Generalization of Schema Mappings for Transformation Reuse

Paolo Atzeni
Università Roma Tre
Italy
atzeni@dia.uniroma3.it

Luigi Bellomarini
Banca d'Italia
Italy
luigibellomarini@bancaditalia.it

Paolo Papotti
EURECOM
France
papotti@eurecom.fr

Riccardo Torlone
Università Roma Tre
Italy
torlone@dia.uniroma3.it

ABSTRACT

The manual specification of transformations between heterogeneous schemas is a key activity in data integration. While there are tools exposing correspondences between elements of schemas, precise schema mappings still need to be manually specified every time, even if the two schemas at hand are similar to others that have already been mapped. In fact, schema mappings are defined over the properties of the original schemas and cannot be *reused* on new ones. We tackle the transformation reuse problem by generalizing schema mappings as *meta-mappings*, a novel formalism describing transformations between generic data structures called *meta-schemas*. We introduce techniques to infer meta-mappings from schema mappings. Once inferred, we organize meta-mappings into a repository, which can be efficiently searched to identify reusable transformations when a new pair of schemas is given. We report effectiveness results from an experimental evaluation over real-world scenarios and show that our system can infer, store, and search millions of meta-mappings in seconds.

1 INTRODUCTION

Schema mappings are widely used as a tool for data exchange and integration. However, although there are systems supporting data architects in the creation of mappings [5], designing them is still a time-consuming task. In this framework, given the overwhelming amount of "enterprise knowledge" stored in traditional data warehouses and in data lakes, reuse is an opportunity of increasing importance [1]. In particular, data transformation scenarios are often defined over schemas that are different in structure but similar in semantics. This is especially true if data sources, which are extremely heterogeneous, have to be mapped to a shared format. It follows that a great opportunity to reduce the effort of transformation design is to *reuse existing schema mappings*. Unfortunately, there is no obvious approach for this problem. Consider the following example.

Example 1.1. A central bank maintains a register with balance data from all companies in the country (Figure 1). This register has schema G, with a relation `Balance` storing, for each company, its gains, zone of operation, and economic sector. External providers send data to the bank in different forms. Provider A adopts a schema S_A, with a relation R_A for companies (firms), with gains, zone of operation, and economic sector, whose code refers to relation `Activity`. Provider B adopts a schema S_B, with a relation R_B for companies (enterprises), their gains, sector, capital, and area, whose code refers

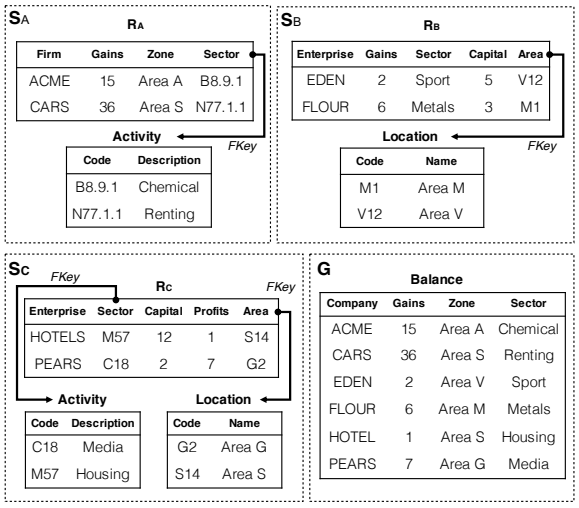

Figure 1: A data transformation scenario

to relation `Location`. Data is moved from S_A and S_B into G, by using two schema mappings:

σ_A: $R_A(f, g, z, s)$, Activity$(s, d) \rightarrow$ Balance(f, g, z, d).

σ_B: $R_B(e, g, s, c, a)$, Location$(a, n) \rightarrow$ Balance(e, g, n, s).

The example shows a *data exchange scenario* where the differences in the mappings are due to the structural differences between S_A and S_B, which are, on the other hand, semantically very similar. Moreover, every new data provider (e.g., S_C in the figure) would require the manual design of a new, ad-hoc mapping, even if there is a clear analogy with the already defined mappings.

Our goal is to reuse σ_A and σ_B and avoid the definition of a new mapping for S_C. The intuition is to collect all available mappings in a repository; then, for any new pair of schemas (e.g., S_C and G in the figure), query such repository to retrieve a suitable mapping. Unfortunately, this form of direct reuse is complicated by the nature of schema mappings. A mapping characterizes the constraint between a pair of schemas at a level of detail that enables both logical reasoning and efficient execution. Yet, a simple variation in a schema, such as a different relation or attribute name or a different number of attributes, makes it not applicable. Our experiments show that mappings from a corpus of 1.000 schema mappings can be reused for new, unmapped pairs of schemas only

in 20% of cases. To be reusable, a mapping should be described in a way that is independent of its specificities but, at the same time, harnesses the essence of the constraint so as to work for similar schemas.

