Schema Mappings: from Data Translation to Data Cleaning

Giansalvatore Mecca[1], Paolo Papotti[2], Donatello Santoro[1]

[1] Università della Basilicata, Italy
[2] Arizona State University, USA

Abstract. Schema mapping management is an important research area in data transformation, integration, and cleaning systems. The reasons for its success can be found in the declarative nature of its building block (thus enabling clean semantics and easy to use design tools) paired with the efficiency and modularity in the deployment step. In this chapter we cover the evolution of schema-mappings through what we identify as three main ages. We start presenting the foundations of schema mapping tools and the first tools aimed at translating data from a source to a target schema in the first, heroic age. We then discuss the silver age, when schema mapping tools have grown their way into complex systems and have been translated into both commercial and open-source tools. Finally, we show how recent results in schema-mapping are stimulating a third, golden age, with novel research opportunities and a new generation of systems capable of dealing with a significantly larger class of real-life applications.

1 Introduction

There are many applications that need to exchange, correlate, and integrate heterogenous data sources. These information integration tasks have long been identified as important problems and unifying theoretical frameworks have been advocated by database researchers [9].
To solve these problems, a fundamental requirement is that of manipulating *mappings* among data sources. The application developer is typically given two schemas – one called the source schema, the other called the target schema – that can be based on radically different models, technologies, and rules. 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, by means of SQL queries for relational data, or XQuery scripts for XML[3]. This latter feature is a key requirement in order to embed the execution of the mappings in more complex application scenarios, that is, to make mappings a plug and play component of integration systems.

[3] Given the importance of XQuery engines in practice, we will treat them as their relational counterpart, even if the two platforms cannot be compared in terms of performance.

Traditionally, data transformation has been approached as a manual task requiring experts to understand the design of the schemas and write scripts to translate data [40]. As this work is time-consuming and prone to human errors, mapping generation tools have been created to make the process more abstract and user-friendly, thus easier to handle for a larger class of people.
In this paper, we outline a history of the different phases that have characterized the research about automatic tools and techniques for schema mappings and data exchange.
We identify three different ages, as follows.

The Heroic Age The heroic age of schema-mappings research started with the seminal papers about the Clio system [35,37]. A first generation of tools was proposed to support the process of generating complex logical dependencies – typically *tuple-generating dependencies* [7] – based on a user-friendly abstraction of the mapping provided by the users. Once the dependencies are computed, these tools transform them into executable scripts to generate a target solution in a scalable and portable way.
Early schema-mapping tools proved to be very effective in easing the burden of manually specifying complex transformations, and were successfully transferred, to some extent, into commercial products (e.g., [27]). However, several years after the development of the initial Clio algorithm, researchers realized that a more solid theoretical foundation was needed in order to consolidate practical results obtained on schema mappings systems. This consideration has motivated a rich body of research about *data exchange* that characterizes the next age.

The Silver Age *Data exchange* [9,15,37] formally studies the semantics of generating an instance of a target database given a source database, a set of mappings, and constraints on the target schema. It has formalized the notion of a *data exchange problem* [15], and has established a number of results about its properties.
After the first data exchange studies, it was clear that a key problem in schema-mappings tools was that of the *quality of the solutions*. In fact, there are many possible solutions to a data-exchange problem, and these may largely differ in terms of size and contents. The notion of the *core of the universal solutions* [17] was identified as the "optimal" solution, since it is the smallest among the solutions that preserve the semantics of the mapping.
An intermediate generation of tools [32,41] emerged to address the problem of generating solutions of optimal quality, while guaranteeing at the same time the portability and scalability of the executable scripts. Nevertheless, despite the solid results both in system and theory fields, the adoption of mapping systems in real-life integration applications, such as ETL workflows or Enterprise Information Integration (EII), has been quite slow. This is due to three main factors:

(a) these systems were not able, at first, to handle functional dependencies over the target, which, as it can be easily understood, is a key requirement in order to obtain solutions of quality;

(*b*) the results were obtained primarily for relational databases, and did not extend to nested models and XML;

(*c*) finally, there was no open-source schema-mapping tool publicly available to the community.

The Golden Age New results [30,13], along with the public availability of the first open-source mapping tools – like ++Spicy [31] and OpenII [39][4] – created a promising starting point towards the solution of these problems and the beginning of a new age for mapping tools.
These works, along with others [1,19,42,34], have given new vitality to schema-mappings research and suggested new applications, beyond traditional data exchange tasks. An important breakthrough has been the adoption of the mappings for the *data cleaning* problem [20]. While the supported cleaning language is a simple extension of the original constraints used in data exchange, the cleaning problem is much more complex, as updates over the data are needed to find suitable repairs [23,24]. For this new challenging application, the system focus has moved from the design of the mappings to their execution, as the previous executable scripts cannot handle the more complex algorithm that computes repairs [25].

In this paper we first expose the basics about schema mappings by presenting the early works of the heroic age in Section 2. Section 3 introduces the main advancements about data exchange brought forth in the silver age. Then, in Section 4 we discuss how recent advances can positively impact several data management problems and become the starting point for a forthcoming golden age. Finally, a conclusion is drawn in Section 5.

