Evolution of the COMA Match System

Sabine Massmann, Salvatore Raunich, David Aumüller, Patrick Arnold, Erhard Rahm

WDI Lab, Institute of Computer Science
University of Leipzig, Leipzig, Germany
{lastname}@informatik.uni-leipzig.de
http://wdilab.uni-leipzig.de

Abstract. The schema and ontology matching systems COMA and COMA++ are widely used in the community as a basis for comparison of new match approaches. We give an overview of the evolution of COMA during the last decade. In particular we discuss lessons learned on strong points and remaining weaknesses. Furthermore, we outline the design and functionality of the upcoming COMA 3.0.

1 Introduction

Schema and ontology matching is the process of automatically deriving correspondences between the elements or concepts of two or more data models, such as XML schemas or formal ontologies. Computed correspondences typically need to be validated and corrected by users to achieve the correct match mappings. Match mappings are needed in many areas, in particular for data integration, data exchange, or to support schema and ontology evolution. Hence, match mappings are the input of many algorithms in these domains, e. g. to determine executable data transformation mappings or to perform ontology merging.

In the last decade, there has been a huge amount of research on schema and ontology matching and on mapping-based metadata management in general. Overviews of the current state-of-the-art are provided in two books [5, 13]. Many dozens of research prototypes have been developed and at least simple (linguistic) automatic match approaches found their way into commercial mapping tools [23]. COMA (Combining Matchers) is one of the first generic schema matching tools. Its development started about ten years ago at the University of Leipzig and is still going on. COMA and its successor COMA++ have been made available to other researchers and have been widely used as a comparison basis for new match approaches.

This paper reflects on the evolution of COMA during the last decade and reports on the major lessons learned. We also describe the design and functionality of a new version, dubbed COMA 3.0. In the next section, we give a short overview of the evolution of COMA and COMA++. We then discuss lessons learned in Section 3 by outlining strong points and remaining weaknesses. The new version of COMA is described in Section 4.

2 System Evolution

Key stations and publications concerning development and use of COMA are as follows:

2002 Initial release and publication of COMA paper at VLDB in Hong Kong [9]. Support for multi-matcher architecture and reuse of previous match mappings.

2003/04 Further evaluations [8] and initial design to support large XML schemas [25].

2005 Release and SIGMOD publication of COMA++ [2]. Support for ontology matching, GUI and fragment matching. Ph.D. Thesis of Hong Hai Do [7].

2006 Support for instance-based matching. Participation at OAEI contest [17].

2007/08 Further evaluations with larger and more diverse models, including web directories [10, 18]. Use of COMA++ within the QuickMig project [11]. Web edition.

2010/11 Redesign and development of COMA 3.0.

The development of COMA started in 2001 and was influenced by the findings and recommendations of the survey article [24] on schema matching. In contrast to most previous approaches at the time, COMA provides a generic approach to support matching of different kinds of schemas (in particular relational and XML schemas) and for different application domains. This is made possible by representing all kinds of schemas by a generic in-memory graph representation on which matching takes place. A key feature of COMA is the flexible support for multiple independently executable matchers that can be executed within a user-controlled match process or workflow. More than ten schema-based matchers were initially supported mainly based on the linguistic and structural similarity of elements. All matchers are evaluated on the Cartesian product of elements from the two input schemas, where each element is represented by a path to the schema root. Furthermore, different approaches to combine matcher results and select correspondences are provided. A unique feature of the initial COMA design was the support for reusing previous match mappings, especially the possibility to compose several existing mappings stored in a repository. An extensive evaluation on XML schemas showed that the combination of several matchers clearly outperforms single matchers.

