AMC – A Framework for Modelling and Comparing Matching Systems as Matching Processes

Eric Peukert[#1], Julian Eberius[#2], Erhard Rahm[*3]

[#] *SAP Research*
01187 Dresden, Germany
[1] eric.peukert@sap.com
[2] julian.eberius@sap.com
[*] *University of Leipzig*
Leipzig, Germany
[3] rahm@informatik.uni-leipzig.de

Abstract—**We present the Auto Mapping Core (AMC), a new framework that supports fast construction and tuning of schema matching approaches for specific domains such as ontology alignment, model matching or database-schema matching. Distinctive features of our framework are new visualisation techniques for modelling matching processes, stepwise tuning of parameters, intermediate result analysis and performance-oriented rewrites. Furthermore, existing matchers can be plugged into the framework to comparatively evaluate them in a common environment. This allows deeper analysis of behaviour and shortcomings in existing complex matching systems.**

I. Introduction

Schema matching is the task of finding mappings between complex metadata structures as required in data integration, ontology alignment or model management. In the last decade numerous matching algorithms were proposed and combined to partly automate that process (see surveys [1] and [4] for an overview). In particular for ontology matching a number of complex matching systems such as COMA++ [2], RIMOM [9], Falcon [6] or ASMOV [7] were developed that try to show their versatility in yearly OAEI evaluations [5]. They use complex matching processes consisting of sequential and iterative executions of several match algorithms, so-called matchers.

Some systems like RIMOM and Falcon analyze the input schemas and intermediate result mappings and include conditions into their process to better adapt to individual match problems of the OAEI evaluations. However, these systems have in common that the underlying execution process consisting of matchers and their combination is largely fixed. Constructing or adapting matching processes for a new problem domain often involves high implementation, tuning and testing effort in current systems. In addition to that, basic matchers are often reimplemented instead of being reused.

Recently, so called meta matching systems were introduced. They focus on tuning combinations of existing matchers by using a given, or synthetically created perfect mapping. These combinations of matchers can be complex workflows or processes as proposed in eTuner [8] or decision trees as used in MatchPlanner [3]. However these processes are not able to automatically adapt to new mapping problems. Additionally these systems do not provide visual support for integrating or selecting matchers and modelling new matching workflows. Moreover, comparing and analyzing basic matching components from existing matching systems is cumbersome and involves a lot of implementation effort.

In this demo paper, we introduce the Auto Mapping Core (AMC), a meta matching system that allows to integrate and compare arbitrary matching components within a so called matching process. AMC offers the following key features:

- The AMC framework can be used to define and execute complex matching processes. It offers a design tool, the *Matching Process Designer*, that visually supports the construction of matching processes. Processes can be executed and debugged by stepping forward and backward in the process and setting breakpoints. A new visualisation technique simplifies the analysis of intermediate matching results.
- Matching processes are adaptive and can easily be changed to new domains. The execution time of matching processes can be automatically optimized by applying rewrite techniques [12].
- Existing matching system components can be plugged into the AMC framework for evaluation and reuse. The AMC evaluation framework allows a uniform comparison of the behaviour and match quality of different matchers and also of combination and selection methods.

II. AMC Overview

Fig. 1 illustrates the architecture of AMC consisting of modules for modelling and execution of matching processes. The system gets a source and target schema as input and computes a mapping. The core of the AMC framework is an extensible operator library consisting of matcher, combination, selection, analyser and blocking operators (see next section). Extensions can be added to the library as loosely coupled plug-ins. Plug-ins implement a thin adaptation layer that calls operators of existing systems and converts inputs and outputs of these operators into internal AMC formats.

