Discovery and Correctness of Schema Mapping Transformations

Angela Bonifati, Giansalvatore Mecca, Paolo Papotti, and Yannis Velegrakis

Abstract Schema mapping is becoming pervasive in all data transformation, exchange and integration tasks. It brings to the surface the problem of differences and mismatches between heterogeneous formats and models, respectively used in source and target databases to be mapped one to another. In this chapter, we start by describing the problem of schema mapping, its background and technical implications. Then, we outline the early schema mapping systems, along with the new generation of schema mapping tools. Moving from the former to the latter entailed a dramatic change in the performance of mapping generation algorithms. Finally, we conclude the chapter by revisiting the query answering techniques allowed by the mappings, and by discussing useful applications and future and current developments of schema mapping tools.

1 Introduction

There are currently many kinds of scenarios in which heterogeneous systems need to exchange, transform, and integrate data. These include ETL ("Extract, Transform and Load") applications, object-relational mapping systems, EII ("Enterprise Information Integration") systems, and EAI ("Enterprise Application Integration") frameworks.

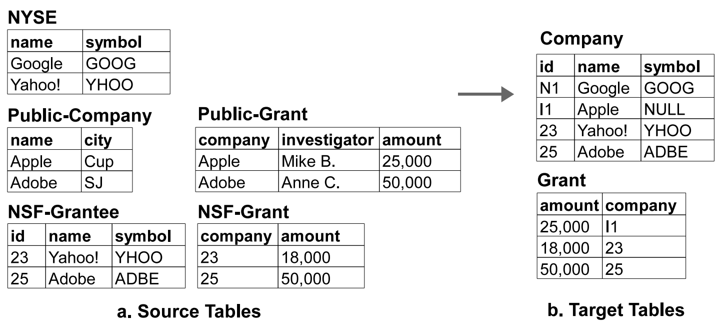
Angela Bonifati
ICAR-CNR – Italy, e-mail: bonifati@icar.cnr.it

Giansalvatore Mecca
Università of Basilicata – Italy e-mail: giansalvatore.mecca@unibas.it

Paolo Papotti
Università Roma Tre – Italy e-mail: papotti@dia.uniroma3.it

Yannis Velegrakis
Università di Trento – Italy e-mail: velgias@disi.unitn.eu

A common feature of all of these applications is that data is organized according to different descriptions, typically based on a variety of data models and formats. To given one example, consider the scenario in Figure 1.



NYSE

name	symbol
Google	GOOG
Yahoo!	YHOO

Public-Company

name	city
Apple	Cup
Adobe	SJ

Public-Grant

company	investigator	amount
Apple	Mike B.	25,000
Adobe	Anne C.	50,000

NSF-Grantee

id	name	symbol
23	Yahoo!	YHOO
25	Adobe	ADBE

NSF-Grant

company	amount
23	18,000
25	50,000

a. Source Tables

Company

id	name	symbol
N1	Google	GOOG
I1	Apple	NULL
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
18,000	23
50,000	25

b. Target Tables

Fig. 1 Mapping Company Information

The inputs to the problem are three data sources about companies, potentially organized according to rather different formats and models: (a) a list of companies from the New York Stock Exchange, *(NYSE)*; (ii) a public database concerning companies and grants, (*Public-Companies*, *Public-Grants*); (iii) and database of grants from the National Scientific Foundation (*NSF-Grantee*, *NSF-Grant*). Notice that, for the purpose of this section, we shall assume that source data are relational tables. However, as it will be clear in the following, they might easily be organized according to more complex data models, for example as nested XML documents.

The expected output is an instance of a target database with the following schema: two tables, *Company* and *Grant*, a key constraint on the *Company.name* attribute, and a foreign-key constraint from *Grant.company* to *Company.id*. Assuming the source data are those in Figure 1.a, it is natural to expect that the target instance obtained by the translation process is the one in Figure 1.b. In fact, informally speaking, such instance has a number of desirable properties: (i) it is a legal instance for the target database; (ii) it is "complete", in the sense that it contains all of the information that is in the source tables; (iii) at the same time, it is "non-redundant", i.e., no piece of information is reported twice.

It can be seen from this example that computing an output solution requires the definition of some form of *mapping* from the source repository to the target repository. Generally speaking, *mappings*, also called *schema mappings*, are expressions that specify how an instance of the source repository should be translated into an instance of the target repository. In order to be useful in practical applications, they should have an executable implementation – for example, under the form of SQL queries for relational data, or XQuery scripts for XML.

There are many ways in which such a transformation can be implemented. Often, this is done in a rather procedural fashion, and developers are forced to write quite a lot of code in order to glue together the various sources. To give an example, in an ETL application a developer would be forced to manually construct a script made of potentially large number of simpler data-transformation steps. In other cases, such as commercial EII systems, transformation steps are often expressed using programming language (such as Java). This procedural style of specifying the mapping has made the problem of exchanging data across different repositories quite a burden, as discussed in (Haas, 2007).

In order to alleviate developers from this burden, we can identify two key requirements that a mapping system should have:

- a first key requirement is represented by *ease of use* and productivity. Developers should not be required to manually specify all of the details about the mapping; on the contrary, users would like to specify only a high-level, abstract and declarative representation of the mapping; then, based on this input, the mapping system should be able to generate the actual mappings, by working out the missing details. To support this process, mapping systems usually provide a graphical user interface using which developers may specify the mapping as a set of *value correspondences*, i.e., correspondences among schema elements. In our example, the input provided to the mapping system would be that shown in Figure 2;
- a second essential requirement is concerned with the generation of the target instances, i.e., with the *quality* and efficiency in the generation of solutions.

In this respect, database researchers have identified two main problems: (i) the first one is that of *schema mapping generation*, largely inspired by the seminal Clio papers (Miller et al, 2000; Popa et al, 2002); this is the problem of generating a set of mappings based on the correspondences provided as input by the user; (ii) the second one is that of solving the actual *data exchange problem*; originally formalized in (Fagin et al, 2005a), this consists in assigning a clear semantics to a given set of mappings, in order to turn them into executable queries on the source, and updates on the target that generate the desired target instance.

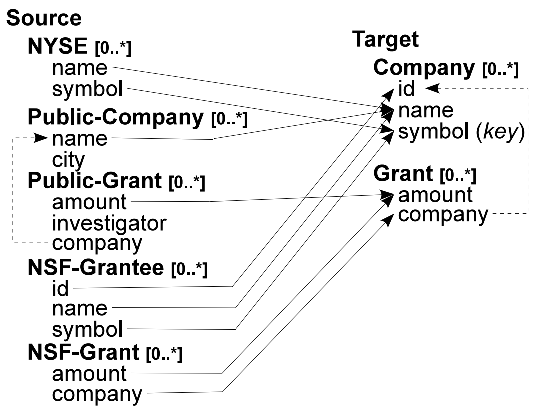


Fig. 2 An Abstract Specification of the Mapping as a Set of Correspondences (dashed arrows denote foreign-key constraints)

Another important application of schema mappings is query answering (Abiteboul and Duschka, 1998). In particular, given a fixed data exchange scenario, target query answering aims at computing the set of answers to a query posed on the target schema. In our example, this amounts to take a query initially expressed on the target tables in Figure 1.b, and to reformulate it according to the source tables in Figure 1.a.

In recent years, research on schema mappings, data exchange and query answering have provided quite a lot of building blocks towards this goal. Interestingly, the of bulk theoretical ideas for solving the data exchange problem were introduced several years after the first mapping generation techniques had been developed. The main motivation was that of providing a clear theoretical foundation for schema mappings, i.e., a solid formalism that systems could use to reason about mappings and their properties, to optimize them and to guarantee that data are exchanged in an optimal way.

In the following sections, we provide an overview of these contributions. More specifically:

- Section 2 provides an overview of data exchange theory, and more specifically of the notions of dependencies, mapping scenario, and solution;
- Section 3 introduces the seminal ideas about schema mapping generation, and the early algorithms developed in the framework of the Clio project (Miller et al, 2000; Popa et al, 2002);
- Section 4 describes the recent advancements in terms of schema mapping rewriting techniques that were introduced to improve the quality of solutions;
- Section 5 provides an overview of the complexity results and algorithms developed for query answering over schema mappings;
- Section 6 discusses a number of other interesting developments and applications of schema mapping techniques;
- finally, Section 7 concludes the chapter by discussing the open problems in this area.

2 Preliminaries

In order to provide a common formalism to be used across the chapter, we first introduce the data model we adopt as a reference. Data exchange was originally formalized for the relation model, so we focus on this data model. Nested sources will be discussed separately in the following sections.

In all of the data exchange theory, databases are considered as collections of relations on two distinct and disjoint domains: a set of *constants*, CONST, a set of *labeled nulls*, NULLS. *Labeled nulls* are used during the generation of solutions in order to “invent” new values in the target that do not appear in the source database. One way to generate labeled nulls through Skolem functions (Hull and Yoshikawa, 1990). A *Skolem function* is an injective function and can be used to produce unique identifiers. It takes one or more arguments and it has the property of producing a unique value for each different set of arguments.

This said, we can formalize the relational model as follows. We fix a set of *labels* $\{A_0, A_1 \ldots\}$, and a set of *relation symbols* $\{R_0, R_1, \ldots\}$. With each relation symbol R we associate a *relation schema* $R(A_1, \ldots, A_k)$. A *schema* $\mathbf{S} = \{R_1, \ldots, R_n\}$ is a collection of relation schemas. An *instance* of a relation schema $R(A_1, \ldots, A_k)$ is a finite set of tuples of the form $R(A_1 : v_1, \ldots, A_k : v_k)$, where, for each i, v_i is either a constant or a labeled null. An *instance* of a schema $\mathbf{S}$ is a collection of instances, one for each relation schema in $\mathbf{S}$. In the following, we will interchangeably use the positional and non positional notation for tuples and facts; also, with an abuse of notation, we will often blur the distinction between a relation symbol and the corresponding instance.

Dependencies and Mapping Scenarios Data exchange systems rely on *embedded dependencies* (Beeri and Vardi, 1984) in order to specify mappings. These dependencies are logical formulas of two forms: *tuple-generating dependencies (tgds)* or *equality-generating dependencies (egds)*; each of them has a precise role in the mapping. Informally speaking (the formal definition are reported below):

- *source-to-target tgds (s-t tgds)*, i.e., tgds that use source relations in the premise, and target relations in the conclusion, are used to specify which tuples should be present in the target based on the tuples that appear in the source; they represent the core of the mapping, since they state how to "move" data from the source to the target;
- *target tgds*, i.e., tgds the only use target symbols; these are typically used to specify foreign-key constraints on the target;
- *target egds*, in turn, are typically used to encode key constraints on the target database.

In our example, the desired mapping can be expressed using the following dependencies:

SOURCE-TO-TARGET TGDS

m_1. $\forall s, n : \mathit{NYSE}(s, n) \rightarrow \exists I : \mathit{Company}(I, n, s)$

m_2. $\forall n, c, a, pi : \mathit{Public\text{-}Company}(n, c) \wedge \mathit{Public\text{-}Grant}(a, pi, n) \rightarrow$
$\exists I, S : \mathit{Company}(I, n, S) \wedge \mathit{Grant}(a, I)$

m_3. $\forall i, n, s : \mathit{NSF\text{-}Grantee}(i, n, s) \rightarrow \mathit{Company}(i, n, s)$

m_4. $\forall a, c : \mathit{NSF\text{-}Grant}(a, c) \rightarrow \mathit{Grant}(a, c)$

TARGET TGDS

t_1. $\forall a, c : \mathit{Grant}(a, c) \rightarrow \exists N, S : \mathit{Company}(c, N, S)$

TARGET EGDS

e_1. $\forall n, n', i, i', s : \mathit{Company}(i, n, s) \wedge \mathit{Company}(i', n', s) \rightarrow (i = i') \wedge (n = n')$

Intuitively, each of the s-t tgds specifies how to map the organization of a portion of the source tables to that of a portion of the target tables. In particular, mapping m_1 copies company names and symbols in the *NYSE* source table to the *Company* table in the target. In doing this, the mapping requires that some value – represented by the I existentially quantified variable – is assigned to the *id* attribute of the *Company* table. The *Public* source contains two relations with companies names and grants that are assigned to them; these information are copied to the target tables by mapping

m_2; in this case, a value – again denoted by the I existentially quantified variable – must be "invented" in order to correlate a tuple in *Grant* with the corresponding tuple in *Company*. Finally, mappings m_3 and m_4 copy data in the *NSF* source tables to the corresponding target tables; note that in this case we don't need to invent any values.