The initial evaluation used relatively small schemas of less than 100 nodes. Hence, further evaluations and adjustments focused on larger schemas, especially for e-business schemas containing shared schema fragments (e. g. for address information) [25]. Several extensions to deal with large schemas were designed and integrated within the next major release of the prototype, called COMA++. COMA++ was introduced in 2005 [2] and represents a major re-implementation of the original COMA design to improve both performance and functionality. It provides a GUI to simplify the definition of match strategies and to correct computed match correspondences. It also supports matching of ontologies, especially OWL ontologies. COMA++ provides additional matchers and operators like merge and diff for post-processing of match mappings.

Several approaches in COMA++ facilitate the matching of larger schemas, in particular to avoid the evaluation of the Cartesian product of schema elements. First, fragment matching is supported that implements one of the first divide-and-conquer approaches where only similar schema fragments need to be matched with each other. Secondly, sequential execution of matchers (or mapping refinement) is supported so that a fast matcher can be executed first and more expensive matchers are only evaluated on more similar pairs of elements. In particular, a strategy called *FilteredContext* performs first matching only for nodes and restricts the evaluation of paths (i. e. node contexts) to the

more similar node pairs. A detailed description and evaluation of these features can be found in [7, 10].

In 2006, two instance matchers were added to COMA++ to prepare the system for participation in the OAEI contest[1] that provides instances for its basic benchmark test cases. Sets of instances are associated to schema elements. One approach is to compare individual instance values with each other and aggregate the similarity values per element pair. Alternatively, all instances are combined within a virtual document and a TF/IDF-like document similarity is determined for element pairs (similar approaches are used in other match systems, e. g. RiMOM [16]). The instance-based matchers and their combination with metadata-based matchers were successfully evaluated on the ontologies of the OAEI benchmark [12] and on web directory taxonomies [18].

In a joint project with SAP, COMA++ was used as the basis for developing mappings for data migration [11]. Furthermore, we created a web edition of COMA++ to support matching without local installation of the tool and its underlying DBMS (MySQL). For evaluation, we also added to COMA++ some advanced approaches for aggregating similarity values of different matchers [22] as being used in other match systems such as Prior+ or OpenII Harmony. Since 2010, we partially redesigned and extended COMA++ as we will describe in Section 4.

Due to numerous requests, we made the binary version of COMA/COMA++ available for free to other researchers. Hundreds of researchers world-wide downloaded the prototype for use and comparison with their own matching approaches.

3 Lessons Learned

Working on and with a system with such a longevity as COMA, there are both positive and negative lessons learned. This section presents both sides.

3.1 Strong Points

In retrospect, we believe that many design decisions of COMA and COMA++ have been right. Several of them have also been adopted by later match systems. The positive points include the following:

Multi-matcher architecture Supporting many diverse matcher algorithms that can be combined within match workflows is key to obtain sufficient match quality for different match tasks. Almost all recent match systems follow such a design with support for linguistic, structural, and instance-based matchers.

Generic approach The generic representation of models as rooted, directed acyclic graphs allowed us to apply all matchers and combination approaches to diverse kinds of schemas and ontologies. By providing the respective parsers we could thus easily extend the scope of COMA to different domains and models. This flexibility also contributed to the popularity of COMA for other researchers.

[1] http://oaei.ontologymatching.org/

Effective default configuration COMA and COMA++ provide a default match configuration that can be used without manual tuning effort. The default configuration was determined for a set of XML schema match tasks and consists of four linguistic and structural matchers and a specific combination approach [10]. The default combination approach is based on taking the average matcher similarity per element pair and applying a so-called *Delta* selection returning the match candidates with the highest match similarity (above a threshold) as well as all further candidates within a small distance (delta) of the top candidate. For improved precision, a stable marriage-like selection is further applied (called *Both*) by default. The default strategy turned out to be surprisingly effective over a wide range of further match problems evaluated by other researchers, including for matching web directories [3], for n-way (holistic) schema matching [14] and even for matching of UML meta-models [15]. An evaluation of different methods to combine matcher results [22] showed that advanced approaches can perform well only in specific cases and are not as robust as simply determining the average matcher similarity applied by default in COMA++.

