Towards large-scale schema and ontology matching

Erhard Rahm

Abstract. The purely manual specification of semantic correspondences between schemas is almost infeasible for very large schemas or when many different schemas have to be matched. Hence, solving such large-scale match tasks asks for automatic or semi-automatic schema matching approaches. Large-scale matching needs especially be supported for XML schemas and different kinds of ontologies due to their increasing use and size, e.g. in e-business, web and life science applications. Unfortunately, correctly and efficiently matching large schemas and ontologies is very challenging and most previous match systems have only addressed small match tasks. We provide an overview about recently proposed approaches to achieve high match quality or/and high efficiency for large-scale matching. In addition to describing some recent matchers utilizing instance and usage data, we cover approaches on early pruning of the search space, divide and conquer strategies, parallel matching, tuning matcher combinations, the reuse of previous match results and holistic schema matching. We also provide a brief comparison of selected match tools.

1. Introduction

Schema matching aims at identifying semantic correspondences between metadata structures or models, such as database schemas, XML message formats, and ontologies. Solving such match problems is a key task in numerous application fields, in particular to support data exchange, schema evolution and virtually all kinds of data integration. Unfortunately, the typically high degree of semantic heterogeneity reflected in different schemas makes schema matching an inherently complex task. Hence, most current systems still require the manual specification of semantic correspondences, e.g. with the help of a GUI. While such an approach is appropriate for matching a few small schemas, it is enormously time-consuming and error-prone for dealing with large schemas encompassing thousands of elements or to match many schemas. Therefore, automatic or semi-automatic approaches to find semantic correspondences with minimal manual effort are espe-

Erhard Rahm
University of Leipzig, Germany
e-mail: rahm@informatik.uni-leipzig.de

cially needed for large-scale matching. Typical use cases of large-scale matching include

- Matching large XML schemas, e.g. e-business standards and message formats (Rahm et al. 2004, Smith et al. 2009)
- Matching large life science ontologies describing and categorizing biomedical objects or facts such as genes, the anatomy of different species, diseases, etc. (Kirsten et al. 2007, Zhang et al. 2007)
- Matching large web directories or product catalogs (Avesani et al. 2005, Nandi and Bernstein 2009)
- Matching many web forms of deep web data sources to create a mediated search interface, e.g. for travel reservation or shopping of certain products (He and Chang 2006, Su et al. 2006).

Schema matching (including its ontology matching variant) has been a very active research area, especially in the last decade, and numerous techniques and prototypes for automatic matching have been developed (Rahm and Bernstein 2001, Euzenat and Shvaiko 2007). Schema matching has also been used as a first step to solve data exchange, schema evolution or data integration problems, e.g. to transform correspondences into an executable mapping for migrating data from a source to a target schema (Fagin et al. 2009). Most match approaches focus on *2-way* or *pairwise schema matching* where two related input schemas are matched with each other. Some algorithms have also been proposed for *n-way* or *holistic schema matching* (He and Chang 2006), to determine the semantic overlap in many schemas, e.g. to build a mediated schema. The result of pairwise schema matching is usually an equivalence mapping containing the identified semantic correspondences, i.e. pairs of semantically equivalent schema elements. Some ontology matching approaches also try to determine different kinds of correspondences, such as is-a relationships between ontologies (Spiliopoulos et al. 2010). Due to the typically high semantic heterogeneity of schemas, algorithms can only determine approximate mappings. The automatically determined mappings may thus require the inspection and adaptation by a human domain expert (deletion of wrong correspondences, addition of missed correspondences) to obtain the correct mapping.

Despite the advances made, current match systems still struggle to deal with large-scale match tasks as those mentioned above. In particular, achieving both good effectiveness and good efficiency are two major challenges for large-scale schema matching. *Effectiveness* (high match quality) requires the correct and complete identification of semantic correspondences and is the more difficult to achieve the larger the search space is. For pairwise schema matching the search space increases at least quadratically with the number of elements. Furthermore, the semantic heterogeneity is typically high for large-scale match tasks, e.g. the schemas may largely differ in their size and scope making it difficult to find all correspondences. Furthermore, elements often have several equivalent elements in the other schema that are more difficult to identify than 1:1 correspondences that are more

likely for small match tasks. Some large-scale problems in the OAEI contest (Ontology Alignment Evaluation Initiative) on ontology matching are still not satisfactorily solved after several years. For example the best F-measure[2] result for the catalog test to match web directories (71%) was achieved in 2007; in 2009 the best participating system achieved merely 63%; the average F-measure was around 50% (Euzenat et al. 2009).

Efficiency is another challenge for large-scale matching. Current match systems often require the schemas and intermediate match results to fit in main memory thereby limiting their applicability for large-scale match tasks. Furthermore, evaluating large search spaces is time consuming especially if multiple matchers need to be evaluated and combined. For some OAEI match tasks and systems, execution times in the order of several hours or even days are observed (Euzenat et al. 2009). For interactive use of schema matching systems such execution times are clearly unacceptable.

In this book chapter we provide an overview of recent approaches to improve effectiveness and efficiency for large-scale schema and ontology matching. We only briefly discuss further challenges such as support for sophisticated user interaction or the evaluation of match quality but these are treated in more detail in other chapters of this book (Falconer and Noy 2011, Bellahsene et al. 2011). For example, advanced GUIs should be supported to visualize large schemas and mappings, to specify automatic match strategies (selection of matchers, parameter tuning), to incrementally start automatic schema matching and adapt match results, etc.

In the next section we introduce the kinds of matchers used in current match systems as well as a general workflow to combine the results of multiple matchers for improved match quality. We also discuss performance optimizations for single matchers and present recently proposed approaches for instance-based and usage-based matching. In Section 3 we present several match strategies that we consider as especially promising for large-scale matching: early pruning of the search space, partition-based matching, parallel matching, self-tuning match workflows and reuse-based matching. We also discuss briefly approaches for n-way (holistic) schema matching. Section 4 contains a short discussion of match support in commercial systems and a comparison of selected research prototypes that have been applied to large match problems.

2. Matchers and match workflows

The developed systems for schema and ontology matching typically support several match algorithms (or matchers) and combine their results for improved match quality. There exists a large spectrum of possible matchers and different imple-

[2] F-Measure combines Recall and Precision, two standard measures to evaluate the effectiveness of schema matching approaches (Do et al. 2003)

mentations as surveyed in (Rahm and Bernstein 2001, Euzenat and Shvaiko 2007), in particular metadata-based and instance-based matchers. Metadata-based matchers are most common and exploit characteristics of schema or ontology elements such as their names, comments, data types as well as structural properties. Instance-based matchers determine the similarity between schema elements from the similarity of their instances; this class of matchers has recently been studied primarily for matching large ontologies and will be discussed in more detail below.

Further matching techniques exploit different kinds of auxiliary (background) information to improve or complement metadata- and instance-based matchers. For example, name matching for both schema elements and instance values can be enhanced by general thesauri such as Wordnet or, for improved precision, domain-specific synonym lists and thesauri (e.g., UMLS as a biomedical reference). Furthermore, search engines can be used to determine the similarity between names, e.g. by using the relative search result cardinality for different pairs of names as a similarity indicator (Gligorov et al. 2007). At the end of this section we will briefly discuss a further kind of match technique, the recently proposed consideration of usage information for matching.