Example 1.2. Consider a "generic" mapping Σ_A, obtained from σ_A by replacing names of the relations and attributes with variables. It could be informally described as follows:

Σ_A: *for each* relation r *with* key f and attributes g, a, s
for each relation r' *with* key s and attribute d
with a foreign key constraint from s of r to s of r'
there exists a relation r'' *with* key f and attributes g, a, d.

If instantiated on S_A, the generic mapping Σ_A expresses a mapping to G that is the same as σ_A. This solution seems a valid compromise between precision, i.e., the ability to express the semantics of the original mapping, and generality, as it can be applicable over different schemas. However, Σ_A falls short of the latter requirement, as it is not applicable on S_B. Indeed, there are no constraints on attribute g and a, and so they could be bound to any of `Gains`, `Sector` and `Capital`, incorrectly trying to map `Capital` into the target.

Example 1.3. Consider now a more elaborated generic mapping that uses constants to identify attributes:

Σ_B^H: *for each* relation r *with* key e and attributes g, s, c, a
for each relation r' *with* key a and attribute d
with a foreign key constraint from a of r to a of r'
where $g = Gains,\ s \neq Gains,\ s \neq Capital,\ c \neq Gains,$
$c \neq Sector$
there exists a relation r'' *with* key e and attributes g, d, s.

This generic mapping is precise enough to correctly describe both σ_A and σ_B and can be re-used with other schemas.

The example shows a combination of attributes, identified by constraints on their names and role, that form a correct and useful generic mapping. Once pinpointed, generic mappings can be stored in a repository, so that it is possible to use them for a new scenario. In our example, given S_C and G, the generic mapping Σ_B^H can be retrieved from the repository and immediately applied.

There are three main challenges in this approach.

- We need a clear characterization of what it means for a generic mapping to correctly describe and capture the semantics of an original schema mapping.
- As a generic mapping is characterized by a combination of conditions on attribute names and roles, for a given schema mapping there is a combinatorial number of generic mappings. We need a mechanism to generate them.
- For a new scenario (e.g., new schemas), there is an overwhelming number of generic mappings that potentially apply, with different levels of "suitability". We need efficient tools to search through and choose among them.

In this work, we address the above challenges with GAIA, a system for mapping reuse. GAIA supports two tasks, as shown in Figure 2: (1) infer generic mappings, called *meta-mappings*, from input schema mappings, and store them in a repository; (2) given a source and a target schema, return a ranked list of meta-mappings from the repository which are used to generate possible mappings between these schemas. GAIA provides the following key contributions:

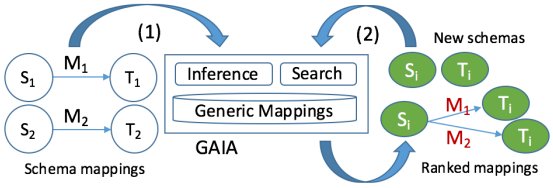

Figure 2: The architecture of GAIA.

- The notion of *fitness*: a semantics to precisely characterize and check when a meta-mapping is suitable for a reuse scenario.
- An algorithm to *infer* meta-mappings from schema mappings with an approach that extends previous efforts for the definition of schema mappings by example; this algorithm is used to populate a repository of meta-mappings supporting schema mapping reuse.
- An approach to reuse based on: (i) the search, in the repository of available meta-mappings, for those that *fit* a new pair of source and target schemas and (ii) the construction, from the retrieved meta-mappings, of possible mappings to be proposed to the designer.

Because of space limitation, algorithms and details are in the full version of the paper [2]. In the rest of this paper, we provide examples of meta-mappings (Section 2) and experimental results from an evaluation of our system with more than 20,000 real-world data transformations over 40,000 schemas (Section 3).

2 MAPPING AND META-MAPPINGS

We recall the notion of schema mapping [6] and introduce that of *meta-mapping*. While the former notion models specific transformations, the latter introduces an abstraction over mappings [7, 8] and models *generic* mappings between schemas. Building on these notions, we illustrate the functionalities of our system.

Schema mappings. Let **S** (the source) and **T** (the target) be two relational schemas and let $Inst(\mathbf{S})$ and $Inst(\mathbf{T})$ denote the set of all possible instances of **S** and **T**, respectively. A *(schema) mapping* M for **S** and **T** is a binary relation over their instances, that is, $M \subseteq Inst(\mathbf{S}) \times Inst(\mathbf{T})$ [3].