2 The Heroic Age: Early Mapping Tools

The first mapping generation tools were created to make the process of defining transformations among schemas easier and more effective with respect to manually developed scripts. This first generation of tools includes primarily Clio [21,27,35,37][5]. We may summarize the features of these early mapping tools as follows.

Value Correspondences The goal of simplifying the mapping specification was pursued by introducing a GUI that allows users to draw arrows, or *correspondences*, between schemas in order to define the desired transformation. Consider the example shown in Figure 1, where data from multiple sources should be transformed into data for a target schema with a foreign key constraints between two relations. A correspondence maps atomic elements of the source schema to elements of the target schema, independently of the underlying data model or of logical design choices, and can be derived automatically with schema matching

[4] Available at `http://www.db.unibas.it/projects/spicy/` and `http://sourceforge.net/projects/openii/`, respectively.

[5] Also Spicy [12] and OpenII [39] incorporate a Clio-like first-generation mapping module.

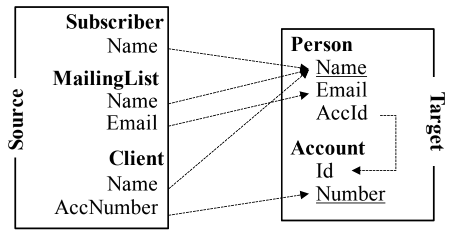


Fig. 1. Schema mapping scenario.

components. Notice that, while correspondences are easy to create and understand, they are a "poor" language to express the full semantics of data transformations. For this reason, a schema mapping tool should be able to interpret the semantics the user wants to express with a set of correspondences.

Mapping Generation Based on value correspondences, mapping systems generate logical dependencies to specify the mapping. These dependencies are logical formulas of two forms: *tuple-generating dependencies* (tgds) or *equality-generating dependencies* (egds). There are two classes of constraints. *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. In an operational interpretation, they state how to "translate" data from the source to the target. Target schemas are also modeled with constraints: *target tgds*, i.e., tgds that only use target symbols, are used to specify foreign-key constraints on the target; while *target egds* are used to encode functional dependencies, such as keys, on the target database.

Consider, for example, the mapping scenario in Figure 1. It has three different source tables: (*i*) a table about subscribers of a service; (*ii*) a table with the email addresses of the people receiving the company mailing list; (*iii*) a table about clients and their check accounts. The target schema contains two tables, one about persons, the second about accounts. On these tables, we have two keys: *name* is a key for the persons, while *number* is a key for the accounts. Based on the correspondences drawn in Figure 1, a Clio-like system would generate the following set of dependencies:

Source-to-Target Tgds

$m_1.\ \forall n: \mathit{Subscriber}(n) \rightarrow \exists Y_1, Y_2: \mathit{Person}(n, Y_1, Y_2)$

$m_2.\ \forall n, e: \mathit{MailingList}(n, e) \rightarrow \exists Y_1: \mathit{Person}(n, e, Y_1)$

$m_3.\ \forall n, acc: \mathit{Client}(n, acc) \rightarrow \exists Y_1, Z: (\mathit{Person}(n, Y_1, Z) \wedge \mathit{Account}(Z, acc))$

Target Egds

$e_1.\ \forall n, e, a, e', a': \mathit{Person}(n, e, a) \wedge \mathit{Person}(n, e', a') \rightarrow (e = e') \wedge (a = a')$

$e_2.\ \forall n, i, i': \mathit{Account}(i, n) \wedge \mathit{Account}(i', n) \rightarrow (i = i')$

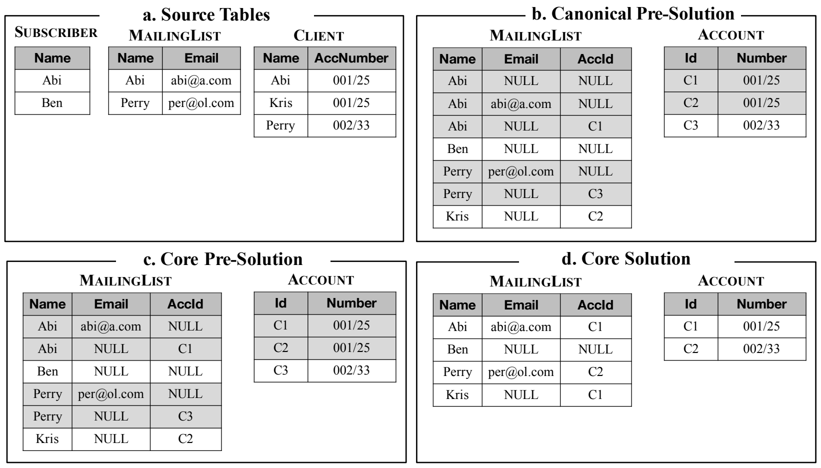
Mapping Execution via Scripts To execute the mappings, schema-mapping systems rely on the traditional *chase procedure* [15]. The chase is a fixpoint algorithm which tests and enforces implication of data dependencies, such as tgds, in a database. To be more specific, a first-generation system, after the mappings had been generated, would discard the target dependencies, and translate the source-to-target ones under the form of an SQL or XQuery script that implements the chase and can be applied to a source instance to return a solution.
Notice, in fact, that the chase of a set of s-t tgds on I can be naturally implemented using SQL. Given a tgd $\phi(\overline{x}) \rightarrow \exists\overline{y}(\psi(\overline{x},\overline{y}))$, in order to chase it over I we may see $\phi(\overline{x})$ as a first-order query Q_ϕ with free variables $\overline{x}$ over the source database. We execute $Q_\phi(I)$ using SQL in order to find all vectors of constants that satisfy the premise and we then insert the appropriate tuple into the target instance to satisfy $\psi(\overline{x},\overline{y})$. Skolem functions [37] are typically used to automatically "generate" some fresh nulls for $\overline{y}$.
However, these systems suffer from a major drawback: they did not have a clear theoretical foundation, and therefore it was not possible to reason about the quality of the solutions.