Efficiently matching large schemas and ontologies implies that every matcher should impose minimal CPU and memory requirements. For improving linguistic matching many techniques for efficiently computing string similarities can be exploited, e.g. for tokenization and indexing (Koudas et al. 2004). Structural matching can be optimized by precollecting the predecessors and children of every element, e.g. in database tables, instead of repeatedly traversing large graph structures (Algergawy et al. 2009). Such an approach can also avoid the need of keeping a graph representation of the schemas in memory that can become a bottleneck with large schemas. The results of matchers are often stored within similarity matrices containing a similarity value for every combination of schema elements. With large schemas these matrices may require millions of entries and thus several hundreds of MB memory. To avoid a memory bottleneck a more space-efficient storage of matcher results becomes necessary, e.g. by using hash tables (Bernstein et al. 2004). In Section 3, we will discuss further performance techniques such as parallel matcher execution.

In the following, we first describe a general workflow-like approach to apply multiple matchers and to combine their results. We then discuss approaches for instance-based ontology matching and usage-based matching.

2.1 Match workflows

Fig. 1a shows a general workflow for automatic, pairwise schema matching as being used in many current match systems. The schemas are first imported into an internal processing format. Further pre-processing may be applied such as analysis of schema features or indexing name tokens to prepare for a faster computation of name similarities. The main part is a sub-workflow to execute several matchers each of which determines a preliminary set of correspondences. After the execution of the matcher sub-workflow there are typically several post-processing steps,

in particular the combination of the individual matcher results and finally the selection of the correspondences from the combined result.

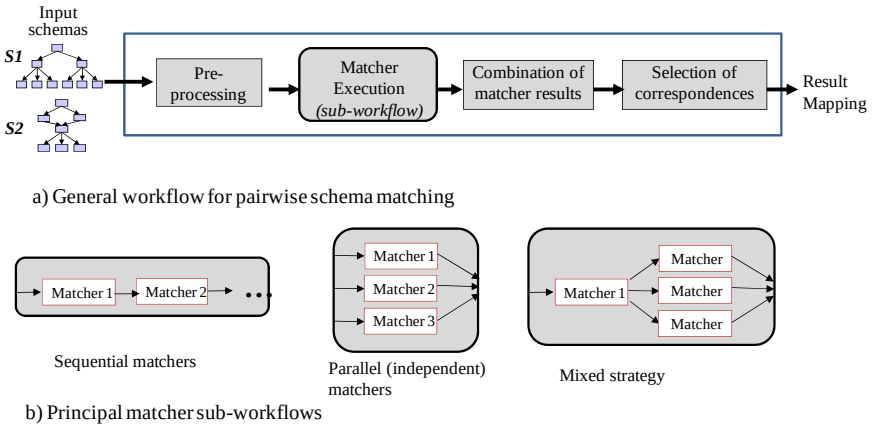

Fig. 1: General match workflows

As indicated in Fig. 1b, the individual matchers may either be executed sequentially, independently (in parallel) or in some mixed fashion. In the sequential approach the matchers are not executed independently but the results of initial matchers are used as input by subsequent matchers. A common strategy, e.g. used in Cupid (Madhavan et al. 2001), is to first execute a linguistic matcher to compare the names of schema elements and then use the obtained similarities as input for structure-based matching. In the parallel matcher strategy, individual matchers are autonomous and can be independently executed from other matchers. This supports a high flexibility to select matchers for execution and combination. Furthermore, these matchers may also physically be executed in parallel, e.g. on multi-core or multi-server hardware. On the other hand, the autonomy of individual matchers may introduce redundant computations, e.g. of name similarities to be used for structural matching. The mixed strategy combines sequential and parallel matcher execution and is thus most complex.

There are different methods to combine match results of individual matchers, e.g., by performing a union or intersection of the correspondences or by aggregating individual similarity values, e.g. by calculating a weighted average of the individual similarities. Similarly there are different methods to finally select the correspondences. Typically correspondences need to exceed some predetermined threshold but may also have to meet additional constraints for improved precision. So it is reasonable for 1:1 mappings to enforce so-called stable marriages, i.e. a correspondence *c1-c1'* is only accepted if *c1'* is the most similar element for *c1* and vice versa. Some ontology matching systems such as ASMOV enforce additional constraints regarding is-a relationships (see Section 4.2).

For interactive schema matching the user may interact with the system and the match workflow in different ways (not shown in Fig. 1), preferably via a user-

friendly GUI. She typically has to specify the workflow configuration, e.g. which matchers should be executed and which strategy/parameters should be applied for the final combination and selection steps. The final results are typically only suggested correspondences that the user can confirm or correct. The match workflow itself could be executed on the whole input schemas or *incrementally* for selected schema parts or even individual elements (Bernstein et al. 2006). The latter approach is a simple but reasonable way to better deal with large schemas as it reduces the performance requirements compared to matching the whole schemas. Furthermore, the determined correspondences can better be visualized avoiding that the user is overwhelmed with huge mappings. (Shi et al. 2009) propose an interesting variation for interactive matching where the system asks the user for feedback on specific correspondences that are hard to determine automatically and that are valuable as input for further matcher executions.

2.2 Instance-based and usage-based matching

Instance-based ontology matching

Instance-based ontology matching determines the similarity between ontology concepts from the similarity of instances associated to the concepts. For example, two categories of a product catalog can be considered as equivalent if their products are largely the same or at least highly similar. One can argue that instances can characterize the semantics of schema elements or ontology concepts very well and potentially better than a concept name or comment. Therefore, instance-based matching holds the promise of identifying high-quality correspondences. On the other hand, obtaining sufficient and suitable instance data for all ontologies and all ontology concepts to be matched is a major problem, especially for large ontologies. Hence, we consider instance-based approaches primarily as a complementary, albeit significant match approach to be used in addition to metadata-based matchers.

As indicated in Fig. 2, two main cases for instance-based ontology matching can be distinguished depending on whether or not the existence of common instances is assumed. The existence of the same instances for different ontologies (e.g., the same web pages categorized in different web directories, the same products offered in different product catalogs, or the same set of proteins described in different life science ontologies) simplifies the determination of similar concept. In this case, two concepts may be considered as equivalent when their instances overlap significantly. Different set similarity measures can be used to measure such an instance overlap, e.g. based on Dice, Jaccard, or cosine similarity. The instance overlap approach has been used to match large life science ontologies (Kirsten et al. 2007) and product catalogs (Thor et al. 2007).

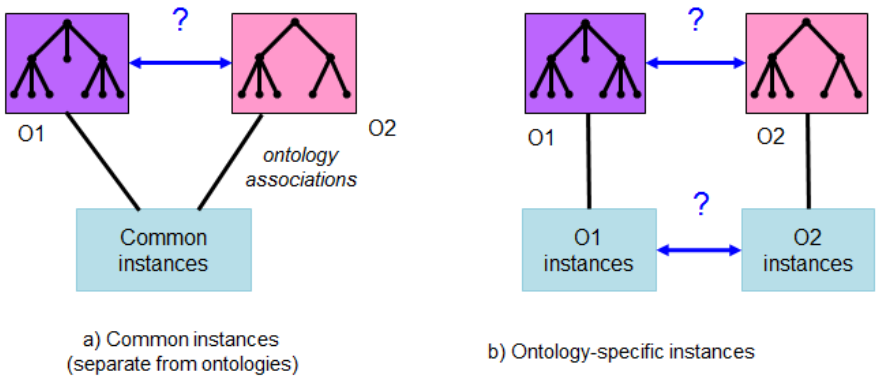

Fig. 2: Two cases for instance-based ontology matching

The more broadly applicable case is when only similar but potentially different instances are used to determine correspondences (Fig. 2b). In this case, determining concept similarity requires determining the similarity between sets of instances which is a variation of the well-studied problem of object matching or entity resolution (Elmagarmid et al. 2007, Koepcke and Rahm 2010). One approach for addressing this task is to combine all instances of a concept into a *virtual document.* Matching is then implemented by comparing such virtual documents with each other based on some document similarity measure, e.g. TF/IDF. This approach is supported in several match prototypes including Rimom and Coma++ (see Section 4.2). (Massmann and Rahm 2008) evaluate instance-based matching for web directories utilizing a virtual document approach for website names and descriptions as well as an instance overlap approach for website URLs. The approaches achieve an average F-measure of about 60% (79% in combination with metadata-based matching) for different match tasks; the largest directory had more than 3000 categories. The OAEI contest also includes a web directory match task however without providing instance data thereby limiting the achievable match quality (as mentioned in the introduction the participating systems could not improve on this task in recent years; the best F-measure in 2009 was 63%).

