The Spicy System: Towards a Notion of Mapping Quality

A. Bonifati,[1,2] G. Mecca[1], A. Pappalardo[1], S. Raunich[1], G. Summa[1]
[1] Dip. di Matematica e Informatica – Univ. della Basilicata – Italy
[2] ICAR – CNR – Italy

ABSTRACT

We introduce the Spicy system, a novel approach to the problem of automatically selecting the best mappings among two data sources. Known schema mapping algorithms rely on value correspondences – i.e. correspondences among semantically related attributes – to produce complex transformations among data sources. Spicy brings together schema matching and mapping generation tools to further automate this process. A key observation, here, is that the quality of the mappings is strongly influenced by the quality of the input correspondences. To address this problem, Spicy adopts a three-layer architecture, in which a schema matching module is used to provide input to a mapping generation module. Then, a third module, the mapping verification module, is used to check candidate mappings and choose the ones that represent better transformations of the source into the target. At the core of the system stands a new technique for comparing the structure and actual content of trees, called structural analysis. Experimental results show that our mapping discovery algorithm achieves both good scalability and high precision in mapping selection.

Categories and Subject Descriptors:
H.2 [Database Management]: Heterogeneous Databases

General Terms: Algorithms, Design, Verification.

Keywords: Mappings, Schema Matching, Mapping Verification.

1. INTRODUCTION

Deriving executable mappings among data sources is nowadays considered a key step in data integration applications [2]. Several systems in recent literature have studied the problem of deriving mappings among sources based on *value correspondences*; each of these correspondences states that an attribute of the target is semantically related to one attribute (or more) in the source, and is usually drawn as a *line* going from the source attribute to the corresponding target attribute (see, for example, [8]).

To alleviate the burden of manually specifying lines, a natural solution is to exploit a *schema matching system*, i.e., a system that automatically or semi-automatically tries to discover matching attributes in a source and target schema. The study of automatic techniques for schema matching has been investigated in many recent works; for a survey see [9, 10]. Unfortunately, schema matching has been recognized as a very challenging problem [6], for which no definitive solution exists: although current schema matching systems perform well in some application categories, in other cases they suffer from poor precision. According to [7], there is no perfect schema matching tool. [7] reports that on a recent benchmark of ontology–matching tasks [1], participating matchers on average achieved 40% precision and 45% recall. Also, even for datasets for which such tools reached higher precision and recall, they still produced inconsistent or erroneous mappings.

As a consequence, outputs of the attribute matching phase are hardly ready to be fed to the mapping generation module, and human intervention is necessary in order to analyze and validate them. It is worth noting that, in general, human intervention is also necessary after mappings have been generated, since several alternative ways of mapping the source into the target may exist. Mapping systems usually produce all alternatives, and offer the user the possibility of inspecting the result of each of them in order to select the preferred one.

Spicy [4], a research effort at Università della Basilicata, is one of the first systems to investigate such coupling in order to automate the generation of executable transformations. Interestingly, the issue of coupling schema matching and mapping generation has received very little attention so far. A crucial observation, here, is that the quality of the mappings produced by the mapping generation system is strongly influenced by the quality of the input lines: starting from faulty correspondences incorrect mappings are inevitably produced, thus impairing the quality of the overall integration process.

The main intuition behind our approach is that in many cases the mapping generation process can be automated to a larger extent by introducing a third step, which we call *mapping verification*, based on a notion of *mapping quality*.

1.1 Mapping Quality

There are several ways to define a notion of quality for schema mappings. These include, for example, the nature of the views used to define the mapping, their efficiency, the expressive power of the mapping language, etc.. These criteria may be considered as "extrinsic" with respect to the mapping scenario. Spicy, on the contrary, adopts a notion of quality that we may consider as "intrinsic" to the mapping scenario.

More specifically, we assume that besides the source schema, the source instance, and the target schema, also one instance of target repository is provided as input to the mapping

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

tool.[1] We consider such instance as a *canonical instance*, i.e., a prototype showing how values should be organized in the target repository. Given a mapping, we evaluate its quality by running the mapping over the given source instance to obtain a translation, and comparing such translation to the canonical instance given as input. The more similar are the two instances, the higher we rank the mapping.

