A Framework for Automatic Schema Mapping Verification Through Reasoning

Paolo Cappellari #1, Denilson Barbosa #2, Paolo Atzeni *3

#University of Alberta
Edmonton, AB, Canada
1cappella@cs.ualberta.ca
2denilson@cs.ualberta.ca

*Università Roma Tre
Roma, Italy
3atzeni@dia.uniroma3.it

Abstract—We advocate an automated approach for verifying mappings between source and target databases in which semantics are taken into account, and that avoids two serious limitations of current verification approaches: reliance on availability of sample source and target instances, and reliance on strong statistical assumptions. We discuss how our approach can be integrated into the workflow of state-of-the-art mapping design systems, and all its necessary inputs. Our approach relies on checking the entailment of verification statements derived directly from the schema mappings and from semantic annotations to the variables used in such mappings. We discuss how such verification statements can be produced and how such annotations can be extracted from different kinds of alignments of schemas into domain ontologies. Such alignments can be derived semi-automatically; thus, our framework might prove useful in also greatly reducing the amount of input from domain experts in the development of mappings.

I. INTRODUCTION

The design of a schema mapping is a complex, time-consuming and expensive process in critical data integration settings. Some estimates put the effort of designing a single schema mapping at 3 man-months [1]. Given such difficulties and the potential catastrophic effects of mapping the incorrect data, it is not surprising that sophisticated tools have been developed to assist in this process. State-of-the-art tools implement a workflow that can be summarized as follows: (1) discovering pairs of semantically related elements between the source and target schemas (the schema correspondences), (2) deriving logically coherent conceptual specifications of how to map the data from the source into the target (the schema mapping), and (3) compiling an executable program to perform the actual data integration/exchange (the executable mapping). Fig. 1 illustrates this process.

The individual steps above have been successfully automated to great extent. Correspondences are discovered through the use of schema matchers [2], [3], and fed into schema mapping tools [4], [5] which produce the final executable programs for mapping the data. While these tools relieve designers from considerable manual work, reducing cost, time and errors, they are far from perfect. First, schema matchers are approximate in nature, leading to ambiguous mappings.

Also, often there are multiple plausible schema mappings associated with a same set of correspondences. Thus, one crucial step where human intervention is still required is in the verification (and/or selection) of candidate mappings, produced during the mapping generation step.

The verification of a schema mapping encompasses several aspects, from checking that schema types are not violated to verifying that the data being mapped conforms with the intended semantics of the target database. Some of these aspects are easy to automate (e.g., verifying that proper data types are used, and that no integrity constraints are violated in the target database), and have been incorporated into the existing tools. However, identifying semantic discrepancies induced by a schema mapping is far more challenging, and,
Motivating Example: Fig. 2(a) shows two schemas $S$ and $T$ in a set of correspondences between their elements. $S$ stores data about projects and the employees involved in those projects, which can be engineers or managers; for the sake of argument, suppose these sets of employees are disjoint. $T$, on the other hand, stores contact information for the supervisors responsible for the projects. In Fig. 2(b) we show three plausible mappings that would be produced by state-of-the-art mapping generation tools. (As customary, we represent mappings as source-to-target tuple generating dependencies.)

The mapping verification step for this example would consist of determining whether each of $\mu_1$, $\mu_2$ and $\mu_3$ is compatible with the semantics of the target database $T$. In other words, we must determine whether $T$ allows only engineers to fill the role of a supervisor (in which case only $\mu_1$ acceptable); whether $T$ allows only managers to fill the role of a supervisor (in which case only $\mu_2$ acceptable); or whether $T$ allows both engineers and managers to fill the role of supervisors (in which case all mappings would be acceptable—although $\mu_3$ would be preferable).

Now, the answer to the questions above cannot be determined from the information which current mapping tools rely on. Hence, the need for human intervention. For example, in Clio [5], the verification is done by asking the designer to inspect sample target instances, generated by applying each candidate mapping to a sample of the source instance.