3 The Silver Age: Theory to the Rescue

Data exchange was conceived as an attempt to formalize the semantics of schema mappings. It formalized many aspects of the mapping execution process, as follows.

3.1 Data Exchange Fundamentals

In a data-exchange setting, the source and target databases are modeled by having two disjoint and infinite sets of values that populate instances: a set of *constants*, CONST, and a set of *labeled nulls*, NULLS [15]. Labeled nulls are used to "invent" values according to existential variables in tgd conclusions. The reference data model is the relational one.
A *mapping scenario* (also called a *data exchange scenario* or a *schema mapping*) is a quadruple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ 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 [15]. Given two disjoint schemas, $\mathbf{S}$ and $\mathbf{T}$, we denote by the pair $\langle \mathbf{S}, \mathbf{T} \rangle$ the schema $\{S_1 \ldots S_n,\ T_1 \ldots T_m\}$. If I is an instance of $\mathbf{S}$ and J is an instance of $\mathbf{T}$, then the pair $\langle I, J \rangle$ is an instance of $\langle \mathbf{S}, \mathbf{T} \rangle$.A target instance J is a *solution* [15] of $\mathcal{M}$ and a source instance 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 $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, a *pre-solution* for $\mathcal{M}$ and a source instance I is a solution over I for scenario $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, obtained from $\mathcal{M}$ 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. Given the source data in Figure 2.a, the canonical pre-solution is reported in Figure 2.b. A mapping scenario may have multiple solutions on a given source instance: each tgd only states an inclusion constraint


a. Source Tables

SUBSCRIBER

Name
Abi
Ben

MAILINGLIST

Name	Email
Abi	abi@a.com
Perry	per@ol.com

CLIENT

Name	AccNumber
Abi	001/25
Kris	001/25
Perry	002/33

b. Canonical Pre-Solution

MAILINGLIST

Name	Email	AccId
Abi	NULL	NULL
Abi	abi@a.com	NULL
Abi	NULL	C1
Ben	NULL	NULL
Perry	per@ol.com	NULL
Perry	NULL	C3
Kris	NULL	C2

ACCOUNT

Id	Number
C1	001/25
C2	001/25
C3	002/33

c. Core Pre-Solution

MAILINGLIST

Name	Email	AccId
Abi	abi@a.com	NULL
Abi	NULL	C1
Ben	NULL	NULL
Perry	per@ol.com	NULL
Perry	NULL	C3
Kris	NULL	C2

ACCOUNT

Id	Number
C1	001/25
C2	001/25
C3	002/33

d. Core Solution

MAILINGLIST

Name	Email	AccId
Abi	abi@a.com	C1
Ben	NULL	NULL
Perry	per@ol.com	C2
Kris	NULL	C1

ACCOUNT

Id	Number
C1	001/25
C2	002/33

Fig. 2. Source instance (a) and three possible solutions (b-d).

and does not fully determine the content of the target. Among the possible solutions we restrict our attention to *universal* solutions, which only contain information from I and $\Sigma_{st} \cup \Sigma_t$. Universal solutions have a crucial property: they have a *homomorphism* (i.e., a constant-preserving mapping of values) into all the solutions for a data exchange problem. Intuitively, this guarantees that the solution does not contain any arbitrary information that does not follow from the source instance and the mappings.
Under a condition of weak acyclicity of the target tgds, an universal solution for a mapping scenario and a source instance can be computed in polynomial time by resorting to the classical *chase procedure* [15]. A solution generated by the chase is called a *canonical solution*. In light of this, we may say that early mapping systems were restricted to generate *canonical pre-solutions*, since they chased s-t tgds only.

