ABSTRACT
In a paper published in the 2001 VLDB Conference, we proposed treating generic schema matching as an independent problem. We developed a taxonomy of existing techniques, a new schema matching algorithm, and an approach to comparative evaluation. Since then, the field has grown into a major research topic. We briefly summarize the new techniques that have been developed and applications of the techniques in the commercial world. We conclude by discussing future trends and recommendations for further work.

1. INTRODUCTION
Schema matching is the problem of generating correspondences between elements of two schemas. A schema is a formal structure that represents an engineered artifact, such as a SQL schema, XML schema, entity-relationship diagram, ontology description, interface definition, or form definition. A correspondence is a relationship between one or more elements of one schema and one or more elements of the other. For example, the correspondences in Figure 1 identify columns that represent the same concepts in the two relational schemas. Often, the relationship is one-to-one, but sometimes it is not, such as Author corresponding to LastName and FirstName in Figure 1. We say that a correspondence has semantics if it constrains the instances of the related schema elements. The common default semantics for one-to-one correspondences is that the instances of two related elements are equal.

There are many applications that require schema matching. In the database field, it is usually the first step in generating a program or view definition that maps instances of one schema into instances of another. For example, it arises in object-to-relational mappings, data warehouse loading, data exchange, and mediated schemas for data integration. In knowledge-based applications, such as life sciences applications and the semantic web, it arises in the alignment of ontologies. For example, it may be used to align gene ontologies or anatomical structures. In health care, it may arise in the alignment of patient records and other medical reports. In web applications, it may be used to align product catalogs. In e-commerce, it may be used to align message formats representing business documents, such as orders and invoices.

This paper recaps the contributions of our VLDB 2001 paper about schema matching [45], summarizes developments since then, and suggests problems that would benefit from further work.

Figure 1: Schema matching is the problem of generating correspondences that identify related elements in two schemas.

2. CONTRIBUTIONS IN VLDB 2001 [45]
Twelve years ago, when we embarked on work in this area, we noticed that schema matching techniques were developed as part of a variety of applications. The techniques were often similar, even when the applications were not. We concluded that the field might move faster and the results might be more reusable if schema matching were studied as a separate topic, independently of the applications that use it. This recommendation was the first contribution of [45].

We then surveyed the literature to identify these common techniques. This resulted in a taxonomy of schema matching techniques, which was the second contribution of [45]. We extended this taxonomy into a survey paper, published later that year in [63]. The taxonomy has often been used as a standard for categorizing subsequent schema matching techniques.

Our third contribution was a new schema matching algorithm, called Cupid, which combined a number of techniques: linguistic matching, structure-based matching, constraint-based matching, and context-based matching. Most of the later approaches to schema matching have used this hybrid matcher approach, which leverages different criteria to arrive at suggested correspondences.

We concluded with an experimental comparison of Cupid with two other algorithms that were reported in the literature, namely MOMIS [6] and DIKE [58]. This was the first such comparison we know of. Such experimental comparisons have become a feature of most of the later work on schema matching.

In summary, our 2001 paper posed schema matching as a problem that could be studied in isolation. It gave a baseline of known techniques. And given the inherently heuristic nature of solutions, it suggested an approach to evaluate those solutions based on experiments. As the references in [45] attest, we were by no means the first to work on schema matching. However, we...
3. SCHEMA MATCHING TECHNIQUES

There are now two books on schema matching [5][26] and two surveys [63][68], so there is little point in our repeating such a survey. However, to give the reader a feel for the scope of the schema matching field, we list many of the known techniques here. We start with techniques that were known in 2001 and that we discussed in [45]:

- Linguistic matching – based on an element’s name or description, using stemming, tokenization, string and substring matching, and information retrieval techniques.
- Using auxiliary information – based on thesauri, acronyms, dictionaries, and mismatch lists.
- Instance-based matching – schema elements are regarded as similar if their instances are similar, based on statistics, metadata, or trained classifiers.
- Structure-based matching – schema elements are similar if they appear in similarly-structured groups, have similar relationships, or have (paths of) relationships to similar elements.
- Constraint-based matching – based on data types, value ranges, uniqueness, nullability, and foreign keys.
- Rule-based matching – based on matching rules that are expressed in first-order logic.
- Hybrid-matching – as explained in the previous section.