There are many drawbacks to such manual verification approach:

- it can only be effective if the designer is very familiar with not only the semantics of both databases but also with their instances;
- different designers may judge the correctness the same mappings differently, depending on their interpretation of the semantics and familiarity with the instances;
- the process is time consuming and does not scale, as too many candidate mappings may be generated in large settings.

Before we present our approach we review the state-of-the-art in mapping verification in the next section.

II. RELATED WORK

An improvement over the manual verification that helps designers in exploring and understanding a schema mapping is the Spider tool [9]. Here, the designer can select subsets of a database to inspect; then Spider shows the routes in the mappings that associated those values created in the target with those that originated them in the source. Inspecting routes offers a better way to understand the origin of specific values in the target database. However, like in the previous approach, the process of understanding which is the mapping producing the right data is entirely driven by the designer. Moreover, this approach also relies on sampling and incurs the same pitfalls as discussed above.

The Spicy tool [7] fully automates the sample-based mapping verification discussed above. Spicy generates sample target instances for the various plausible mappings and compares each generated instance with a sample target instance. Such comparisons are performed on a statistical basis on the values in the available target database and those in the generated samples. The output of the comparison is used to refine the mapping at both the correspondences definition and schema mapping level. The main advantage of the approach is that it is automatic and can, hence, scale on large settings. Spicy, however, relies on the strong assumption that there are statistically significant and detectable patterns in the data that allow the verification of a mapping. Recall that in the example above the only data about the supervisor stored in the target is her name and email address. It is unlikely that one can determine whether supervisors can be engineers or not based on their names and email addresses alone. Another downside is that it requires the availability of both source and target
instances.

One common shortcoming of all methods above is their reliance on sampling, which means they cannot work without enough real data. This can be problematic in some cases, e.g., due to privacy or commercial restrictions. There are also the known problems with sampling; e.g., the sample may not be representative of the source database and, as a consequence, the sample target data produced by the mapping might not be representative of the effects of a mapping which, leading to inaccurate decision from the designer.

In summary, the state-of-the-art in mapping verification is to rely heavily on the expertise of the designer as well as on sampling. Next we propose a radically different approach, based on exploiting semantic annotations and turning the mappings into statements about a domain ontology, which can be checked automatically.

III. TOWARDS AUTOMATIC VERIFICATION

In this paper we advocate a generic framework for helping in the automation of the verification step in the design of schema mappings. Our fundamental difference to previous work is that we propose that: the semantics of the source and target schema be explicitly captured (as annotations), and that we use formal reasoning to verify such mappings. Fig. 3 illustrates the relationship of our framework for the verification of schema mappings with the mapping design step. The workflow of Fig. 1 remains virtually unchanged, except for the fact that the manual verification step is replaced by an automated one, and the designer now may provide other kinds of input as we will discuss later.

A. Semantic Annotations

In our vision, semantic annotations are assignments of semantics to the variables in the (queries defining) schema mappings. These annotations are used to derive verification statements which are checked against the ontology, to detect incompatibilities imposed by the mapping. Recall the example of Fig. 2, and assume for the sake of argument that the target database $T$ admits only engineers as project supervisors. Thus, $\mu_1$ in Fig. 2(a) is unacceptable because it maps names (and emails) of managers into names (and emails) of engineers, which is not allowed in this setting. In other words, the semantics of variable $n$ (and also $c$) in the LHS and the RHS of $\mu_1$ are incompatible. Consider now $\mu_2$: it maps engineers into engineers and thus uses compatible semantics in both its LHS and RHS.

Now, we make more precise what we mean by annotation. A source-to-target tuple-generating dependency [10], [11], or $ST$-TGD, is an expression of the form $(\forall x)(n_1 x \rightarrow (\exists y)q_T(x,y))$, where $q_S(x)$ is a conjunctive query over the source database and $q_T(x,y)$ is a conjunctive query over the target database. Moreover, every variable in $x$ appears in $q_S(x)$. A schema mapping is a set of $ST$-TGDs between the same pair of schemas $S,T$: $\mu = \{\sigma_1, \ldots, \sigma_n\}$.