GUI The graphical user interface in COMA++ significantly improves the usability compared to the original COMA implementation, in particular for importing and visualizing schemas, configuring match strategies, and inspecting and correcting match mappings. Furthermore, it allows a simplified evaluation of different match strategies.

Customizability Even when using the default match strategy, it is possible to customize linguistic matching by providing domain-specific dictionaries, in particular synonym and abbreviation lists. These simple lists can significantly improve match quality and are an effective approach to leverage and reuse background knowledge. Particularly, they avoid the risk of many wrong match candidates (poor precision) when using general-purpose dictionaries such as Wordnet. There are many more possibilities to customize the match strategy, in particular the selection of promising matchers. For example, instance matching can be selected if instances are available for matching.

Advanced match strategies While not part of the default match strategy, COMA++ supports several advanced match strategies that are especially helpful to deal with large schemas and ontologies. These strategies include the reuse of previous match results, fragment matching, as well as mapping refinements such as in the *FilteredContext* strategy. While the approaches have been quite effective, there are still opportunities for improvement, e. g. for a more comprehensive reuse and for schema partitioning as discussed in [23].

Repository We store all imported schemas and ontologies as well as confirmed mappings in a repository for later inspection and reuse. The repository avoids the repeated import and matching of the same schemas and is a prerequisite for the reuse matchers.

3.2 Weak Points

Given the broad need for schema and ontology matching and increasing demands w. r. t. the size of match tasks and the use of mappings, we also encountered a set of difficulties with our match system asking for improvements. Most current match systems

share similar limitations although some of the associated challenges have already been addressed in recent research [28].

Scalability issues The number of paths is generally much higher than the number of nodes per schema so that COMA's path-based matching leads to memory and runtime problems for large match tasks with millions of path pairs to evaluate in the Cartesian product. Memory problems are primarily caused by storing the similarity values for all matchers and path pairs in memory in addition to the schema graphs. These problems can be alleviated to some degree by applying fragment matching and node-based matching but a more general solution with reduced memory requirements is needed. Furthermore, additional performance techniques such as parallel matching on several processors are desirable.

Configuration effort While the default match strategy is often effective it cannot guarantee the best match quality and runtime performance for all match tasks. Finding better match strategies, however, is difficult even for experts due to the high flexibility that comes with the possibility to select from many matchers and combination options and having to find suitable parameter settings. Therefore, the system should help users to choose a match strategy based on an automatic analysis of the match task to solve.

Limited semantics of match mappings The system only determines match mappings consisting of simple equality correspondences between elements of schemas or ontologies. More expressive mappings are desirable supporting additional kinds of relationships such as containment or is-a relationships. Furthermore, applications such as data exchange or schema evolution need executable mappings that can be applied to instance data.

Limited accessibility COMA++ is a stand-alone tool designed for interactive use, not for use by programs. To improve the accessibility, the provision of an API, e. g. based on web services is desirable. The web edition of COMA++ is not widely used and a redesigned browser-based GUI would be attractive.

4 COMA 3.0

Since 2010 we have partially redesigned and extended COMA++ at the Web Data Integration Lab (WDI Lab) of the University of Leipzig. The mission of the WDI Lab is the development of advanced data integration tools and approaches that can be used for practical applications. Major work areas include support for schema and ontology integration, entity resolution as well as mashup-like data integration workflows.

In this section, we outline the new version of our match tool called COMA 3.0 that includes support for enriched mappings and ontology merging. We start with an overview of the architecture and functionality of COMA 3.0. We then discuss the mapping enrichment and ontology merging components and present preliminary evaluation results. COMA 3.0 will be made available in 2012. We plan to provide two versions: a community version as open source and a professional version to be distributed by a WDI Lab spinoff.

