Rule-based Construction of Matching Processes

Eric Peukert
SAP Research Center Dresden
01187 Dresden, Germany
eric.peukert@sap.com

Julian Eberius
Dresden University of Technology
Dresden, Germany
julian.eberius@tu-dresden.de

Erhard Rahm
University of Leipzig
Leipzig, Germany
rahm@informatik.unileipzig.de

ABSTRACT
Semi-automatic schema matching systems have been developed to compute mapping suggestions that can be corrected by a user. However, constructing and tuning match strategies still requires a high manual effort. We therefore propose a self-configuring schema matching system that is able to automatically adapt to the given mapping problem at hand. Our approach is based on analyzing the input schemas as well as intermediate match results. A variety of matching rules use the analysis results to automatically construct and adapt an underlying matching process for a given match task. The evaluation shows that our system is able to robustly return good quality mappings across different mapping problems and domains.

Categories and Subject Descriptors
D.2.12 [**Interoperability**]: Data mapping

General Terms
Algorithms

Keywords
Matching system, rewrite rules, features, matching process

1. INTRODUCTION
Finding mappings between complex metadata structures is a critical task in a number of domains such as data integration, ontology alignment or model transformation. A multitude of schema matching systems and matching algorithms were proposed (see [1] for an overview) to speed up that process. Except for some domain-specific matchers, the algorithms used in the different systems are often similar, e.g. they consider the linguistic and structural similarity of schema elements or the similarity of instance data. Many systems are constructed for a single schema type or domain and may even be tuned for specific benchmarks such as the OAEI Benchmark [5]. Moreover, schema matching publications typically report the maximally achieved quality of automatically computed mapping suggestions using some specially tuned parameter configuration. This construction and tuning is a complex, manual and time-consuming task that often requires substantial matching experience and expert knowledge as well as given perfect mappings.

However, such an approach cannot be adopted for applying a schema matching system in practice onto fully unknown matching problems. Additionally, given default configurations may not be robust enough to cope with largely differing matching problems of diverse domains. Hence, there is a need for adaptive and robust matching systems that return good mappings across different matching problems without manual tuning.

There have already been some attempts to make parts of a matching system more adaptive and self-tuning [9][13][10]. However, these adaptations are typically fixed in the code or seem tailored to specific tasks. Other approaches like eTuner [14] perform tuning based on a synthetically constructed gold standard. However, this approach seems not applicable in practice since a synthetic gold standard could differ a lot from the given mapping problem. Learning techniques like YAM [3] or [7] were also proposed, but in practice, gold mappings for learning are not available. The concept of meta-level learning already aims at adaptive matching processes based on schema features [4]. However they still rely on learning from existing mappings.

We aim at a comprehensive approach for a fully self-configuring schema matching system that can automatically construct and adapt a matching process for a given mapping problem. To this end, we make the following contributions:

- We introduce so-called *features* that are computed from the input schemas as well as from intermediate mapping results.
- Based on the features, we introduce the concept of *matching rules* that represent expert knowledge on how to define or adapt a schema matching process.
- We propose an adaptive matching approach that integrates features and matching rules. A matching process is iteratively extended, rewritten and executed.
- We evaluate our approach on different match problems and show its robustness.

2. PRELIMINARIES
We first provide some preliminaries on the assumed model of schemas and schema matching.

A **schema** consists of a number of schema elements. Each element carries a name, a data type, and optionally a description (called annotation) as well as instances. The kind of schema is not restricted and can refer to any metadata structure such as XML schema trees, ontologies, database schemas or meta-models. The goal of a schema matching system is to compute a mapping suggestion between a source schema S and a target schema T. For computing the mapping, most matching systems use several matchers as well as other operators for aggregation and selection.