3.2 Tools of the Intermediate Generation

Once the theory of data-exchange had become more mature, it was clear that producing solutions of quality was a critical requirement. The notion of a *core solution* [17] was formalized as the "optimal" solution, since it is universal, and among the universal solutions is the one of the smallest size. In our example, the core solution is the one reported in Figure 2.d. Sophisticated algorithms were developed [17,26] to post-process a canonical solution generated by a schema-mapping tool, and minimize it to find its core [36]. These tools have the merit of being very general, but fail to be scalable: even though the algorithms are polynomial, their implementation requires to couple complex recursive computations with SQL to access the database, and therefore hardly scale to large databases. In

fact, empirical results show that they are hardly usable in practice due to unacceptable execution times for medium size databases [32].
It was therefore clear that, in order to preserve the effectiveness and generality of mapping tools, reasoning about the mapping was necessary. First, a number of approaches were proposed to optimize schema mappings in order to improve the efficiency of their execution and manipulation in real-world applications. In fact, schema mappings may present redundancy in their expressions, due for example to the presence of unnecessary atoms or unrelated variables, thus negatively affecting the data management process [16,36]. Following efforts have aimed at optimizing such dependencies by identifying two kinds of equivalence aside from standard logical equivalence: the relaxed notions of data-exchange (DE) equivalence and conjunctive-query (CQ) equivalence [16]. DE and CQ equivalences coincide with logical equivalence when the mapping scenario is made only of s-t tgds (i.e., $\Sigma = \Sigma_{st}$), but differ on richer classes of equivalences, such as second-order tgds, and scenarios with both Σ_{st} and Σ_t.
A different approach to the generation of core solutions was undertaken in [32,41]. In these proposals, scalability is a primary concern. Given a mapping scenario composed of source-to-target tgds (s-t tgds), the goal is to rewrite the tgds in order to generate a runtime script, for example in SQL, that, on input instances, materializes core solutions. This is a key requirement in order to embed the execution of the mappings in more complex application scenarios, that is, in order to make data-exchange techniques a real "plug and play" feature of integration applications. +Spicy [32] is an example of mapping tool of this generation. These works exploit the use of *negation* in the premise of the s-t tgds to rewrite them intercepting possible redundancy. Consider again our running example; algorithms for SQL core-generation would rewrite m_1 to make sure that no redundant data are copied to the target from the relation *Subscriber*:

$$m'_1.\ \mathit{Subscriber}(n) \wedge \neg(\mathit{MailingList}(n, E)) \wedge \\ \neg(\mathit{Client}(n, A)) \rightarrow \mathit{Person}(n, Y_1, Y_2)$$

Experiments [32] show that, in the computation of the core solution, with executable scripts there is a gain in efficiency of orders of magnitude with respect to the post-processing algorithms. This is not surprising, as these mapping rewriting approaches preserve the possibility to execute transformations in standard SQL, with the guarantee of scalability to large databases and of portability to existing applications.

However, these tools still have some serious limitations, that prevent their adoption in real-life scenarios. We may summarize these limitations as follows.

(*a*) *They have limited support for target constraints.* Handling target constraints – i.e., keys and foreign keys, represented by *egds* and *target tgds* [15], respectively – is a crucial requirement in many mapping applications. Notice that foreign-key constraints were at the core of the original schema-mapping algorithms, and, under appropriate hypothesis, can always be rewritten as part of the source-to-target tgds [16]. Unfortunately this is not the case for target edgs.

Consider again the running example; the best a tool from this generation can obtain with executable scripts is the core pre-solution reported in Figure 2.c, where the redundancy coming from the source-to-target tgds has been removed, but the solution lacks the enforcement of the target key constraints.

(*b*) *They are limited to relational scenarios, and cannot handle XML or nested datasets.* This is a consequence of the fact that data-exchange research has primarily concentrated on the relational setting, and for a long time no notion of data exchange for more complex models was available. In a way, this is a setback with respect to the early systems, which had supported nested relations since the beginning with a pragmatical approach. In fact, they were able to produce results for XML setting, but without the precise definition of quality that core solutions provide. It is interesting to note that benchmarks for mapping systems have been proposed [2,6]. However, tools of the intermediate generation cannot be fully evaluated using such benchmarks – for example in order to compare the quality of their solutions – since several scenarios in the benchmarks refer to nested structures, and these systems are not capable to generate core solutions for a nested data model.

4 Time for a Golden Age: Data Cleaning

Recent results have faced these problems and paved the way towards the emergence of a fully-fledged new-generation of schema-mapping and data-exchange tools.
An important aspect is the extension to nested relations and XML. The theoretical properties of data exchange in a general XML setting have been recently studied [4,13,3], and, due to the generality, have been shown to exhibit several negative properties. However, important results were established for the fragment of XML data exchange in which the data model is restricted to correspond to nested relations [38]. A very important result was reported in [13]: the authors show that the generation of universal solutions for a nested scenario can be reduced to the generation of solutions for a traditional, relational scenario, even in the presence of target constraints. The authors also provide an algorithm to perform the reduction.
A second important advancement is related to the management of functional dependencies over the target. Although it is not always possible, in general, to enforce a set of egds using a first-order language as SQL, it has been proposed a best-effort algorithm that rewrites the above mapping into a new set of s-t tgds that directly generate the target tuples that are produced by chasing the original tgds first and then the egds [30,31]. As egds merge and remove tuples from the pre-solution, to correctly simulate their effect the algorithm puts together different s-t tgds and uses negation to avoid the generation of unneeded tuples in the result. It has been recently shown how rewriting the subclass of egds defined by Functional Dependencies (FDs) can also lead to significant reductions of the execution times in the solution computation [11].