For an $ST$-TGD $\sigma = (\forall x)(n_1 x \rightarrow (\exists y)q_T(x,y))$, and a variable $x \in x$, we assign two (possibly unrelated) concepts from a common domain ontology into $x$, which we call $\Gamma_{q_S}(x)$ and $\Gamma_{q_T}(x)$. Intuitively, the semantic annotation $\Gamma_q(x)$ assigns meaning to the variable $x$ in the query $q$ relative to the domain ontology.

To make this concrete, consider the sample domain ontology in Fig. 4, describing disjoint career paths for engineers and managers (as discussed in our example). Returning to mapping $\mu_1$, we can capture the intended meaning in the example as follows:

\[
\begin{align*}
\Gamma_{q_S}(n) &= \text{Manager} \\
\Gamma_{q_S}(c) &= \text{Manager} \\
\Gamma_{q_S}(s) &= \text{Project} \\
\Gamma_{q_T}(n) &= \text{Leader} \\
\Gamma_{q_T}(c) &= \text{Leader} \\
\Gamma_{q_T}(s) &= \text{Leader}
\end{align*}
\]
B. Mapping Verification

Once with \( \Gamma_{q_e}(x) \) and \( \Gamma_{q_T}(x) \) for a given ST-TGD \( \sigma \), we build a verification statement to check whether \( \sigma \) uses \( x \) in a way that is consistent with the semantics of \( S, T \). Because \( \sigma \) moves data from \( S \) into \( T \), the corresponding verification statement \( u(\sigma, x) = \Gamma_{q_e}(x) \sqsubseteq \Gamma_{q_T}(x) \). We say that \( \sigma \) uses \( x \) in a consistent way if \( u \) is compatible with the domain ontology \( O \). To determine this, we check whether \( O \models u(\sigma, x) \) is entailed.

The verification of a mapping \( \mu = \{ \sigma_1, \ldots, \sigma_n \} \) amounts to checking, for every variable \( v_i \), in every \( \sigma_j \) whether

\[
O \models u(\sigma_j, v_j).
\]

To see how this method would detect that \( \mu_1 \) (Recall Fig. 2(b)) is incompatible with \( T \), observe that it would generate the following verification statements:

\[
\text{for variable } n : \quad \text{Manager} \sqsubseteq \text{Leader} \quad (1)
\]

\[
\text{for variable } c : \quad \text{Manager} \sqsubseteq \text{Leader} \quad (2)
\]

\[
\text{for variable } s : \quad \text{Project} \sqsubseteq \text{Project} \quad (3)
\]

Because only statement (3) above is entailed by the ontology in Figure 4, \( \mu_1 \) is deemed incompatible.

IV. Obtaining Semantic Annotations

The framework discussed in the previous section provides a fully automatic way of verifying schema mappings given semantic annotations for the variables in each ST-TGD in each mapping involved. The natural question then is where such annotations come from? Clearly, while such annotations can be provided manually by the design experts, the practicality of the solution hinges on minimizing the intervention by the expert in producing such annotations. We discuss this problem next.

Closely related to our notion of semantic annotations, two prominent proposals have been suggested for defining and exploiting alignments between relational schemas and ontologies to help in two complimentary aspects of data integration. An et al. [12] define LAV-like schema alignments to derive more detailed “logical entities” (which are the building blocks for semantic mappings in Clio), resulting in better schema mappings [13] (in the sense that users in a case study preferred the mappings produced by their tool as opposed to those produced by Clio). Conversely, Calvanese et al. [14] define fact-generating alignments to semantically integrate data, by deriving facts from the data in the database into a knowledge base, where all the integration really happens.

LAV-like alignments, as the name suggests, treat the database schema as a local view of the ontology, aligning each relation in the schema separately. In effect, this approach assigns concepts to individual columns in the schema. Fact-generating alignments employ a GLAV style, and are strictly more expressive: they align views of the relational schema into views of the ontology (effectively allowing partial alignments). In our previous work [15], we detailed how to extract semantic annotations from LAV-like alignments, which we briefly revisit next. Then we discuss how to deal with fact-generating alignments.