An alternate approach for instance-based matching using machine learning has been implemented in the GLUE and SAMBO systems (Doan et al. 2003), (Lambrix et al 2008). The SAMBO approach focuses on matching life science ontologies based on the similarity of publications (Pubmed abstracts) referring to the ontology concepts. Both GLUE and SAMBO perform a training phase per ontology to learn concept classifiers for the available instances. These classifiers are then mutually applied to the instances from the other ontology to determine the concepts an instance is predicted to belong to. The instance-concept associations are aggregated, e.g. by a Jaccard-based set similarity measure, to derive concept similarities and concept correspondences. The approaches do not require shared instances but only similar ones for classification. Furthermore, they can utilize many existing instances in applications such as matching product catalogs or web direc-

tories. On the other hand, the classification problem becomes inherently more difficult to solve for increasing numbers of concepts. The GLUE evaluation in (Doan et al. 2003) was restricted to comparatively small match tasks with ontology sizes between 31 and 331 concepts. The SAMBO approach was evaluated for even smaller (sub-) ontologies (10-112 concepts). Effectiveness and efficiency of the machine learning approaches to large-scale match tasks with thousands of concepts is thus an open issue.

Usage-based matching

Two recent works propose the use of query logs to aid in schema matching. In (Emelgee et al. 2008), SQL query logs are analyzed to find attributes with similar usage characteristics (e.g. within join conditions or aggregations) and occurrence frequencies as possible match candidates for relational database schemas. The Hamster approach (Nandi and Bernstein 2009) uses the click log for keyword queries of an entity search engine to determine the search terms leading to instances of specific categories of a taxonomy (e.g. product catalog or web directory). Categories of different taxonomies sharing similar search queries are then considered as match candidates. Different search terms referring to the same categories are also potential synonyms that can be utilized for matching but also for other purposes such as the improvement of user queries.

A main problem of usage-based matching is the difficulty to obtain suitable usage data, which is likely more severe than the availability of instance data. For example, the click logs for the Hamster approach are only available to the providers of search engines. Furthermore, matching support can primarily be obtained for categories or schema elements receiving many queries.

3. Techniques for large-scale matching

In this section, we provide an overview about recent approaches for large-scale pairwise matching that go beyond specific matchers but address entire match strategies. In particular, we discuss approaches in four areas that we consider as especially promising and important:

- Reduction of search space for matching (early pruning of dissimilar element pairs, partition-based matching)
- Parallel matching
- Self-tuning match workflows
- Reuse of previous match results

We also discuss proposed approaches for holistically matching n schemas.

3.1 Reduction of search space

The standard approach for pairwise schema matching is to compare every element of the first schema with every element with the second schema to determine matching schema elements, i.e. evaluation of the cross join. Such an approach has at least a quadratic complexity w.r.t. schema size and does not scale well. There are not only efficiency problems for large schemas but the large search space makes it also very difficult to correctly identify matching element pairs. Hence, it is important for large match tasks to reduce the search space in order to improve at least efficiency and potentially match quality.

To reduce the search space for matching we can adopt similar approaches as in the area of entity resolution (or object matching) where the number of objects and thus the search space is typically much larger than for schema matching. The initial step to reduce the search space for entity matching has been called blocking and there exist numerous approaches for this task, e.g. based on clustering on selected object attributes (Elmagarmid et al. 2007).

For schema and ontology matching, two main types of approaches have been considered to reduce the search space that we discuss in the following:

- Early pruning of dissimilar element pairs
- Partition-based matching

Early pruning of dissimilar element pairs

The idea is to limit the evaluation of the cartesian product to at most a few initial steps in the match workflow, e.g. one matcher, and to eliminate all element pairs with very low similarity from further processing since they are very unlikely to match. This idea is especially suitable for workflows with sequential matchers where the first matcher can evaluate the cartesian product but all highly dissimilar element pairs are excluded in the evaluation of subsequent matchers and the combination of match results.

Quick Ontology Matching (QOM) was one of the first approaches to implement this idea (Ehrig and Staab 2004). It iteratively applies a sequence of matchers and can restrict the search space for every matcher. The considered approaches to restrict the search space include focusing on elements with similar names (labels) or similar structural properties. The authors showed that the runtime complexity of QOM can be reduced to ($O(n \cdot \log(n))$ instead of $O(n^2)$ for ontologies of size n.

(Peukert et al 2010a) propose the use of filter operators within match workflows to prune dissimilar element pairs (whose similarity is below some minimal threshold) from intermediate match results. The threshold is either statically predetermined or dynamically derived from the similarity threshold used in the match workflow to finally select match correspondences. (Peukert et al 2010a) also propose a rule-based approach to rewrite match workflows for improved efficiency, in particular to place filter operators within sequences of matchers.

Partition-based matching

Partition-based matching is a divide-and-conquer strategy to first partition the input schemas / ontologies and then perform a partition-wise matching. The idea is to perform partitioning in such a way that every partition of the first schema has to be matched with only a subset of the partitions (ideally, only with one partition) of the second schema. This results in a significant reduction of the search space and thus improved efficiency. Furthermore, matching the smaller partitions reduces the memory requirements compared to matching the full schemas. To further improve performance, the partition-based match tasks may be performed in parallel.

There are many possible ways to perform partitioning and finding the most effective approaches is still an open research problem. COMA++ was one of the first systems to support partition-based schema matching by a so-called *fragment matching* (Aumueller et al. 2005), (Do and Rahm 2007). Fragment matching works in two phases. In the first phase the fragments of a specified type (e.g. user-specified fragments or subschemas such as relational tables or message formats in large XML schemas) are determined and compared with each other to identify the most similar fragments from the other schema worth to be fully matched later. The search for similar fragments is some kind of light-weight matching, e.g. based on the similarity of the fragment roots. In the second phase, each pair of similar fragments is independently matched to identify correspondences between their elements. The fragment-based match results are finally merged to obtain the complete output mapping. In the evaluation for large XML schemas in (Do and Rahm 2007) fragment matching not only improved execution times significantly but also lead to a slight improvement of match quality.

The ontology matching system *Falcon-AO* also supports partition-based matching to reduce the search space (Hu et al. 2008). The approach is similar to fragment matching but uses a structural clustering to initially partition the ontologies into relatively small, disjoint blocks. Matching is then restricted to the most similar blocks from the two ontologies. To determine the block similarity, Falcon-AO utilizes so-called *anchors*. Anchors are highly similar element pairs that are determined before partitioning by a combined name/comment matcher. Fig. 3 illustrates the idea to limit matching to pairs of similar partitions sharing at least one anchor. In the shown example only partitions of the same color are matched with each other (e.g., B_{S2} with B_{T2}), while partitions without shared anchor (B_{T3}) are excluded from matching. An extension of the Falcon approach has been proposed for the *Taxomap* system (Hamdi et al. 2009). They first partition only one of the ontologies like in Falcon and then try to partition the second ontology accordingly. In particular it is tried to achieve that the anchors per partition can be localized in few partitions of the other ontology to reduce the number of pairs to be matched.