The ability to handle functional dependencies paves the way to the application of schema mappings for *Data-fusion*. Consider again the example in Figure 1 and its tgds. It can be considered as a typical data-fusion scenario, in which it is required to merge together data from three different source databases into a common target. However, by using the s-t tgds only, the best we can achieve is to generate a core pre-solution, as shown in Figure 2.c. This solution can be generated efficiently, but it violates the required key constraints and suffers from an unwanted *entity fragmentation* effect: information about the same entities (e.g., *Abi*, *Perry*, or the account number *001/25*) is spread across several tuples, each of which gives a partial representation of the entity. If we take into account the usual dimensions of data quality [10], it is clear that such an instance must be considered of low quality in terms of compactness (or minimality). Based on these requirements, it is natural to desire the generation of a solution as the one shown in Figure 2.d, where the null values are complemented by constants [10]. Such optimal solution can be materialized by chasing the dependencies above with a post-processing step to minimize the pre-solution, but in practice there are orders of magnitude between the execution time needed to compute the pre-solution and the one needed to achieve the optimal one (e.g., seconds vs hours for the same database) [32].

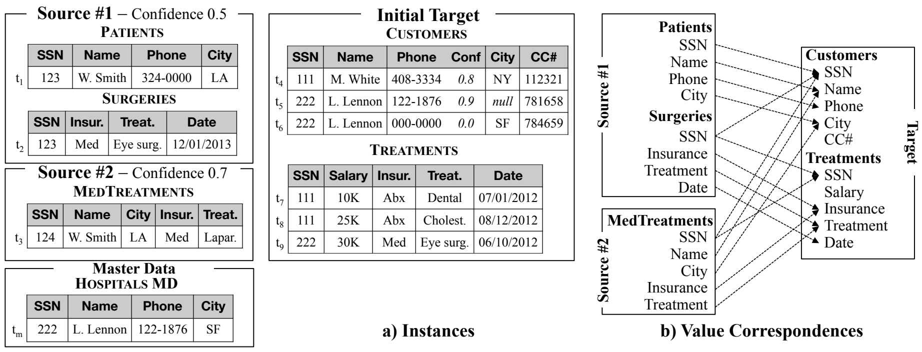
As discussed above, there is a large class of cases where the best-effort algorithm is able to rewrite the given mapping into a new set of s-t tgds that enforce the target egds [30]. However, these algorithms handle the traditional data-exchange case: when two constant values conflict in the conclusion of an egd, the chase fails as there is not valid solution to the problem. Unfortunately, in practice it is very common to have conflicting values, and a lot of effort is spent for their removal. The challenge of *cleaning data* has motivated a lot of research, and it is very interesting to see how schema mappings and data exchange principles have proven to be fundamental tools to attack also this problem, as we discuss next.

4.1 Mapping and Cleaning

Commercial data cleaning systems are based on approaches in which cleaning actions have to be explicitly specified by users by using transformation operations. They usually focus on data profiling, to identify data quality issues[6], and record matching, to remove duplicate entities[7], by using ad-hoc techniques and rules with special attention for specific types of data, such as addresses or phone numbers. A principled approach to data cleaning is based on constraints [22,20]. Data repairing uses declarative constraints, like functional and inclusion dependencies, to detect and repair inconsistencies in the data.
It is natural to think of data exchange and data repairing as two strongly interrelated activities. In fact, the source databases are often structured according to different conventions and rules, and may be dirty. As a

[6] http://www.trifacta.com, http://www.informatica.com/PowerCenter
[7] http://www.tamr.com, http://www.ibm.com/software/products/en/ibminfoqual

consequence, inconsistencies and errors often arise when the sources are brought together into a target schema that comes with its own integrity constraints.
For long time, the database literature has studied these two problems in isolation, with the consequence that there was neither a clear semantics, nor adequate techniques to handle data translation and data repairing in an integrated fashion. One might expect that pipelining data exchange algorithms [15] and data repairing algorithms like those in [20] is sufficient. Unfortunately, this is not the case. In fact, it has been shown that schema mappings and data quality constraints interact in such a way that simply pipelining the two semantics often does not return solutions [24].


Source #1 – Confidence 0.5

PATIENTS

	SSN	Name	Phone	City
t_1	123	W. Smith	324-0000	LA

SURGERIES

	SSN	Insur.	Treat.	Date
t_2	123	Med	Eye surg.	12/01/2013

Source #2 – Confidence 0.7

MEDTREATMENTS

	SSN	Name	City	Insur.	Treat.
t_3	124	W. Smith	LA	Med	Lapar.