A. Deriving Annotations from LAV-like alignments

Fig. 5 shows LAV-like alignments for our example. Observe that, according to Fig. 5, S.Engineer stores all engineers in the company (including those who do not lead a project), whereas S.Project keeps only those engineers who are women as well as technical leaders of at least 3 projects (recall the cardinality constraint for SeniorLeader in Fig. 4). In [15] we detail an algorithm that assigns concepts into columns of tables in the database by directly parsing LAV-like alignments. In our example, such assignment would look like the following:

\[
\begin{align*}
\alpha_{T.\text{Program}} &= \{1 \to \text{Leader}, 2 \to \text{Leader}, \\
& \qquad \quad 3 \to \text{Project} \} \\
\alpha_{S.\text{Engineer}} &= \{1 \to \text{Engineer}, 2 \to \text{Engineer}, \\
& \qquad \quad 3 \to \text{Engineer} \} \\
\alpha_{S.\text{Manager}} &= \{1 \to \text{Manager}, 2 \to \text{Manager}, \\
& \qquad \quad 3 \to \text{Manager} \} \\
\alpha_{S.\text{Project}} &= \{1 \to \text{Project}, 2 \to \text{Manager}, \\
& \qquad \quad 3 \to \text{SeniorLeader} \cap \text{Woman} \}
\end{align*}
\]

Given a query \( q \) and a variable \( x \), we derive \( \Gamma_{q}(x) \) from a set of LAV-like alignments as follows. Let \( R.c \) be a column from relation \( R \) in the database schema, such that \( R \) is used as a subgoal in \( q \) and \( x \) appears in the same position as \( c \). Let \( C_i \) be the concept assigned to column \( R.c \) by the LAV-like annotations. (To compute this, we must reason about keys in the relation: the concept assigned to column \( c \) is the conjunction of that indicated in the alignment and those concepts assigned to all columns that functionally determine \( c \).) Then, let \( C_1, \ldots, C_k \) be the concepts associated with all occurrences of \( x \) in \( q \); we define \( \Gamma_{q}(x) = C_1 \sqcap \ldots \sqcap C_k \).

\[
\begin{align*}
T.\text{Program}(x, y, z) &\sim \exists l, p : O.\text{Leader}(l), O.\text{Project}(p), \\
& O.\text{leads}(l, p), O.\text{hasName}(l, x), \\
& O.\text{hasEmail}(l, y), \\
& O.\text{hasLocation}(p, z) \\
& (a) \text{ Target schema.}
\end{align*}
\]

\[
\begin{align*}
S.\text{Engineer}(x, y, z) &\sim O.\text{Engineer}(x), O.\text{Name}(x, y), \\
& O.\text{hasEmail}(x, z) \\
S.\text{Manager}(x, y, z) &\sim O.\text{Manager}(x), O.\text{Name}(x, y), \\
& O.\text{hasEmail}(x, z) \\
S.\text{Project}(x, y, z) &\sim \exists p, l, m : O.\text{Project}(p), \\
& O.\text{hasLocation}(p, x), \\
& O.\text{SeniorLeader}(l), O.\text{Woman}(l), \\
& O.\text{leaders}(l, p), O.\text{Manager}(m), \\
& O.\text{manages}(m, p) \\
& (b) \text{ Source schema.}
\end{align*}
\]

Fig. 5: LAV-like alignments of schemas in Fig. 2 into the ontology of Fig. 4.
\[ T.\text{Program}(x, y, z) \sim \exists l, p \Omega : \text{Leader}(l), \Omega : \text{Project}(p), \]
\[ \Omega : \text{leads}(l, p), \Omega : \text{Name}(l, x), \]
\[ \Omega : \text{hasEmail}(l, y), \]
\[ \Omega : \text{hasLocation}(p, z) \]
\]

(a) Target schema.

\[ S.\text{Engineer}(x, y, z) \sim \Omega : \text{Engineer}(x), \Omega : \text{Name}(x, y), \]
\[ \Omega : \text{hasEmail}(x, z) \]
\]

\[ S.\text{Manager}(x, y, z) \sim \Omega : \text{Manager}(x), \Omega : \text{Name}(x, y), \]
\[ \Omega : \text{hasEmail}(x, z) \]
\]

\[ S.\text{Project}(x, y, z), x = \text{Edmonton'} \sim \exists p, l, m \Omega : \text{Project}(p), \]
\[ \Omega : \text{hasLocation}(p, x), \]
\[ \Omega : \text{SeniorLeader}(l), \Omega : \text{Woman}(l), \]
\[ \Omega : \text{leaves}(l, p), \Omega : \text{Manager}(m), \]
\[ \Omega : \text{manages}(m, p) \]
\]

(b) Source schema.

