++Spicy: an Open-Source Tool for Second-Generation Schema Mapping and Data Exchange

Bruno Marnette[1*] Giansalvatore Mecca[2] Paolo Papotti[3]
Salvatore Raunich[4] Donatello Santoro[2,3]

[1] INRIA Saclay & ENS Cachan, France – [2] Università della Basilicata, Potenza, Italy
[3] Università Roma Tre, Roma, Italy – [4] University of Leipzig, Leipzig, Germany

ABSTRACT

Recent results in schema-mapping and data-exchange research may be considered the starting point for a new generation of systems, capable of dealing with a significantly larger class of applications. In this paper we demonstrate the first of these second-generation systems, called ++Spicy. We introduce a number of scenarios from a variety of data management tasks, such as data fusion, data cleaning, and ETL, and show how, based on the system, schema mappings and data exchange techniques can be very effectively applied to these contexts. We compare ++Spicy to the previous generations of tools, to show that this is much-needed advancement in the field.

1. INTRODUCTION

There are many different classes of applications that need to exchange, correlate, or integrate data. An essential requirement of these applications is that of manipulating *mappings* between sources. Mappings are executable transformations that specify how an instance of the source repository should be translated into an instance of the target repository. We may identify two broad research lines in the recent literature.

On one side, we have studies on practical tools and algorithms for *schema-mapping generation*. In this case, the focus is on the development of systems that take as input an abstract specification of the mapping, usually made of a bunch of correspondences between the two schemas, and generate the mappings – typically under the form of *tuple-generating dependencies*(*tgds*) [3] – and the executable scripts needed to perform the translation. This research topic was largely inspired by the seminal papers about the Clio system [19, 21].

On the other side, we have theoretical studies about *data exchange*. Several years after the initial schema-mapping algorithms had been proposed, researchers developed a rich body of research in which the notion of a *data exchange problem* [9] was formalized, and a number of theoretical results were established. In this context, the focus is not on the generation of the mappings, but rather on the characterization of their properties and of their solutions.

We can sketch a line of evolution in schema-mappings and data exchange systems, through three main generations.

First-Generation Systems The first generation of schema-mapping systems – primarily Clio [21], but also HePToX [5], and the early version of Spicy [6][1] – were focused on the process of generating complex logical dependencies (i.e., tgds) based on a nice and user-friendly abstraction of the mapping provided by users under the form of value correspondences. It is interesting to note that they proposed a rather general data model, based on nested relations, that allowed for the treatment of both relational and XML-based mapping tasks.

These systems also had a limited data-exchange support, in that they were able to generate scripts (for example in SQL or XQuery) to execute the mappings and materialize a target solution. In this process, several key notions were introduced, like the one of Skolem functions to handle existentially quantified variables [21].

However, at that stage, the systems suffered from a major drawback: these systems failed to generate solutions of good quality. They were mainly restricted to *canonical solutions* [9], which tend to include significant redundancy, as shown in [16, 17].

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 universal solution* [11] was formalized as the "optimal" solution, since it is *universal*, i.e., it does not contain any arbitrary information that does not follow from the source instance and the mappings, and among the universal solutions is the one of the smallest size.

Sophisticated algorithms were developed [11, 13, 14] to post-process a canonical solution generated by a schema-mapping tool, and minimize it to find its core [20]. 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

*Work funded by the ERC grant Webdam

Proceedings of the VLDB Endowment, Vol. 4, No. 12

[1] Notice that also the recent OpenII [23] integration suite – that supports a broader class of integration tasks – incorporates a Clio-like first-generation mapping module.

show that they are hardly usable in practice due to unacceptable execution times for medium size databases [16, 17].

A different approach to the generation of core solutions was undertaken in [24, 16, 17]. 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[2] [16, 18] is an example of mapping tool of this generation.

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* [9], 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 [10]. Therefore, the main problem is represented by key constraints. This intermediate generation of tools cannot handle key constraints in a scalable way. Either they employ the post-processing approach to enforce keys – in which case the computation of the core fails to scale to large databases, as shown in [15] – or were limited to generate scalable SQL script for scenarios with source-to-target tgds only, and no egds [16, 18].

(*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. It is interesting to note that a benchmark for mapping systems has been recently proposed [1]. 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.

++Spicy: a Second-Generation Tool Two recent results have paved the way towards the emergence of a fully-fledged second-generation schema-mapping and data-exchange tool.