The target tgd encode the foreign key on the target. The target egd simply states that *symbol* is key for *Company*.

To formalize, given two schemas, **S** and **T**, an *embedded dependency* (Beeri and Vardi, 1984) is a first-order formula of the form $\forall \bar{x}(\phi(\bar{x}) \rightarrow \exists \bar{y}(\psi(\bar{x},\bar{y})))$, where $\bar{x}$ and $\bar{y}$ are vectors of variables, $\phi(\bar{x})$ is a conjunction of atomic formulas such that all variables in $\bar{x}$ appear in it, and $\psi(\bar{x},\bar{y})$ is a conjunction of atomic formulas. $\phi(\bar{x})$ and $\psi(\bar{x},\bar{y})$ may contain equations of the form $v_i = v_j$, where v_i and v_j are variables.

An embedded dependency is a *tuple–generating dependency* if $\phi(\bar{x})$ and $\psi(\bar{x},\bar{y})$ only contain relational atoms. It is an *equality generating dependency (egd)* if $\psi(\bar{x},\bar{y})$ contains only equations. A tgd is called a *source-to-target tgd* if $\phi(\bar{x})$ is a formula over **S** and $\psi(\bar{x},\bar{y})$ over **T**. It is a *target tgd* if both $\phi(\bar{x})$ and $\psi(\bar{x},\bar{y})$ are formulas over **T**.

A *mapping scenario* (also called a *data exchange scenario* or a *schema mapping*) is a quadruple $\mathscr{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where **S** is a source schema, **T** is a target schema, Σ_{st} is a set of source-to-target tgds, and Σ_t is a set of target dependencies that may contain tgds and egds. If the set of target dependencies Σ_t is empty, we will use the notation $(\mathbf{S}, \mathbf{T}, \Sigma_{st})$.

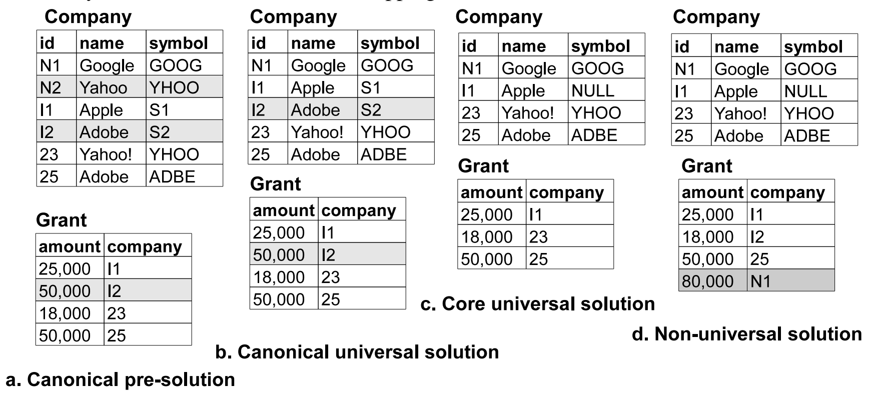
Solutions We can now introduce the notion of a *solution* for a mapping scenario. In order to do this, given two disjoint schemas, **S** and **T**, we shall denote by $\langle \mathbf{S}, \mathbf{T} \rangle$ the schema $\{S_1 \dots S_n, T_1 \dots T_m\}$. If I is an instance of **S** and J is an instance of **T**, then the pair $\langle I, J \rangle$ is an instance of $\langle \mathbf{S}, \mathbf{T} \rangle$.

A target instance J is a *solution* of $\mathscr{M}$ and a source instance I (denoted $J \in \mathsf{Sol}(\mathscr{M}, I)$) iff $\langle I, J \rangle \models \Sigma_{st} \cup \Sigma_t$, i.e., I and J together satisfy the dependencies.

Given a mapping scenario $\mathscr{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, with s-t and target dependencies, we find it useful to define a notion of a *pre-solution* for $\mathscr{M}$ and a source instance I as a solution over I for scenario $\mathscr{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, obtained from $\mathscr{M}$ by removing target constraints. In essence, a pre-solution is a solution for the s-t tgds only, and it does not necessarily enforce the target constraints.

Figure 3 shows several solutions for our example scenario on the source instance in Figure 1. In particular, solution **a** is a pre-solution, since it satisfies the s-t tgds but it does not comply with the key constraints and therefore it does not satisfy the egds. Solution **b** is a solution for both the s-t tgds and the egds. We want however to note that a given scenario may have multiple solutions on a given source instance. This is a consequence of the fact that each tgd only states an inclusion constraint, but it does not fully determine the content of the target. To give an example, beside solution **b** in Figure 3, also the two target instances **c** and **d** are solutions for the same source instance.

By looking at these solutions, we notice two things: (i) solution **c** is more compact than solution **b**; it can be seen that the grayed tuples in solution **b** are somehow "redundant", since they do not add any information to that contained in solution **c**;



Company

id	name	symbol
N1	Google	GOOG
N2	Yahoo	YHOO
I1	Apple	S1
I2	Adobe	S2
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
50,000	I2
18,000	23
50,000	25

a. Canonical pre-solution

Company

id	name	symbol
N1	Google	GOOG
I1	Apple	S1
I2	Adobe	S2
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
50,000	I2
18,000	23
50,000	25

b. Canonical universal solution

Company

id	name	symbol
N1	Google	GOOG
I1	Apple	NULL
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
18,000	23
50,000	25

c. Core universal solution

Company

id	name	symbol
N1	Google	GOOG
I1	Apple	NULL
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
18,000	I2
50,000	25
80,000	N1

d. Non-universal solution

Fig. 3 Several Solutions for the Companies Scenario

(ii) solution **d** contains a tuple (the one with a gray background) with a ground value $(80,000)$ that does not belong to the source instance. In essence, the space of solutions is quite various: on one side, solutions may have different sizes; intuitively, we prefer those of smaller size; on the other side, some of them may contain some "arbitrary" values, that do not really follow from the content of the source instance and from the constraints in $\Sigma_{st} \cup \Sigma_t$.

It is natural to state a couple of quality requirements for solutions to a mapping scenario:

- first, we would like to restrict our attention to those solutions – which we call *universal* – that only contain information that follows from I and $\Sigma_{st} \cup \Sigma_t$;
- among universal solutions, we would like to select the ones of the smallest size – called the *core universal solutions*.

To formalize these two notions, we introduce the notion of a *homomorphism* among solutions. Given two instances J, J' over a schema $\mathbf{T}$, a *homomorphism* $h : J \to J'$ is a mapping of the values of $dom(J)$ to the values in $dom(J')$ such that it maps each constant to itself, i.e., for each $c \in const()(J)$, $h(c) = c$, and it maps each tuple in J to a tuple in J', i.e, for each $t = R(A_1 : v_1, \ldots, A_k : v_k)$ in J it is the case that $h(t) = R(A_1 : h(v_1), \ldots, A_k : h(v_k))$ belongs to J'. h is called an *endomorphism* if $J' \subseteq J$; if $J' \subset J$ it is called a *proper endomorphism*.

In essence, a homomorphism is a constant-preserving mapping that can be used to turn one instance into a subset of another. Whenever a homomorphism h turns a tuple t of J into a tuple t' of J', we may be certain that t' contains at least "as much information as" t. Similarly, if h maps J into J', then J' contains at least as much information as J. If, on the contrary, there exists a tuple t in J that contains a constant (like $80,000$ in our example) that does not appear in J', i.e., if J contains some "extra" information that is not in J', then there cannot be any homomorphism of t into a tuple of J' and therefore no homomorphism of J itself into J'.

This allows us to formalize the notion of a *universal solution*. A solution J for $\mathscr{M}$ and source instance I is *universal* (Fagin et al, 2005a) (denoted $J \in \mathsf{USol}(\mathscr{M}, I)$)

iff for every other solution K for $\mathscr{M}$ and I there is an homomorphism from J to K. In the following, we shall only consider universal solutions.

Among these, we prefer those of minimal size. Given a scenario $\mathscr{M}$, and an instance I, a *core universal solution* (Fagin et al, 2005b) $J \in \mathsf{USol}(\mathscr{M},I)$, denoted $\mathbf{C} \in \mathsf{Core}(\mathscr{M},I)$, is a subinstance $\mathbf{C} \subseteq J$ such that there is a homomorphism from J to $\mathbf{C}$, but there is no homomorphism from J to a proper subinstance of $\mathbf{C}$. Cores of universal solutions are themselves universal solutions (Fagin et al, 2005b), and they are all isomorphic to each other. It is therefore possible to speak of *the core solution* as the "optimal" solution, in the sense that it is the universal solution of minimal size (Fagin et al, 2005b).

The Chase A natural question is how it is possible to derive universal solutions for a mapping scenario and a source instance. It turns out that this can be done by resorting to the classical *chase procedure* (Fagin et al, 2005a).

The chase works differently for tgds and egds. Given a vector of variables $\overline{v}$, an *assignment* for $\overline{v}$ is a mapping $a : \overline{v} \rightarrow \textsc{const} \cup \textsc{nulls}$ that associates with each universal variable a constant in $\textsc{const}$. Given a formula $\phi(\overline{x})$ with free variables $\overline{x}$, and an instance I, we write $I \models \phi(a(\overline{x}))$ whenever I *satisfies* the formula $\phi(a(\overline{x}))$, that is whenever I contains all the atoms in $\phi(a(\overline{x}))$.

Given instances I,J, during the *naive chase* (ten Cate et al, 2009)[1] a tgd $\phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x},\overline{y}))$ is fired for all assignments a such that $I \models \phi(a(\overline{x}))$; to fire the tgd, a is extended to $\overline{y}$ by injectively assigning to each $y_i \in \overline{y}$ a fresh null, and then adding the facts in $\psi(a(\overline{x}),a(\overline{y}))$ to J. Consider tgd m_2 in our example:

$$m_2.\ \forall n,c,a,pi,n: \mathit{Public\text{-}Company}(n,c) \wedge \mathit{Public\text{-}Grant}(a,pi,n) \rightarrow \\ \exists I,S: \mathit{Company}(I,n,S) \wedge \mathit{Grant}(a,I)$$

On source tuples *Public-Company(Adobe, SJ), Public-Grant(Adobe., Anne C., 50,-000)* it will generate two target tuples, $\mathit{Company}(N_1,\mathit{Adobe},N_2)$, and $\mathit{Grant}(50,000, N_1)$, where N_1,N_2 are fresh nulls.

A solution generated by the (naive) chase is called a *canonical solution*. It is possible to prove (Fagin et al, 2005a) that each canonical solution is a universal solution. Chasing the s-t tgds in our example scenario generates the canonical, universal pre-solution in Figure 3.**a**. In (Fagin et al, 2005a), the notion of a *weakly-acyclic* set of tgds was introduced to guarantee that the chase terminates and generates a universal solution.

After a canonical pre-solution has been generated by chasing the s-t tgds, to generate an actual universal solution it is necessary to chase the target dependencies. Notice that the chase of target tgds can be defined exactly in the same way, with the variant that it only works for assignments such that $J \models \phi(a(\overline{x}))$. However, in this example, there is no need to chase the target tgd: the pre-solution is also a solution for tgd t_1. In fact, the target tgd states that, whenever a tuple is inserted into the *Grant* table, a corresponding tuple must exist in the the *Company* table, and this is the case in our pre-solution. Generating tgds that have this property is one of the

[1] We refer to naive chase rather than to the *standard* chase used in Fagin et al (2005a), since the naive chase is much simpler and rather straightforward to implement in SQL. Such chase is sometimes calles *oblivious chase*, e.g. in (Marnette, 2009)

main intuitions behind the Clio algorithms Miller et al (2000); Popa et al (2002), which will be discussed in more details in Section 3.

To chase an egd $\phi(\bar{x}) \rightarrow (x_i = x_j)$ over an instance J, for each assignment a such that $J \models \phi(a(\bar{x}))$, if $h(x_i) \neq h(x_j)$, the chase tries to equate the two values. We distinguish two cases: (i) both $h(x_i)$ and $h(x_j)$ are constants; in this case, the chase procedure *fails*, since it attempts to identify two different constants; (ii) at least one of $h(x_i)$, $h(x_j)$ is a null – say $h(x_i)$ – in this case chasing the egd generates a new instance J' obtained from J by replacing all occurrences of $h(x_i)$ by $h(x_j)$. To give an example, consider egd e_1:

$$e_1.\ \forall n,n',i,i',s\colon Company(i,n,s) \land Company(i',n',s) \rightarrow (i = i') \land (n = n')$$

On the two tuples generated by chasing the tgds, *Company* $(23, Yahoo!, YHOO)$, *Company* $(N_2, Yahoo!, YHOO)$, chasing the egd equates N_2 to the constant 23, based on the same value for the symbol attribute, *YHOO*. Chasing the egds returns the canonical universal solution in Figure 3.**b**. Notice how the canonical universal solution is not the core universal solution, which in turn is represented in Figure 3.**c**.