Master Data

HOSPITALS MD

	SSN	Name	Phone	City
t_m	222	L. Lennon	122-1876	SF

Initial Target

CUSTOMERS

	SSN	Name	Phone	Conf	City	CC#
t_4	111	M. White	408-3334	*0.8*	NY	112321
t_5	222	L. Lennon	122-1876	*0.9*	*null*	781658
t_6	222	L. Lennon	000-0000	*0.0*	SF	784659

TREATMENTS

	SSN	Salary	Insur.	Treat.	Date
t_7	111	10K	Abx	Dental	07/01/2012
t_8	111	25K	Abx	Cholest.	08/12/2012
t_9	222	30K	Med	Eye surg.	06/10/2012

a) Instances

Fig. 3. A Hospital Mapping and Cleaning Scenario.

Consider the data scenario shown in Figure 3. Several different hospital-related data sources must be correlated to one another. The first repository has information about Patients and Surgeries. The second one about MedTreatments. Our goal is to move data from the source database into a target database that organizes data in terms of Customers with their addresses and credit-card numbers, and medical Treatments paid by insurance plans. The mappings are informally specified under the form of value correspondences in Figure 3.b, that are then translated into a set of s-t tgds.
Besides deciding how to populate the target to satisfy the mappings above, users must also deal with the problem of generating instances that comply with target constraints, as follows.

(i) *Functional and Inclusion Dependencies*: Traditionally, database architects have specified constraints of two forms: inclusion constraints and functional dependencies. In our example, we have a foreign-key constraint stating that the *SSN* attribute in the Treatments table references the *SSN* of a customer in Customers. The target database also comes with a number of functional dependencies: $d_1 = (SSN, Name \rightarrow Phone)$, $d_2 = (SSN, Name \rightarrow CC\#)$ and $d_3 = (Name, City \rightarrow SSN)$ on table Customers. Here, d_1 requires that a customer's social-security number (*SSN*) and name uniquely determine his or her phone number (*Phone*). Similarly for d_2 and d_3. Differently for traditional data exchange, there is no assumption that the target database is empty. In fact, in Figure

3.a we have reported an instance of the target. There, the pair of tuples $\{t_5, t_6\}$ violates both d_1 and d_2; the database is thus not consistent wrt the constraints, i.e., "dirty".

(*ii*) *Conditional Dependencies*: Besides standard functional and inclusion dependencies, more advanced forms of constraints are often necessary [20]. Therefore, an expressive data-cleaning tool needs to support a larger class of data-quality rules. Here we mention *conditional functional dependencies* and *conditional inclusion dependencies.* We assume two conditional functional dependencies (CFDs): (*i*) a CFD $d_4 = (\mathit{Insur}[\text{Abx}] \rightarrow \mathit{Treat}[\text{Dental}])$ on table Treatments, expressing that insurance company *'Abx'* only offers dental treatments (*'Dental'*). Tuple t_8 violates d_4, adding more dirtiness to the target database. (*ii*) In addition, we also have an *inter-table* CFD d_5 between Treatments and Customers, stating that the insurance company *'Abx'* only accepts customers who reside in San Francisco (*'SF'*). Tuple pairs $\{t_4, t_7\}$ and tuples $\{t_4, t_8\}$ violate this constraint.

(*iii*) *Master Data and Editing Rules*: Finally, as it is common in corporate information systems, an additional *master-data table* is available; this table contains highly-curated records whose values have high accuracy and are assumed to be clean. We also consider an additional *editing rule*, d_6, stating that whenever a tuple t in Customers agrees on the *SSN* and *Phone* attributes with some master-data tuple t_m in Hospitals MD, then the tuple t must take its *Name* and *City* attribute values from t_m, i.e., $t[\mathit{Name, City}] = t_m[\mathit{Name, City}]$. Tuple t_5 does not adhere to this rule as it has a missing city value (NULL) instead of *'SF'* as provided by the master-data tuple t_m.

As in traditional data exchange, the example requires to map different source databases into a given target, but assumes that the target database may be non-empty, and that the sources and the target instances may generate inconsistencies when the mappings are executed. Given the source and the target instances, the goal is to generate a target instance that satisfies the mappings and the target constraints.

Interestingly, the data cleaning constraints listed above can be all expressed with a specific form of egds. More specifically, besides relation atoms, this form considers *equation atoms* of the form $t_1 = t_2$, where t_1, t_2 are either constants in CONST or variables, and allows for both source and target atoms in the premise. Examples of translation to egds for the constraints in our running example are expressed as follows:

$$\begin{array}{l}
d'_1.\ \mathsf{Customers}(\boldsymbol{s}, \boldsymbol{n}, p, c, k), \mathsf{Customers}(\boldsymbol{s}, \boldsymbol{n}, p', c', cc') \rightarrow p = p' \\
d'_4.\ \mathsf{Treatments}(s, a, i = \text{`}\mathit{Abx}\text{'}, t, d) \rightarrow t = \text{`}\mathit{Dental}\text{'} \\
d'_6.\ \mathsf{Customers}(\boldsymbol{ssn}, n, \boldsymbol{p}, c, k), \mathsf{HospMD}(\boldsymbol{s}, n', \boldsymbol{p}, c') \rightarrow n = n' \\
d''_6.\ \mathsf{Customers}(\boldsymbol{ssn}, n, \boldsymbol{p}, c, k), \mathsf{HospMD}(\boldsymbol{s}, n', \boldsymbol{p}, c') \rightarrow c = c'
\end{array}$$