(*a*) On the one side [15], the core-oriented rewriting algorithms developed in [16, 18] have been extended to handle a very large class of mapping scenarios with target functional dependencies, i.e., target egds. This is a significant advancement, as keys are very important in all cases in which data coming from different sources needs to be integrated and correlated.

(*b*) On the other side, a theory of XML data exchange [2, 7] was developed. While the XML setting studied in these papers is very general, and, for its generality, leads to several negative results, important properties were established for the fragment of XML data exchange in which the data model is restricted to correspond to nested relations [22]. 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.

++Spicy[3] is the first example of a second-generation mapping tool based on these very recent algorithms. In this demo:

(*i*) we show how ++Spicy 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;

(*ii*) we compare ++Spicy with previous-generation and commercial mapping systems, and show how it is capable to generate optimized SQL and XQuery code in a fully automated way based on a very simple and intuitive graphical specification of the mapping, even in the presence of nested sources and target constraints; with respect to previous systems, which were restricted to rather simplistic and unrealistic examples, or failed to generate compact solutions, we show that the algorithms embedded in the engine enable the management of more realistic and complex tasks;

(*iii*) we show that ++Spicy can efficiently compute core solutions even for large databases and large scenarios; moreover, its capability of producing executable scripts in SQL or XQuery facilitates the process of executing the actual exchange outside of the system, by running the scripts in conventional engines.

Notice that, by the time of the conference, the system will be made freely available under an open-source license.

2. DEMONSTRATION

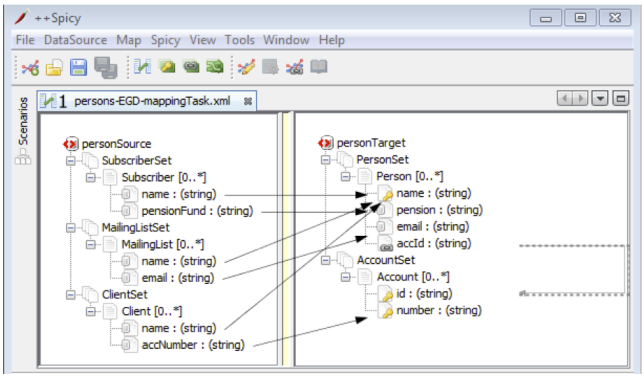

Figure 1: Mapping GUI.

In the demonstration we show how the user can specify mapping tasks with ++Spicy in a natural and friendly way by using the GUI in Figure 1. As it is common, the user loads and manipulates schemas and instances – both relational and XML – with their constraints, possibly adding target keys by dragging and dropping attributes. Then, s/he draws arrows among elements of the schemas in order to define the desired transformation. Such arrows express the

[2]Pronounced "more spicy".

[3]Pronounced "much more Spicy".

equivalence relationship among elements, independently of the underlying data model or of logical design choices; as an alternative, a module is available to load a set of pre-defined tgds in logical form from a text file.

Inputs to the system are a source and a target schema, along with their key and foreign-key constraints, an abstract specification of the mapping in terms of correspondences, and one or more source instances. The output is an executable transformation, that can be run on the source instance to generate a core solution for the target. The system hides the technical details of the tgd generation and rewriting phase, and automatically generates an SQL or XQuery script that can be fed to an external engine (e.g., PostgreSQL, Saxon, etc) to generate the target instance.

In the demonstration we let the audience interact with ++Spicy using a set of schemas including relational ones with keys, nested XML schemas with keys, and scenarios taken from the schema mapping benchmark [1]. In particular, we focus on four classes of scenarios, as discussed in the following paragraphs, to highlight the practical impact of a second generation mapping system. For each scenario, we compare the output of ++Spicy to that of alternative systems (previous generation ones, commercial ones [4]) in terms of quality results and of the execution times.