4.1 Overview

The revised architecture of COMA is shown in Figure 1. Several components are analogous to COMA++, including *Import* of schemas and ontologies, auxiliary information and mappings, an *Export* to output models and mappings, and a *Repository* where these information as well as match configurations are kept. Furthermore, there is an *Execution Engine* to execute defined match strategies using matchers and preprocessing and postprocessing steps from different libraries. The following components are largely new:

Configuration Engine which supports both a manual configuration (*expert mode*) as well as an initial kind of automatic configuration (*simple mode*) and validation of workflow configurations. The automatic configuration uses the default match strategy as a starting point and decides about which matchers to add or drop and how to change the approach for combining matcher results. For this purpose, the input schemas/ontologies are checked for the availability of instances, comments, and diverse data types to determine whether the matchers requiring these features can be used at all. Furthermore, we test whether the reuse of previously determined match mappings is applicable. Similar to other systems [16], we also calculate the degree of linguistic and structural similarity of the input models to decide about the use of linguistic and structural matchers. We also check the input sizes to decide about the use of fragment matching.

Enrichment Engine which supports a semi-automatic or manual enrichment of simple 1:1 correspondences into more complex mapping expressions including functions, e. g. to support data transformations. Furthermore, is-a and inverse is-a correspondences between ontologies can be determined as input for ontology merging. More details will be described in subsection 4.2.

Merge and Transformation Engines which support match-driven ontology merging (see subsection 4.3), as well as the generation of executable mappings (queries) for data transformation. The latter functionality is based on the approaches developed for the +Spicy mapping system [20].

User Interfaces. A new GUI as well as programmatic access to the COMA functionality is planned by offering an API and web service interfaces.

We retain virtually all matchers and advanced strategies (*reuse matching, fragment matching, FilteredContext*) of COMA++ and add some new ones. We implemented an additional kind of fragment matching for large schemas/ontologies where fragments are automatically identified by a clustering algorithm based on the structural similarity within schemas/ontologies. The approach is described and evaluated in [1]. For matching life science ontologies, we added a domain-specific matcher called `NameSyn` that exploits the frequent availability of name synonyms for concepts. It therefore considers two concepts to match if either their names or one of their synonyms are highly similar.

There are numerous implementation enhancements to improve performance and scalability. Since the matcher results (similarity matrices) can contain a very large number of similarity values we support their storage either in memory or in a database table. Both implementations support the same interface and are thus internally usable in the same way. Linguistic matching is now based on a set of optimized and more versatile string matchers (Trigram, Jaccard, Levensthein, TF/IDF, etc.) that have been devised in

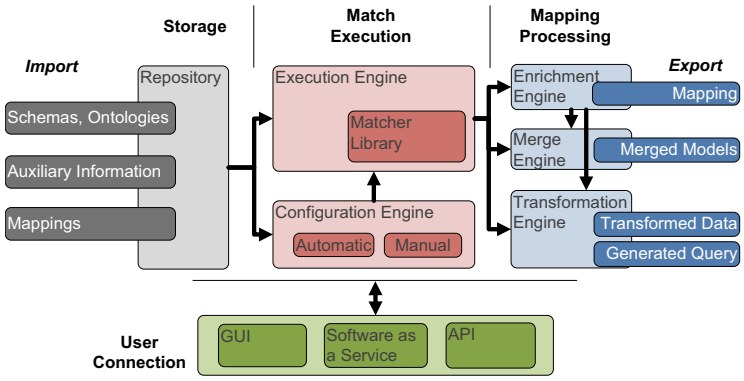

Fig. 1. Architecture of COMA 3.0

the WDI Lab for both entity resolution and schema/ontology matching. In particular the input and output of these matchers can either be in memory, in a file or in a database. Common optimizations of the string matchers include a preprocessing of input strings (stop word removal, tokenization, resolution of abbreviations/synonyms etc.) as well as an early pruning of highly unsimilar pairs of strings.