Without loss of generality, we consider mappings expressed by a single *source-to-target tuple-generating-dependency* (st-tgd) σ: $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ where $\mathbf{x}$ and $\mathbf{y}$ are two disjoint sets of variables, $\phi(\mathbf{x})$ (the left-hand-side, LHS) is a conjunction of atoms involving relations in **S** and $\psi(\mathbf{x}, \mathbf{y})$ (the right-hand-side, RHS) is a conjunction of atoms involving relations in **T**. The dependency represents a mapping M in the sense that $(I, J) \in M$ if and only if (I, J) satisfies σ. In this case, J is called a *solution* of I under σ. We can compute a suitable J in polynomial time by applying the *chase procedure* to I using σ [6]: the result may have labeled nulls denoting unknown values and is called the *universal* solution, since it has a homomorphism to any possible solution J', that is, a mapping h of the nulls into constants and nulls such that $h(J) \subseteq J'$.

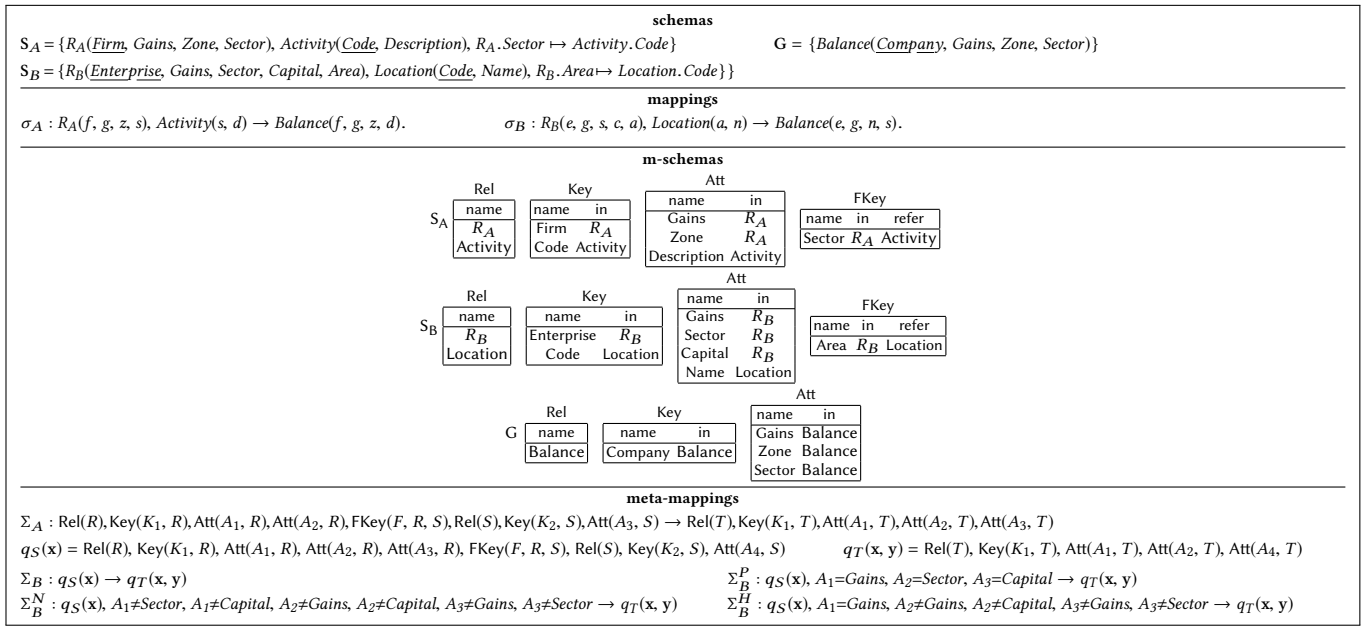

Figure 3: Schemas, m-schemas, mappings, and meta-mappings discussed along the paper.

Example 2.1. Consider the schemas $\mathbf{S}_A$, $\mathbf{S}_B$ and $\mathbf{G}$ of Figure 1, which we recall in Figure 3 with all the formalisms that will be discussed throughout the paper.

A mapping between $\mathbf{S}_A$ and $\mathbf{G}$ is the st-tgd σ_A discussed in the Introduction and reported in Figure 3 (quantifiers are omitted for the sake of readability). Intuitively, the application of the chase to the instance of $\mathbf{S}_A$ using σ_A enforces this dependency by generating one tuple in the target for each pair of tuples in the source for which there is a binding to the LHS of the dependency. The result includes the first two tuples in relation `Balance` in Figure 1. Besides, a mapping from $\mathbf{S}_B$ to $\mathbf{G}$ is represented by the s-t tgd σ_B in Figure 3.