Data-fusion Consider the *data-fusion* scenario in Figure 2.

a. Source Tables

Subscriber

name	*pFund*
Abi	TRS
Ben	PERS

MailingList

name	*email*
Abi	abi@a.com
Perry	per@ol.com

Client

name	*accNumber*
Abi	001/25
Kris	001/25
Perry	002/33

b. Expected Solution

Person

name	*pension*	*email*	*accId*
Abi	TRS	abi@a.com	C1
Ben	PERS	NULL	NULL
Perry	NULL	per@ol.com	C2
Kris	NULL	NULL	C1

Account

id	*number*
C1	001/25
C2	002/33

c. Pre-Solution (does not enforce keys)

Person

name	*pension*	*email*	*accId*
Abi	TRS	NULL	NULL
Abi	NULL	abi@a.com	NULL
Abi	NULL	NULL	C1
Ben	PERS	NULL	NULL
Perry	NULL	per@ol.com	NULL
Perry	NULL	NULL	C3
Kris	NULL	NULL	C2

Account

id	*number*
C1	001/25
C2	001/25
C3	002/33

Figure 2: Mapping person data.

It requires to merge together financial data from three different source tables (Figure 2.a.): (*i*) a table about subscribers of pension funds; (*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 *Person*, while *number* is a key for *Account*.

Based on the correspondences among elements, as represented in Figure 1, a first-generation mapping system generates for this scenario several s-t tgds, which specify how data should be moved from the source to the target. Target egds can be used to encode the required key constraints on the target, as follows. Note how the third tgd, m_3, invents a value to perform a vertical partition of the *Client* table.

[4]Such as Altova MapForce (`http://www.altova.com/mapforce`) or StylusStudio (`http://www.stylusstudio.com`).

$$m_1.\ \forall n, pf: Subscriber(n, pf) \rightarrow \exists Y_1, Y_2: Person(n, pf, Y_1, Y_2)$$
$$m_2.\ \forall n, e: MailingList(n, e) \rightarrow \exists Y_1, Y_2: Person(n, Y_1, e, Y_2)$$
$$m_3.\ \forall n, acc: Client(n, acc) \rightarrow \exists Y_1, Y_2, Z: (Person(n, Y_1, Y_2, Z) \wedge Account(Z, acc))$$
$$e_1.\ \forall n, p, e, a, p', e', a': Person(n, p, e, a) \wedge Person(n, p', e', a') \rightarrow (p = p') \wedge (e = e') \wedge (a = a')$$
$$e_2.\ \forall n, i, i': Account(i, n) \wedge Account(i', n) \rightarrow (i = i')$$

However, as first-generation systems ignore egds, by using such s-t tgds the best we can achieve is to generate efficiently a *pre-solution*, i.e., a solution for the s-t tgds only, as shown in Figure 2.c. It is easy to see that it is unsatisfactory as 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 *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 [4], such an instance must be considered of very low quality in terms of compactness (or minimality). In fact, on large source instances, the level of redundancy due to entity fragmentation can seriously impair both the efficiency of the translation and the quality of answering queries over the target database.

Based on these requirements, it is natural to desire the generation of a solution as the one shown in Figure 2.b.

During the demo, we show how previous generation systems are unable to generate the desired solution. We also demonstrate how the core solution can be materialized by chasing the dependencies above with a post-processing step to minimize the pre-solution. Unfortunately, existing chase engines that are capable of performing this task hardly scale to large databases [16]. Using the algorithms in [15], the engine in ++Spicy generates an SQL script that is order of magnitude faster (e.g., seconds vs hours for the same database).

STBenchmark STBenchmark [1] proposes various mapping scenarios with nested sources. For each scenario, it proposes one or more sample source instances, and the expected solutions. Interestingly, there are no key constraints in the benchmark. A key observation is that, for all mapping scenarios in the benchmark, the expected solution always corresponds to the core.

During the demonstration, we will compare the various mapping systems, including commercial ones, based on their performance on the mapping tasks in STBenchmark. Some scenarios will be enriched by adding key constraints. In all cases in which sources are nested, relational mapping systems are not applicable. The remaining ones show significant differences in terms of quality of the generated solutions. It is also important to note that there is a significant trade-off between the quality of the output and the amount of effort required to specify the mapping, especially for commercial systems. ++Spicy provides a very good compromise, since it generates core solutions in a scalable way with minimal user interaction.

Data-cleaning Commercial data cleaning systems are based on approaches in which cleaning actions have to be explicitly specified by users using transformation operations. They usually focus on data profiling, to identify data quality issues, and record matching, to remove duplicate entities, by using ad-hoc techniques and rules with special attention for specific types of data, such as addresses or phone

numbers. A more principled approach to cleaning is based on constraints [12]. Consider for instance the database E in Figure 3.a as given. In ++Spicy we let the user de-

a. Employees

name	*age*	*salary*
Paul	1978	NULL
Paul	NULL	29,000
Paul	1979	NULL
Melanie	1990	NULL
Bob	NULL	37,000
Bob	1977	NULL
Charlie	1978	32,000

b. Employees

name	*age*	*salary*
Paul	?	29,000
Melanie	1990	NULL
Bob	1977	37,000
Charlie	1978	32,000

Paul.age = 1978 OR 1979 ?