As mentioned, we plan to make the COMA functionality available as a web service (SaaS). The goal is to broaden the usability of COMA to diverse applications thereby reducing the need to repeatedly re-implement match functionality. Possible consumers include mashups or mobile applications as well as match-driven tools, e. g. for data migration or ontology merging. Furthermore, the match functionality can be utilized within new graphical user interfaces, e. g. a browser-based light-weight GUI for match processing. The core functionality to be provided as web service includes the possibility to load schemas and ontologies, to specify and execute a match strategy and to return computed match mappings. A challenge still to be addressed is support for multi-tenancy so that the data and execution of many users can be isolated from each other.

4.2 Enrichment Engine

The match process of COMA identifies a set of 1:1 equality match correspondences between elements of the input schemas or ontologies. While elements may participate in multiple such correspondences, there is no direct support for complex correspondences interrelating several elements per input schema. For example, there may be correspondences *Name–FirstName* and *Name–LastName* that should in fact be a complex correspondence *Name* – {*FirstName, LastName*}. One task of the enrichment engine is to semi-automatically determine such complex correspondences and extend them with data transformation functions, e. g. to specify that a split of *Name* should be applied to obtain the *FirstName* and *LastName* elements. The second task of the enrichment engine is the derivation of more semantic correspondences for ontologies, in particular is-a and their inverse is-a relationships that can be utilized for ontology merging.

We have extended the mapping structure of COMA to support complex correspondences with data transformation functions. We support similar functions as in commercial

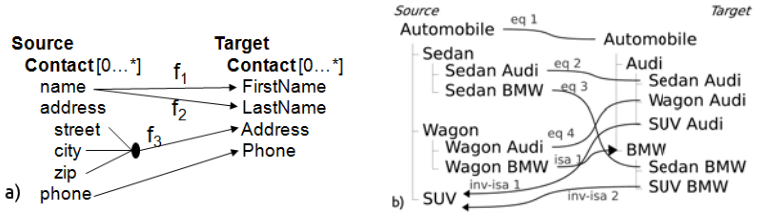

Fig. 2. a) A Mapping Scenario from STBenchmark – b) A Merging Scenario

mapping tools such as BizTalk Server[2], in particular numerical functions (e. g. sum, multiply), string functions (e. g. concat, lowercase, replace), date/time functions, conversion functions, logical functions, relational functions, etc. The semi-automatic determination of complex correspondences and mapping expressions is very complex and only few research approaches have been proposed so far, e. g. iMAP [6].

To find complex correspondences we analyze the set of 1:1 correspondences. In particular we check whether sets of correspondences involving the same source or target element should be combined within a complex correspondence, e. g. if they refer to elements in close proximity such as in the mentioned *Name* example. As usual, all suggestions are subject to user feedback for confirmation or correction.

Our approach to identify applicable transformation functions depends on instance data but also uses element names and structural information. For all correspondences we test and verify the applicability of the type-specific transformation functions. For illustration, consider the example from the STBenchmark[3] shown in Figure 2(a), where three functions are needed to correctly transform source instances into target instances. In particular, f_1 and f_2 describe how to split the source element *name* into the two target elements *FirstName* and *LastName*; similarly, the concatenation function f_3 defines how to combine the three elements *street, city, zip* into the target element *Address*. Our approach is able to find out, for example, the last complex correspondence by starting from the 1:1-correspondence *address–Address* automatically detected by the system and performing a structural analysis of these two elements. Since data is not stored in the inner node *address* but in its children, a complex correspondence will be created out of the subnodes. Furthermore, analyzing instance data is helpful to determine in which order the source elements should be concatenated by f_3.