The taxonomy matching system *AnchorFlood* implements a dynamic partition-based matching that avoids the a-priori partitioning of the ontologies (Hanif and Aono 2009). It also utilizes anchors (similar concept pairs) but takes them as a starting point to incrementally match elements in their structural neighborhood until no further matches are found or all elements are processed. Thus the partitions

(called segments) are located around the anchors and their size depends on the continued success of finding match partners for the considered elements.

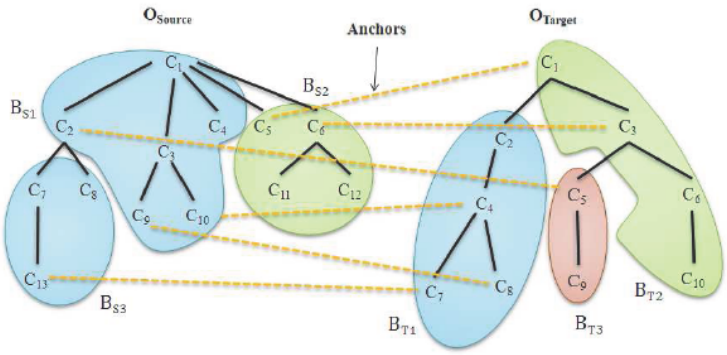

Fig. 3: Partition-based matching in Falcon-AO and Taxomap (from (Hamdi et al. 2009))

(Zhong et al. 2009) focus on the case when a small ontology is matched with a much larger one, e.g. one that is obtained from merging several others. They determine the subontology (partition) from the larger ontology that is most similar to the smaller ontology and consider only this subontology for matching to improve efficiency. Finding the subontology is performed in two steps. First, a name matcher is applied on the Cartesian product of elements to determine the most similar ontology elements from the large ontology. Then, the subontology is determined by evaluating the subgraphs around the similar elements found in the first step.

3.2 Parallel matching

A relatively straight-forward approach to reduce the execution time of large-scale matching is to run match processes in parallel on several processors. As discussed in (Gross et al. 2010), two main kinds of parallel matching are applicable: inter- and intra-matcher parallelization. *Inter-matcher parallelization* enables the parallel execution of independently executable (parallel) matchers in match workflows. This kind of parallelism is easy to support and can utilize multiple cores of a single computing node or multiple nodes. However, inter-matcher parallelization is limited by the number of independent matchers and not applicable for sequential matchers. Furthermore, matchers of different complexity may have largely different execution times limiting the achievable speedup (the slowest matcher determines overall execution time). Moreover, the memory requirements for matching are not reduced since matchers evaluate the complete ontologies.

Intra-matcher parallelization is more versatile and deals with internal parallelization of matchers, typically based on a partitioning of the schemas or ontologies to be matched. Partitioning leads to many smaller match tasks that can be executed in

parallel with reduced memory requirements per task. By choosing appropriate partition sizes the approach becomes very flexible and scalable. Furthermore, intra-matcher parallelism can be applied for sequential as well as independently executable matchers, i.e., it can also be combined with inter-matcher parallelism.

The partition-based matching discussed in Section 3.1 inherently supports intra-matcher parallelization as well as a reduction of the search space by limiting matching to pairs of similar partitions. However, intra-matcher parallelization could also be applied without reduced search space by matching all partition pairs, i.e. to evaluate the Cartesian product in parallel. As discussed in (Gross et al. 2010) such a simple, generic parallelization is applicable for virtually all element-level matchers (e.g. name matching) but can also be adapted for structural matching. In this case, one can also choose a very simple, size-based partitioning (same number of elements per partition) supporting good load balancing.

3.3 Self-tuning match workflows

The match workflows in most current systems need to be manually defined and configured. This affects the choice of matchers to be applied and specification of the methods to combine matcher results and to finally select match correspondences. Obviously, these decisions have a significant impact on both effectiveness and efficiency and are thus especially critical for large-scale match tasks. Unfortunately, the huge number of possible configurations makes it very difficult even for expert users to define suitable match workflows. Hence, the adoption of semi-automatic tuning approaches becomes increasingly necessary and should especially consider the challenges of matching large schemas.

The companion book chapter (Bellahsene and Duchateau 2011) provides an overview of recent approaches including tuning frameworks such as Apfel and eTuner (Ehrig et al. 2005), (Lee et al. 2007). Most previous approaches for automatic tuning apply supervised machine learning methods. They use previously solved match tasks as training to find effective choices for matcher selection and parameter settings such as similarity thresholds and weights to aggregate similarity values, e.g. (Duchateau et al. 2009). A key problem of such approaches is the difficulty of collecting sufficient training data that may itself incur a substantial effort. A further problem is that even within a domain the successful configurations for one match problem do not guarantee sufficient match quality for different problems, especially for matching large schemas. Therefore, one would need methods to preselect suitable and sufficient training correspondences for a given match task, which is an open challenge.

(Tan and Lambrix 2007) propose an alternative approach that recommends a promising match strategy for a given match problem. They first select a limited number of pairs of small segments from the schemas to be matched (e.g. subgraphs with identically named root concepts) and determine the perfect match result for these segments. These results are used to comparatively evaluate the effectiveness and efficiency of a predetermined number of match strategies from which the best performing one is recommended for the complete match task. The ap-

proach thus requires manual "training" for matching the preselected segment pairs; this effort pays off if it helps to avoid a larger amount of manual post-processing. On the other hand, the number of reasonable match strategies can be very high (many combinations of available matchers, many possible similarity thresholds, etc.) so that likely only a small subset of them can be evaluated (in the evaluation merely 30 strategies are considered).

Several match systems first analyze the schemas to be matched and determine their linguistic and structural similarity. These similarity characteristics are then used to select matchers or to weight the influence of different matchers in the combination of matcher results (Pirro and Talia 2010). The Rimom system (Li et al. 2009) uses such similarity factors for dynamically selecting matchers for a specific match task. For example, they use string measures for name matching only if the input schemas have highly similar names; otherwise, they rely on thesauri such as Wordnet. Similarly, they apply structural matching only if the input schemas are deeply structured and structurally similar. Falcon-AO uses the linguistic and structural similarities to combine matcher results, in particular to optimize individual similarity (cutoff) thresholds (Hu et al. 2008). For example, if the linguistic similarity is high, Falcon-AO uses lower thresholds for linguistic matchers so that more of their correspondences are considered.

A versatile approach to combine the results of individual matchers is to determine a weighted average of the individual matcher similarities per correspondence and to accept a correspondence if the combined similarity exceeds some threshold. Several approaches try to tune matcher combination by applying task-specific weights and combination methods, e.g. by favoring higher similarity values (Ehrig and Staab 2004), (Mao et al. 2008), (Mork et al. 2008). For example, the approach of (Mao et al. 2008), used in the PRIOR+ match prototype, combines similarity values according to the matchers' so-called harmony value that is defined as the ratio of element pairs for which a matcher achieved the top similarity values. The comparative analysis in (Peukert et al. 2010b) showed that such combination approaches can be effective in some cases but that they are mostly less effective and less robust than generic approaches such as taking the average matcher similarity.

Optimizing complete match workflows is so far an open challenge, especially since most match systems prescribe workflows of a fixed structure, e.g. regarding which matchers can be executed sequentially or in parallel. As discussed in section 3.1, (Peukert et al 2010a) propose a first approach for tuning match workflows focusing on reducing the search space for improved efficiency.