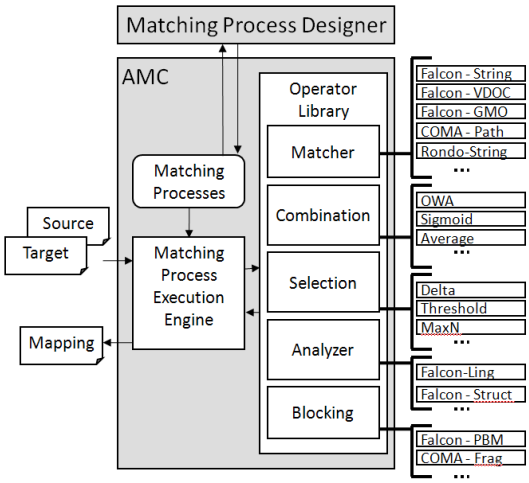

Fig. 1 System Architecture of our Meta Matching System

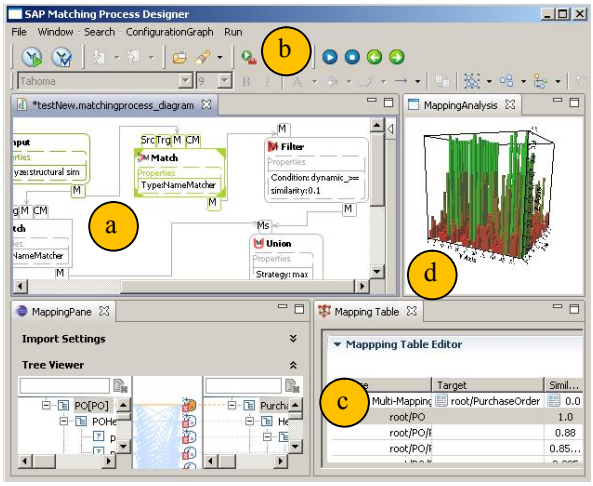

Fig. 2 Matching Process Designer

Currently, we integrate only operators for which the sources are available and licensing does permit their use. However, binaries can also be integrated as plug-ins if their use is permitted. Using the operator library, matching processes can be defined using a GUI-based process editor, the *Matching Process Designer* (see Fig. 2). The created matching processes are executed by an execution engine that calls the respective operators from the library.

The Matching Process Designer is used to define an order of operators that used to be fixed for existing matching systems. Parameters of individual components can be set directly within the graphical representation of the process (Fig. 2a). It can also be used to iteratively step through the process (Fig. 2b) and to investigate intermediate results of individual operators (Fig. 2c). For that purpose a new visualisation technique for analysing intermediate results (introduced in Section IV) is used (Fig. 2d).

III. Matching Processes

A matching process is a directed acyclic graph that describes the execution order of selected operators. Operators in the graph take either a source schema and a target schema or mappings as input and return a mapping as output. The output of one operator is used as input for the next one.

The AMC uses a uniform mapping data structure as basis for integration of external operators. A **mapping** between a source schema and a target is represented by a similarity matrix $A = (a_{ij})$ that has $|S| \times |T|$ cells to store a match result. Each cell a_{ij} contains a similarity value between 0 and 1 representing the strengths of similarity between the i-th element of the source schema and the j-th element of the target schema. Additionally a mapping contains a so-called comparison matrix that is defined as $CM = (cm_{ij})$ with $|S| \times |T|$ cells. Each cell cm_{ij} defines whether the i-th and j-th element should be compared in the following operations. Please refer to [12] for further details on the comparison matrix and how it can help reducing the search space for improved performance.

The AMC operator library consists of different operator types, namely matcher, combination, selection, analyser and blocking operators.

Matcher operators form a crucial part of the library. They take either a source schema and a target schema or a mapping as input and compute a mapping as output. Matchers compute a similarity for each element pair for which the comparison matrix entry is set to true. This can be used for refining already computed mappings. In addition to the mapping and schema input, so called constituent mappings as already supported in COMA++ [2] can be provided as input to a matcher operator. This can be used in particular by structural matchers that compute a similarity from a combination of pre-computed constituent similarity values. For example a name-path matcher computes the path similarity by combining the pre-computed name similarities of elements in the path.

The **combination operators** take multiple mappings as input and compute a single mapping as output. The input mappings can result from other matcher executions but also from previous selection or combination operations. For each pair of schema elements a combined similarity value is computed.

The **selection operators** get a mapping as input and produce a mapping as output. Selection operators filter the most promising mapping pairs using some filter strategy. The result is a sparse mapping where similarity values are often only set for few element pairs per schema element.

Analyser operators get source and target schemas or mappings as input and compute a characteristic feature value, for example the linguistic or structural similarity of the input schemas. These feature values are used within the matching process to define conditions for executing or not executing sub-processes. This lets the process automatically adapt to the concrete matching problem.

The **blocking operators** are used to reduce the often necessary cross-product computation to a number of smaller, possibly intersecting, block mapping problems. This reduces the search space since only similar blocks are matched. For that purpose clustering, partitioning and also fragmentation techniques can be used. A **for-each operator** is used to

execute a sub-process for each block mapping problem. It finally aggregates the results to a single mapping.
Besides the mentioned operators there are auxiliary operators for splitting or filtering schema elements and mappings. Also a difference operation that subtracts a mapping from another mapping is included.