1.2 Mapping Selection

Suppose now we have run a schema matching tool on the source and instance repositories and have obtained a number of candidate lines. In order to discard incorrect correspondences, we may proceed as follows: (*a*) we select a set of candidate correspondences; (*b*) we run the mapping generation algorithm to produce an executable mapping,; (*c*) we execute the transformation to obtain a translation of the source instance; (*d*) we evaluate the quality of the mapping by comparing the translation to the canonical instance. Based on such comparison, we may identify incorrect transformations due to wrong correspondences, and suggest to users the ones that more likely represent correct translations of the source.

So far, the only step in the direction of verifying mappings produced by a mapping generation system has been represented by Spider [5]. Spider is an ad hoc tool that serves as a debugger for the mapping generation process, since it allows the user to trace and inspect mappings step by step during their generation, similarly to source code debuggers. Note that our approach is significantly different from the one pursued in [5], since we aim at reducing human intervention, while human users play a major role in the debugging process.

2. ARCHITECTURE OF SPICY

Our mapping scenarios can be informally described as follows: given (a portion of) a repository T, called the *target*, and a repository S, called the *source*, find a transformation query – i.e., an executable view – that can be used to map instances of S into instances of T. In the process, we aim at minimizing the amount of human intervention. Note that we do not assume that lines are provided in advance.

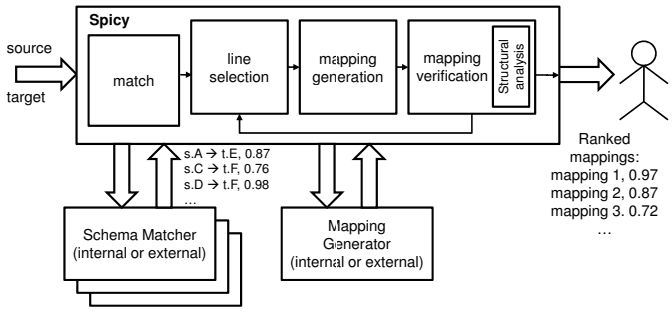

Figure 1: Architecture of Spicy

The overall system architecture is shown in Figure 1. In order to solve the problem, we assume the availability of one or more schema matchers, and of a mapping generation module. Given the variety of schema matching systems available, the system architecture has been designed in such a way that any of those can be easily integrated. In addition, Spicy provides its own instance–based matcher. The system also assumes the availability of a mapping generation module. The mapping generation module takes as input a set of correspondences, and generates a number of mappings under the form of *source to target dependencies (TGDs)* [8]. To derive mappings in Spicy, we have implemented a version of Clio's mapping algorithm [8]. However, other mapping generation tools, like, for example, HepToX [3], could be easily integrated into the system.

Besides pipelining schema matching and mapping generation, Spicy decouples these two modules by a number of intermediate additional modules. These modules coordinate the mapping generation process in order to select the best results.

2.1 Mapping Verification

The first step of our mapping discovery procedure consists of running the schema matcher to produce a number of candidate correspondences. Each correspondence is usually labeled with a similarity measure, i.e., a level of confidence. Note that, in most cases, the schema matching system will not be able to produce a single correspondence for each target attribute. It will rather produce a number of compatible source attributes for each target attribute, with different degrees of similarity. Since in our setting each target attribute must be matched to a single source attribute, we need to consider these as alternative lines to be given as input to the mapping generation module.

Example 1: *Consider for example the data sources in Figure 2. As it is common in data integration systems, data structures are represented in the system using an abstract, graph-based model. It can be seen that the data model is essentially a tree-based representation of a nested-relational model, in which set and tuple nodes alternate. Instances are trees as well; while leaf nodes have atomic values – strings, integers, dates etc. – values for intermediate nodes are oids. Foreign key constraints are drawn as dashed edges going from the foreign key to the corresponding primary key.*

Assume that, when run on the two data sources in our example, the selected schema matcher suggests the following correspondences:

projectDB.project.name →
  companyDB.projects.project.projectName, 0.95
projectDB.project.budget →
  companyDB.projects.project.budget, 0.92
projectDB.project.manager →
  companyDB.divisions.division.managers.manager.name, 0.87
projectDB.project.manager →
  companyDB.divisions.division.employees.employee.name, 0.90