Based on these ideas, it is possible to introduce the following procedure to solve a mapping scenario $\mathscr{M}$ given a source instance I:

- first, chase the s-t tgds in Σ_{st} on I to generate a canonical pre-solution, J_{pre};
- then, chase the target constraints (target tgds and especially egds) on J_{pre}, to generate a canonical universal solution, J;
- minimize J by looking for endomorphic subsets that are still universal solutions, to generate the core universal solution, J_0

There currently exist chase engines capable of doing this (Savenkov and Pichler, 2008), which we will discuss thoroughly in the remainder of this chapter.

Chasing with SQL As an alternative, the naive chase of a set of tgds on a given source instance I can be naturally implemented using SQL. Given a tgd $\phi(\bar{x}) \rightarrow \exists\bar{y}(\psi(\bar{x},\bar{y}))$, in order to chase it over I we may see $\phi(\bar{x})$ as a first-order query Q_ϕ with free variables $\bar{x}$ over $\mathbf{S}$. We may execute $Q_\phi(I)$ using SQL in order to find all vectors of constants that satisfy the premise.

We now need to insert the appropriate tuples into the target instance to satisfy $\psi(\bar{x},\bar{y})$. However, in order to do this, we need to find a way to properly "invent" some fresh nulls for $\bar{y}$. To do this, Skolem functions (Hull and Yoshikawa, 1990) are typically used. Given a vector of k universally quantified variables $\bar{x}$, a *Skolem term*[2] over $\bar{x}$ is a term of the form $f(\bar{x})$ where f is a function symbol of arity k. Skolem terms are used to create fresh labeled nulls on the target. Given an assignment of values a for $\bar{x}$, with the Skolem term above we (injectively) associate a labeled null $N_{f(a(\bar{x}))}$.

Based on this, in order to implement the chase by means of SQL statements, as a preliminary step we replace existential variables in the conclusion by means of Skolem terms. More specifically, for each tgd $m : \phi(\bar{x}) \rightarrow \exists\bar{y}(\psi(\bar{x},\bar{y}))$, we use

[2] While Skolem terms are usually nested, for the sake of simplicity here we only consider flat terms.

a different Skolem function f_{m,y_i} for each variable $y_i \in \bar{y}$, and take as argument all universal variables that appear in the conclusion.

To give an example of how the process works, consider tgd m_2 above.

$$m_2.\ \forall n,c,a,pi,n: \textit{Public-Company}(n,c) \land \textit{Public-Grant}(a,pi,n) \rightarrow \\ \exists I,S: \textit{Company}(I,n,S) \land \textit{Grant}(a,I)$$

In order to implement the chase in SQL, the tgd is first rewritten using Skolem terms as follows:

$$m_2'.\ \forall n,c,a,pi,n: \textit{Public-Company}(n,c) \land \textit{Public-Grant}(a,pi,n) \rightarrow \\ \exists I,S: \textit{Company}(f_I(n,a),n,f_S(n,a)) \land \textit{Grant}(a,f_I(n,a))$$

As an example, we show below one of the two SQL statements to which m_2' is translated (we omit the second on *Grant* for space reasons):

```
INSERT into Company
   SELECT append(`fI(',c.name, `,', g.amount, `)'), c.n,
          append(`fS(',c.name, `,', g.amount, `)')
   FROM Public-Company c, Public-Grant g
   WHERE c.name = g.company
```

3 Schema Mappings: The Early Years

The design of mappings had for a long time been a manual task. Transformation designers had to express their mappings in complex transformation languages and scripts, and this only after they had obtained a good knowledge and understanding of the semantics of the schemas and of the desired transformation. As schemas started to become larger and more complex, it was soon realized that the manual design of the mappings was at the same time laborious, time-consuming and error-prone. While seeking support for mapping designers, mapping tools were created with the intention of raising the level of abstraction an the automated part of the tasks. This section provides an overview of the developments in mapping generation since the very first need of data transformations, until the development of the first schema mapping tools under the form they are widely understood today. Having defined the data exchange problem, this section describes how a mapping scenario can be constructed. The presented algorithm, which is the basis of the Clio (Popa et al, 2002) mapping scenario generation mechanism, has the additional advantage that generates scenarios in which the mappings respect the target schema constraints. In that sense, generating the target instance can be done by taking into consideration only the mappings of the mapping scenario and not the target schema constraints. This kind of mappings are more expressive that other formalisms like simple correspondence lines (Rahm and Bernstein, 2001) or morphisms (Melnik et al, 2005).

3.1 The First Data Translation Systems

Since the beginning of data integration, a major challenge has been the ability to translate data from one format to another. This problem of *data translation* has been studied for many years, in different variants and under different assumptions. One of the first systems was EXPRESS (Shu et al, 1977), a system developed by IBM. A series of similar but more advanced tools have followed EXPRESS. The TXL language (Abu-Hamdeh et al, 1994), initially designed to describe syntactic software transformations, offered a richer set of operations and soon became popular in the data management community. It was based on transformation rules that were fired upon successful parsing of the input data. The problem became more challenging when data had to be transformed across different data models, a situation that was typically met in wrapper construction (Tork-Roth and Schwarz, 1997). MDM (Atzeni and Torlone, 1997) was a system for this kind of transformations that was based on patterns (Atzeni and Torlone, 1995).

Some later works (Beeri and Milo, 1999) proposed a tree-structured data model for describing schemas, and showed that the model was expressive enough to represent relational and XML schemas, paving the way for the later introduction of tree based transformations. A formal foundation for data translation was created, alongside a declarative framework for data translation (Abiteboul et al, 1997). Based on this work, the TranScm system (Milo and Zohar, 1998) used a library of transformation rules and pattern matching techniques to select the most applicable rules between two schemas, in an effort to automate the whole data translation task. Other transformation languages developed in parallel emphasized on the type checking (Cluet et al, 1998) task or on integrity constraint satisfaction (Davidson and Kosky, 1997).

3.2 Correspondences

The first step towards the creation of mappings between two schemas was to understand how the elements of the different schemas relate to each other. This relationship had to be expressed in some high level specification. That specification was materialized in the form of correspondences.

A *correspondence* maps atomic elements of a source schema to atomic elements of the target schema. This specification is independent of logical design choices such as the grouping of elements into tables (normalization choices), or the nesting of records or tables (for example, the hierarchical structure of an XML schema). In other words, one need not specify the logical access paths (join or navigation) that define the associations between the elements involved. Therefore, even users that are unfamiliar with the complex structure of the schema can easily specify them. Correspondences can be represented graphically through simple arrows or lines that connect the elements of the two schemas.

The efficacy of using element-to-element correspondences is greatly increased by the fact that they need not be specified by a human user. They could be in fact the result of an automatic component that matches the elements of the two schemas, and then the mapping designer simply verifies the correctness of the results. This task is found in the literature under the name *schema matching* and has received considerable attention, and has led into a variety of methodologies and techniques (Rahm and Bernstein, 2001).

A correspondence can be formally described as a tgd with one and only one existentially quantified variable being equal to one of the universally quantified variables, and one term on each side of the dependency (for the case of the relational schemas). The correspondence states that every value of the source schema element represented by the first variable should also exist in the instance values of target schema element represented by the second.

In certain cases, correspondences that involve more than one source schema elements may exist, but there should always be one existentially quantified variable whose value is determined as a function of the universally quantified variables representing the participated source schema elements.

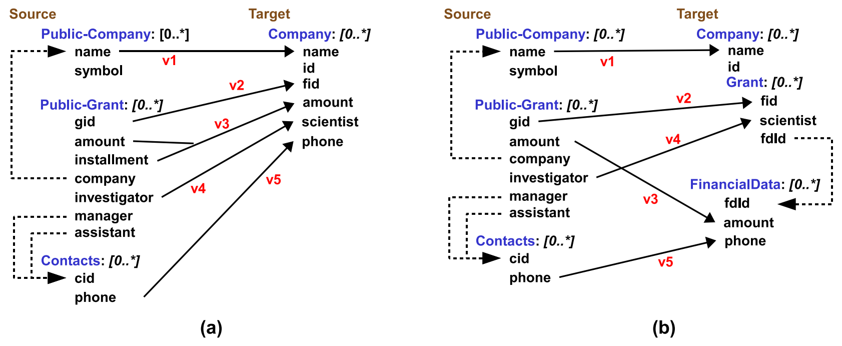


Fig. 4 Two Schema Matching Examples

Consider the example of Figure 4(a) which is a variation of the example previously presented. Here the first source consists of only the three relational tables Public-Company, Public-Grant, and Contact, while the target consists of only the table Company. As before, the intra-schema lines represent schema constraints, and in the particular example are foreign key constraints. The red dotted inter-schema lines, represent the correspondences. Note that the appearance of an attribute with the same or similar name in both schemas, does not necessarily mean that the two attributes represent the same fact. For instance, consider the attributes symbol and id. Although in the companies world these terms may be used interchangingly , in the specific example, the lack of a line among them may be justified by a case in which the attribute id may represent the fiscal number of the company while the attribute symbol may be the symbol with which the company appears in the stock exchange.

The line v_1 from the attribute name of the Public-Company table to the attribute name of the Company table, represents a correspondence declaring that the latter has to be populated with values from the former. Its tgd representation is:

$$v_1\colon \forall na, sy\ Public-Company(na, sy)\ \rightarrow$$
$$\exists na2, id, fi, am2, sc, ph2\ \ Company(na2, id, fi, am2, sc, ph2),\ na2 = na$$

It can be seen in the above logical expression that among all the existential variables of its right-hand side, only the value of the $na2$ is determined by a source variable, i.e., one of the universally quantified.

A situation that demonstrate the case in which an attribute value in the target schema is created by a combination of attribute values from the source is the one of amount. Although the attribute amount appears in both schemas, it may be the case that in the first, amount means the amount of an installment while in the second amount may mean the total figure. In that case, the value of the latter is composed by the value of the former multiplied by the number of installments that is stored in the attribute installements. The tgd of the correspondence is:

$$v_3\colon\ \ \forall gi, am, in, co, re, ma, as\ Public-Grant(gi, am, in, co, re, ma, as) \rightarrow$$
$$\exists na2, id, fi, am2, sc, ph2\ \ Company(na2, id, fi, am2, sc, ph2),\ am2 = am * in$$

Note that even in this case, there is only one target schema variable whose value is determined by source variable values.

While easy to create, understand and manipulate, element correspondences are not expressive enough to describe the full semantics of a transformation. As a consequence, they are inherently ambiguous. There may be many mappings that are consistent with a set of correspondences, and still, not all of them have the same semantics. A mapping generation tool needs to be able to identify what the mapping designer had in mind when he/she provided a given set of correspondences, and generate plausible interpretations to produce a precise and faithful representation of the transformation, i.e., the mappings. For instance, in the schema mapping scenario of Figure 4(a), consider only the correspondence v_1. One possible mapping that this correspondence alone describes is that for each Public-Company in the source instance, there should be in the target instance a Company with the same name. Based on a similar reasoning for correspondence v_2, for every Public-Grant with identifier gid in the source instance, it is expected that there should be a Company tuple in the target instance with that grant identifier as attribute fid. By noticing that a Public-Grant is related to a Public-Company through the foreign key on attribute company, one can easily realized that a more natural interpretation of these two correspondences is that every public grant identifier found in a target schema tuple of table Company should have as an associated company name the name of the respective public company that the public grant is associated in the source. Yet, it is not clear, whether public companies with no associated grants should appear in the target table Company with a null fid attribute, or should not appear at all. Furthermore, notice that the target schema relation has an attribute phone that is populated from the homonym attribute from the source. This value, should not be random, but somehow related to the company and the grant. However, notice that the Contact table in which the phone is located, is related to the grant information through two different join paths, i.e. one on the manager and one on the assistant. The information provided by the correspondence on the phone, is not enough to specify whether the target should be populated with the phone of the manager or the phone of the assistant.

The challenging task of interpreting the ambiguous correspondences gave raise to the schema mapping problem as it has been introduced in Section 2.