3.4 Reuse of previous match results

A promising approach to improve both the effectiveness and efficiency of schema matching is the reuse of previous match results to solve a new but similar match task (Rahm and Bernstein 2001). An ideal situation for such a reuse is the need to adapt a mapping between two schemas after one of them evolves to a new schema version. By reusing the previous match mapping for the unchanged schema parts a significant amount of match effort can likely be saved. The reuse of previously de-

termined correspondences and match results may also be applicable in other cases, especially when different schemas share certain elements or substructures, such as standardized address information. Exploiting the reuse potential requires a comprehensive infrastructure, in particular a repository to maintain previously determined correspondences and match results. Furthermore, methods are necessary to determine the schema elements and fragments for which match reuse is applicable. Reuse can be exploited at three mapping granularities: for individual element correspondences, for mappings between common schema fragments and for complete mappings and schemas.

Coma and its successor Coma++ support the reuse of complete match mappings (Do and Rahm 2002). They apply a so-called MatchCompose operator for a join-like combination of two or more match mappings to indirectly match schemas. For example, a mapping between schemas *S1* and *S3* can be obtained by combining preexisting *S1*-*S2* and *S2*-*S3* mappings involving another schema *S2*. For two schemas to be matched, the reuse matcher of Coma searches the repository for all applicable mapping paths connecting the two schemas and combines the composition results just like other matcher results. The reuse matcher can also be combined with regular matchers. Furthermore, the compose approach allows the adaptation of an old mapping after one of the schema evolves. Fig. 4 shows a Coma++ screenshot for such a reuse scenario where the target schema (shown on the right) has been slightly changed. The mapping between the source schema and the old target schema (shown in the middle) can be reused by composing it with the mapping between the old and the new target schema. The latter mapping has to be newly determined but this is easy when the two schema versions are highly similar as in the example. The evaluations in (Do and Rahm 2002) and (Do 2006) showed the high effectiveness and efficiency of the reuse approach even when only completely automatically determined match results are composed.

The corpus-based match approach of (Madhavan et al. 2005) uses a domain-specific corpus of schemas and match mappings and focuses on the reuse of element correspondences. They augment schema elements with matching elements from the corpus and assume that two schema elements match if they match with the same corpus element(s). They use a machine learning approach to find matches between schema and corpus elements. In particular, for each corpus element c a model based on several matchers is learned to decide whether schema elements s match c. The approach thus requires a substantial effort for learning the models and applying the models to determine the matches, especially for a large corpus and large schemas.

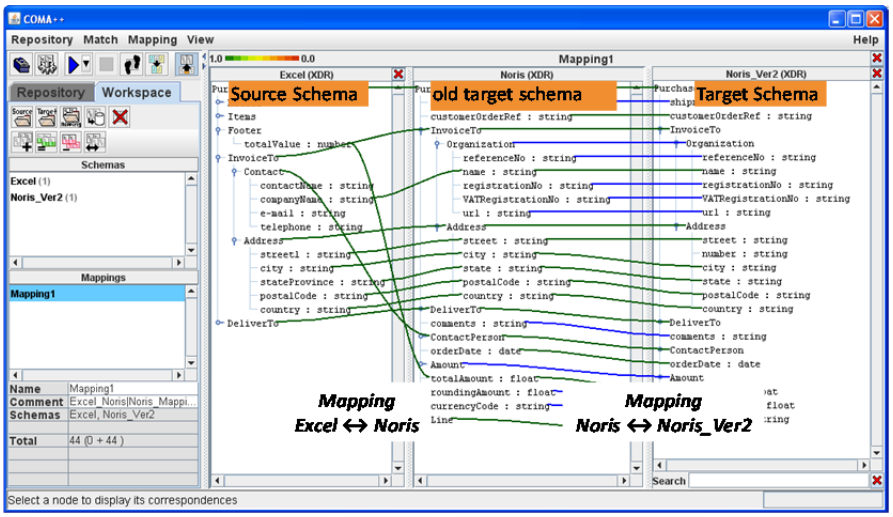

Fig.4: Coma++ reuse scenario after the evolution of the target schema

There are several other attempts to provide repositories of schemas and mappings for matching or information integration, in general. For example, the OpenII project is developing an infrastructure for information integration that includes a repository of schemas and mappings to permit their reuse (Seligman et al. 2010). While the OpenII schema matcher, Harmony, does not yet exploit this reuse potential there are several other tools to explore and visualize the schemas and mappings. In particular, the Schemr search tool determines a ranked list of schemas in the repository that are similar to a given schema fragment or list or keywords (Chen et al 2009). For this purpose, Schemr uses an index on schema element names to first find repository schemas that are linguistically similar to the search input. In a second step, the candidate schemas are matched with the input schema to obtain refined schema similarities used for ranking. The search tool could thus be useful to determine relevant schemas for reuse.

A new project at the IBM Almaden research center investigates the repository-based reuse of schema fragments and mappings, in particular for enhancing schema matching (Alexe et al. 2009). The repository stores conceptual schema fragments called UFOs (Unified Famous Objects) such as address or employee structures that are in use or relevant for different applications and schemas. By maintaining mappings between similar UFOs in the repository, these mappings may be reused when matching schemas that contain the respective UFOs. Successfully implementing such an idea is promising but also highly complex and apparently not yet finished. First the repository has to be built and populated; a first design is sketched in (Gubanov et al. 2009). For schema matching, the schemas to be matched have to be analyzed whether they include schema fragments from the repository for which mappings exist. Finally the fragment mappings need to be properly assembled (and combined with the results of other matchers) in order to obtain a complete schema mapping. (Saha et al. 2010) focuses on the second step

and describes an approach called *schema covering* to partition the input schemas such that the partitions can be matched to schema fragments in the repository. They first apply a filter step to determine relevant repository fragments with some similarity with the schemas to be matched. Then for each of the remaining repository fragments the maximal subgraphs in the schemas are determined that can be covered by the fragment. To speed-up the similarity computations, filtering and subgraph identification utilize a predetermined index of the schema element names and their positions in the repository schemas and the schemas to be matched.

SAP also works on an ambitious project called Warp10 to exploit the reuse of XML schemas and mappings for business integration including improved schema matching (SAP 2010). As indicated in Fig. 5, the key idea is to maintain a central repository maintaining a global schema (called consolidated data model) that includes semantically consolidated versions of individual schemas or schema fragments. Consolidation is based on the CCTS (UN/CEFACT Core Component Technical Specification) rules for uniformly naming and structuring concepts. The global schema is initially populated by standard business schemas and data types (e.g., from SAP) and can be semi-automatically and collaboratively extended by integrating specific schemas. The correct integration of such schemas is likely a complex merge operation needing the control by domain experts. Once mappings between schemas and the global schema are established, they can be reused for quickly solving new integration tasks, in particular for matching between schemas (to match schemas *S1* with *S2* one has to compose their mappings with the global schema *G*, *S1*-*G* and *G*-*S2*). Unfortunately, the details about how the global schema is maintained are not yet published.