It can be seen how we are deliberately assuming that the schema matcher considers employee names to be highly similar to names of project managers in projectDB*, even more than managers' names in* companyDB*. It is also apparent that such similarities cannot be fed to the mapping generation algorithm as they are; since attribute* project.manager *has been matched to two different source attributes, the system must choose between two different consistent line sets: in both sets project names and budgets are correctly mapped to their counterparts; then, in set* (a) *managers' names are mapped to managers' names, while in set* (b) *to employees' names.*

If we were to select the preferred lines exclusively based on similarity values, we would choose the second candidate set.

[1]This is typical in many settings, like, for example, Web data sources.

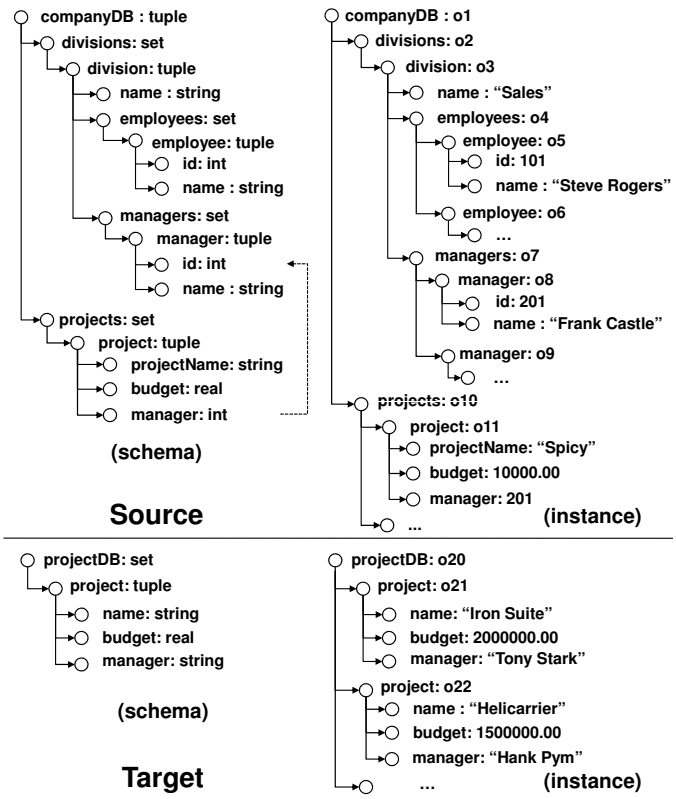

Figure 2: A sample mapping task

However, let us reason for a moment on the two transformations that would be produced.

Set (a) *yields the following transformation (we adopt the syntax used in [8]):*

```
for     d in companyDB.divisions, m in d.managers,
        p in companyDB.projects
where   p.project.manager = m.manager.id
exists  p' in projectDB
where   p'.project.name = p.project.projectName and
        p'.project.budget = p.project.budget and
        p'.project.manager = m.manager.name
UNION
for     d in companyDB.divisions, m in d.managers,
exists  p' in projectDB
        p'.project.manager = m.manager.name
```

The transformation is the union of two TGDs. The first one states that, in order to obtain an instance of the target, we need to join projects and managers in the source, and then produce a tuple for each project name, budget and manager name. The second mapping, on the contrary, maps manager names. However, it only contributes to the result for those managers that do not have projects and therefore do not participate to the join above. If, how it is reasonable, we assume that the vast majority of managers have projects, very few tuples will be generated by the second TGD. Based on these observations, we may say that the translated target instance – which we report in tabular form for space reasons – would look as follows:

name	budget	manager
Spicy	10000.00	Frank Castle
Noodles	15000.00	Scott Summers
...	...	...