3.3 Schema Mapping as Query Discovery

One of the first mapping tools to systematically study the schema mapping problem was Clio (Miller et al, 2000), a tool developed by IBM. The initial algorithm of the tool considers each target schema relation independently. For each relation R_i, it creates a set V^{R_i} of all the correspondences that are on a target schema element that belongs to the relation R_i. Naturally, all these sets are mutually disjoint. For each such set, a query $Q_{V^{R_i}}$ will be constructed to populate the relation R_i. The latter query is constructed as follows. The set V^{R_i} of correspondences is further divided into maximal subsets such that each such maximal subset $M_k^{V^{R_i}}$ contains at most one correspondence for each attribute of the respective target schema relation. For all the source schema elements used by the correspondences in each such subset, the possible join paths connecting them are discovered, and combined to form the union of join queries. These queries are then combined together through an outer union operator to form the query $Q_{V^{R_i}}$.

3.4 Schema Mapping as Constraint Discovery

The algorithm for managing the schema mapping problem as query discovery failed to handle two important cases. The first, was the complex nesting schema situations, and the second was the management of unspecified attributes, i.e., attributes in the target schema for which there is no correspondence to specify their value, yet, the target schema specification either does not permit a null value, or even if it does, its use will lead to loss of information. Furthermore, it became clear that the schema information in conjunction will the correspondences could not always lead into a full specification of the target instance, but only into a constraint relationship between the source and the target instance. Thus, the notion of a mapping stopped being the actual transformation script and became this notion of inter-schema constraint, expressed as a tgd. This is a more natural view of the mapping problem since with schemas being heterogeneous, it is natural to expect that not all the information represented in the source can also exist in the target, and vice versa. Since a mapping describes only the data that is to be exchanged between the schemas, the information described by the mapping is a subset of the information described by the schemas.

Consider the example of Figure 4(b). The situation is more or less the same as the one on its left, with the small difference that the target schema has all the grant information grouped and nested within the company in which the grant belongs. Furthermore, the amount of the grand is not stored within the grand but separately in the FinancialData structure. Note that the Grant structure has an attribute fdid used by no correspondence, thus it could have remained null, if the target schema specification permits it. If not, a random value could have been generated to deal with this restriction. Unfortunately, either of the two actions would break the relationship of the funding information with its amount, since the attribute fdid is actually the foreign key relationship that connects their respective structures.

To discover the intended meaning of the correspondences and generate the mappings, it is important to realize how the elements within a schema relate to each other. This relationship will guide the combination of correspondences into groups and the creation of the expected mappings. The idea for doing so comes from the work on the universal relation (Maier et al, 1984). The universal relation provides a single-relation view of the whole database, in a way that the user does not have to specify different tables and join paths. The construction of the universal relation is based on the notion of logical access paths, or *connections*, as they were initially introduced, and are groups of attributes connected either by being in the same table or by following foreign key constraints (Maier et al, 1984).

A generalized notion of a *connection* is that of the *association* (Popa et al, 2002). Intuitively, an *association* represents a set of elements in the same schema alongside their relationships. An association is represented as a logical query whose head consists of a relation with all the attributes mentioned in the body. For simplicity the head of the association is most of the time omitted. As an example, the following logical query body:

$$A(x,y,z),\ B(u,v,w),\ x=u$$

represents an association that consists of the six attributes of the tables A and B, for which the first is equal to the fourth. Obviously, not all associations are semantically meaningful. In database systems, there are many ways one can specify semantic relationships between schema elements, but three are the most prevalent, the schema structure, the schema constraints, and the user specification, which define three respective kinds of associations.

The structural associations are based on the aggregation of schema elements as it has been specified by the database designer. For instance, the placement of a number of attributes in the same tables means that these attributes are related to each other, most probably by describing different characteristics of the entity that the respective table represents. In a relational schema there is one structural association for each set of attributes in a table. For a nested schema, there is a structural association for each set element, at any level of nesting. The association is constructed by collecting all the non-set subelements of the set element alongside all the non-set subelements of every set element ancestor. Due to the way structural associations are constructed in nested schemas, they are also known broadly as *primary paths*. The source schema of Figure 4(a) has the following three primary paths: (i) $Public{-}Company(na,sy)$,

(ii) $Public{-}Grant(gi, am, in, co, re, ma, as)$, and (iii) $Contact(ci, ph)$, while the target schema has only the $Company(na2, id, fi, am, sc, ph2)$.

For the scenario of Figure 4(b), the primary paths of the source schema are the same, while those of the target schema are: (i) $Company(na2, id)$, (ii) $Company(na2, id, Grant)$, $Grant(fi, sc, fd)$, and (iii) $FinancialData(fd2, am2, ph2)$. Note that the structural association that contains the elements of the set element Grant, those of the set element Company are also included since the former is nested within the latter.

Schema formalisms may not always be enough to describe the full semantics of the data. A data administrator may have some knowledge about sets of attributes that are associated that is nowhere recorded. Based on this user knowledge, an association can be constructed. These kinds of associations are known as *user associations* (Velegrakis, 2005).

Apart from the schema structure, another way database designers can specify semantic relationships between schema elements is the use of schema constraints. This lead to the form of association called the *logical associations*. A logical association is a maximal set of schema elements that are related either through user specification (user association), either through structural construction (structural association), or through constraints. Since logical associations are based on constraints, they can be used as an alternative for computing different join paths on the schema.

Logical associations, also known in the literature, as logical relations, are computed by using the chase (Maier et al, 1979), a classical method that has been used in query optimization (Popa and Tannen, 1999), although originally introduced to test implications of functional dependencies. A chase procedure is a sequence of chase steps. A chase step is is an enhancement of an association using a schema constraint. In particular, when the left part of the tgd that expresses a referential constraint is logically implied by the logical representation of the association, then the latter is enhanced with the terms and the conditions of the right-hand side of the tgd of the constraint. This intuitively means that the association is expanded to include the referenced attributes of the constraint. The procedure is repeated until no more schema constraints can be applied, in which case the association has become maximal. This maximal association is a logical association. Maier et al. (Maier et al, 1979) have shown that for the relational model, two different chase sequences with the same set of dependencies, i.e., constraints, generate identical results. Popa (Popa, 2000) has shown a similar result for the case of the nested relational model. These two results mean that the the result of the chase of a user or a structural association with a set of constraints is unique. To illustrate how the logical relations are computed, consider again the example on Figure 4(b). Let A represent the structural association $Public{-}Grant(gi, am, co, in, ma, as)$. The tgd expressing the foreign key constraint from the attribute company to name is $Public{-}Grant(gi, am, co, in, ma, as)$ $\rightarrow Public{-}Company(na, sy)$, $na = co$. Its left-hand side is the same as A, thus, the question on whether it is logically implied by A is yes which means that a chase step can be applied on A using the specific constraint. This will enhance A with the contents of the right-hand side of the tgd of the constraint, bringing the association into the form:

$Public{-}Grant(gi, am, co, in, ma, as)$, $Public{-}Company(na, sy)$, $na = co$

Further chase steps on the association using the foreign key constraints on the attributes manager and assistant will further expand the association into the form:
$Public{-}Grant(gi, am, co, in, ma, as)$, $Public{-}Company(na, sy)$,
$Contact(cim, phm)$, $Contact(cia, pha)$, $na = co \wedge cim = ma \wedge cia = as$
Since no other constraint can be further applied to it A, A in its last form is a logical association.

Associations form the basis for understanding how the correspondences can be combined together to form groups that will produce semantically meaningful mappings. The technique presented here forms the basis of the Clio (Fagin et al, 2009) mapping tool. Given a set of correspondences, Clio generates a mapping scenario with nested tgds. Similar technique has also been adapted by other tools, such as Spicy (Bonifati et al, 2008) or HePToX (Bonifati et al, 2005). This is done by considering pairs of source and target logical associations. For each such pair, the set of correspondences covered by the pair is discovered. A correspondence is said to be covered by the pair A, B of a source and a target association, if its left and right part (apart from the equality condition) are logically implied by A and B, respectively. A mapping is formed by creating a tgd whose left-hand side consists of association A, and whose right-hand side is the association B enhanced with the conditions of the covered correspondences. Note that the covering of a correspondence is based on the notion of homomorphism. If there are multiple homomorphisms, then there are multiple alternative mappings. Consider for instance the source-target logical association pair $Public{-}Company(na, sy)$ and $Company(na2, id, Grand)$. Only the correspondence $v1$ is covered by this pair. This leads to the mapping m_1: $Public{-}Company(na, sy) \rightarrow Company(na2, id, Grand)$, $na2 = na$, where the last equation is the one that was on the tgd representation of the correspondence $v1$.
For the source-target logical association pair A and B, where A is
$Public{-}Grant(gi, am, co, in, ma, as)$, $Public{-}Company(na, sy)$,
$Contact(cim, phm)$, $Contact(cia, pha)$, $na{=}co \wedge cim{=}ma \wedge cia{=}as$
and B is
$Company(na2, id, Grand)$, $Grant(fi, sc, sd)$,
$FinancialData(fd2, am2, ph2)$, $fd2{=}fd$,
all the correspondences illustrated in Figure 4 are covered. However, for $v5$ there are two ways that it can be covered, which leads to two different mappings. The first is:

$$\begin{aligned}
&Public{-}Grant(gi, am, co, in, ma, as),\ Public{-}Company(na, sy),\\
&\quad Contact(cim, phm),\ Contact(cia, pha),\ na{=}co \wedge cim{=}ma \wedge cia{=}as\\
&\qquad \rightarrow Company(na2, id, Grand),\ Grant(fi, sc, sd),\\
&\qquad\quad FinancialData(fd2, am2, ph2),\ fd2{=}fd \wedge\\
&\qquad\quad na2{=}na \wedge fi{=}gi \wedge am2{=}am \wedge re{=}sc \wedge ph2{=}pha
\end{aligned}$$

The second mapping is exactly the same with the only difference that the last equality is $ph2{=}phm$ instead of $ph2{=}pha$.

Note that through the right-hand side conditions, the mappings guarantee to generate data that does not violate the constraints of the target schema, which is why finding a solution to a Clio generated scenario does not need to take into consideration the target schema constraints, since they have been taken into consideration in the source-to-target tgd construction.

3.5 Data Exchange and Query Generation

Once the mapping scenario has been constructed, the next step it to find a solution (see Section 2). Clio (Popa et al, 2002) was the first tool to consider mappings in a nested relational setting, thus not only offering mappings that were nested tgds, but also offering an algorithm for generating nested universal solutions.

The algorithm mainly starts by creating a graph of the target schema in which every node corresponds to a schema element, i.e., a table or an attribute in the case of a relational schema. Then, the nodes are annotated with source schema elements from where the values will be derived. These annotations propagate to other nodes based on the nested relationship and on integrity constraint associations. The value for the unspecified elements is the result if a Skolem function that gets as arguments the values of all the source schema elements of the source. The annotations that have been made on the set elements are used to create Skolem functions that drive the right nesting. More details can be found in (Fagin et al, 2009).

At this point, the final queries, or transformation scripts in general, can be constructed. First, the variable of every unspecified target schema element in the mapping is replaced by its Skolem function expression. In its simplest brute-force form, the final query is generated by first executing the query described on the left-hand side of the mapping tgd expression for every nesting level, i.e., for every set element of any nesting depth of the target. Then, the Skolem functions that have been computed for the set elements of the target are used to partition the result set of these queries and place them nested under the right elements. The full details of this task can be found in (Fagin et al, 2009).

4 Second-Generation Mapping Systems

Inspired by the seminal papers about the first schema mapping system (Miller et al, 2000; Popa et al, 2002), in the following years a rich body of research has proposed algorithms and tools to improve the easiness of use of mapping systems (An et al, 2007; Raffio et al, 2008; Cabibbo, 2009; Mecca et al, 2009b) (see Section 7 and Chapter 9) and the quality of the solutions they produce. As experimentally shown in (Fuxman et al, 2006; Mecca et al, 2009a), different solutions for the same scenario may differ significantly in size and for large source instances the amount of redundancy in the target produced by first generation mapping systems may be very large, thus impairing the efficiency of the exchange and the query answering process. Since the core is the smallest among the solutions that preserve the semantics of the exchange, it is considered a desirable requirement for a schema mapping system to generate executable scripts that materialize core solutions for a mapping scenario.

In this Section, we present results related to this latest issue and we show how novel algorithms for mapping generation and rewriting have progressively addressed the challenge of producing the best solution for data exchange.