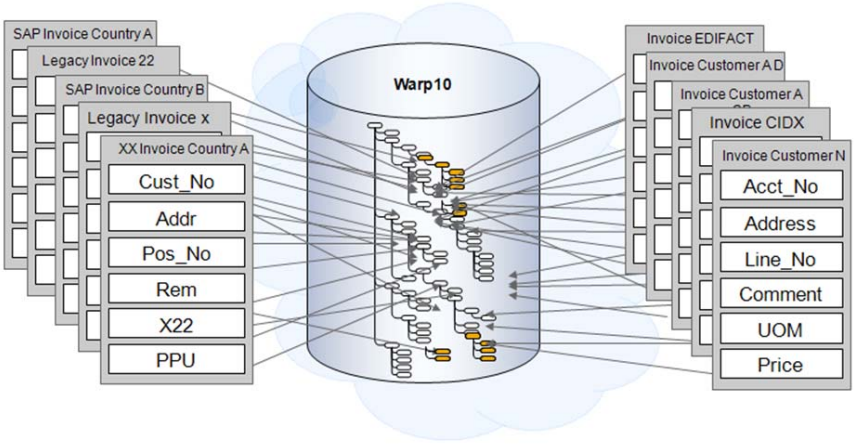

Fig 5: Mappings between schemas and the consolidated data model in Warp10 (from (SAP 2010))

3.5 Holistic schema matching

While most of the previous match work focuses on pairwise matching, there has also been some work on the generalized problem of matching n schemas. Typically, the goal is to integrate or merge the n schemas such that all matching elements of the n schemas are represented only once in the integrated (mediated) schema. N-way matching can be implemented by a series of 2-way match steps and some systems such as Porsche follow such an approach and incrementally merge schemas (Saleem et al. 2008). The alternative is a holistic matching that clusters all matching schema elements at once.

The holistic approach has primarily been considered for the use case of matching and integrating web forms for querying deep web sources (He et al. 2004), (He and Chang 2006), (Su et al. 2006). While there are typically many web forms to integrate in a domain, the respective schemas are mostly small and simple, e.g. a list of attributes. Hence, the main task is to group together all similar attributes. Matching is primarily performed on the attribute names (labels) but may also use additional information such as comments or sample values. A main observation utilized in holistic schema matching is the correlation of attribute names, in particular that similar names between different schemas are likely matches but similar names within the same schema are usually mismatches. For example, attributes *first name* and *last name* do not match if they co-occur in the same source.

The DCM (Dual Correlation Mining) approach of (He and Chang 2006) utilizes these positive and negative attribute correlations for matching. It also utilizes negative correlations to derive complex relationships, e.g. that attribute *name* matches the combination of both *first name* and *last name*. The HSM approach of (Su et al. 2006) extends the DCM scheme for improved accuracy and efficiency. HSM also utilizes that the vocabulary of web forms in a domain tends to be relatively small and that terms are usually unambiguous in a domain (e.g. *title* in a book domain). A main idea is to first identify such shared attributes (and their synonyms) in the input schemas and exclude such attributes from matching the remaining attributes for improved efficiency and accuracy.

(Das Sarma et al. 2008) propose to determine a so-called *probabilistic mediated schema* from n input schemas, which is in effect a ranked list of several mediated schemas. The approach observes the inherent uncertainty of match decisions but avoids any manual intervention by considering not only one but several reasonable mappings. The resulting set of mediated schemas was shown to provide queries with potentially more complete results than with a single mediated schema. The proposed approach only considers the more frequently occurring attributes for determining the different mediated schemas, i.e. sets of disjoint attribute clusters. Clustering is based on the pairwise similarity between any of the remaining attributes exceeding a certain threshold as well as the co-occurrence of attributes in the same source. The similarity between attributes can also be considered as uncertain by some error margin which leads to different possibilities to cluster such attributes within different mediated schemas. The probabilistic mapping approach is further described in the companion book chapter (Das Sarma et al. 2010).

We finally note that some of the partition-based and reuse-based match approaches discussed above dealt with multiple subschemas so they also implement some form of n-way schema matching. An important building block in such advanced match strategies is to search a collection of n (sub) schemas for the schema that is most similar to a given schema. There are many other applications for the schema search problem, e.g. finding similar peer schemas in P2P data integration or the discovery of suitable web services (Dong et al. 2004, Algergawy et al. 2010).

4 Selected match systems

To further illustrate the state of the art, we discuss in this section the schema matching capabilities in commercial tools as well as in selected research prototypes. For better comparability, we restrict ourselves on systems for pairwise schema matching.

4.1 Commercial match tools

In commercial tools, schema matching is typically a first step for generating executable mappings (e.g., for data transformation) between schemas, in particular XML schemas or relational database schemas. Systems such as IBM Infosphere Data Architect, Microsoft Biztalk server, SAP Netweaver Process Integration or Altova MapForce provide a GUI-based mapping editor but still require a largely manual specification of the match correspondences. In recent years, support for automatic matching has improved and all mentioned systems can present users equally named schema elements (typically within preselected schema fragments) as match candidates. The Infosphere mapping editor also supports approximate name matching and the use of external thesauri for linguistic matching. The mapping tool of Microsoft Biztalk server 2010 has significantly improved for better matching large schemas (www.microsoft.com/biztalk). It supports an enhanced user interface to better visualize complex mappings similar as described in (Bernstein et al 2006). Furthermore, it supports approximate name matching by a new search functionality called “indicative matching”.

The increasing support in commercial tools underlines the high practical importance of automatic schema matching. However, the tools need much further improvement to reduce the manual mapping effort especially for large match tasks. For example, commercial tools do neither support structural matching nor any of the advanced techniques discussed in Section 3.

4.2 Research prototypes

As already discussed in the previous sections, in research more advanced approaches for semi-automatic schema and ontology matching have been developed. In fact, hundreds of prototypes and algorithms for schema and ontology matching have been developed in the last decade, many of which are surveyed in (Euzenat and Shvaiko 2007). To illustrate the state of the art in current tools, we briefly

compare a few recent prototypes that have successfully been applied to large-scale match problems, in particular within the OAEI (Ontology Alignment Evaluation Initiative) benchmark competition (http://oaei.ontologymatching.org). Table 1 provides a rough comparison between six match prototypes. All of them are capable of matching (OWL) ontologies, two systems (Coma++, Harmony) can also match relational and XML schemas. The shown years of introduction are estimates based on the oldest publication found per system.

The table indicates that all systems include linguistic and structural matchers. Linguistic matching can always be performed either on element names and comments; furthermore, external dictionaries such as synonym lists or thesauri can be utoilized. Most systems (except Falcon and Harmony) also support instance-based matching. A comprehensive GUI for interactive matching is provided by Coma++, AgreementMaker and Harmony; for the other systems no specific details on this issue could be found. The individual matchers may either be executed independently or sequentially within a predetermined, fixed match workflow (not mentioned in Table 1). Partitioning of large schemas is currently supported by two of the considered systems (Coma++, Falcon), a self-tuning by dynamically selecting the matchers to execute only by Rimom. Coma++ is the only system supporting the reuse of previous mappings for matching. All systems except Harmony have successfully participated in OAEI ontology matching contests; some systems even implemented specific extensions to better solve certain contest tasks. Not shown in the table is that most systems focus on 1:1 correspondences, although the elements of a schema may participate in several such correspondences (the fact that element Name in one schema matches to the combination of Firstname and Lastname in the second schema can thus not directly be determined). Parallel matching (section 3.2) is also not yet supported in the tools.

In the following, we provide some specific details for the considered prototypes.

Coma++

Coma++ (Aumueller et al. 2005), (Do and Rahm 2007) and its predecessor Coma (Do and Rahm 2002) are generic prototypes for schema and ontology matching developed at the University of Leipzig, Germany. They were among the first systems to successfully support the multi-matcher architecture and match workflows as introduced in Section 2. Initially the focus was on a metadata-based matching; instance-based matching was added in 2006. Coma++ supports the partitioning and reuse approaches discussed in the previous section.

Coma++ is available for free for research purposes and hundreds of institutes worldwide have used and evaluated the prototype. Surprisingly, the default match workflow of Coma++ (combining four metadata-based matchers) proved to be competitive in many diverse areas, in particular for matching XML schemas (Algergawy et al. 2009), web directories (Avesani et al. 2005) or even meta-models derived from UML (Kappel et al. 2007). Coma++ successfully participated in the ontology matching contest OAEI 2006.