Set (b)*, on the contrary, yields a quite different transformation:*

```
for     p in companyDB.projects
exists  p' in projectDB
where   p'.project.name = p.project.projectName and
        p'.project.budget = p.project.budget
UNION
for     d in companyDB.divisions, e in d.employees
exists  p' in projectDB
where   p'.project.manager = e.employee.name
```

In this case, both mappings contribute to the result, because nobody among the employees manages projects. The resulting instance is quite different from the previous one, since the two mappings generate a number of null values:

name	budget	manager
Spicy	10000.00	NULL
Noodles	15000.00	NULL
NULL	NULL	Frank Castle
NULL	NULL	Scott Summers
...	...	...

If we compare this second instance to the target instance shown in Figure 2, we may see how information "flows" quite differently through them. This might allow us to correctly infer that set (a) *more likely represents the correct choice.*

Although this example has been kept very small and rather simple for clarity's sake, we may summarize by saying that schema matchers typically output correspondences that may be assembled in different ways. This leads to different candidate transformations. We advocate that, while an a priori selection of the right correspondences is often very difficult and typically leads to poor results, an a posteriori comparison of the instances produced by the various transformations to the original target instance may give very useful insights on the quality of these transformations, and therefore on the correctness of the associated lines.

A fundamental role in our architecture is played by the *mapping verification module*. Besides the availability of a source schema and instance, we assume that instances of the target data source are also available, and not only its schema; this is typical in many settings, like, for example, Web data sources. In these cases, whenever we select a set of candidate correspondences produced in the schema matching phase, we may think of running the mapping generation algorithm to obtain the transformation induced by these correspondences; then, we may check such transformation by running the query on (a subset of) the source, and comparing the result to the available target instance. The level of similarity obtained by this comparison is taken as an estimate of the *quality* of the mapping. Based on such comparison, we may on the one side identify incorrect transformations due to wrong correspondences produced during the schema matching phase, and on the other side rank the remaining candidates to suggest to users the ones that more likely represent correct translations of the source.

Spicy allows for various verification strategies. All strategies adopt a feature–based approach. Suppose we are given

a portion s of a repository S and a portion t of a repository T; in order to compare them, we might proceed as follows: (i) analyze s using the selected strategy in order to derive a number of descriptive features; (ii) do the same for t; (iii) compare the features computed by the model and return an index of similarity that is proportional to the similarity of the selected features. Besides the more traditional attribute–oriented feature analysis, SPICY provides an innovative comparison strategy called *structural analysis*.

2.2 Structural Analysis

The main intuition behind this choice is that a data structure can be seen as a "generator of information"; in essence, we may imagine that its elements – for example, its atomic fields like strings or numbers – tend to produce a flow of information in the repository – i.e., they generate an internal *stress*; each element, in turn, has some *consistency* due to its nature – for example, numbers may be considered of different consistency with respect to strings or dates; from the interaction between stress and consistencies we may predict the flow of information. Intuitively, two instances will be considered to be similar when information flows similarly through them.

Electrical circuits exhibit nice and elegant properties that make them a very effective means to study similarities about data structures. In [4] we formalize a transformation function, that, when applied to a tree-like portion of a data source, generates an electrical circuit that can be used for the purpose of comparison. One example is shown in Figure 3. For now, let us mention some features of this transformation. For further details we refer the reader to [4].

First, the resulting circuit has a tree-like topology that is isomorphic to that of the original tree; also, the circuit embodies instance-based features; in fact, values for resistors and voltage generators associated with attributes are derived from numerical features of a sample of instance values; hence, it is an ideal means to model the informative content of the data structure.