4.1 Problems with canonical solutions

To see how translating data with mapping systems from a given source database may bring to a certain amount of redundancy into the target data, consider again the mapping scenario in Figure 2 and its source instance in Figure 1. To simplify the discussion, in the following we drop the target egd constraints as they are not handled by most mapping systems during the schema mapping generation. Based on the schemas and the correspondences in the scenario, a constraint-driven mapping system such as Clio would rewrite the target tgd constraints into a set of s-t tgds (using the logical associations described in Section 3), like the ones below.

m_1. $\forall s,n: NYSE(s,n) \rightarrow \exists I: Company(I,n,s)$
m_2. $\forall n,c,a,pi: Public\text{-}Company(n,c) \wedge Public\text{-}Grant(a,pi,n) \rightarrow$
$\qquad \exists I,S: Company(I,n,S) \wedge Grant(a,I)$
m_3. $\forall i,n,s: NSF\text{-}Grantee(i,n,s) \rightarrow Company(i,n,s)$
m_4. $\forall a,c: NSF\text{-}Grant(a,c) \rightarrow \exists M,S: Company(c,M,S) \wedge Grant(a,c)$

Notice that these expressions are different from those in Section 2. In fact, the mapping tool is taking care of the foreign key constraint in m_4 and produces the canonical universal solution in Figure 5.a. While this instance satisfies the s-t tgds (and the original target tgd), still it contains many redundant tuples, those with a gray background.

Company

id	name	symbol
N1	Google	GOOG
N2	Yahoo	YHOO
I1	Apple	S1
I2	Adobe	S2
23	Yahoo!	YHOO
25	Adobe	ADBE
23	M1	S3
25	M2	S4

Grant

amount	company
25,000	I1
50,000	I2
18,000	23
50,000	25

a. Canonical universal solution

Company

id	name	symbol
N1	Google	GOOG
I1	Apple	NULL
23	Yahoo!	YHOO
25	Adobe	ADBE

Grant

amount	company
25,000	I1
18,000	23
50,000	25

b. Core universal solution

Fig. 5 Canonical and Core Solutions for the Mapping Scenario

Consider for example the tuple $t_1 =$ *(N2, Yahoo!, YHOO)* in the *Company* table; it can be seen that the tuple is redundant since the target contains another tuple $t_2 =$ *(23, Yahoo!, YHOO)* for the same company, which in addition to the company name also gives information about its id in the target database (i.e., there is an homomorphism from t_1 to t_2). A similar argument holds for the tuples *(I2, Adobe, S2)* and *(50,000, I2)*, where *I2* and *S2* are the values invented by executing tgd m_2, while there are tuples with real id and symbol values for the same company and grant. The core in this example is the solution reported in Figure 5.b.

Therefore, a natural requirement for a schema mapping system becomes that of materializing core solutions. We now review the algorithms that have been proposed to compute the core solution in a relational data exchange setting.

4.2 Theoretical results on core computation

The first approach that has been studied to generate the core for a relational data exchange problem is to generate the canonical solution, and then to apply a post-processing algorithm for its core identification. It is known that computing the core of an arbitrary relational instance with variables is NP-complete, as many NP-complete problems (e.g., computing the core of a graph (Fagin et al, 2005b; Hell and Nešetřil, 1992) or conjunctive query optimization (Chandra and Merlin, 1977)) can be reduced to it. In contrast with the case of computing the core of an arbitrary instance, computing the core of a universal solution in data exchange can be done in polynomial time.

In Fagin et al. (Fagin et al, 2005b), an algorithm is presented that computes the core in polynomial time in a restricted setting, that is, for a data exchange problem whose source-to-target constraints are tgds and whose target constraints consist of arbitrary egds only. More specifically, they proved that the core of a universal solution can be computed in polynomial time in two settings: (i) when the set of target constraints is empty, (ii) when the set of target constraints contains egds only. To address these goals, two different methods are provided.

A *greedy* algorithm, given a source instance I, first computes an universal solution J for I, if it exists, and then computes its core by successively removing tuples from J, as long as I and the instance resulting in each step satisfy the s-t tgds and the target constraints. Although the greedy algorithm is conceptually simple, it requires the availability of the source instance I for the execution of the algorithm.

The *blocks* method does not require the availability of the source instance and is based on the relationships among the labeled nulls of a canonical solution J. The *Gaifman graph* of the nulls of J is an undirected graph in which the nodes are the nulls of J and there exists an edge between two labeled nulls whenever there exists some tuple in some relation of J in which both labeled nulls occur. A *block* of nulls is the set of nulls in a connected component of the Gaifman graph of the nulls. Given J as the result of applying the source-to-target tgds to a ground source instance S, the block method starts from the observation that the Gaifman graph of the labeled nulls of the result instance J consists of connected components whose size is bounded by a constant b. The main step of the algorithm relies on the observation that checking whether there is a homomorphism from any $J \in K$, where K is any set of instances with such bound b, into any arbitrary other instance J_0 is feasible in polynomial time. The algorithm works also for the case where the target constraints consist of egds, which, when applied, can merge blocks by equating variables from different blocks. Thus, after chasing J with egds, the resulting J' can lost the bounded block-size property. However, the authors show an algorithm that looks at the nulls in J and computes its core by successively finding and applying a sequence of small *useful* endomorphisms; where *useful* means that at least one null disappears. More practically, (i) the algorithm starts computing a canonical universal solution J_0, (ii) then it recursively generates a sequence of intermediate instances such that, given the intermediate instance J_i, there is a useful endomorphism that is the identity everywhere except for a block from J_i to J_{i+1}; (iii) when the algorithm stops, the instance J_i is the core solution. The polynomial-time bound is due to the total number of endomorphisms that the algorithm explores, which is at most n^b for each block of J_0, where b is the maximum number of existentially quantified variables over all the s-t tgds and n is the number of tuples in J_0.

Gottlob and Nash (Gottlob and Nash, 2008) extended previous results by introducing an algorithm that computes, still in polynomial time, the core solution of a data exchange problem whose target constraints are (weakly-acyclic) tgds and arbitrary egds. The authors introduce novel technical intuitions to compute the core of universal solutions and prove two complexity bounds. Using an exhaustive enumeration algorithm they get an upper bound of $O(vm|dom(J)|^b)$, where v is the number of variables in the canonical solution J, m is the size of J, and b is the block size of J. There exist cases where a better bound can be achieved by relying on hypertree decomposition techniques. In such cases, the upper bound is $O(vm^{[b/2]+2})$, with special benefits if the target constraints of the data exchange scenario are LAV tgds.

The main algorithm in (Gottlob and Nash, 2008) has been revised in (Savenkov and Pichler, 2008) (by removing the simulation of egds with full tgds) and in (Marnette, 2009) (by replacing a key component of the algorithm with a faster one). Also, an implementation of the core-computation algorithm in (Gottlob and Nash, 2008) has been developed (Savenkov and Pichler, 2008): the prototype uses a relational DBMS to chase tgds and egds, and a specialized engine to find endomorphisms and minimize the universal solution.

The algorithms above provide a very general solution to the problem of computing core solutions for a data exchange setting and made significant steps towards the goal of integrating core computations into schema mapping systems. However, experience with these algorithms shows that, although polynomial, they require very high computing times since they look for all possible endomorphisms among tuples in the canonical solution (Savenkov and Pichler, 2008; Mecca et al, 2009a). As a consequence, recursive algorithms iterating on intermediate instances hardly scale to large mapping scenarios: the necessity to study more scalable solutions than post-processing approaches motivated the works that follow.

4.3 Generating Core Solutions with SQL scripts

The fact that s-t tgds produced by first-generation schema mapping systems may generate redundancy in the target has motivated several practical proposals towards the goal of removing such redundant data. Unfortunately, some of these works are applicable only in some cases and do not represent a general solution to the problem.

Only recently there are been proposals for general rewriting techniques that are able to obtain core solution with executable scripts. As we discuss next, all the mapping systems that attempted to reduce the redundancy in the solutions started from the formalism and the algorithms in (Popa et al, 2002).

Early attempts *Nested mappings* (Fuxman et al, 2006) are s-t tgds that extend the language of s-t tgds allowing the arbitrary nesting of mapping formulas within other mapping formulas. As an example, the schema mapping from the example in the right-hand side of Figure 4 can be defined by means of nested s-t tgds. We omit the quantifiers for the sake of readability (variables on the right that do not appear on the left are existentially quantified), while the atomic variables are in lower-case and the set variables start with upper-case, as follows:

$$\begin{aligned} m'_1.\ & \mathit{Public\text{-}Company}(sn, ss) \rightarrow [\mathit{Company}(sn, ti, \mathit{Grant}) \\ & \quad \wedge [\mathit{Public\text{-}Grant}(sg, sa, sn, sr, sm, sa) \wedge \mathit{Contact}(sm, ph) \wedge \mathit{Contact}(sa, ph2) \rightarrow \\ & \qquad\qquad \mathit{Grant}(sg, sr, tf) \wedge \mathit{FinancialData}(tf, sa, ph)]] \end{aligned}$$

The second mapping is exactly the same with the only difference that the last variable in atom *FinancialData* is $ph2$ instead of ph.

Intuitively, whenever a tgd m_1 writes into a target relation R_1 and a tgd m_2 writes into a relation R_2 nested into R_1, it is possible to "correlate" the two mappings by nesting m_2 into m_1. The correlation among inner and outer mappings can be observed by the variable *sn* both in *Public-Company* and *Public-Grant* in the example above. This rewritten mapping reduces the amount of redundant tuples in the target, since the same data is not mapped twice in the generated target instance. The same intuition applies if R_2 contains a foreign key pointing to relation R_1. Nested mappings are correlated in a rewriting step based on a *nestable* property for a given pair of mappings. The property is verified with a syntactical check based on the structures of the schemas involved in the mappings and the correspondences between them. Once the property has been verified for all the mappings composing a scenario, the nesting algorithm constructs a DAG, where a node is a mapping having edges to other mappings for which it is nestable. The DAG represents all the possible ways in which mappings can be nested under other mappings. The algorithm identifies root mappings for the DAG (mappings that are not nestable), for each root mapping traverses the DAG to identify a tree of mappings, and generates a nested mapping for each tree rewriting the variables accordingly to the structure.

As nested mappings factor out common subexpressions, there are many benefits in their use: (i) it is possible to produce more efficient translation queries by reducing the number of passes over the source; (ii) the generated data have less redundancy as the same data are not mapped repeatedly by s-t tgds sharing common parts.

Another attempt to reduce the redundancy generated by basic mappings has been proposed by (Cabibbo, 2009). The work introduced a revision of both the mapping and the query generation algorithms. In the mapping generation phase, the presence of nullable attributes is considered to introduce an extended notion of logical associations, a modified chase procedure to compute them, and novel pruning rules used together with the subsumption and implication rules from (Popa et al, 2002).

The query generation algorithm is also revised to ensure the satisfaction of target key constraints or to unveil unsatisfiability of such keys. Moreover, when there are key conflicts between groups of logical mappings with the same target relation, an algorithm tries to resolve those conflicts by rewriting conflicting logical mappings in queries with negations. Such interaction between mapping and query generation algorithms allows to have similar benefits to those gained by nested mappings in different relational settings. In particular, those techniques generate target data with less redundancy, as source data involved in the same target key constraints is copied by the generated queries only once.

Unfortunately, the approaches above are applicable only in some specific cases: the above techniques benefits apply only when schemas and correspondences obey to certain structures or require the presence of key constraints to reduce the redundancy in the output. Therefore, those approaches do not represent a general solution to the problem of generating neither core nor compact universal solutions.

SQL core-generation algorithms The following systems introduce core computation algorithms that, given a set of s-t tgds, enable a more efficient implementation by means of executable scripts that scale well to large databases. This problem has been first approached in (Chiticariu, 2005), where an algorithm is presented for schema mappings specified by the limited class of s-t tgds with single atomic formulas (without repetition of existential quantified variables) in the conclusions.