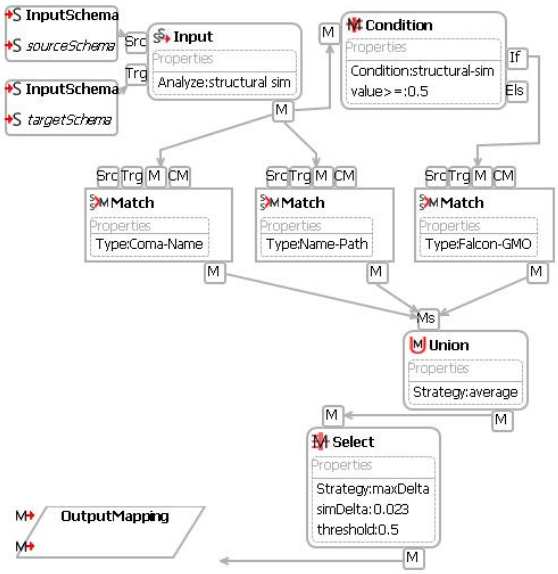

Fig. 3 Sample Matching Process

Fig. 3 shows a sample matching process. It consists of three matcher operators (Coma-Name, Name-Path and Falcon-GMO), a combination operator that computes the average of the matcher results, a condition and a selection. The condition is based on a structural similarity feature of the input schemas that is computed by an analyser. Only if the structural similarity is bigger than 0.5 the Falcon-GMO matcher is executed. The result of the executed matchers is combined in an average combination and the most promising element pairs are selected for the output mapping.

IV. Process Modelling and Tuning

Matching processes are modelled in a graphical process editor by dragging operators onto the surface and connecting them. Individual parameters of operators can be set directly in the graph. Source and target schema fragments can be selected and a stepwise process execution can be started. The user can step forward and backward in the process or set breakpoints. The currently executed operator is highlighted and the intermediate matching result is shown.

The user can investigate computed similarities of schema element pairs. With higher numbers of elements, line based and table based visualisations provide too much information for the user. For that reason we propose a new visualisation of intermediate results (see Fig. 4 for examples). It represents a mapping as a cube. The x- and y-axis represent the source and target elements and the z-axis represents the similarity values. Such an aggregated representation turned out to be quite intuitive for analysis of large intermediate results. The user can analyse the overall distribution of similarities of mappings. Also, conclusions about the selectivity of the mapping can be drawn. A selective mapping consists of a number of steep peaks whereas a non- selective mapping produces flat blocks. In many cases the mapping representation produces steps that give a good indication of possible thresholds for later selection and filters.

At the bottom of the example cube of Fig. 4a many low-valued similarities are found that we classify as noise. A number of similarities are higher than 0.9 and only few more are higher than 0.6. These values can be tested as selection and filter parameters. Each matcher produces a characteristic representation in our visualisation that tends to be similar for different test cases. So some matcher results show a big variance of similarity values (see Fig. 4a for the name matcher result), whereas for other matchers like the name-path matcher the variance is much smaller (see Fig. 4b).

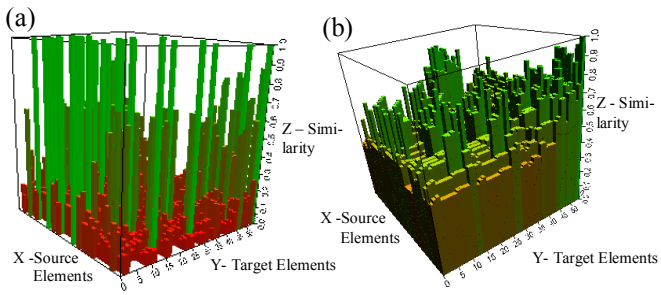

Fig. 4 Comparing Intermediate Results

V. Comparison of Matching Components

Previous evaluations [5] of matching systems compared whole systems with default configurations with each other. The shortcomings and strength of individual systems could not be identified. Some systems perform well due to their combination and selection methods whereas others have very strong matchers.

The AMC framework is able to integrate and compare individual components of different matching systems. This allows the user to better compare and evaluate the behaviour of these systems at the level of individual matchers and other components. We extended the AMC operator library by a number of plug-ins from different matching systems such as COMA++, FalconAO, and Rondo [11]. Additionally, AMC provides its own operator implementations.

Our library now consists of a number of element-based and also structural matchers. Different combination and selection techniques and also two blocking operators are included, that are the COMA++ Fragmentation [2] and the Falcon [6] partitioning technique. Moreover the AMC framework integrates schema and mapping property analysers from Falcon that can be used to adapt the matching process to the mapping problem.

One observation we made is that all systems implement their own name-based and structural matchers. Thus almost no reuse is done across matching systems. To demonstrate the features of AMC we performed a brief comparative evaluation of the element-based matchers. For that purpose, we chose a set of small purchase order test cases T1 to T10 from the COMA evaluations [1]. We then created a number of simple

matching processes that consist of the individual element-based matchers. For all processes, the same selection operator is included. AMC applies the often used Precision, Recall and F-Measure (FM) as evaluation metrics. Within the AMC Matching Process Designer, the test cases can be selected and the comparison process can be started. The evaluation result from Fig. 5 can also be visualized directly in the tool.

As can be seen, the result quality of the individual matchers on the given data set is different across the basic matcher implementations. In comparison to the others, the AMC Name matcher was the favourable matcher for the given test cases. However, the other matchers are closely following. With the help of the Matching Process Designer we could identify the reasons for that.