Table 1: Comparison of match prototypes (AM=AgreementMaker)

		COMA++	Falcon	Rimom	Asmov	AM	Harmony
year of introduction		2002/2005	2006	2006	2007	2007	2008
Input	*relational*	√	-	-	-	-	√
schemas	*XML*	√	-	-	-	(√)	√
	ontologies	√	√	√	√	√	√
compreh. GUI		√	(√)	?	?	√	√
Matchers	*linguistic*	√	√	√	√	√	√
	structure	√	√	√	√	√	√
	Instance	√	-	√	√	√	-
use of ext.dictionaries		√	?	√	√	√	√
schema partitioning		√	√	-	-	-	-
parallel matching		-	-	-	-	-	-
dyn. matcher selection		-	-	√	-	-	-
mapping reuse		√	-	-	-	-	-
OAEI participation		√	√	√	√	√	-

Falcon

Falcon-AO is an ontology matching prototype developed at the Southeast University in Nanjing, China (Hu and Qu 2008). As discussed in Section 3.1, it supports a partitioning approach to reduce the search space and uses coefficients of the linguistic and structural schema similarity to control the combination of matcher results. Instance-based matching is not yet provided. In the OAEI contests 2005-2007 it was among the best performing systems.

Rimom

Rimom is an ontology matching prototype developed at Tsinghua University in Beijing, China (Li et al. 2009). It was one of the first systems implementing a dynamic selection of matchers, as discussed in Section 3.3. The schema elements and their instances are first linguistically matched; structural matching is only applied if the schemas exhibit sufficient structural similarity. There are several methods for linguistic matching including one that uses a virtual document per element consisting of the name, comments and instance values of the element. Rimom is among the best performing prototypes in the OAEI contests until 2009.

Asmov

ASMOV (Automated Semantic Matching of Ontologies with Verification) prototype (Jean-Mary et al. 2009) is among the best performing systems at the recent OAEI match contests. Its most distinctive feature is an extensive post-processing of the combined matcher results to eliminate potential inconsistencies among the set of candidate correspondences. Five different kinds of inconsistencies are checked including the avoidance of so-called crisscross correspondences, e.g. to prevent that for a correspondence between classes *c1* and *c1'* there is another correspondence mapping a child of *c1* to a parent of *c1'*.

AgreementMaker

This ontology matching prototype is developed at the University of Illinois at Chicago (Cruz et al. 2009). It provides a sophisticated GUI so that the user can control the iterative execution of matchers. AgreementMaker was among the best performing systems in the OAEI 2009 contest.

Harmony

Harmony is the match component within the Open Information Integration project on developing a publicly available infrastructure for information integration (Seligman et al. 2010). It provides many of the known features of previous match prototypes as well as a GUI. Instance-based matching is not yet supported. The combination of matcher results uses a non-linear combination of similarity values to favor matchers with higher similarity values (Mork et al. 2008) as briefly discussed in Section 3.3. According to (Smith et al. 2009) Harmony is able to match larger schemas with about 1000 elements each. However, so far no detailed evaluation of Harmony's effectiveness and efficiency has been published.

6. Conclusions

We have provided an overview of selected approaches and current implementations for large-scale schema and ontology matching. Commercial systems increasingly support automatic matching but still have to improve much to better handle large schemas. The current research prototypes share many similarities, in particular a multi-matcher architecture with support for combining linguistic, structural and instance-based matching. We discussed first approaches in several areas that seem promising for large-scale matching, in particular partition-based matching, parallel matching, self-tuning of match workflows and reuse of previously determined match mappings. Such techniques are not yet common in current match systems and more research is needed in all these areas.

Research on holistic (n-way) schema matching mostly focused on very simple schemas such as web forms. More research is therefore needed for n-way matching (clustering) of more complex schemas. An important variation of this problem is searching the most similar schemas for a given schema. Within advanced match

strategies for large schemas such search approaches are also needed for finding relevant sub-schemas.

Fully automatic schema matching is possible and may provide sufficient match quality for simple schemas such as web forms. This is especially the case for the idea of probabilistic mediated schemas considering several alternatives for clustering attributes. For large schemas and ontologies, on the other hand, user interaction remains necessary to configure match workflows, perform incremental matching on selected schema portions, and to provide feedback on the correctness of match candidates. Integrating the various match techniques within a usable and effective data integration infrastructure is challenging and also requires much more work.

References

Alexe B, Gubanov M, Hernandez MA, Ho H, Huang JW, Katsis Y, Popa L, Saha B, Stanoi I (2009) Simplifying Information Integration: Object-Based Flow-of-Mappings Framework for Integration. Proc. BIRTE08 (Business Intelligence for the Real-Time Enterprise) workshop, Springer Lecture Notes in Business Information Processing, Vol. 27

Algergawy A, Schallehn E, Saake G (2009) Improving XML schema matching performance using Prüfer sequences. Data Knowl. Eng. 68(8):728-747

Algergawy A et al (2010) Combining Schema and Level-Based Matching for Web Service Discovery. Proc. 10th Int. Conf. Web Engineering (ICWE)

Aumueller D, Do HH, Massmann S, Rahm E (2005) Schema and ontology matching with COMA++. Proc. ACM SIGMOD Conf., demo paper

Avesani P, Giunchiglia F, Yatskevich M (2005) A Large Scale Taxonomy Mapping Evaluation. Proc. Int. Conf. Semantic Web (ICSW), Springer LNCS 3729

Bellahsene Z, Bonifati A, Duchateau F, Velegrakis Y (2011) On Evaluating Schema Matching and Mapping, In: Bellahsene, Z, Bonifati A, Rahm E (eds): Schema Matching and Mapping, Data-Centric Systems and Applications Series, Springer

Bellahsene Z, Duchateau F (2011) Tuning for Schema Matching. In: Bellahsene, Z, Bonifati A, Rahm E (eds): Schema Matching and Mapping, Springer Data-Centric Systems and Applications Series

Bernstein PA, Melnik S, Petropoulos M, Quix C (2004) Industrial-Strength Schema Matching. ACM SIGMOD Record 33(4): 38-43

Bernstein PA, Melnik S, Churchill JE (2006) Incremental schema matching.

Proc. VLDB, demo paper

Chen K, Madhavan J, Halevy AY (2009) Exploring schema repositories with Schemr. Proc. ACM SIGMOD Conf., demo paper

Cruz IF, Antonelli FP, Stroe C (2009) AgreementMaker: Efficient Matching for Large Real-World Schemas and Ontologies. PVLDB 2(2), demo paper

Das Sarma A, Dong X, Halevy AY (2008) Bootstrapping pay-as-you-go data integration systems. Proc. ACM SIGMOD Conf.

Das Sarma A, Dong X, Halevy AY (2011) Uncertainty in Data Integration and Dataspace Support Platforms. In: Bellahsene Z, Bonifati A, Rahm E (eds): Schema Matching and Mapping, Springer Data-Centric Systems and Applications Series

Do HH (2006) Schema Matching and Mapping-based Data Integration. Dissertation, Dept of Computer Science, Univ. of Leipzig

Do HH, Rahm E (2002) COMA – A System for Flexible Combination of Schema Matching Approaches. Proceedings VLDB Conf., pp 610–621

Do HH, Melnik S, Rahm E (2003) Comparison of Schema Matching Evaluations. In: Web, Web-Services, and Database Systems, Springer LNCS 2593

Do HH, Rahm E (2007) Matching large schemas: Approaches and evaluation. Inf. Syst. 32(6): 857-885

Doan A, Madhavan J, Dhamankar R, Domingos P, Halevy AY (2003) Learning to Match Ontologies on the Semantic Web. VLDB Journal, 12(4):303-319

Dong X, Halevy AY, Madhavan J, Nemes E, Zhang J (2004) Similarity Search for Web Services. Proc. VLDB Conf.