The **matcher** operator computes a similarity value for each pair of schema elements from the source schema S and the target schema T and constructs a **similarity matrix** as output. An entry in the similarity matrix is a value between 0 and 1 representing the similarity between two elements with 0 representing low and 1 representing high similarity. An **aggregation** operator is used to combine multiple similarity matrices computed by different

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

matchers to a single aggregated similarity matrix. Finally, a **selection** operator extracts the most probable element pairs from a similarity matrix and sets all other values to zero (see [15] for an overview to selection strategies). From the finally selected matrix a **mapping** between a source schema S and a target schema T can be constructed. A mapping consists of a set of correspondences (s, t, sim) referring to a source- and a target element as well as a similarity value.

Matching systems not only differ in the implementation of these basic operators but also by the order in which these operators can be executed. In this paper we adopt the notion of matching processes similar to eTuner [14]. A **matching process** (or matching strategy) is represented by a directed acyclic graph describing the execution order of operators such as match, aggregation or selection. It contains all steps necessary to come from two input schemas to a final mapping. Operators in the graph have one or more similarity matrices as input and return a similarity matrix as output. The topology of a matching process can be very different. Simple topologies that are commonly used are parallel, sequential and iterative execution of matchers. These basic topologies may also be combined in more sophisticated match strategies.

In general, tuning a matching system involves defining the underlying matching process structure, selecting the appropriate operators and parameterizing individual operators. Our system should be able to automatically choose promising matchers, aggregation and selection operations and their parameters. Also the structure of a matching process should be automatically adaptable.

3. ADAPTIVE MATCHING SYSTEM

We took inspiration from how a matching expert would interactively develop and execute a matching process. She analyzes the source and target schema, selects appropriate matchers and constructs an initial matching process. The process is executed and the result is inspected. Depending on that result, certain parts or parameters of the matching process can be changed and extended manually.

Our approach performs similarly but in an automatic way. In order to automate the analysis step, we rely on so-called features. These are computed from the input schemas but also from intermediate mapping results. Features try to give some indication about schema properties or the quality of a mapping. Computed features are used by so-called *matching rules* representing expert knowledge about a relation between features and operators or process patterns. Finally, an adaptive process execution system (see Figure 1) selects and applies rules and incrementally executes the constructed process. Our adaptive matching system gets two schemas as input and returns a mapping as output. Ideally, no further parameterization input should be needed when running the system. All necessary parameters should be defined automatically.

The system consists of a registry that contains a number of so called feature analyzers, matching rules as well as an operator library that contains all necessary operators in particular the matchers, aggregation or selection operators. The core component of the system is the adaptive process construction that implements a staged rule application approach that we describe below. In a preprocessing step all schema features of the input schemas are computed. The adaptive process construction starts with an empty process and applies rewrite rules to add or change operators. After every change of the process the matching process execution is called to execute the new operators. This creates new intermediate similarity matrices that can be analyzed by matrix feature analyzers.

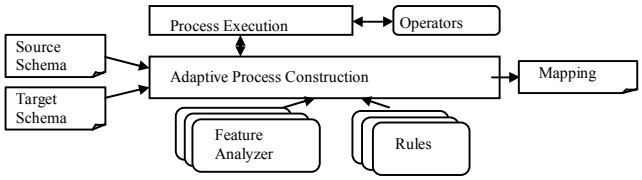

Figure 1: Adaptive Matching System

In the following, each of the components is described (Please see extended version for more details[1]).

3.1 Features

In general a **feature** takes one or several schemas or similarity matrices as input and computes a value between 0 and 1 as output. We distinguish between schema features, and matrix features. The notion of matrix features is newly introduced by this paper.

Schema features try to describe properties of schemas that can be computed in a preprocessing step before actually executing a matching process. In simple cases they reflect the schema size or the relative frequency of schema element properties such as the availability of element descriptions or data type information. More complex features rely on value distributions of schema elements or structural properties. For instance, the average length of paths in a schema tree gives some indication of whether to use a path matcher evaluating the name similarity of elements and their predecessors. Some schema features evaluate the degree of similarity between both input schemas. For example the structural and linguistic schema similarity can be used to decide about the appropriateness of applying a structure-based or name-based matcher [9].