The first complete proposal of an algorithm for rewriting s-t tgds in order to generate core solutions was introduced in (Mecca et al, 2009a). This work is based on the exploiting of two key ideas: the notion of *homomorphism among formulas* and the use of *negation* to rewrite tgds.

m_1. $\forall s,n: NYSE(s,n) \rightarrow \exists I: Company(I,n,s)$
m_2. $\forall n,c,a,pi: Public\text{-}Company(n,c) \wedge Public\text{-}Grant(a,pi,n) \rightarrow$
$\exists I,S: Company(I,n,S) \wedge Grant(a,I)$
m_3. $\forall i,n,s: NSF\text{-}Grantee(i,n,s) \rightarrow Company(i,n,s)$
m_4. $\forall a,c: NSF\text{-}Grant(a,c) \rightarrow Grant(a,c)$

The first intuition is that it is possible to analyze the set of formulas in order to recognize when two tgds may generate redundant tuples in the target. This happens when it is possible to find a homomorphism between the right-hand sides of the two tgds. Consider the right-hand sides of the s-t tgds m_1 and m_3 from the Example in Section 2 reported here for convenience; with an abuse of notation, we treat the two formulas as sets of tuples, with existentially quantified variables that correspond to nulls. It can be seen that the conclusion $Company(I,n,s)$ of m_1 can be mapped into the conclusion $Company(i,n,s)$ of m_3 by the following mapping of variables: $I \rightarrow i$, $n \rightarrow n$, $s \rightarrow s$; in this case, they say that m_3 *subsumes* m_1. This gives a condition to intercept possible redundancy that is general (i.e., key constraint on the target are not required to identify causes of redundancy) and necessary, since the actual generation of endomorphisms among facts in the target data depends on values coming from the source. From the complexity viewpoint, checking for the presence of homomorphisms among formulas, i.e., conclusions of tgds, is completely different than doing the same check among instance tuples: since the number of tgds is typically order of magnitudes smaller than the size of an instance, the check among formulas can be carried out very quickly.

Based on these ideas, the algorithm finds all possible homomorphisms among s-t tgd conclusions. More specifically, it looks for variable mappings that transform atoms in the conclusion of one tgd into atoms belonging to the conclusions of other tgds, with the constraint that universal variables are mapped to universal variables. There are two homomorphisms of this form in the running example. The first one is from the right hand side of m_1 to the rhs of m_3, as discussed above. The second one is from the rhs of m_2 to the union of the conclusions of m_3 and m_4 by the following mapping: $I \rightarrow i$, $n \rightarrow n$, $S \rightarrow s$, $a \rightarrow a$, $I \rightarrow c$; in this case the homomorphisms to be valid imply a condition $i = c$ and they say that m_3, m_4 *cover* m_2.

A second intuition is that, whenever two tgds m, m' such that m subsumes m' are identified, it is possible to prevent the generation of redundant tuples in the target instance by executing them according to the following strategy: first, generate the target tuples for m, the "more informative" mapping; then, generate for m' only those tuples that actually add some new content to the target. In the running example, the original s-t tgds can be rewritten as follows:

$$
\begin{aligned}
&m_3.\ \forall i,n,s: \textit{NSF-Grantee}(i,n,s) \rightarrow \textit{Company}(i,n,s)\\
&m_4.\ \forall a,c: \textit{NSF-Grant}(a,c) \rightarrow \textit{Grant}(a,c)\\
&m'_2.\ \forall n,c,a,pi,s,i: \textit{Public-Company}(n,c) \wedge \textit{Public-Grant}(a,pi,n) \wedge\\
&\qquad\qquad\qquad \wedge \neg(\textit{NSF-Grantee}(i,n,s) \wedge \textit{NSF-Grant}(a,i)) \rightarrow\\
&\qquad\qquad \exists I,S: \textit{Company}(I,n,S) \wedge \textit{Grant}(a,I)\\
&m'_1.\ \forall s,n,i: \textit{NYSE}(s,n) \wedge \neg(\textit{NSF-Grantee}(i,n,s)) \rightarrow \exists I: \textit{Company}(I,n,s)
\end{aligned}
$$

Once the original tgds have been rewritten in this form, which are called *core schema mappings*, it is easy to generate an executable transformation under the form of relational algebra expressions where negations become difference operators. The algebraic expressions can then be implemented in an executable script, to be run in any database engine. The authors experimentally show that, in the computation of the target instance, with executable scripts there is a gain in efficiency of orders of magnitude with respect to the post-processing algorithms(Fagin et al, 2005b; Gottlob and Nash, 2008; Savenkov and Pichler, 2008).

In (ten Cate et al, 2009) the authors independently developed an algorithm to rewrite a set of s-t tgds as a *laconic mapping*, that is, a new set of dependencies from which to generate an SQL script that computes core solutions for the original scenario. The algorithm is more general than the one proposed in (Mecca et al, 2009a), since it can be applied to dependencies that make use of arbitrary first-order formulas in the premises, and not only conjunctive formulas.

The main algorithm to rewrite schema mappings as laconic is composed of four step. In the first step, it constructs a finite list of potential patterns of tuples in the core. This step is done by an exhaustive analysis of the target right hand side of each s-t tgd in the input mapping. The number of patterns is finite, but exponential in the size of the schema mapping in general. In the running example, the patterns are the right hand sides of the four original mappings. In the second step, the main algorithm computes for each pattern a *precondition*: a first-order formula over the source schema that is able to identify the cases when the core solution will contain the current pattern. This crucial task is done by relying on a procedure called *certain()*, which rewrites the certain answers of a query on the target as a query on the source. Given a source instance I, a schema mapping M, and a query q on the target schema, the set of *certain answers* to q in I with respect to M, is the intersection of the results from the query $q(J_i)$ over all the possible solutions J_i to the mapping. The authors introduce a practical version of the algorithm in which *certain()* relies on a variant of the MiniCon algorithm (Pottinger and Halevy, 2001), which works for conjunctive formulas, and they also announce (ten Cate and Kolaitis, 2009) a more general algorithm to compute *certain()* on arbitrary FO queries. In the example, the precondition for the pattern $Company(I,n,s)$ is the left hand side of mapping m'_1 above. In the third step, the algorithm generates additional *side-conditions* to handle special cases with self-joins in the conclusion, i.e., s-t tgds in which the same relation symbols occurs more than once in the right-hand side. Side-conditions are Boolean combination of formulas with inequalities. In our example, side-conditions are not generated as there are not self-joins. In the final step, the algorithm put together the laconic schema mapping with preconditions and side-conditions in the left-hand-side and the respective pattern in the right-hand-side, thus generating mappings such as m'_1 above.

In terms of dependencies generated by the algorithm, laconic mappings from the algorithm in (ten Cate et al, 2009) tends to contain a lower number of dependencies with more complex premises with respect to the core schema mappings from (Mecca et al, 2009a), which typically contain more rules. In fact, laconic mappings reason on patterns at a "global" level, while the rewriting algorithm for core schema mappings works at a "local" level, i.e., at the tgd level.

5 Query Answering in Mapping Scenarios

An important application of schema mappings arises in all the scenarios in which queries are formulated against one of the two schemas connected by the mappings and need to be translated against the other schema. In the early years, the semantics of query answering in indefinite databases adopted the notion of 'certain answers'. This notion has also been adopted in data exchange (Fagin et al, 2005a), while studying the computational complexity of target query answering, i.e. the problem of computing certain answers for a target query q.

As already explained in Section 4.3, to represent all possible databases, we must consider the set of all possible target instances J_i consistent with $\mathscr{M}$ and the source instance I. Since there may be several target instances J_i, we must consider the intersection $\bigcap_{J_i} q(J_i)$, the intersection being called the set of the *certain answers* of q.

In (Fagin et al, 2005a), the semantics of query answering has been defined by considering the universal solutions. Indeed, it is important to ascertain whether the certain answers of a query can be computed by query evaluation on the ‘good’ target instance that has been chosen for materialization. In Section 2, we have already introduced universal solutions for a data exchange scenario. Sufficient conditions for the existence of a universal solution have been defined for weakly acyclic tgds (Fagin et al, 2005a). In this special case, polynomial-time algorithms can be defined to determine whether a solution exists and to produce a particular solution, the canonical universal solution (as defined in Section 2). By analyzing query answering issues in greater details, (Fagin et al, 2005a) focuses on determining which target queries can be answered using solely the materialized target instance, and studies the computational complexity of computing certain answers for target queries.

Given a fixed data exchange scenario $\mathscr{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, for each target query q, the problem is to study the computational complexity of the following problem: given a source instance I, find the certain answers of q with respect to I. If q is a union of conjunctive queries, the certain answers of q can be computed on an arbitrary canonical universal solution. Having this solution homomorphisms to all solutions, and being computable in polynomial time, it implies that the certain answers of q as union of conjunctive queries can also be computed in polynomial time. However, if conjunctive queries have inequalities, computing the certain answers becomes a coNP-complete problem (Abiteboul and Duschka, 1998). In particular, in (Fagin et al, 2005a), it is shown that computing the certain answers of unions of conjunctive queries with at most two equalities per disjunct is a coNP-complete problem. Beyond the intractability result for the case with two or more inequalities, (Fagin et al, 2005a) shows that there is a polynomial time algorithm for computing certain answers of queries with at most one inequality per disjunct (thus overcoming the result in (Abiteboul and Duschka, 1998)).

(Fagin et al, 2005a) focuses on the relational case, whereas (Yu and Popa, 2004) extends the target query answering to the XML data model, by also covering the presence of target constraints (also called *nested equality-generating dependencies (NEGDS)*. The latter presence further complicates the problem of defining the correct query answering semantics, since merging rules at the target have to be taken into account. In (Yu and Popa, 2004), a nested extension of the relational chase (Fagin et al, 2005a) is used to accomodate XML target instances. A basic query rewriting algorithm is presented, that consists of four phases, precisely rule generation, query translation, query optimization and query assembly. The basic version ignores the presence of target constraints. Rule generation is done by creating a rule for each root of the target schema, and by taking the mappings into consideration. The goal of this phase is to set of mapping rules that fully specify the target in terms of the sources, and to prepare the target expressions that will be substituted by source expressions in the next phase. Query translation is done by translating the target query into a set of decorrelated source queries, by exploiting the set of mappings. Optimization is then performed to eliminate equal Skolem terms that have been introduced in the previous phase and to guarantee the minimization of the rewriting, as one of the cases in (Deutsch et al, 1999). Finally, decorrelated queries get assembled into nested source queries.

The above steps are modified when target constraints are present, since the above query rewriting becomes incomplete. A resolution step needs to be performed before query optimization takes place, in order to exhaustively explore all possible rewritings and the application of resolution to them. The computation is a tree, whose branching factor corresponds to the multiple ways of applying a resolution step to a query. The resolution step terminates if the set of source constraints obtained by translating the target constraints is acyclic.

However, the query rewriting algorithm may still be incomplete, as it explicitly avoids recursion. The validity of the incomplete results is proved experimentally, by measuring their approximation with respect to the complete set of certain answers. However, it is still an open problem how to bridge the completeness gap in an efficient way.

Target query answering is addressed in HePToX (Bonifati et al, 2010) as backward translation, i.e. translation of a query q over the target schema and against the direction of the mappings. In HePToX, the opposite direction of query translation, namely the forward translation, is also considered, to highlight the importance of having bidirectional mappings, that can be traversed either ways. A key complication in forward translation is that μ, the mapping that transforms instances of $\mathbf{S}$ to those of $\mathbf{T}$, may not be invertible (Fagin, 2007). In this direction, the semantics of query answering is still based on certain answers over all possible pre-images I^k for which $J = \mu(I^k)$. This direction is novel and has not been handled in previous work.

To handle this translation, the query q posed against $\mathbf{S}$ is transformed into a tree pattern (for simplicity, only one tree pattern is considered, although the query translation module can handle joins of tree patterns). The tree pattern is matched against each of the rule bodies in Σ_{st}; this phase is called *expansion*. The tree pattern, possibly augmented with dummy nodes at the end of the expansion, is translated against the rules in Σ_{st}, leading to the *translation* phase. Several translated tree patterns may be merged in the *stitching* phase, and dummy and redundant nodes may be eliminated in the *contraction* phase.

6 Developments and Applications

In this chapter, we discuss the recent developments and applications of schema mapping. Schema mapping is widely known as the 'AI-complete' problem of data management, and, as such, exhibits strong theoretical foundations, as it has been highlighted in the previous sections. However, the question we ask ourselves is: what are the real application scenarios in which schema mapping is used? is schema mapping an everyday life problem? All the scenarios that entail the access to multiple heterogenous datasets represent natural applications of schema mapping (Halevy, 2010). For instance, executing a Web search leads to dispatch the request to several web sites, that are differently structured and have possible overlapping content.