Meta-mappings. A *meta-mapping* describes generic mappings between relational schemas and is defined as a mapping over the catalog of a relational database [8]. Specifically, in a relational meta-mapping, source and target are both defined over the following schema, called *(relational) dictionary*: Rel(*name*), Att(*name*, *in*), Key(*name*, *in*), FKey(*name*, *in*, *refer*) (for the sake of simplicity, we consider here a simplified version of the relational model). An instance S of the dictionary is called *m-schema* and describes relations, attributes and constraints of a (standard) relational schema S. Figure 3 shows the m-schemas of the schemas $\mathbf{S}_A$, $\mathbf{S}_B$, and $\mathbf{G}$ of the running example.

We assume, hereinafter, that, given a schema, its corresponding m-schema is also given, and vice versa. A meta-mapping is expressed by means of an st-tgd over dictionaries that describes how the elements of a source m-schema map to the elements of a target m-schema.

Example 2.2. Mapping σ_A of Example 2.1 can be expressed, at the dictionary level, by meta-mapping Σ_A in Figure 3. This st-tgd describes a generic transformation that takes two source relations R and S linked by a foreign key F and generates a target relation T obtained by joining R and S on F that includes: the key K_1 and the attributes A_1 and A_2 from relation R and the attribute A_3 from S.

Given a source m-schema S and a meta-mapping $\mathscr{M}$, a target m-schema T is generated by applying the chase procedure to S using $\mathscr{M}$.

Example 2.3. The chase of $\mathbf{S}_A$ using Σ_A, both in Figure 3, generates the following target m-schema where $\perp_R$ is a labelled null denoting a relation name.

Rel: name | $\perp_R$
Key: name in | *Firm* $\perp_R$
Att: name in | *Gains* $\perp_R$ | *Zone* $\perp_R$ | *Description* $\perp_R$

This m-schema describes the relational schema:
R(*Firm*, *Gains*, *Zone*, *Description*)

A meta-mapping operates at schema level rather than at data level and thus provides a means for describing generic transformations. Subtleties could arise from the chase procedure in presence of existential quantifications in meta-mappings producing duplications of relations in the result. This is avoided by assuming that, for each existentially quantified variable, there is a target *equality generating dependency* (egd) [6] ensuring that whenever two relations in the target have the same structure, then they coincide.

From meta-mappings to mappings. Given a source schema $\mathbf{S}$ and a meta-mapping Σ, it is possible not only to generate a target schema by using the chase, as shown in Example 2.3, but also to automatically obtain a schema mapping σ that represents the *specialization* of Σ for $\mathbf{S}$ and $\mathbf{T}$ [8]. The *schema to data exchange transformation* (SD transformation) generates from $\mathbf{S}$ and Σ a complete schema mapping made of $\mathbf{S}$, a target schema $\mathbf{T}$ (obtained by chasing the m-schema of $\mathbf{S}$ with the meta-mapping), and an s-t tgd σ between $\mathbf{S}$ and $\mathbf{T}$. The correspondences between LHS and RHS of

σ are derived from the provenance information computed during the chase step, in the same fashion as the provenance computed over the source instance when chasing schema mappings [4].

Example 2.4. Consider again the scenario in Figure 3. If we apply the *SD* transformation to the schema $\mathbf{S}_A$ and the meta-mapping Σ_A, we obtain the target m-schema of Example 2.3 and the following mapping from $\mathbf{S}_A$ to $\perp_R$:

$$\sigma : R_A(f, g, z, s), \text{Activity}(s, d) \rightarrow \perp_R(f, g, z, d).$$

Thus, we get back, up to a renaming of the target relation, the mapping σ_A in Figure 3 from which Σ_A originates.

From mappings to meta-mappings While previous work focused on generating a mapping from a given meta-mapping [8], we tackle the more general problem of *mapping reuse*, which consists of: (i) generating a repository of meta-mappings from a set of user-defined schema mappings, and (ii) given a new pair of source and target schemas, generating a suitable mapping for them from the repository of meta-mappings.

Example 2.5. For the scenario in Figure 3, our system first generates several fitting meta-mappings from σ_A between $\mathbf{S}_A$ and $\mathbf{G}$. Once $\mathbf{S}_B$ and $\mathbf{G}$ are given as input for a new transformation, the system scans the corpus of existing meta-mappings and identifies Σ_A as a fitting meta-mapping from $\mathbf{S}_B$ to $\mathbf{G}$. This meta-mapping is then instantiated to generate a schema mapping between them, according to the SD transformation above.