We additionally analyze intermediate similarity matrices after executing operators of the matching process to derive **matrix features**. They are used to evaluate the quality or similarity value distributions of similarity matrices. For instance a so-called Noise feature computes the number of low valued entries in a similarity matrix in relation to the top-1 values in each row and column. The resulting feature can be used to evaluate the quality of a matrix and thus the operator that has generated it. Some matrix features take more than two matrices as input to describe the degree of commonalities and differences between them. For instance a feature could measure the overlap of top-1 values of different similarity matrices. If the overlap is higher, more confidence could be put in the different matrices.

3.2 Matching Rules

We use schema and matrix features within so-called matching rules. A matching rule captures a design decision a matching process expert would take, e.g. to select and add matchers..

A **matching rule** consists of the following parts:

- A **pattern** that describes a part of a process graph where the rule can be applied to. The pattern can be empty, in particular within rules that start the process construction.

[1] http://arxiv.org/abs/1108.1925

- An **action** that applies a defined change to instances of the found process pattern, e.g. additions and changes of operators.
- A **relevance** function that computes the relevance of the respective rule for the current matching process and match task. It is based on computed schema and matrix features on the input schemas and already computed similarity matrices and computes a relevance value between 0 and 1. The relevance is used to decide whether a rule is executed.
- An optional **check** function that is used after a rule was applied to a process. It rates the quality of the changes that were introduced by the action. It also relies on matrix features to compute a value between 0 and 1.

We identified different types of matching rules that are starting-, aggregation-, rewrite-, refine and selection rules. **Starting rules** can be applied to an empty process when no intermediate matrix was computed yet. Each application of a starting rule creates so called dangling nodes that are possible end-points of the process and do not have following nodes. **Aggregation rules** add aggregation operators to a process and combine a number of dangling operators from a matching process. **Rewrite rules** change the structure of a given process without changing the input and dangling output nodes. For instance the order of operators could be changed or additional operations can be added in between others. Selection Rules are used to add a selection operator to the last dangling node of the current matching process.

3.3 Adaptive process construction

The different kinds of rules are applied in a certain order to control the degree of process adaptations. Different rule classes depend on each other since some rules add operators and others combine the output of several operators as done by the aggregation rule. In order to restrict complexity we perform the application of rules only within a fixed number of stages as shown in the left part of Figure 2. This reduces the structural diversity an adaptively created process can have but simplifies the rule selection process significantly. If all rules would be able to compete in all stages of the process side-effects of rule application could not be controlled and termination could not be ensured.

The process starts with importing the input schema and analyzing them to compute the schema features. An empty matching process is created. In the next stage starting rules can be selected and applied. Starting rules mostly add element-based matchers. However, also more complex starting rules can be defined that already construct an advanced matching process structure, e.g. to enforce the sequential execution of several matchers. In the next stage the dangling nodes from applying the starting rules need to be combined and the result matrices are aggregated. After the aggregation, rewrite rules can be applied. If no relevant rewrite rules can be found, a selection rule is applied. Based on the selection result the process can be finished or refine rules can be applied iteratively.

Within each stage of the process a predefined rule selection process is started (see right state chart from Figure 2). The selection process begins with filtering all rules that can be executed within the current stage. This set of rules is only created once within a stage. If the remaining set of rules is empty the stage can be finished directly. If there are rules left to be applied, their relevance is computed for each rule using the rules relevance functions. The most relevant rule is selected and applied.

