A Short History of Schema Mapping Systems (Extended Abstract)

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

[1] Università della Basilicata – Potenza, Italy
[2] Qatar Computing Research Institute (QCRI) – Doha, Qatar

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 [5].

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 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 or XQuery scripts. This latter feature is a key requirement in order to embed the execution of the mappings in more complex application scenarios, that is, in order to make mappings a plug and play component of integration systems.

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. 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 ten years ago with the seminal papers about the Clio system [17, 19]: a first generation of tools was proposed to support the process of generating complex logical dependencies – typically *tuple-generating dependencies* [4] – 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., [13]). 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* [5, 8, 19] formally studies the semantics of generating an instance of a target database given a source database and a set of mappings. It has formalized the notion of a *data exchange problem* [8], 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* [10] was identified as the "optimal" solution, since it is the smallest among the solutions that preserve the semantics of the mapping.

In the last three years an intermediate generation of tools [16, 21] have 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, has been quite slow. This seems to be due to three main factors: (*a*) these systems were not able, at first, to handle functional dependencies over the target, which 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 available to the community.

The Golden Age A number of recent results [14, 7], along with the public availability of the first open-source mapping tools – like ++Spicy [15] and OpenII [20] – seem to be a promising starting point towards the solution of these problems and the beginning of a new, golden age for mapping tools. These works, along with others [1, 11], are giving new vitality to schema-mappings research and suggest new applications, beyond traditional data exchange and data integration tasks.

In the following, we first describe the three ages in Section 2 and we then draw some conclusions in Section 3.

2 A History of Schema Mappings In Three Ages

2.1 The Heroic Age

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 [12, 13, 17, 19] and systems [6, 20] which incorporate a Clio-like first-generation mapping module. We 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.

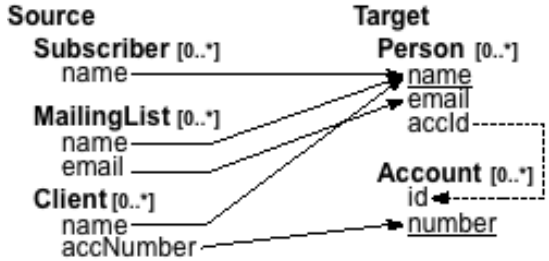

Fig. 1. Schema mapping scenario.

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 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.

The mapping scenario in Figure 1 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 tgds:

Source-to-Target Tgds
$m_1.\ \forall n\colon \mathit{Subscriber}(n) \rightarrow \exists Y_1, Y_2 : \mathit{Person}(n, Y_1, Y_2)$
$m_2.\ \forall n, e\colon \mathit{MailingList}(n, e) \rightarrow \exists Y_1 : \mathit{Person}(n, e, Y_1)$
$m_3.\ \forall n, acc\colon \mathit{Client}(n, acc) \rightarrow \exists Y_1, Z\colon (\mathit{Person}(n, Y_1, Z) \wedge \mathit{Account}(Z, acc))$

In addition, the following egds translate the key constraints on the target schema:

TARGET EGDS
e_1. $\forall n, e, a, e', a' : Person(n, e, a) \wedge Person(n, e', a') \rightarrow (e = e') \wedge (a = a')$
e_2. $\forall n, i, i' : Account(i, n) \wedge Account(i', n) \rightarrow (i = i')$

Mapping Execution via Scripts To execute the mappings, schema-mapping systems rely on the traditional *chase procedure* [8]. 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 [19] 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.

2.2 The Silver Age

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.

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 [8]. 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 [8].

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* [8] 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

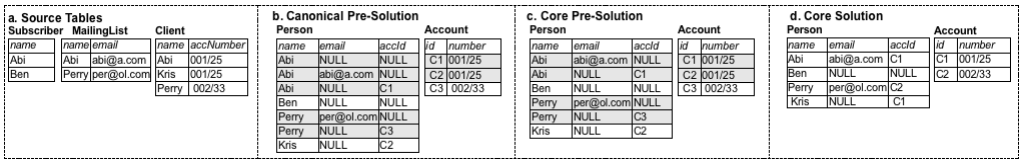

Fig. 2. Source instance (a) and three possible solutions (b-d).

given source instance: each tgd only states an inclusion constraint 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* [8]. 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.

Tools of the Intermediate Generation. Once the theory of data-exchange had become mature, it was clear that producing solutions of quality was a critical requirement. The notion of a *core solution* [10] 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 reported in Figure 2.d.

Sophisticated algorithms were developed to post-process a canonical solution generated by a schema-mapping tool, and minimize it to find its core [10, 18]. 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 do not scale nicely 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 [16].

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 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 [9, 18].

A different approach to the generation of core solutions was undertaken in [16, 21]. 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 [16] 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 [16] 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* [8], 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 [9]. 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 a benchmark for mapping systems has been recently proposed [2]. However, none of the tools of the intermediate generation can be evaluated using the benchmark – for example in order to compare the quality of their solutions – since most of the scenarios in the benchmark refer to nested structures, and these systems are not capable to generate core solutions for a nested data model.

2.3 Time for a Golden Age

Recent results have faced these problems and paved the way towards the emergence of a new-generation of schema-mapping and data-exchange tools.

A first 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 [14]. 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. Other approaches and semantics for the rewriting of s-t tgds have also been recently introduced [1].

Another 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 [3, 7], 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. A very important result was reported in [7]: 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.

We believe that these recent results, together with important theoretical studies, such as the ones on mapping invertion [11], open new possibilities for research on schema mappings. A notable example of a new generation of tool has been recently presented [15]. The ++Spicy tool can deal with different data management tasks, including data fusion, data cleaning and ETL scenarios, which, in our opinion, represent very promising areas of application of the latest schema-mappings and data-exchange techniques.

3 Conclusions

Schema mapping management is an important research area in data transformation, exchange and integration systems. From the early prototypes developed ten years ago, important results have been consolidated, but, despite the good results, the adoption of mapping systems in real-life integration applications has been 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.

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. M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. *J. of the ACM*, 55(2):1–72, 2008.
4. C. Beeri and M. Vardi. A Proof Procedure for Data Dependencies. *J. of the ACM*, 31(4):718–741, 1984.
5. P. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *SIGMOD*, pages 1–12, 2007.
6. A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *EDBT*, pages 85 – 96, 2008.
7. R. Chirkova, L. Libkin, and J. Reutter. Tractable XML Data Exchange via Relations. In *CIKM*, 2011.
8. R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.
9. R. Fagin, P. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *ACM PODS*, pages 33–42, 2008.
10. R. Fagin, P. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.
11. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. *Schema Matching and Mapping*, chapter Schema Mapping Evolution Through Composition and Inversion. 2011.
12. 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.
13. 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.
14. B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.
15. 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.
16. G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *SIGMOD*, 2009.
17. R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB*, pages 77–99, 2000.
18. R. Pichler and V. Savenkov. DEMo: Data Exchange Modeling Tool. *PVLDB*, 2(2):1606–1609, 2009.
19. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.
20. 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.
21. 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.