Besides discovering complex matches, the enrichment engine supports the determination of semantic correspondences between ontologies, in particular is-a and inverse is-a relationships, that can be used for ontology merging. We provide a basic algorithm based on the analysis of labels that can detect textual containment relationships between concepts; this approach can be extended using external linguistic oracles (e. g. WordNet) providing semantic relationship indicators. For illustration, consider the simple example in Figure 2(b) where the catalog of a new online car shop (source) should be merged into the catalog of a price comparison portal (target). Analyzing the labels of the input con-

[2] http://www.microsoft.com/biztalk
[3] http://www.stbenchmark.org/

cepts, we find that the source label *Wagon BMW* "is contained" in the target label *BMW* and then derive that *Wagon BMW "is a" BMW*; similarly for concepts *SUV, SUV Audi* and *SUV BMW*, but in the opposite direction, identifying the inverse-is-a relationships shown in Figure 2(b).

A related approach for semantic matching is supported by S-Match [14] where different matchers are used to discover semantic relations, like equivalence, less general, more general and disjointness between concepts. The less and more general relationships correspond to is-a and inverse-is-a relationships in our approach.

4.3 Ontology Merging

A major extension of COMA 3.0 is the inclusion of an ontology merging component that consumes the match mapping as input and produces an integrated ontology as output, called merged ontology. The main approach is called AUTOMATIC TARGET-DRIVEN ONTOLOGY MERGING (ATOM) and is described in more detail in [27]. The current version is restricted to is-a taxonomies; multiple inheritance and instance data for leaf concepts are supported.

Ontology merging is a difficult problem since there often exists no ideal unique solution. Previous ontology merge approaches are largely user-controlled and do not clearly separate matching from merging resulting in complex approaches, [4], [21], [19], [29]. By utilizing a manually verified match mapping as input, our merge approach is largely automatic. We support two kinds of automatic ontology merging: a *symmetric* or *full merge* as well as an *asymmetric*, *target-driven* merge.

The symmetric approach fully preserves both input ontologies, combining equivalent concepts and maintaining all remaining concepts and relationships of both input ontologies. The main problem of such a full merge result is that maintaining different organizations of the same information can reduce its understandability and introduce multiple inheritance and semantic overlap.

As an alternative we therefore support the new asymmetric ATOM approach that merges the source (first) input ontology into the target (second) input ontology. It thus gives preference to the target ontology and preserves it fully while redundant source concepts and relationships might be dropped. We find that such an asymmetric merge is highly relevant in practice and allows us to incrementally extend the target ontology by additional source ontologies. For example, the product catalog of a merchant may have to be merged as a new source into the existing catalog of a price comparison portal.

Figure 3 shows a COMA 3.0 screenshot on the use of ATOM for merging the ontologies of Figure 2(b). The merge result is shown in the middle. The lines specify mappings between the input ontologies and the merged ontology that are also automatically determined and that can be used to migrate instances. If ATOM is provided with semantic correspondences as discussed in the previous Section, the merge result can be improved by finding a better placement of concepts. The screenshot in Figure 3 shows already the outcome when using the correspondences labeled isa_1, $inv\text{-}isa_1$ and $inv\text{-}isa_2$ in Figure 2(b). These correspondences could be used, for example, to place the concept *Wagon BMW* as a subclass of the *BMW* concept, which would not have been possible with equivalence correspondences alone. More details on our merging approach and its evaluation can be found in [26] and [27].

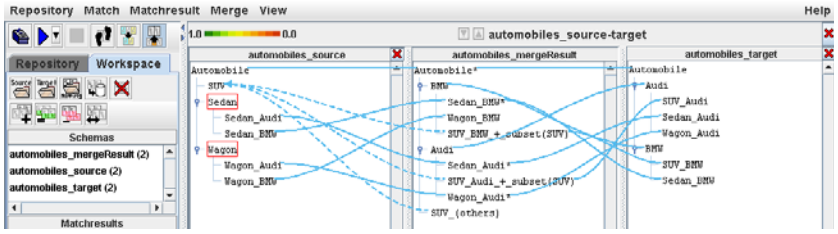

Fig. 3. A merging scenario in COMA 3.0

4.4 Preliminary Evaluation

We have just started to evaluate COMA 3.0 and can therefore discuss only some preliminary results. We report some results for the OAEI Anatomy match task. Some statistics of the two anatomy ontologies are shown in Table 1. The ontologies have around 3000 concepts resulting in about 9 million pairs to evaluate in the Cartesian product. The number of root paths is much higher due to multiple inheritance etc. resulting in about 470 million pairs to evaluate for the Cartesian product. COMA++ used to run into memory problems on the path-based evaluation of the anatomy match task and could thus only solve it with a node-based execution strategy.