Thus, providing a common semantic layer that lets obtain a uniform answer from multiple sites, by means of explicit or implicit correspondences, is the common objective of schema mapping tools. There are several directions on which researchers have focused their attention, and achieved promising results, namely: (*i*) extending the expressiveness of schema mappings to cover data-metadata conflicts (Bonifati et al, 2010, 2005; Hernández et al, 2008); (*ii*) extending them to complex data models, such as XML (Arenas and Libkin, 2008; Amano et al, 2009) and ontologies (Calì et al, 2009b,a); (*iii*) using mappings in large-scale distributed scenarios (Bonifati et al, 2010, 2005; Ives et al, 2004); (*iv*) normalizing and optimizing schema mappings (Gottlob et al, 2009; Fagin et al, 2008). We underline that all the above achievements correspond to the need of addressing problems that arise in real life applications. Indeed, if we focus on the first direction, we can easily think of heterogeneus data management scenarios in which instances and schemas contain the same content and need to be bridged (Bonifati et al, 2010, 2005; Hernández et al, 2008). As an example, health care environments have typically the information about patients, diseases and therapy. However, such information is structured quite differently across the various health care databases. Whereas in one database, the diseases are data instances, it may happen that such values become schema components in another database. Such conflicts, known as data-metadata conflicts, may arise in various other situations, such as multimedia databases, data-intensive web sites, heterogeneous parallel and distributed databases. We illustrate the implications of dealing with data-metadata conflicts, and discuss the schema mapping tools that support data-metadata mappings in Section 6.1. Whereas data integration and exchange tasks have been extensively studied for relational schema mappings, only recently a similar theory has been developed for XML schema mappings (Jiang et al, 2007). Such mappings allow navigational queries with joins and tree patterns, thus enlarging the scope of relational queries (Arenas and Libkin, 2008; Amano et al, 2009). Along the same line, disparate data sources may be expressed by means of ontological languages, which are more or less expressive fragments of OWL-2 (OWL-Full, 2004). Such languages rely on expressive constructs, in which sophisticate semantic relationships are better represented and goes far beyond the expressive power of the relational and XML models. Notwithstanding the complexity of handling such languages to express instances, they are becoming more and more important in data modeling, information integration and development of the Semantic Web. In particular, there has been in the latest years a paradigm shift from decidability issues on ontologies to scalable query answering for suitable fragments, such as Datalog+- (Calì et al, 2009b,a). We discuss the issues behind the treatment of both XML and ontological instances and schema mappings tailored to such instances in Section 6.2. Third, we focus on the mapping scalability issues that arise in real scenarios exhibiting distributed heterogeneous data. In such cases, not only the semantic of mappings should be correctly interpreted, but also the efficiency of data exchange and query answering should be guaranteed. Examples of such distributed architectures are numerous if we think of Internet-scale applications and novel highly distributed peer-to-peer systems. In Section 6.3, we introduce the systems that so far have addressed this problem, discuss their differences and the future work in this area. Fourth, schema mappings expressed as source-to-target dependencies may be redundant, due to the presence of unnecessary atoms, and unrelated variables. Recent efforts have aimed at simplifying such dependencies, by obtaining a normal form (Gottlob et al, 2009) and by identifying various classes of equivalences (Fagin et al, 2008). These optimizations are very important in applications, in which mappings are required to be minimal, for efficiency reasons. We discuss the recent approaches (Gottlob et al, 2009; Fagin et al, 2008) in Section 6.4.

6.1 Bridging Data and Metadata

HePToX (Bonifati et al, 2010, 2005) has been the first system to introduce data-metadata correspondences, that drive the trasformation from the schema components in the source schema to the instance values in the target schema and vice-versa. Such novel correspondences enrich the semantics of the transformation, while at the same time posing new research challenges. HePToX uses a Datalog-based mapping language called TreeLog; being an extension of SchemaLog, it is capable of handling schema and data at par. TreeLog expressions have been inferred from arrows and boxes between elements in the source schema and instances in the target schema, that rely on an ad-hoc graphical notation. By virtue of a bidirectional semantics for query answering, correspondences also involving data-metadata conflicts can be traversed by collecting the necessary components to answer the queries. Queries are expressed in XQuery and the underlying data is expressed in XML to maintain the connection with TreeLog expressions, which are intrinsically nested.

Recently, MAD (MetadatA-Data) mappings (Hernández et al, 2008) have been studied as useful extensions in Clio (Popa et al, 2002), that extend the basic mappings expressed as s-t tgds. Contrarily to HePToX, such mappings are used for data exchange. To this purpose, output dynamic schemas are defined, since the result of data exchange cannot be determined a priori whenever it depends of the instances. MAD mappings in Clio are also generated from visual specifications, similarly to HePToX and then translated to executable trasformations. The translation algorithm is a two-step algorithm in which the first step ‘shreds’ the source data into views that offer a relational partitioning of the target schema, and the second step restructures the result of the previous step by also taking into account user-defined grouping in target schema with nested sets.

To summarize, Clio derives a set of MAD mappings from a set of lines between a source schema and a target schema. Applying these transformations computes a target instance that adheres to the target schema and to the correspondences. Similarly, HePToX derives a set of TreeLog mapping rules from element correspondences (i.e., boxes and arrows) between two schemas. TreeLog rules are similar in spirit to s-t tgds, although TreeLog has a second-order syntax. However, the problems solved by Clio and HePToX are different. In Clio, the goal is data exchange, while in HePToX turns to be query reformulation in an highly distributed setting, as we will further discuss in Section 6.3.

6.2 Extending schema mappings to complex data models

We have recently seen research aiming to study the extensions needed to handle the XML data model for schema mapping (Arenas and Libkin, 2008; Amano et al, 2009), data transformation (Jiang et al, 2007) and query rewriting (Yu and Popa, 2004). The latter (Yu and Popa, 2004) starts from proposing novel algorithms to reformulate target queries against the source schemas, based on the mappings and on target constraints. Given that the data is at the sources, such queries need to be efficiently evaluated and this work considers for the first time both relational and XML schemas. The presence of the target constraints make the problem ways more complicated by the fact that the data transformed according to the mapping needs to be 'merged' afterwards. A further complication bears from the fact that the target constraints can enable each other in a recursive way and interact with the mappings as well. A canonical target instance is defined that takes into account the presence of target constraints and mappings, and the semantics of query answering is decided upon this target instance. Moreover, a basic query rewriting algorithm focuses on only mappings first, and extends to XML queries and XML mappings the relational techniques for query rewriting using views. The target constraints, namely the *nested equality-generating dependencies (NEGDS)*, covering XML schema key constraints among the others, are then considered in a query resolution algorithm. Schema mapping for XML data has been studied in (Jiang et al, 2007), as an extension of the Clio system. In particular, data transformations involving such a complex data model requires more complex transformation primitives, than previously relational efforts. For instance, a key challenge arises with XML-to-XML data transformation if the target data is generated as a hierarchy with multiple levels of grouping (as in 4.b in Section 3). In such a case, a deep union operator must be natively implemented in the transformation engine (and this is done in Clio), as XML query languages, such as XQuery, XSLT and SQL/XML, are not yet suitable for such task of hierarchically merging XML trees. Reasoning about the full structure of XML documents and developing a theory of expressive XML schema mapping has been only recently tackled (Arenas and Libkin, 2008; Amano et al, 2009). In particular, (Arenas and Libkin, 2008) focuses on extending the theory of data exchange to XML, and introduced the XML tree patterns as XML schema mappings. Along the same lines, (Amano et al, 2009) presents an analog of source-to-target dependencies for XML schema mappings, discusses their properties, including their complexity, and presents static analysis techniques for determining the 'consistency' between source schemas and target schemas. The problem of consistency was also dealt with in (Arenas and Libkin, 2008), and in (Amano et al, 2009) it is extended to consider all forms of navigational axes and joins for XML query languages.

Recently, database vendors are extending their products to support ontological reasoning capabilities. Following this direction, research on schema mapping and query rewriting (Calì et al, 2009b,a) is focusing on the extension of classical logical formalisms, such as Datalog, to support query answering over ontologies. Datalog$^{\pm}$ enriches Datalog with existential quantifiers in the rule head, and allows a set of restrictions to guarantee efficient ontology querying. In particular, the tractable fragment of Description Logics, namely *DL-Lite*[15] can be represented with Datalog$^{\pm}$ by filling the gap between databases and the Semantic Web. Suitable fragments of Datalog$^{\pm}$ are embodied by: (i) guarded TGDs (GTGDs); (ii) linear TGDs (LTGDs); (iii) (i) or (ii) with equation-generating dependencies and negative constraints. A TGD σ is guarded iff it contains an atom in its body that has all universally quantified variables of σ. A subset of guarded TGDs is represented by linear TGDs, iff it contains only a singleton body atom. If we look at the source-to-target TGDs illustrated in Section 2, then m_1, m_3 and m_4 are linear TGDs (and, thus, guarded) and m_2 is a non-guarded TGD. The main result of (Calì et al, 2009b) is that query answering with (iii) that do not conflict with the TGDs is feasible in polynomial time in the data complexity and thus is first-order rewritable.

6.3 Distributing Schema Mappings across several sites

We are currently witnessing a substantial interest in distributed database management systems, called PDMS, that are based on highly decentralized P2P infrastructures. Such PDMSs might share heterogeneous data and exchange such data in a seamless fashion.

In Piazza (Ives et al, 2004), each peer stores semantic mappings and storage descriptions. Semantic mappings are equalities or subsumptions between query expressions, provided in XQuery. Storage descriptions are equalities or subsumptions between a query and one or more relations stored on a peer. In Piazza, semantic mappings are first used to do query rewriting using the MiniCon algorithm (Pottinger and Halevy, 2001). When semantic mappings cannot be applied further, storage descriptions are used to do query reformulation. The result of this phase is a reformulation of peer relations into stored relations, which can be either in GAV or LAV style. Query routing in Piazza requires a centralized index that stores all the mappings at a global level.

In HePToX (Bonifati et al, 2005, 2010), the exact mapping rules are derived automatically from correspondences, which are intuitively displayed in a peer-based GUI. In contrast to Piazza, HePToX is totally decentralized and its scalability is less than linear (i.e., logarithmic, as in DHT-based systems). Thus, mappings are locally stored on each peer and used at need when doing query reformulation.

HePToX query rewriting can be done in both directions, along and against the mappings, leading to forward and backward query translations. The semantics of HePToX's forward query translation is similar to answering queries using views (Levy et al, 1995). However, HePToX can leverage Skolem functions and the form of the mapping rules to perform forward translation efficiently. Backward query translation is totally new and was never defined in other systems.

Orchestra (Ives et al, 2008) extends PDMSs for life scientists. It focuses on provenance, trust, and updates. While it can be extended to XML, it uses the relational model. Orchestra's mapping rules translate from tgds to Datalog, rather than HePToX's mapping rules which translate from a visual language to TreeLog. Unlike

HePToX, which supports the user in easily creating the mapping between schemas, Orchestra relies on other systems to create the initial mappings. Moreover, the Q system, which is the query module in Orchestra, focuses on keywords queries rather than on XQuery queries.

Calvanese et al. (Calvanese et al, 2004) address data interoperability in P2P systems using expressive schema mappings, also following the GAV/LAV paradigm, and show that the problem is in PTIME only when mapping rules are expressed in epistemic logic.

6.4 Normalizing schema mappings

Schema mappings, as high-level specifications describing the relationships between two database schemas, are subject to optimization. (Fagin et al, 2008) lays the foundations of schema mapping optimization, by introducing three kinds of equivalence: *(i)* logical equivalence, stating that two schema mappings $\mathscr{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathscr{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ [3], are logically equivalent if for every source instance I and target instance J, we have that $(I,\ J) \models \Sigma$ if and only if $(I,\ J) \models \Sigma'$; *(ii)* data-exchange equivalence, if for every source instance I, the set of universal solutions for I under $\mathscr{M}$ coincides with the set of universal solutions for I under $\mathscr{M}'$; *(iii)* conjunctive-query equivalence, if for every target conjounctive query Q and for every source instance I, the set of solutions for I under $\mathscr{M}$ is empty if and only if the set of solutions for I under $\mathscr{M}'$ is empty, and, whenever they are not empty, the set of certain answers of Q on I under $\mathscr{M}$ coincides with the set of certain answers of Q on I under $\mathscr{M}'$. Equivalences *(ii)* and *(iii)* coincide with equivalence *(i)* when $\Sigma = \Sigma_{st}$, but differ on richer classes of equivalences, such as second-order tgds and sets of both Σ_{st} and Σ_t. The assumption of logical equivalence has also been done in (Gottlob et al, 2009), which focuses on the normalization of schema mappings with respect to four optimality criteria, precisely cardinality-minimality, antecedent-minimality, conclusion-minimality and variable-minimality. Following these criteria, given a set of st-tgds in Σ, the total number of st-tgds in this set, the total number of atoms in the antecedent and conclusion of each st-tgd shall be minimal, along with the total number of existentially quantified variables in the conclusion. The presence of egds is not considered in (Gottlob et al, 2009) and represents a natural extension. Other than that, much work remains to be done towards defining new heuristics for schema mapping optimization, extending the above criteria to larger classes of rules, and considering the impact of all the equivalences discussed above.