Fig. 6: Fact-generating alignments of schemas in Fig. 2 into the ontology of Fig. 4.

Returning to our example, the verification statements for mapping \( \mu_2 \) (which is valid) in Fig. 2(b) that would be inferred from the LAV-like alignment in Fig. 5 are:

- for variable \( n : \text{Engineer} \sqcap \text{SeniorLeader} \sqcap \text{Woman} \sqsubseteq \text{Leader} \)
- for variable \( c : \text{Engineer} \sqcap \text{SeniorLeader} \sqcap \text{Woman} \sqsubseteq \text{Leader} \)
- for variable \( s : \text{Project} \sqsubseteq \text{Project} \)

In \( \mu_2 \), variable \( i \) defines a join between \( S.\text{Engineer} \) and \( S.\text{Project} \) and is also the primary key of \( S.\text{Engineer} \). Thus, \( \Gamma_{\mu_2}(i) = \text{Engineer} \sqcap \text{Leader} \). Because \( i \) is assigned to the column corresponding to the primary key of \( S.\text{Engineer} \), \( \Gamma_{\mu_2}(i) \) is “added” (by means of a logical conjunction) to all other variables that appear in the goal corresponding to \( S.\text{Engineer} \) (namely \( n, c \)).

B. Deriving Annotations from fact-generating alignments

As mentioned above, fact-generating alignments are more expressive than LAV-like alignments as they map queries over the source schema into queries over the ontology. Thus, in effect, this approach allows alignments of views over the relational schema into views over the domain ontology. This added expressibility leads to much finer-grained control over the semantics of the schema mappings but comes with an added cost as well. To illustrate the trade-off, consider the fact-generating alignments in Fig. 6, which carry essentially the same semantics as those in Fig. 5 except that they refine the semantics of \( S.\text{Project} \); those projects hosted in Edmonton are led by my senior engineers who are also women, whereas projects in other locations are led by regular engineers.

The machinery for finding semantic annotations discussed above and in our previous work does not apply directly to fact-finding alignments. Consider again mapping \( \mu_2 \) in Fig. 2(b). Because it does not restrict the locations of the projects that it maps, we cannot use either alignment rule in Fig. 6 pertaining to \( S.\text{Project} \) in isolation. Instead, we need to combine both when identifying the semantic annotations, leading to these interpretations:

\[
\Gamma_{\mu_2}(n) = \text{Engineer} \sqcap ((\text{SeniorLeader} \sqcap \text{Woman}) \sqcap \text{Leader})
\]
\[
\Gamma_{\mu_2}(c) = \text{Engineer} \sqcap ((\text{SeniorLeader} \sqcap \text{Woman}) \sqcap \text{Leader})
\]
\[
\Gamma_{\mu_2}(s) = \text{Project}
\]
\[
\Gamma_{\mu_2}(l) = \text{Leader}
\]
\[
\Gamma_{\mu_2}(c) = \text{Leader}
\]
\[
\Gamma_{\mu_2}(s) = \text{Leader}
\]

That is, given the alignments, \( \mu_2 \) maps either senior female engineers or engineers who lead at least one project into the supervisor role in the target. (In passing, the verification statements that result from the annotations above are all entailed by the domain ontology in this case.)