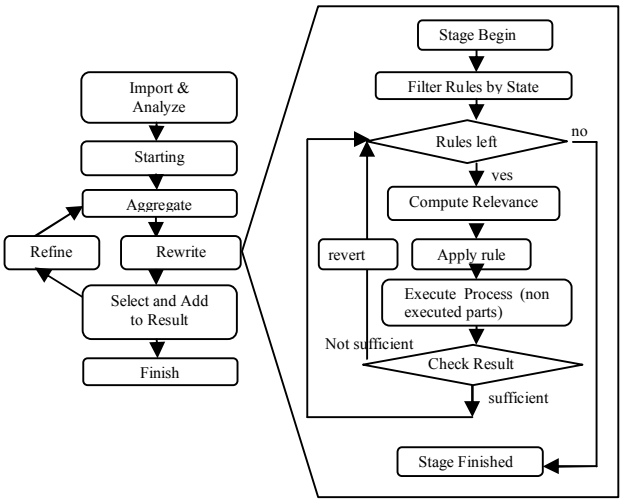

Figure 2: Stages and rule selection

Applying a rule implies changes to the current matching process. After that the current process is executed. However no operator is executed twice and only new or changed parts are executed. The matrix result of executing the most recently applied rule is evaluated using the rules check function. It often happens that rules are rated as relevant due to the existence of certain attributes in the source and target schema. However, after executing the matchers that were added by the rule the matrix result quality is sometimes very low, indicating that the most recent rule should be ignored. For that purpose the most recent rule effect is reverted. After executing a rule it is removed from the remaining list of rules. Again the most relevant rule is selected and the process runs on until the rule set is empty.

3.4 Example

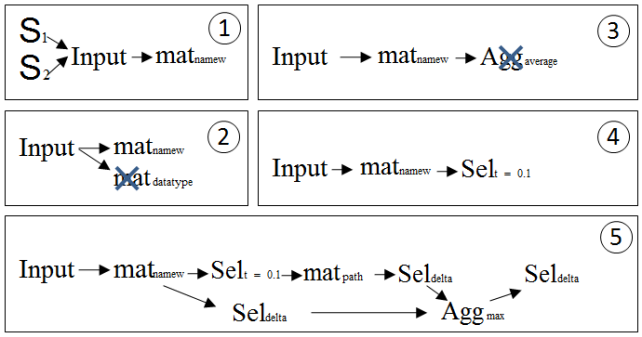

Figure 3: Example adaptive construction

For illustration we explain the adaptive process construction for matching two anatomy taxonomies that are also used in our evaluation. In Figure 4 different states of the adaptive construction are visualized. Our system starts with constructing a new process with an input node that imports the source and target schemas and computes the schema features. In (1) a starting rule is applied that measures the number of repeating tokens in element names and selects a name matcher with term weighting. In (2) a rule adds the datatype matcher and executes it. However it is then removed due to a low quality detected by the check function. After all relevant starting rules are executed, an aggregation rule is executed (3) that combines multiple dangling nodes (these are operators that need to be consumed by other operators to complete the matching process). In the example, only one dangling node exists so that the

Aggregation operator can be removed. In (4) a rewrite rule has been found relevant and applied; it adds a selection operator after the matcher operator. In (5) a refine rule that adds a path matcher after the dangling selection to refine the intermediate match result. Finally an aggregate/selection rule is applied that combines the results of both matchers.

4. EVALUATION

In the evaluation we want to investigate the effectiveness and robustness of our approach. For that reason we compare our adaptive schema matching system to currently known alternative approaches for diverse problems. We currently have implemented 20 feature analyzers and 18 rules. For details, please refer to the extended version of the paper.

We consider a wide range of schema mapping problems from different domains. To be comparable to existing work we include the OAEI-Benchmark and Anatomy as well as the ModelCVS Benchmark [6] in our evaluation. The Purchase Order dataset and mappings were already used in the early evaluations from COMA [2] and they are publicly available. We also use them for computing a default configuration to compare against.

We precomputed a best configuration for the Purchase Order dataset similar to the way the default strategy proposed in [2] was computed. We call this computed strategy our DEFAULT configuration. It consists of the matchers WeightedName, Path, Children, Leaf, Sibling and Datatype. The best selection strategy found was parameterized with delta=0.021 and a threshold=0.5 (see [2] for details on selection strategies).