With COMA 3.0 eliminating the memory bottleneck we are able to solve this match task for both node-based and path-based evaluation. Due to the high linguistic similarity of the input ontologies, we use a combination of mainly linguistic matchers: a simple `Name` matcher for concepts, the domain-specific `NameSyn` matcher as well as a `Path` matcher comparing the concatenated names of the concepts on the root path.

		Source	Target	Comparisons
OAEI Anatomy	Nodes	2,746	3,306	9 million
	Paths	12,362	39,001	468 million

Table 1. Statistics of the match task

	Source	Target
`Name`	18	18
`Path`	137	156

Table 2. Average string length

Figure 4(a) gives precision, recall, and F-measure results, evaluated against the gold standard containing about 1500 correspondences. The numbers in the combinations denote the weights of the individual matchers as used in the combined similarity value. The best F-measure value of 0.88 is achieved using the combination of `NameSyn`, `Path`, and `Parents` matchers. Figure 4(b) depicts execution times needed for the single matchers on an average workspace PC. The `Name` matcher itself is the fastest with an execution time of around 10 seconds, thereby still achieving 0.81 F-measure. The `Path` matcher requires in total over 41 minutes of which 32 are consumed by the evaluation of the 468 million pairs of paths, which are due to high number of path elements (cf. Table 2). The remaining time mainly is attributed to the combination of similarity values of the multiple paths per concept.

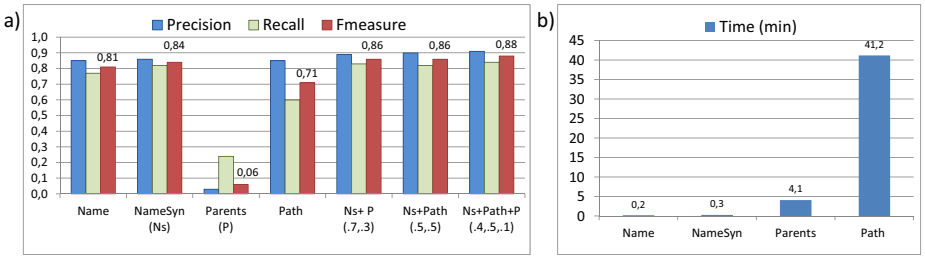

Fig. 4. Results for anatomy matching

5 Conclusions

We described the evolution of the COMA match systems during the last decade, discussed lessons learned, and sketched the upcoming version COMA 3.0. We find that many features and design decisions of the original COMA system are still valid today, and are being used in most other schema and ontology matching tools, such as the multi-matcher architecture. Further strong points include an effective default match configuration and advanced match strategies such as reuse of previous mappings and fragment matching. We also outlined current limitations of COMA++ and discussed how they are partially addressed in COMA 3.0. In particular, the new system provides improved scalability and initial support for self configuration. Furthermore, it supports the generation of enhanced mappings as well as automatic ontology merging. We are currently evaluating and completing the implementation of COMA 3.0. Further extensions are planned for future versions, such as parallel matching.

6 Acknowledgements

We thank Hong Hai Do for implementing COMA and helping in developing COMA++.