Figure 3: Data cleaning example.

fine a key constraint for the attribute `name`. To enforce the new constraint, the system rewrites the corresponding egd e. $Employees(n, a, s) \wedge Employees(n, a', s') \rightarrow (a = a') \wedge (s = s')$ into a data exchange from the given database E to a new empty one with the same structure plus the egd e. In this setting, the algorithm described in the previous example produces a schema mapping which outputs the database in Figure 3.b. Then, thanks to the key constraint on the *Employees* table, the system detects the inconsistency on `Paul`'s age, and reports it to the user, which must decide how to handle it by properly curating the data. We will show how the resulting scripts scale extremely well even with large databases with hundreds of thousands of tuples.

ETL ETL tools are widely used in data warehousing environments to express data transformations as a composition of operators in a procedural fashion. 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.

Compared to mapping systems, the superior popularity of ETL systems is due to their richer semantics, which allow them to express more operations [8], and to the declarative nature of schema mapping tools that can become a limit with complex transformations where intermediate steps are needed. For this reason, it is important to support scenarios where flows of mappings, defined using intermediate results, are preferable to a single, monolithic mapping with a large number of complex s-t tgds. ++Spicy allows the design of chains of mappings and introduces functional dependencies in the target, thus enabling operations that were not possible with first-generation mapping tools. We will show how the expression of data exchange scenarios by mapping tools is preferable to ETL systems in terms of easiness of use, without losing efficiency in the execution, by comparing the same scenario implemented with the two paradigms. To give

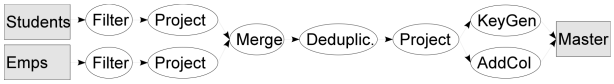

Figure 4: ETL graph.

an intuition of the minimal input required by ++Spicy, consider that for the ETL scenario in Figure 4 only two lines and two labels are required to express the same data exchange scenario, as exemplified by the following s-t tgd:

$$m.\ Students(n_1, b_1, c_1, p_1) \wedge Emps(n_1, d_1, p_1, e_1) \wedge (p_1 = \texttt{`Msc'}) \rightarrow Master(N_1, b_1, d_1, `M')$$

To support complex flows of mappings, ++Spicy introduces two main operators. The first is used for *chaining* mapping scenarios. The second can be used to *merge* the output of different scenarios.

3. REFERENCES

[1] B. Alexe, W. Tan, and Y. Velegrakis. Comparing and Evaluating Mapping Systems with STBenchmark. *PVLDB*, 1(2):1468–1471, 2008.

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

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

[4] J. Bleiholder and F. Naumann. Data fusion. *ACM Comp. Surv.*, 41(1):1–41, 2008.

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

[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. Technical report, North Carolina State University, 2010.

[8] S. Dessloch, M. A. Hernandez, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316, 2008.

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

[10] R. Fagin, P. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *ACM PODS*, pages 33–42, 2008.

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

[12] 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.

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

[14] B. Marnette. Generalized Schema Mappings: From Termination to Tractability. In *ACM PODS*, pages 13–22, 2009.

[15] B. Marnette, G. Mecca, and P. Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.

[16] G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *SIGMOD*, pages 655–668, 2009.

[17] G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings: Scalable Core Computations in Data Exchange. Technical Report Spicy WR-01-2011, Dipartimento di Matematica e Informatica - Università della Basilicata, 2010.

[18] G. Mecca, P. Papotti, S. Raunich, and M. Buoncristiano. Concise and Expressive Mappings with +Spicy. *PVLDB*, 2(2):1582–1585, 2009.

[19] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *VLDB*, pages 77–99, 2000.

[20] R. Pichler and V. Savenkov. DEMo: Data Exchange Modeling Tool. *PVLDB*, 2(2):1606–1609, 2009.

[21] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[22] A. Roth, M. F. Korth, H. and A. Silberschatz. Extended Algebra and Calculus for Nested Relational Databases. *ACM TODS*, 13:389–417, October 1988.

[23] 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.

[24] 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.