Secondly we implemented the Meta-level learning proposed in [4]. This approach is a valid comparison since it includes schema features in a learning process of a decision tree to increase adaptivity. The learning also takes the Purchase Order dataset as gold mappings. We include all schema features in the learning process. The learning was implemented using the weka[2] library. The computed decision trees are then translated into matching processes without loss of information. In order to reduce possible overfitting we restricted the size of the decision tree. We then executed the default configuration, the decision tree and our new adaptive matching system with all provided mapping problems.

4.1 Results

The results of our evaluations are summarized in Figure 4.

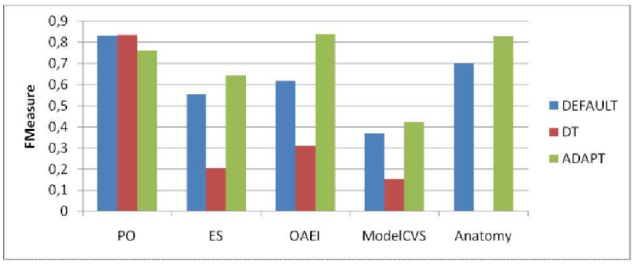

Figure 4: Evaluation results for adaptive execution

For each dataset we compared the average achieved F-Measure of the individual approaches DEFAULT, DT and ADAPT. Our adaptive approach (ADAPT) returned good results across all different test cases and thus proved its viability. For the Purchase Order data set ADAPT is only closely behind the DEFAULT configuration and the computed decision tree (DT). The slightly better outcome for DEFAULT is not surprising since DEFAULT was tuned on this data set by testing all possible parameter configurations. DT had problems to learn a good process since the diversity of matches is too big to derive a representative rule in a decision tree. We also experienced strong overfitting which was already experienced by the authors of [4].

5. CONCLUSIONS

We proposed a new self-configuring and adaptive schema matching system. It uses a series of schema and matrix features and different matching rules to automatically construct and adapt matching processes. Our evaluations show that the approach is able to return good results for diverse match problems and that it can compete with manually tuned approaches.The proposed approach is in its initial version and we plan further improvements, e.g. to support more complex adaptations.

6. REFERENCES

[1] Rahm, E. and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. The VLDB Journal 10.

[2] Do , H. H. and Rahm, E. 2002. COMA - A System for Flexible Combination of Matching Approaches. VLDB Proc.

[3] Duchateau, F., et. al. 2009. YAM: a schema matcher factory, CIKM.

[4] Eckert, K., Meilicke C., Stuckenschmidt, H. 2009. Improving ontology matching using meta-level learning. In ESWC 2009.

[5] Euzenat, J., et. al. 2010. Results of the Ontology Alignment Evaluation Initiative 2010. Workshop on Ontology Matching.

[6] Falleri, J.-R., et. al. 2008. Metamodel matching for automatic model transformation generation. MoDELS '08

[7] Gal, A., and Sagi, T., 2010, Tuning the Ensemble Selection Process of Schema Matchers, Information Systems 35

[8] Gusfield, D., Irving, R. W. 1989. The stable marriage problem: structure and algorithms. MIT Press.

[9] Hu, W. and Qu, Y. 2008. Falcon-AO: A practical ontology matching system. Web Semant., 6(3).

[10] Li, J. et al. 2009. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Transactions on Knowledge and Data Engineering, 21(8).

[11] Meilicke, C., Stuckenschmidt, H. 2007. Analyzing Mapping Extraction Approaches. ISWC - Workshop on Ontology Matching.

[12] Noy, N. F. and Musen, M. A. 2003. The PROMPT Suite:Interactive Tools for Ontology Merging and Mapping. Int. J. Hum.-Comput. Stud.

[13] Pirrò, G.,Talia, D., 2010: UFOme: An ontology mapping system with strategy prediction capabilities. Data Knowl. Eng. 69(5)

[14] Y. Lee et. al. 2007. eTuner: tuning schema matching software using synthetic scenarios. The VLDB Journal, 16(1).

[2] http://www.cs.waikato.ac.nz/~ml/weka/index.html