References

1. Alsayed Algergawy, Sabine Massmann, and Erhard Rahm. A Clustering-based Approach For Large-scale Ontology Matching. *Proc. ADBIS*, 2011.
2. David Aumüller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and Ontology Matching with COMA++. In *Proc. of ACM SIGMOD*, 2005.
3. Paolo Avesani, Fausto Giunchiglia, and Mikalai Yatskevich. A Large Scale Taxonomy Mapping Evaluation. In *Proc. Int. Conf. Semantic Web (ICSW), LNCS 3729*. Springer-Verlag, 2005.
4. Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comp. Surv.*, 18(4), 1986.
5. Zohra Bellahsene, Angela Bonifati, and Erhard Rahm. *Schema Matching and Mapping*. Data-centric Systems and Applications. Springer, 2011.
6. Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy, and Pedro Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *Proc. of ACM SIGMOD*, 2004.

7. Hong Hai Do. *Schema Matching and Mapping-based Data Integration: Architecture, Approaches and Evaluation*. VDM Verlag, Saarbrücken, Germany, 2007.
8. Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of Schema Matching Evaluations. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*. Springer-Verlag, 2003.
9. Hong-Hai Do and Erhard Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. of VLDB*, 2002.
10. Hong-Hai Do and Erhard Rahm. Matching Large Schemas: Approaches and Evaluation. *Inf. Syst.*, 32, September 2007.
11. Christian Drumm, Matthias Schmitt, Hong-Hai Do, and Erhard Rahm. QuickMig - Automatic Schema Matching for Data Migration Projects. In *Proc. of CIKM*, 2007.
12. Daniel Engmann and Sabine Massmann. Instance Matching with COMA++. In *BTW Workshops*, 2007.
13. Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer, 2007.
14. Fausto Giunchiglia, Aliaksandr Autayeu, and Juan Pane. S-Match: An Open Source Framework for Matching Lightweight Ontologies. *Semantic Web*, 2011.
15. Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber, Martina Seidl, Michael Strommer, and Manuel Wimmer. Matching Metamodels with Semantic Systems - An Experience Report. In *BTW workshop on Model Management*, 2007.
16. Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Trans. Knowl. Data Egineering*, 21(8), 2009.
17. Sabine Massmann, Daniel Engmann, and Erhard Rahm. COMA++: Results for the Ontology Alignment Contest OAEI 2006. *Int. Workshop on Ontology Matching*, 2006.
18. Sabine Massmann and Erhard Rahm. Evaluating Instance-based Matching of Web Directories. *11th International Workshop on the Web and Databases (WebDB)*, 2008.
19. Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. An Environment for Merging and Testing Large Ontologies. In *KR*, 2000.
20. Giansalvatore Mecca, Paolo Papotti, Salvatore Raunich, and Marcello Buoncristiano. Concise and Expressive Mappings with +Spicy. *Proc. VLDB Endow.*, August 2009.
21. Natalya Fridman Noy and Mark A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, 2000.
22. Eric Peukert, Sabine Massmann, and Kathleen Koenig. Comparing Similarity Combination Methods for Schema Matching. In Klaus-Peter Faehnrich and Bogdan Franczyk, editors, *GI Jahrestagung (1)*, volume 175 of *LNI*. GI, 2010.
23. Erhard Rahm. Towards Large-scale Schema and Ontology Matching. In *Schema Matching and Mapping*. Springer, 2011.
24. Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB J.*, 10, April 2001.
25. Erhard Rahm, Hong-Hai Do, and Sabine Massmann. Matching Large XML Schemas. *SIGMOD Record 33(4)*, 2004.
26. Salvatore Raunich and Erhard Rahm. Target-driven Merging of Taxonomies. Technical report, University of Leipzig, 2010.
27. Salvatore Raunich and Erhard Rahm. ATOM: Automatic Target-driven Ontology Merging. In *Proc. of ICDE*, 2011.
28. Pavel Shvaiko and Jérôme Euzenat. Ten Challenges for Ontology Matching. *Proc. OTM Conferences*, 2008.
29. Gerd Stumme and Alexander Maedche. FCA-MERGE: Bottom-Up Merging of Ontologies. In *IJCAI*, 2001.