A new unified semantics has been proposed to handle this kind of constraints, together with source-to-target and target tgds [24]. The semantics is a conservative extension of the one of data exchange and incorporates many of the features found in data-repairing algorithms. The new semantics not only preserves the fundamental concepts of data exchange, such as the preference relationship among alternative solutions [29], but also naturally extends the chase procedure to define an algorithms that produces solutions for the new setting. As expected from the negative result on the enforcing of target "classic" egds, executable scripts, such as SQL, cannot express the full recursive power that is needed to solve data cleaning scenarios. This challenge has motivated new recent work on chase engines, with experimental results that show scalability in the computation of solution for mapping scenarios with millions of tuples [8].

4.2 More applications: ETL

We now briefly discuss an example of a novel applications for the recent advancements in mapping systems.
The most widely used systems in data warehousing environments to express data transformations as a composition of operators in a procedural fashion are known as ETL tools. Operators vary from simple data mappings between tables to more complex manipulations, such as joins, splits of data, and merging of data from different sources. Usually, these tools are used by developers that want to achieve an efficient implementation of a data exchange task. The superior diffusion of ETL systems compared to mapping systems is due to their richer semantics, which allow them to express more operations [14], and to the declarative nature of schema mapping tools that can become a limit with complex transformations where many intermediate steps to manipulate data are needed [33]. For this reason, it became important to study scenarios where flows of mappings, defined using simple intermediate results, are preferable to a single, monolithic mapping with a large number of complex s-t tgds. Preliminary results fill part of this gap by introducing several novel features that allow the possibility to manipulate flows of mappings. This is done by enabling their composition in sequences over intermediate results [18], the ability to invert mappings [5] and to mix data and metadata in them [28], the automatic combination of "parallel" mappings [1], and the reuse of transformations defined in similar settings as components [42]. Moreover, the new results discussed above enable not only to enforce functional dependencies in the target with schema mappings only [30], but also to pair mapping and cleaning specification to ultimately obtain better quality in the solutions [24].
We can give the intuition of how the expression of data exchange scenarios by mapping tools is preferable to ETL systems in terms of easiness of use by comparing a very simple scenario implemented with the two paradigms. Consider the source schema with relations *Students*(name,

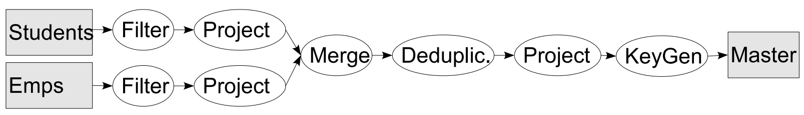


Fig. 4. A simple ETL graph.

birthdate, course, program) and *Emps*(name, dept, role), and the target schema *Master*(id, name, birthdate). The desired transformation joins employees and students in order to create a new database where supervisors with a master of science are assigned a new id. The scenario is informally depicted as an ETL transformation in Figure 4 to show how many small procedural steps are needed in order to generate the desired target instance with an ETL approach, while in a mapping system, given the two schemas, only two lines in a GUI and two manually entered strings are required for it, as exemplified by the following s-t tgd:

$$\begin{aligned} m_a.\ & \mathit{Students}(n_1, b_1, c_1, p_1) \wedge \mathit{Emps}(n_1, d_1, r_1) \\ & \wedge(p_1 = \text{`}\mathtt{Msc}\text{'}) \wedge (r_1 = \text{`}\mathtt{S}\text{'}) \rightarrow \mathit{Master}(I_i, n_1, b_1) \end{aligned}$$

While, in general, ETL tools are more sophisticated than schema-mapping ones and can handle a larger class of problems, in some cases schema mappings may provide a more abstract, less labor-intensive way of specifying portions of the transformation. Something very similar also happens for other comparable formalisms (like, for example, graphical editors for complex languages as XQuery or XSLT). This does not mean that schema mappings may replace these alternative formalisms, but certainly opens up a number of opportunities to merge these different approaches.

5 Conclusions

Schema mapping management has become an important research area in data transformation, exchange and integration systems. From the early data translation prototypes, important results have been consolidated, but, despite the good results, the adoption of mapping systems in real-life integration applications has been quite slow. We have shown how emerging trends are overcoming the limits of the initial proposal and are going to encourage the developing of more systems based on schema mappings. On one side, novel theoretical results are paving the way to the creation of innovative applications for real world problems. On the other side, a new generation of tools for the creation and optimization of schema mappings are widening the opportunities offered by such technology.
As we discussed in the previous section, we believe that there are quite a lot of interesting research opportunities in this area. Notable examples of the new generation of tools are ++Spicy [31] and Llunatic [25]. These tools can deal with different data management tasks, including data fusion, data cleaning and ETL scenarios, which represent very promising areas of application of the latest schema-mappings and data-exchange techniques.