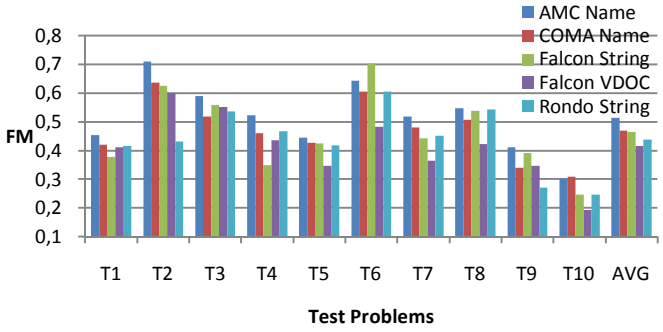

Fig. 5 Comparing Element-based Matchers

It needs to be further evaluated in future work how these matchers perform in other domains such as OAEI. We also plan to use AMC for more advanced evaluations such as for comparing blocking operators or propagation-based structure matchers, e.g. Similarity Flooding [10] and the Falcon GMO Matcher [6]. We also need to investigate in techniques to cope with possible main memory problems when evaluating larger mapping problems.

VI. Demonstration Description

In our demonstration we illustrate how AMC is used to model and tune a matching process that integrates existing operators from different matching systems.

For that purpose we introduce a small scenario: A consultant needs to define mappings for a number of similar purchase order schemas. He wants to make use of semi-automatic matching to speed up the mapping process.

A matching expert is asked to create a tuned matching process using AMC for the specific purchase order mapping problems. For that purpose, the consultant defines a few mappings manually. The matching expert takes the default matching process from COMA consisting of name, namepath, leaf, datatype and children matcher that showed good results across different domains. By using the Process Designer and with the help of the provided mappings he is able to tune the given process to the given domain of purchase order schemas. We show how the matching expert can incrementally execute operators from the initial process and investigate the intermediate result mapping in our new visualisation. With the help of the intermediate visualisation we can show that the name-path matcher produces some noise at the bottom of the matrix that should be removed. Therefore it is recommended to introduce a filter operator after the name-path matcher that removes noise from the intermediate result and improves result quality. The filter threshold can be easily derived from our new visualisation by putting it just above the noise level. After executing further matchers, we can show that for the given example test cases the data type matcher does not return any usable result, so that it can be dropped or replaced by a new matcher. After having built and tested the matching process we demonstrate how performance-oriented rewrites can be applied. The final matching process can be saved and handed over to the consultant as a tuned matching process for the specific purchase order mapping scenario.

We further demonstrate how existing operators from different matching systems can be compared. The comparison diagrams are generated and analysed. By going back to the matching process we investigate the result of individual components and show what the differences are to the AMC Name matcher mapping using a difference operator.

Within our demo, visitors can directly model their own matching process, play around with parameterisations and start comparison tasks.

Acknowledgment

This work was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference “01MQ07012”. A special thanks goes to Veronika Thost for fruitful discussions and some implementation work on the Matching Process Designer.

References

[1] H. H. Do & E. Rahm. *COMA - A System for Flexible Combination of Schema Matching Approaches*. VLDB Proc., 2002

[2] H. H. Do and E. Rahm. *Matching large schemas: Approaches and evaluation*. Inf. Syst., 32(6), 2007.

[3] F. Duchateau, Z. Bellahsene, R. Coletta: *A Flexible Approach for Planning Schema Matching Algorithms*. OTM Conferences (1) 2008

[4] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer-Verlag, 2007.

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

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

[7] Y. R Jean-Mary, E. P Shironoshita, and M. R Kabuka. *Ontology matching with semantic verification*. Web Semant. 7, 3 Sep. 2009

[8] Y. Lee et. al. *eTuner: Tuning Schema Matching Software using Synthetic Scenarios*. The VLDB Journal, 16(1), 2007.

[9] J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: *A Dynamic Multistrategy Ontology Alignment Framework*. IEEE Transactions on Knowledge and Data Engineering, 21(8), 2009.

[10] S. Melnik, H. Garcia-Molina, and E. Rahm. *Similarity flooding: a versatile graph matching algorithm and its application to schema matching*. Proc. ICDE, 2002.

[11] S. Melnik, E. Rahm, P.A. Bernstein. Rondo: *A Programming Platform for Generic Model Management*, Proc. ACM SIGMOD 2003

[12] E. Peukert, H. Berthold and E. Rahm: *Rewrite Techniques for Performance Optimization of Schema Matching Processes*. EDBT 2010

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

[14] Hu, W. and Qu, Y., *Block matching for ontologies*. In: LNCS, vol. 4273. Springer. pp. 300-313.