[3] We do not distinguish here between Σ_{st} and Σ_t and consider Σ as a set of generic constraints.

7 Conclusions and Future Work

In this chapter, we have discussed the state of the art of schema mapping algorithms, along with their most recent developments and applications.

We believe that there are quite a lot of open problems in this area, which we attempt to briefly discuss below.

First of all, within the data exchange theory the core has been studied only for relational settings, to date there is not a formal definition of core solutions for nested scenarios. We believe such a notion is needed in many practical scenarios.

Post-processing algorithms (Fagin et al, 2005b; Gottlob and Nash, 2008; Savenkov and Pichler, 2008; Marnette, 2009) can handle scenarios with arbitrary target constraints, while by using the rewriting algorithms in (Mecca et al, 2009a; ten Cate et al, 2009), the best we can achieve is to generate a solution that does not consider target tgds and edgs. This is especially unsatisfactory for egds, since the obtained solution violates the required key constraints and it is not even a legal instance for the target. As shown in (Marnette et al, 2010), this may lead to a high level of redundancy, that can seriously impair both the efficiency of the translation and the quality of answering queries over the target database.

In fact, handling egds is a complicate task. As conjectured in (ten Cate et al, 2009), it has recently been shown (Marnette et al, 2010) that it is not possible, in general, to get an universal solution that enforces a set of egds using a first-order language as SQL. For the class of target egds that correspond to functional dependencies, the most common in practical settings, (Marnette et al, 2010) introduced a best-effort rewriting algorithm that takes as input a scenario with s-t tgds and egds and, whenever this is possible, it rewrites it into a new scenario without egds. Moreover, this algorithm can be combined with existing mapping rewriting algorithms (Mecca et al, 2009a; ten Cate et al, 2009) to obtain SQL scripts that generate core solutions. The paper shows that handling target egds efficiently is possible in many practical cases. This is particularly important in real-world applications of mappings, where key constraints are often present and play an important role.

Another important open problem concerns the expressibility of the GUI of a schema mapping tool. Indeed, many GUIs are limited in the set of primitives they use to specify the mapping scenarios and need to be enriched in several ways. For instance, it would be useful to be able to duplicate sets in the source and in the target and, thus, handle tgds that contain duplicate tables. To a further extent, full control over joins in the two data sources becomes a crucial requirement of schema mapping GUIs, in addition to those corresponding to foreign key constraints; by using this feature, users can specify arbitrary join paths, like self-joins themselves.

This richer set of primitives poses some challenges with respect to the mapping generation and rewriting algorithms as well. In particular, duplications in the target correspond to different ways of contributing tuples to the same set. As we discussed above, this makes the generation of core solutions more delicate, since there exist tgds that write more than one tuple at a time in the same target table, and therefore redundancy can be generated not only across different tgds, but also by firing a single tgd (Mecca et al, 2009a; ten Cate et al, 2009).

Second generation mapping systems have certainly enlarged the class of mappings scenarios that can be handled using a GUI, but a formal characterization of the exact class of mappings that can be expressed with them is still missing. For instance, it is still unclear if every mapping made of conjunctive queries can be expressed by existing GUIs.

Finally, another important problem is the use of mappings in practical user scenarios and applications, thus making them the building blocks of general-purpose data transformation tools. Although previous attempts have been done in this direction (as explained in Section 6), more work is still left to fill this gap.

References

Abiteboul S, Duschka OM (1998) Complexity of answering queries using materialized views. In: PODS, pp 254–263

Abiteboul S, Cluet S, Milo T (1997) Correspondence and Translation for Heterogeneous Data. In: ICDT, Delphi, Greece, pp 351–363

Abu-Hamdeh R, Cordy J, Martin T (1994) Schema Translation Using Structural Transformation. In: CASCON, pp 202–215

Amano S, Libkin L, Murlak F (2009) Xml schema mappings. In: PODS, pp 33–42

An Y, Borgida A, Miller R, Mylopoulos J (2007) A Semantic Approach to Discovering Schema Mapping Expressions. In: ICDE, pp 206–215

Arenas M, Libkin L (2008) XML Data Exchange: Consistency and Query Answering. J of the ACM 55(2):1–72

Atzeni P, Torlone R (1995) Schema Translation between Heterogeneous Data Models in a Lattice Framework. In: Data Semantics Conference, pp 345–364

Atzeni P, Torlone R (1997) MDM: A Multiple-Data Model Tool for the Management of Heterogeneous Database Schemes. In: SIGMOD, pp 528–531

Beeri C, Milo T (1999) Schemas for Intergration and Translation of Structured and Semi-structured Data. In: ICDT, pp 296–313

Beeri C, Vardi M (1984) A Proof Procedure for Data Dependencies. J of the ACM 31(4):718–741

Bonifati A, Chang EQ, Ho T, Lakshmanan L, Pottinger R (2005) HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In: VLDB, pp 1267–1270

Bonifati A, Mecca G, Pappalardo A, Raunich S, Summa G (2008) Schema Mapping Verification: The Spicy Way. In: EDBT, pp 85 – 96

Bonifati A, Chang EQ, Ho T, Lakshmanan L, Pottinger R, Chung Y (2010) Schema Mapping and Query Translation in Heterogeneous P2P XML Databases. VLDB J 41(1):231–256

Cabibbo L (2009) On Keys, Foreign Keys and Nullable Attributes in Relational Mapping Systems. In: EDBT, pp 263–274

Calì A, Gottlob G, Lukasiewicz T (2009a) Datalog$^{\pm}$: A Unified Approach to Ontologies and Integrity Constraints. In: ICDT, pp 14–30

Calì A, Gottlob G, Lukasiewicz T (2009b) A general datalog-based framework for tractable query answering over ontologies. In: PODS, pp 77–86

Calvanese D, De Giacomo G, Lenzerini M, Rosati R (2004) Logical Foundations of Peer-to-Peer Data Integration. In: ACM PODS, pp 241–251

ten Cate B, Kolaitis PG (2009) Structural characterizations of schema-mapping languages. In: ICDT, pp 63–72

ten Cate B, Chiticariu L, Kolaitis P, Tan WC (2009) Laconic Schema Mappings: Computing Core Universal Solutions by Means of SQL Queries. PVLDB 2(1):1006–1017

Chandra AK, Merlin PM (1977) Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp 77–90

Chiticariu L (2005) Computing the Core in Data Exchange: Algorithmic Issues. MS Project Report, unpublished manuscript

Cluet S, Delobel C, Siméon J, Smaga K (1998) Your Mediators Need Data Conversion! In: SIGMOD, pp 177–188

Davidson S, Kosky A (1997) WOL: A Language for Database Transformations and Constraints. In: ICDE, pp 55–65

Deutsch A, Popa L, Tannen V (1999) Physical data independence, constraints, and optimization with universal plans. In: VLDB, pp 459–470

Fagin R (2007) Inverting Schema Mappings. ACM TODS 32(4)

Fagin R, Kolaitis P, Miller R, Popa L (2005a) Data Exchange: Semantics and Query Answering. TCS 336(1):89–124

Fagin R, Kolaitis P, Popa L (2005b) Data Exchange: Getting to the Core. ACM TODS 30(1):174–210

Fagin R, Kolaitis P, Nash A, Popa L (2008) Towards a Theory of Schema-Mapping Optimization. In: ACM PODS, pp 33–42

Fagin R, Haas LM, Hernandez M, Miller RJ, Popa L, Velegrakis Y (2009) Conceptual Modeling: Foundations and Applications by A. Borgida, V. Chaudhri, P. Giorgini, E. Yu, Springer, chap Clio: Schema Mapping Creation and Data Exchange, pp 198–236

Fuxman A, Hernández MA, Howard CT, Miller RJ, Papotti P, Popa L (2006) Nested Mappings: Schema Mapping Reloaded. In: VLDB, pp 67–78

Gottlob G, Nash A (2008) Efficient Core Computation in Data Exchange. J of the ACM 55(2):1–49

Gottlob G, Pichler R, Savenkov V (2009) Normalization and Optimization of Schema Mappings. PVLDB 2(1):1102–1113

Haas LM (2007) Beauty and the Beast: The Theory and Practice of Information Integration. In: ICDT, pp 28–43

Halevy AY (2010) Technical perspective - schema mappings: rules for mixing data. Commun CACM 53(1):100

Hell P, Nešetřil J (1992) The Core of a Graph. Discrete Mathematics 109(1-3):117–126

Hernández MA, Papotti P, Tan WC (2008) Data Exchange with Data-Metadata Translations. PVLDB 1(1):260–273

Hull R, Yoshikawa M (1990) ILOG: Declarative Creation and Manipulation of Object Identifiers. In: VLDB, pp 455–468

Ives ZG, Halevy AY, Mork P, Tatarinov I (2004) Piazza: mediation and integration infrastructure for semantic web data. J Web Sem 1(2):155–175

Ives ZG, Green TJ, Karvounarakis G, Taylor NE, Tannen V, Talukdar PP, Jacob M, Pereira F (2008) The orchestra collaborative data sharing system. SIGMOD Record 37(3):26–32

Jiang H, Ho H, Popa L, Han W (2007) Mapping-Driven XML Transformation. In: WWW Conf., pp 1063–1072

Levy AY, Mendelzon A, Sagiv Y, Srivastava D (1995) Answering Queries Using Views. In: PODS

Maier D, Mendelzon AO, Sagiv Y (1979) Testing implications of data dependencies. ACM TODS 4(4):455–469

Maier D, Ullman JD, Vardi MY (1984) On the Foundations of the Universal Relation Model. ACM TODS 9(2):283–308

Marnette B (2009) Generalized Schema Mappings: From Termination to Tractability. In: ACM PODS, pp 13–22

Marnette B, Mecca G, Papotti P (2010) Scalable data exchange with functional dependencies. PVLDB 3(1)

Mecca G, Papotti P, Raunich S (2009a) Core Schema Mappings. In: SIGMOD, pp 655–668

Mecca G, Papotti P, Raunich S, Buoncristiano M (2009b) Concise and Expressive Mappings with +SPICY. PVLDB 2(2):1582–1585

Melnik S, Bernstein P, Halevy A, Rahm E (2005) Supporting executable mappings in model management. In: SIGMOD, pp 167–178

Miller RJ, Haas LM, Hernandez MA (2000) Schema Mapping as Query Discovery. In: VLDB, pp 77–99

Milo T, Zohar S (1998) Using Schema Matching to Simplify Heterogeneous Data Translation. In: VLDB, pp 122–133

OWL-Full (2004) OWL Web Ontology Language Reference. Http://www.w3.org/TR/owl-ref/#OWLFull

Popa L (2000) Object/Relational Query Optimization with Chase and Backchase. PhD thesis, University of Pennsylvania

Popa L, Tannen V (1999) An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In: ICDT, pp 39–57

Popa L, Velegrakis Y, Miller RJ, Hernandez MA, Fagin R (2002) Translating Web Data. In: VLDB, pp 598–609

Pottinger R, Halevy A (2001) Minicon: A scalable algorithm for answering queries using views. VLDB J 10(2-3):182–198

Raffio A, Braga D, Ceri S, Papotti P, Hernández MA (2008) Clip: a Visual Language for Explicit Schema Mappings. In: ICDE, pp 30–39

Rahm E, Bernstein PA (2001) A Survey of Approaches to Automatic Schema Matching. VLDB J 10:334–350

Savenkov V, Pichler R (2008) Towards Practical Feasibility of Core Computation in Data Exchange. In: LPAR, pp 62–78

Shu NC, Housel BC, Taylor RW, Ghosh SP, Lum VY (1977) EXPRESS: A Data EXtraction, Processing and REstructuring System. ACM TODS 2(2):134–174

Tork-Roth M, Schwarz PM (1997) Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In: VLDB, pp 266–275

Velegrakis Y (2005) Managing Schema Mappings in Highly Heterogeneous Environments. PhD thesis, University of Toronto

Yu C, Popa L (2004) Constraint-based xml query rewriting for data integration. In: SIGMOD Conference, pp 371–382