References

1. B. Alexe, M. A. Hernández, L. Popa, and W. C. Tan. MapMerge: Correlating independent schema mappings. *PVLDB*, 3(1):81–92, 2010.
2. B. Alexe, W. Tan, and Y. Velegrakis. Comparing and Evaluating Mapping Systems with STBenchmark. *PVLDB*, 1(2):1468–1471, 2008.

3. S. Amano, C. David, L. Libkin, and F. Murlak. XML schema mappings: Data exchange and metadata management. *J. ACM*, 61(2):12:1–12:48, 2014.
4. M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. *J. of the ACM*, 55(2):1–72, 2008.
5. M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Query language-based inverses of schema mappings: semantics, computation, and closure properties. *The VLDB Journal*, 21(6):823–842, 2012.
6. P. C. Arocena, B. Glavic, R. Ciucanu, and R. J. Miller. The ibench integration metadata generator. *PVLDB*, 9(3):108–119, 2015.
7. C. Beeri and M. Vardi. A Proof Procedure for Data Dependencies. *J. of the ACM*, 31(4):718–741, 1984.
8. M. Benedikt, G. Konstantinidis, G. Mecca, B. Motik, P. Papotti, D. Santoro, and E. Tsamoura. Benchmarking the chase. In *PODS*, 2017.
9. P. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *SIGMOD*, pages 1–12, 2007.
10. J. Bleiholder and F. Naumann. Data fusion. *ACM Comp. Surv.*, 41(1):1–41, 2008.
11. A. Bonifati, I. Ileana, and M. Linardi. Functional Dependencies Unleashed for Scalable Data Exchange. In *SSDBM*, 2016.
12. A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *EDBT*, pages 85 – 96, 2008.
13. R. Chirkova, L. Libkin, and J. Reutter. Tractable XML Data Exchange via Relations. In *CIKM*, 2011.
14. S. Dessloch, M. A. Hernandez, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316, 2008.
15. R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.
16. R. Fagin, P. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *ACM PODS*, pages 33–42, 2008.
17. R. Fagin, P. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.
18. R. Fagin, P. Kolaitis, L. Popa, and W. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *ACM TODS*, 30(4):994–1055, 2005.
19. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. *Schema Matching and Mapping*, chapter Schema Mapping Evolution Through Composition and Inversion, pages 191–222. Springer, 2011.
20. W. Fan and F. Geerts. *Foundations of Data Quality Management.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
21. A. Fuxman, M. A. Hernández, C. T. Howard, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *VLDB*, pages 67–78, 2006.
22. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

23. F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9), 2013.
24. F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and Cleaning. In *ICDE*, pages 232–243, 2014.
25. F. Geerts, G. Mecca, P. Papotti, and D. Santoro. That's all folks! LLUNATIC goes open source. *PVLDB*, 7(13):1565–1568, 2014.
26. G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *J. of the ACM*, 55(2):1–49, 2008.
27. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: from Research Prototype to Industrial Tool. In *SIGMOD*, pages 805–810, 2005.
28. M. A. Hernández, P. Papotti, and W. C. Tan. Data Exchange with Data-Metadata Translations. *PVLDB*, 1(1):260–273, 2008.
29. B. Kimelfeld, E. Livshits, and L. Peterfreund. Detecting ambiguity in prioritized database repairing. In *ICDT*, 2017.
30. B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.
31. B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *PVLDB*, 4(11):1438–1441, 2011.
32. G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *SIGMOD*, pages 655–668, 2009.
33. G. Mecca, P. Papotti, S. Raunich, and D. Santoro. What is the IQ of your Data Transformation System? In *CIKM*, pages 872–881, 2012.
34. G. Mecca, G. Rull, D. Santoro, and E. Teniente. Semantic-based mappings. In *Conceptual Modeling - 32th International Conference, ER 2013, Hong-Kong, China, November 11-13, 2013. Proceedings*, pages 255–269, 2013.
35. R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB*, pages 77–99, 2000.
36. R. Pichler and V. Savenkov. DEMo: Data Exchange Modeling Tool. *PVLDB*, 2(2):1606–1609, 2009.
37. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
38. A. Roth, M. F. Korth, H. and A. Silberschatz. Extended Algebra and Calculus for Nested Relational Databases. *ACM TODS*, 13:389–417, October 1988.
39. L. Seligman, P. Mork, A. Halevy, K. Smith, M. J. Carey, K. Chen, C. Wolf, J. Madhavan, A. Kannan, and D. Burdick. OpenII: an Open Source Information Integration Toolkit. In *SIGMOD*, pages 1057–1060, 2010.
40. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing and REstructuring System. *ACM TODS*, 2(2):134–174, 1977.
41. B. ten Cate, L. Chiticariu, P. Kolaitis, and W. C. Tan. Laconic Schema Mappings: Computing Core Universal Solutions by Means of SQL Queries. *PVLDB*, 2(1):1006–1017, 2009.
42. R. Wisnesky, M. A. Hernández, and L. Popa. Mapping polymorphism. In *ICDT*, pages 196–208, 2010.