To compute \( \Gamma_q(x) \) w.r.t. fact-generating alignments given as a set of rules \( r_1, \ldots, r_m \), we proceed as follows. For each sub-goal \( g \) of \( q \), we find first a minimal relevant set of rules \( r_j, \ldots, r_k \) among those in the alignment. A set of rules is relevant if the union of their LHS is equivalent to the goal \( g \) (in the query containment sense [16]). Once such set is found, we apply the procedure for the LAV-style alignments using the RHS of each such rule, yielding the set of concepts \( \Gamma_q^{j_1}(x), \ldots, \Gamma_q^{j_k}(x) \). Finally, we define

\[
\Gamma_q(x) = \Gamma_q^{j_1}(x) \sqcup \ldots \sqcup \Gamma_q^{j_k}(x).
\]

V. Conclusion

We have described a way of automating the verification of schema mappings based on checking whether each \( ST-TGD \) in such a mapping maps data in a way that is consistent with semantic annotations for both the source and target schemas. Our solution avoids serious problems in the state-of-the-art, namely the need for real (and potentially large) sample instances, and the reliance on sampling and strong statistical assumptions. Our solution can be implemented today, as all of its necessary components are readily available: our approach does not rely on any particular reasoner nor formalism, and there are many practical reasoners at the time of writing that deal with fairly expressive description logics.

Our approach is centred on the premise that the domain knowledge of the experts will be represented formally in a domain ontology. One immediate question is where such domain ontologies will come from. We believe that, as the semantic web effort progresses, more and better infrastructure will become available, facilitating the development and
improvement of such domain ontologies. Relying on formal knowledge bases has another advantage besides re-using tools: it allows for the direct re-use of expert knowledge. As the experts find errors in their mappings, they may improve the domain ontology, the alignments, or both. Moreover, it should be noted that in our approach, the source and the target schemas are aligned into the domain ontology independently of each other. Intuitively, this is also an improvement over the state-of-the-art: it should be much easier for an expert to align a schema into a descriptive domain ontology than to judge the correctness of a complex schema mapping between disparate systems.

Another potential source of schema alignments are modern software engineering tools, in which semantic models of the systems are produced (in rich languages such as UML) and from which relational schemas are automatically derived. Considerable efforts have been made into incorporating semantics into such software engineering languages and tools [17]. Another encouraging development for our framework is that considerable progress has been made in providing effective semi-automatic reverse-engineering tools for discovering schema alignments and expressing them formally [12]. We believe that the use of formal reasoning has great potential benefits for the development of schema mappings. For instance, reasoning is based on finding proofs of correctness, which can serve as explanations for failure in the case when mappings are deemed incompatible. We conceive such explanations being used in future mapping design tools to help the experts fix incorrect mappings. Second, such formal proofs are structures (usually graphs) that can be reasoned upon as well. We believe that it should be possible to derive fitness scores from such formal proofs to allow the ranking of mappings (i.e., determining which among consistent mappings is preferable over the others).

A. Future Work

Our work is very much ongoing, and we believe there are many challenges to be overcome if our approach is to become part of real mapping verification systems. We conclude by discussing some of them.

One immediate challenge is the cost of formal reasoning. As we indicate in [15], while there are practical reasonsers capable of tackling real problems, it remains to be seen the kinds of practical limitations imposed by reasoning and how to overcome them in the context of designing schema mappings. We believe that novel hybrid reasoning methods that rely on input from experts might hold the key for success here.

Another difficulty in implementing our framework is that it requires a large-scale validation, which is very difficult to perform and verify independently as there are no defined benchmarks for large-scale, critical data integration settings. Moreover, it is easy to see that the benefits of our framework will increase over time, as more and more mappings are designed, verified, and improved. This will lead to better domain ontologies, and better alignments. Thus, we envision several challenges in properly validating our approach.

A final, and more technical, challenge lies on the use of fact-generating alignments for obtaining annotations. On one hand, such alignments are attractive for being highly expressive and flexible for the designer (who can focus on specific subsets of the schema at a time). However, our current solution requires that we find a minimal subset of the alignment rules that is equivalent to a given sub-goal in an ST-TGD of a mapping.

It is easy to see that this boils down to solving the query containment problem for sets of queries, which is expensive. Furthermore, it is plausible that the alignments are such that no subset fully contains a given sub-goal in a query; leading to potentially expensive optimization problems. We believe that further research is needed to determine the right balance between expressiveness and performance here.

Acknowledgements

This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and the Alberta Ingenuity Fund.

References