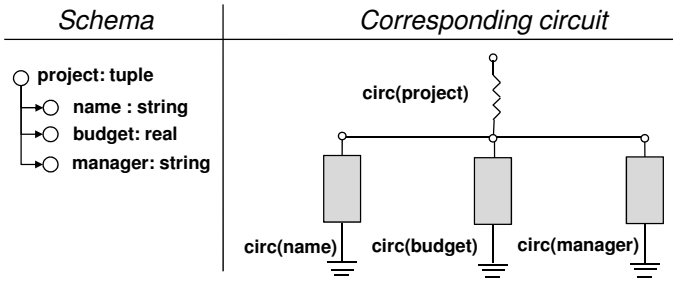

Figure 3: Example of Circuit

Second, circuits are an excellent summarization technique for the purpose of verifying mappings. Recall that our goal is to evaluate the quality of a mapping by checking the similarity of the instance obtained by executing the mapping with respect to the instance originally provided with the target data source. To do this, in principle, we might adopt a rather straightforward attribute-to-attribute comparison approach, as follows: (i) after the translated instance has been generated, we might consider each attribute A_i in the target separately; (ii) take a sample of values for A_i in both the original and the translated instance, and derive a number of numerical features based on this sample; (iii) calculate the overall similarity between the two instances based on the similarity of the various features. However, combining independent similarity measures into a single similarity value is known to be a hard problem [6]. Circuits provide a nice and elegant solution to this problem. They naturally lend themselves to a black-box analysis to characterize the behavior of the two trees, through which we can collect a small number of descriptive parameters – output current, average current, internal voltages – that can be effectively used to measure the similarity of the original structure to other structures. Our experiments show that circuits perform better than attribute-to-attribute feature comparisons.

Finally, the comparison technique based on circuits is fully automatic and does not require any additional inputs; it is also quite efficient, since, for the kind of stationary, continuous current circuits used in the system, solving the circuit amounts to solve a system of linear equations of the form $Ax = b$.

2.3 The Mapping Discovery Algorithm

The first step of our integration procedure consists of running the schema matcher to produce a number of candidate element correspondences. Each correspondence is usually labeled with a similarity measure, i.e., a level of confidence. Note that in most cases the schema matching system will not be able to produce a single correspondence for each target attribute. It will rather produce a number of compatible source attributes for each target attribute, with different degrees of similarity. Since in our setting each target attribute must be matched to a single source attribute, we need to consider these as alternative lines to be given as input to the mapping generation module.

In view of this, our algorithm is based on a feedback loop in which we initially fix a similarity threshold; at each step: (i) we consider only candidate correspondences with similarity above the threshold; (ii) we combine these correspondences into consistent sets, and run the mapping generation algorithm to generate the corresponding transformations; (iii) we use each transformation to translate (a sample of) the source instance into an instance of the target, and then use the mapping verification module to compare the obtained instances to the original target instance; (iv) we rank transformations – and therefore correspondence sets – according to such similarity measure. If no satisfactory mappings are produced at a given iteration, we progressively lower the threshold and analyze more alternative mappings. In essence, using this process, users are not required to manually inspect and select correspondences before mappings are generated; also, when the process stops, the system will present to them a ranked list of source-to-target transformations, suggesting which ones are believed to better reproduce the target.

3. DEMONSTRATION

The system described in the paper has been implemented in a working prototype. The prototype was developed in Java and is equipped with a graphical user interface that allows users to explore all steps of the mapping generation process, as schematically shown in Figure 4.

After the user has chosen a mapping task by loading the source and the target, step 1 consists in the match phase; either the internal or one external schema matcher is run against the two data sources, to provide a number of candi-