Duchateau F, Coletta R, Bellahsene Z, Miller RJ (2009) (Not) yet another matcher. Proc. CIKM, poster paper

Ehrig M, Staab S (2004) Quick ontology matching. Proc. Int. Conf. Semantic Web (ICSW), Springer LNCS 3298

Ehrig M, Staab S, Sure Y (2005) Bootstrapping Ontology Alignment Methods with APFEL. Proc. Int. Conf. Semantic Web (ICSW), Springer LNCS 3729

Elmagarmid AK, Ipeirotis PG, Verykios VS (2007) Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. 19(1): 1-16

Elmeleegy H, Ouzzani M Elmagarmid AK (2008): Usage-Based Schema Matching. Proc. ICDE Conf.

Euzenat J, Shvaiko P (2007) Ontology Matching. Springer

Euzenat J et al (2009) Results of the Ontology Alignment Evaluation Initiative 2009. Proc. ICSW workshop on Ontology Matching

Fagin R, Haas LM, Hernández MA, Miller RJ, Popa L, Velegrakis Y (2009) Clio: Schema Mapping Creation and Data Exchange. In: Conceptual Modeling: Foundations and Applications, Springer LNCS 5600

Falconer SM, Noy NF (2011) Interactive techniques to support ontology mapping. In: Bellahsene Z, Bonifati A, Rahm E (eds): Schema Matching and Mapping, Springer Data-Centric Systems and Applications Series

Gligorov R, ten Kate W, Aleksovski Z, van Harmelen F (2007) Using Google distance to weight approximate ontology matches. In: Proceedings WWWConf., pp 767–776

Gross A, Hartung M, Kirsten T, Rahm E (2010) On Matching Large Life Science Ontologies in Parallel. Proc. 7th Int. Conf. Data Integration in the Life Sciences (DILS), Springer LNCS 6254

Gubanov M et al (2009) IBM UFO repository: Object-oriented data integration. PVLDB, demo paper

Hamdi F, Safar B, Reynaud C, Zargayouna H (2009) Alignment-based Partitioning of Large-scale Ontologies. In: Advances in Knowledge Discovery And Management. Springer Studies in Computational Intelligence Series

Hanif MS, Aono M (2009) An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. J. Web Sem. 7(4): 344-356

He H, Meng W, Yu CT, Wu Z (2004) Automatic integration of Web search interfaces with WISE-Integrator. VLDB J. 13(3): 256-273

He B, Chang KC (2006) Automatic complex schema matching across Web query interfaces: A correlation mining approach. ACM Trans. Database Syst. 31(1): 346-395

Hu W et al (2008) Matching large ontologies: A divide-and-conquer-approach. Data Knowl. Eng. 67(1): 140-160

Jean-Mary YR, Shironoshita EP, Kabuka MR (2009) Ontology matching with semantic verification. J. Web Sem. 7(3): 235-251

Kappel G et al (2007) Matching metamodels with semantic systems-an experience report. Proc. BTW workshop on Model management

Kirsten T, Thor A, Rahm E (2007) Instance-Based Matching of Large Life Science Ontologies. Proc. Data Integration in the Life Sciences (DILS), Springer LNCS 4544, 172-187

Koepcke H, Rahm E (2010) Frameworks for entity matching: A comparison. Data Knowl. Eng. 69(2):197-210

Koudas N, Marathe A, Srivastava D (2004) Flexible String Matching Against Large Databases in Practice. Proc. VLDB Conf.

Lambrix P, Tan H, Xu W (2008) Literature-based Alignment of Ontologies. Proc.

ICSW Ontology Matching (OM) workshop

Lee Y, Sayyadian M, Doan A, Rosenthal A (2007) eTuner: tuning schema matching software using synthetic scenarios. VLDB Journal 16(1):97-122

Li J, Tang J, Li Y, Luo Q (2009) RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. IEEE Trans. Knowl. Data Eng. 21(8): 1218-1232

Madhavan J, Bernstein P A, Rahm E (2001) Generic SchemaMatching with Cupid. In: Proceedings VLDB Conf., pp 49–58

Madhavan J, Bernstein PA, Doan A, Halevy AY (2005) Corpus-based Schema Matching. Proc. ICDE Conf.

Massmann S, Rahm E (2008) Evaluating Instance-based Matching of Web Directories. Proc. WebDB

Mao M, Peng Y, Spring M (2008) A Harmony Based Adaptive Ontology Mapping Approach. Proc. Int. Conf. on Semantic Web and Web Services (SWWS)

Mork P, Seligman L, Rosenthal A, Korb J, Wolf C (2008) The Harmony Integration Workbench. J. Data Semantics 11: 65-93

Nandi A, Bernstein PA (2009) HAMSTER: Using Search Clicklogs for Schema and Taxonomy Matching. PVLDB 2(1)

Peukert E, Berthold H, Rahm E (2010a) Rewrite Techniques for Performance Optimization of Schema Matching Processes. Proc. 13th Int. Conf. on Extending Database Technology (EDBT)

Peukert E, Massmann S, König K (2010b) Comparing Similarity Combination Methods for Schema Matching. Proc. 40th Annual Conf. of the German Computer Society (GI-Jahrestagung)

Pirrò G, Talia D (2010) UFOme: An ontology mapping system with strategy prediction capabilities. Data Knowl. Eng. 69(5): 444-471

Rahm E, Bernstein PA (2001) A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4):334–350

Rahm E, Do, HH, Massmann S (2004) Matching Large XML Schemas. SIGMOD Record 33(4): 26-31

Saha B, Stanoi I, Clarkson KL (2010) Schema covering: a step towards enabling reuse in information integration. Proc. ICDE Conf.

Saleem K, Bellahsene Z, Hunt E (2008) PORSCHE: Performance Oriented SCHEma mediation. Inf. Syst. 33(7-8): 637-657

SAP (2010) Warp10 community-based integration.
https://cw.sdn.sap.com/cw/docs/DOC-120470 (white paper),
https://cw.sdn.sap.com/cw/community/esc/cdg135, April 2010

Seligman L, Mork P, Halevy AY et al (2010) OpenII: An Open Source Infor-

mation Integration Toolkit, Proc. ACM SIGMOD Conf.

Shi F, Li J et al (2009) Actively learning ontology matching via user interaction. Proc. Int. Conf. Semantic Web (ICSW)

Smith K, Morse M, Mork P et al (2009) The Role of Schema Matching in Large Enterprises. Proc. CIDR

Spiliopoulos V, Vouros GA, Karkaletsis V (2010) On the discovery of subsumption relations for the alignment of ontologies. J. Web Sem. 8(1): 69-88

Su W, Wang J, Lochovsky FH (2006) Holistic Schema Matching for Web Query Interfaces. Proc. Int. Conf. on Extending Database Technology (EDBT)

Tan H, Lambrix P (2007) A Method for Recommending Ontology Alignment Strategies. Proc. Int. Conf. Semantic Web (ICSW), Springer LNCS 4825

Thor A, Kirsten T, Rahm E (2007) Instance-based matching of hierarchical ontologies. Proc. 12th BTW Conf. (Database systems for Business, Technology and Web)

Zhang S, Mork P, Bodenreider O, Bernstein PA (2007) Comparing two approaches for aligning representations of anatomy. Artificial Intelligence in Medicine 39(3):227-236

Zhong Q, Li H et al (2009) A gauss function based approach for unbalanced ontology matching. Proc. ACM SIGMOD Conf.