Composing mappings among data sources. In: Proceedings of the 29th international conference on Very large data bases, VLDB Endowment, Berlin, Germany, vol. 29 (2003)
18. Magiridou, M., Sahtouris, S., Christophides, V., Koubarakis, M.: RUL: A Declarative Update Language for RDF. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 506–521. Springer, Heidelberg (2005)
19. 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 In-tegration of Distributed and Heterogeneous Data Sources in ACGT. In: Proceedings of the First International Conference on Health Informatics (HEALTHINF 2008), Funchal, Madeira, Portugal, pp. 301–306 (2008)
20. Mohania, M., Dong, G.: Algorithms for Adapting Materialised Views in Data Warehouses. In: CODAS, pp. 309-316 (1996)
21. Nash, A., Bernstein, P.A., Melnik, S.: Composition of mappings given by embedded dependencies. ACM Trans. Database Syst. 32, 4 (2007)
22. Noy, N.F.: Semantic integration: a survey of ontology-based approaches. SIGMOD Rec 33, 65–70 (2004)
23. Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A Framework for Ontology Evolution in Collaborative Environments. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 544–558. Springer, Heidelberg (2006)

24. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems 6, 428–440 (2004)
25. Plessers, P., Troyer, O.D.: Ontology Change Detection Using a Version Log. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 578–592. Springer, Heidelberg (2005)
26. Ra, Y.-G., Rundensteiner, E.A.: A Transparent Schema-Evolution System Based on Object-Oriented View Technology. IEEE Trans. on Knowl. and Data Eng. 9, 600–624 (1997)
27. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-Driven Ontology Evolution Management. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 285–300. Springer, Heidelberg (2002)
28. Stojanovic, L., Maedche, A., Stojanovic, N., Studer, R.: Ontology evolution as reconfigura-tion-design problem solving. In: Proceedings of the 2nd international conference on Knowledge capture. ACM, Sanibel Island (2003)
29. Stojanovic, L., Motik, B.: Ontology Evolution Within Ontology Editors. In: Proceedings of the OntoWeb-SIG3 Workshop, pp. 53–62 (2002)
30. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking Database Representations of RDF/S Stores. In: International Semantic Web Conference, pp. 685–701 (2005)
31. Tzitzikas, Y., Kotzinos, D. (Semantic web) evolution through change logs: problems and solutions. In: Proceedings of the 25th conference on Proceedings of the 25th IASTED Interna-tional Multi-Conference: artificial intelligence and applications. ACTA Press, Innsbruck (2007)
32. Velegrakis, Y., Miller, J., Popa, L.: Preserving mapping consistency under schema changes. The VLDB Journal 13, 274–293 (2004)
33. Velegrakis, Y., Miller, R.J., Mylopoulos, J.: Representing and Querying Data Transformations. In: Proceedings of the 21st International Conference on Data Engineering. IEEE Computer Society, Los Alamitos (2005)
34. Xuan, D.N., Bellatreche, L., Pierra, G.: A Versioning Management Model for Ontology-Based Data Warehouses, DawaK, Poland (2006)
35. Yu, C., Popa, L.: Semantic adaptation of schema mappings when schemas evolve. In: Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, Trondheim, Norway (2005)
36. Zeginis, D., Tzitzikas, Y., Christophides, V.: On the Foundations of Computing Deltas Between RDF Models. In: ISWC/ASWC, pp. 637–651 (2007)