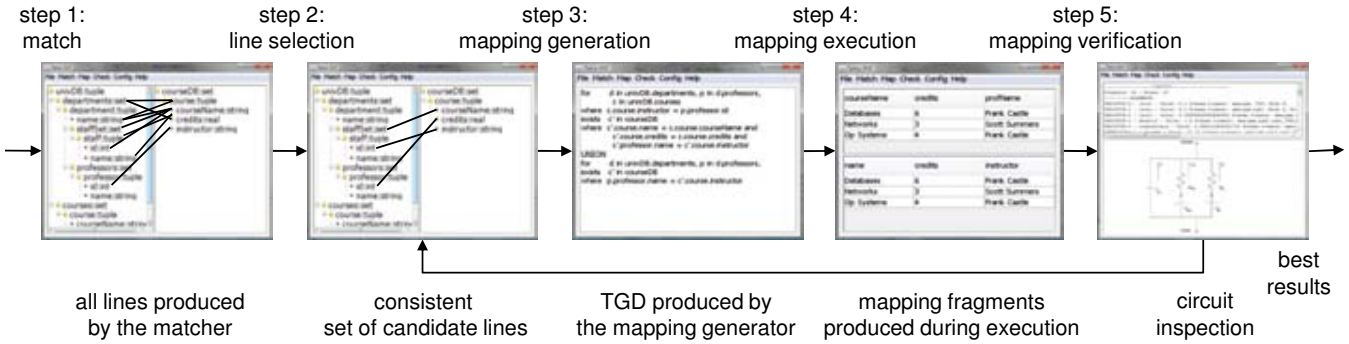

Figure 4: Main Steps in the Demonstration

date lines, each with a given level of confidence. These lines can be inspected and manually edited by the user. Then, the mapping selection iterative process starts. Based on the current threshold, the line selection module selects a number of consistent line subsets (step 2). The user may accept the subsets selected by the module or discard them. Each selected subset is fed to the mapping generation module (step 3) and generates a TGD. At step 4 the TGD is executed by the internal execution engine. During execution, fragments of instances can be inspected to understand how operators are transforming the source instance. Then (step 5) the translated instance is compared to the original target instance to measure the quality of the mapping. At this step the user may see the circuits that have been generated and explore their features. If the quality of the current mapping is considered unsatisfactory, then the process is iterated and new mappings are explored. Otherwise, the mapping is added to the output.

A key issue to validate our approach is to compare the quality obtained by selecting attribute matches *a posteriori*, i.e., after mapping generation and translation, with respect to selecting them *a priori*, i.e., relying only on the output of the attribute matcher. To test this, the system provides several alternative configurations.

Using the *a priori* configuration, the selected attribute matcher is run to find candidate correspondences; then, the unambiguous line sets (one or more) with the highest confidence are selected; attribute similarities can be aggregated in various ways, including average, harmonic mean and euclidean distance.

To assess a posteriori strategies, a second important point is how structural analysis performs with respect to more traditional comparison procedures, based on attribute features alone. To do this, we have several different *a posteriori* configurations.

Using the *attribute–features configuration*, the best line set is chosen a posteriori, as follows: the mapping search algorithm is run as described above; candidate transformations are run on the source to obtain a translated instance of the target and their quality is measured by comparing it to the original target instance in a standard fashion, i.e., by comparing each attribute in the translated instance to its counterpart using instance-based features.

Using the *structural–analysis configuration*, we use structural analysis – i.e., circuits – to measure the quality of mappings.

To compare the various strategies, we have selected a number of experiments, based on different data sources, both relational and XML. Besides well known data sources like DBLP, Mondial[2], and Amalgam[3], we tested several real–life databases serving the information system of our School of Computer Science, and some synthetic datasets. Some of these experiments will be executed during the demonstration to show that a posteriori strategies perform significantly better that a priori ones, and that structural analysis, i.e., circuits, outperforms other attribute–feature configurations.

Acknowledgments

Salvatore Raunich was partially supported by the European Social Fund (Fondo Sociale Europeo).

4. REFERENCES

[1] The Ontology Alignment Evaluation Initiative – 2007. http://oaei.ontologymatching.org/2007/.

[2] P. A. Bernstein and S. Melnik. Model management 2.0: Manipulating richer mappings. In *Proc. of ACM SIGMOD*, pages 1–12, 2007.

[3] A. Bonifati, E. Q. Chang, T. Ho, L. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *Proc. of VLDB*, pages 1267–1270, 2005.

[4] A. Bonifati, G. Mecca, A. Pappalardo, S Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *Proc. of EDBT*, 2008.

[5] L. Chiticariu and W. C. Tan. Debugging Schema Mappings with Routes. In *Proc. of VLDB*, pages 79–90, 2006.

[6] A. Gal. Why is Schema Matching Tough and What We Can Do About It. *Sigmod Record*, 35(4):2–5, 2006.

[7] A. Gal. The Generation Y of XML Schema Matching (Panel Description). In *Proceedings of XML Database Symposium*, pages 137–139, 2007.

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

[9] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB J.*, 10:334–350, 2001.

[10] P. Shvaiko and J. Euzenat. A Survey of Schema Based Matching Approaches. *J. of Data Semantics*, IV - LNCS 3730:146–171, 2005.

[2] http://www.dbis.informatik.uni-goettingen.de/Mondial/

[3] http://www.cs.toronto.edu/~miller/amalgam/