3 EXPERIMENTAL EVALUATION

We implemented GAIA in PL/SQL 11.2 for Oracle 11g. All experiments were conducted on Oracle Linux, with an Intel Core i7@2.60GHz, 16GB RAM.

Datasets and Transformations. We used data transformations that are periodically executed to store data coming from several data sources into the *Central National Balance Sheet* (CNBS) database, an archive of financial information of about 40K enterprises. The schema of CNBS has 5 relations with roughly 800 attributes in total. Source data come from three different providers. The Chamber of Commerce provides data for 20K companies (datasets **Chamber**). While the schemas of these datasets are similar, the differences require one mapping per company between relations of up to 30 attributes with an average of 32 (self) joins in the LHS of the mappings. Data for 20K more companies is collected by a commercial data provider in a single database (**CDP**). This database is different both in structure and attributes names from the schemas in **Chamber** and requires one mapping involving 15 joins in its LHS. Finally, data for further 1,300 companies (**Stock**) is imported from the stock exchange, with each company requiring a mapping with 40 joins on average.

Transformation scenarios. For the inference of the meta-mappings, we consider two configurations: *single*, where a meta-mapping is inferred from one mapping, and *multiple*, where the inference is computed on a group of mappings for companies in the same business sector. We observe that the results stabilize with 10 schema mappings in the group and do not improve significantly with larger numbers, therefore consider 10 mappings for the *multiple* configuration.

Search precision. Let σ be a mapping from a source $\mathbf{S}$ to a target $\mathbf{T}$ and let $\mathscr{Q}$ be the set of meta-mappings inferred from σ. We measure, in terms of *search precision*, the ability of the system to return the meta-mappings in $\mathscr{Q}$ when queried with $\mathbf{S}$ and $\mathbf{T}$, i.e., how well the system retrieves correct cases.

We use a *resubstitution* approach: we populate an initially empty repository by inferring the meta-mappings from a set Θ of mappings. For each schema mapping σ in Θ, we will denote by $\mathscr{Q}_\sigma$ all the meta-mappings inferred from σ. We then pick a schema mapping σ' from Θ and search the repository by providing as input the source and target schemas of σ'. We then collect the top-10 results according to the coverage and compute the percentage of correctly retrieved meta-mapping, that is, those that belong to $\mathscr{Q}_{\sigma'}$.

We test search precision with a corpus of mappings of increasing size (from 200 to 21,301). In each test, we query the repository of meta-mappings inferred from the given mappings by using 50 different source-target pairs and report their average search precision. The experiment is executed on single and multiple configurations. The largest repository contains about 700K explicit meta-mappings in single configuration and 110K in multiple configuration. In the multiple scenario, the test is considered successful on a mapping σ when the retrieved meta-mapping originates from the group that includes σ. On average, a meta-mapping includes 12 equality and/or inequality constraints. For this test, the inference for all meta-mappings took less than a minute and the search took always less than 5 seconds.

The results shows that search precision is equal to 1 with up to 5k mappings in the repository and it slightly decreases with the repository size up to 0.98 with more than 20k mappings in the single scenario and up to 0.995 with the multiple scenario. This is because the larger numbers of mappings lead to an increasing number of false positives in the result. When the **CDP** mapping is inserted in the repository, it is immediately identified with both configurations. This shows that specific structures and constants lead to meta-mappings that are easy to retrieve. Meta-mappings derived from multiple schema mappings improve search precision because are more general than the ones from single mappings, thus reducing the risk of overfitting.

Experiments show that our approach efficiently identifies useful mappings for a new scenario, while the conventional approach would require a user to completely define a new transformation.

REFERENCES

[1] D. Abadi et al. The Beckman report on database research. *Commun. ACM*, 59(2):92–99, Jan. 2016.
[2] P. Atzeni, L. Bellomarini, P. Papotti, and R. Torlone. Meta-mappings for schema mapping reuse. *PVLDB*, 12(5):557–569, 2019.
[3] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, 2007.
[4] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.
[5] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling*, 2009.
[6] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.
[7] M. A. Hernández, P. Papotti, and W. C. Tan. Data exchange with data-metadata translations. *PVLDB*, 1(1):260–273, 2008.
[8] P. Papotti and R. Torlone. Schema exchange: Generic mappings for transforming data and metadata. *Data Knowl. Eng.*, 68(7):665–682, 2009.