Modeling Matching Systems using Matching Process Design Patterns

Eric Peukert

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

1 Introduction

Many schema- and ontology matching systems were developed to compute mapping suggestions for a user. Most of these systems are black boxes that often re-implement basic matching components which are extended by a few domain specific matchers. We observe that most systems mainly differ in their internal execution order and combination of matchers.

In this paper, we advertise using a matching process model to unify a broad set of different matching systems. That allows making the order of execution within a matching system explicit. Moreover, we identify a set of so called matching process design patterns that are often used and combined to build strong matching systems.

2 Process Model and Design Patterns

A *matching process* is represented by a directed matching process graph as was also proposed by [2]. The vertices represent operations and edges determine the execution order and data flow. The result of a matching process is a mapping *MA* between a source schema *S* and a target schema *T* that consists of correspondence links between schema elements. With a *schema* we refer to any meta data structure such as trees, ontologies, or meta models. Mappings are computed with the help of *similarity matrices* that contain similarity values between schema source and target elements. Additionally, we introduce a so called *comparison matrix*. A comparison matrix consists of $|S| * |T|$ cells. Each cell contains a boolean value representing whether a comparison within subsequent matching operations should be performed. The comparison matrix is crucial for controlling the flow of element comparisons within a matching process. Our set of operators is based on the operators we introduced in [1] that are Match, Combine, Select, Filter, Input and Output. *Match* computes a similarity matrix using some matching algorithm. *Combine* aggregates multiple matrices and *Select* reduces the matrix to most likely mapping candidates. Filter is used to reduce the number of comparisons for subsequent operations by setting boolean values in the comparison matrix. Additionally we introduce a *Condition* to allow conditional execution of process parts and *Split/Loop* to model processes of systems like Falcon or RiMOM [3, 5]. Based on these operators we are able to model a variety of matching systems internal matching processes using the framework and tools described in [1]. Moreover, we were able to identify an initial set of reusable matching process design patterns that are often used and combined to build strong matching processes (see Figure 1).

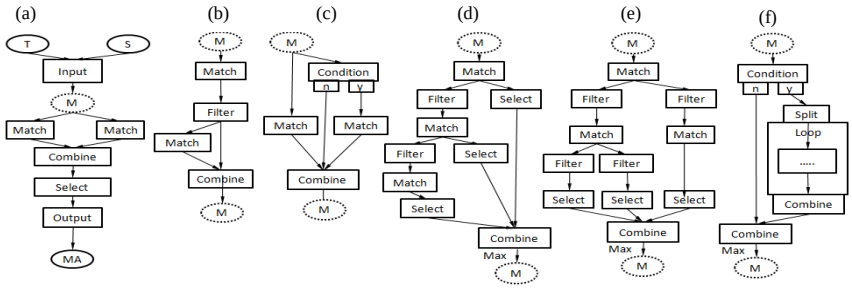

Fig. 1. Matching Process Design Patterns

Parallel Composition (a) is often applied to combine a set of matching algorithms and was introduced in [4]. *Refinement Sequence* (b) tries to increase precision by refining the results of a matcher within subsequent matchers in a process. *Adaptive Matcher Selection* (c) is often used to select the most appropriate matcher for a given matching problem based on some pre-computed feature value. The *Skimming* pattern (d) extracts the most probable correspondences from every matcher individually. These correspondences are "skimmed". This approach is useful if individual matchers have a high precision for a domain of mapping problems. *Divide and Conquer* (e) divides the set of comparisons based on some property and distributes these comparisons to the most appropriate, possible different matchers. This pattern is extensively used in decision tree based matching systems. Finally *Blocking&Clustering* is applied in systems that repeatedly execute process parts. A typical application is the fragmentation of the matching task into smaller blocks that are executed independently. In addition to the visualized patterns we propose two further patterns that are *Iteration* and *Matcher Hierarchies.* Iteration repeatedly executes process parts until a given condition is met. *Matcher Hierarchies* are implicitly used by many matching systems to build complex structure-based matchers. Within that pattern, the output of a matcher is directly used as input for a second matcher.

In our evaluation we were able to show that the parallel composition pattern behaves very robust to solve different matching problems with high quality. However, by combining the pattern with skimming and refinement parts the quality can further be improved as was implicitly done in the internal process of Falcon and RiMOM.

References

1. Peukert, E., Eberius, J., Rahm, E.: AMC - A Framework for Modeling and Comparing Matching Systems as Matching Processes. ICDE (2011)
2. Lee, Y. et. al.; eTuner: Tuning Schema Matching Software Using Synthetic Scenarios. The VLDB Journal, 16(1), (2007)
3. Li, J. et. al.: RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Transactions on Knowledge and Data Engineering, 21(8), (2009)
4. Do, H. H. and Rahm, E.: COMA - A System for Flexible Combination of Matching Approaches. VLDB (2002)
5. Hu, W. and Qu. Y.: Falcon-AO: A Practical Ontology Matching System. Web Semant., 6(3), (2008)