Since 2001, many other techniques have been developed. These include algorithms that use new types of information. For example:

- Graph matching – based on comparing the relationships between elements in the schema graphs by, for example, either fixed-point computations on a similarity propagation graph [53], or probabilistic constraint satisfaction algorithms [22].
- Usage-based matching – based on analyzing database query logs for hints about how users relate schemas, e.g., by equating elements in join clauses [25]. Taxonomy paths can be matched by finding web pages that represent the paths and then analyzing keyword-query logs to determine if the pages are accessed via similar query distributions [55].
- Document content similarity – where instances of a schema element are grouped into a document that is then matched with other such documents based on the information retrieval measure TF-IDF (term frequency times inverse document frequency) [44][49].
- Document link similarity – where concepts in two ontologies are regarded as similar if the entities referring to those concepts are similar [42].

Strategies have been proposed to flexibly combine multiple matching algorithms and to scale gracefully to compare large schemas. For example:

- Workflows-like strategies to independently or sequentially execute matchers and to combine their results [12][19][67].
- Self-tuning match workflows – where for a given match task or domain of match tasks, a tuner selects the match components to be combined and/or assigns values to the various parameters that affect how component match results are combined [24][43][44].
- Early search space pruning – where a fast matcher is used to eliminate unlikely matches from consideration so that a manageable-small number of elements can be matched using more expensive and accurate techniques [23][57].
- Partition-based matching – where to reduce the space of possible matches, the input schemas are partitioned followed by partition-wise matching [20][39][73].
- Parallel matching – where different steps of the matching algorithm are run in parallel or different partitions of the schemas are matched in parallel [34].
- Optimizations for large schemas such as using string matching optimizations [40], pre-collecting predecessors and children of each element to avoid repeated traversal [2], and using space-efficient similarity matrices [12].

Approaches have been proposed where multiple schemas in a domain are collectively matched. For example:

- Reuse-based matching – where matches between schema fragments are harvested from validated mappings and used as auxiliary information to help future match tasks in the same domain [20][46][65].
- Holistic matching – where a single mediated schema is constructed for a domain by aligning elements of a large corpus of schemas, such as web forms covering a particular domain. Similar element names appearing in the same schema are regarded as mismatches [37][38][66][69].

Strategies have been proposed to incorporate user interaction and feedback in the matching process. For example:

- GUI support to interactively inspect and correct computed correspondences [3][11][16][31].
- Incremental matching – where given a user-selected element of one schema, the matcher calculates the best match or matches (top-k) in the other schema [11].
- Top-k matching – where instead of computing a complete mapping between two schemas, the matcher computes the top-k matches of each element of one schema to elements of the other schema [11][32].
- Collaborative, wiki-like user involvement to provide, improve, and reuse mappings [50][72].

Finally, algorithms have been proposed that extend the semantics of matches beyond that of simple correspondences. For example:

- Semantic tagging – where correspondences are tagged with semantic relationships, such as equality, containment, disjointness, and unknown. [33][35][48].
- Conditional tagging – where correspondences are refined to be valid only for certain values of another element. For example, if productType = “book” then Invoice.Code = ISBN [14][33].

4. SCHEMA MATCHING TOOLS

Most of the listed techniques have been implemented in a large number of tools for schema and ontology matching [26][62]. Figure 2 shows a comparative overview of selected tools: Cupid [45], COMA++ [3][19][20], ASMOV [40], Falcon-AO [39], RiMON [44], AgreementMaker [16], OII Harmony [67]. Most
recent prototypes support match workflows and the combined use of different linguistic, structural and instance-based matchers. External dictionaries such as synonym lists or thesauri are commonly used to improve linguistic matching. GUI support is often provided, albeit still with limitations [31]. A few systems are able to match both schemas and ontologies [3][16][67]. As indicated in Figure 2, advanced techniques such as schema partitioning, parallel matching, mapping reuse and self-tuning capabilities (e.g., a dynamic selection of matchers for a given match task) are still only supported to a limited extent in current match prototypes.

Match tools have been intensively evaluated but typically under different conditions and for smaller match problems [4][18]. For ontology matching, the Ontology Alignment Evaluation Initiative (OAEI) organizes yearly contests that include some larger problems, e.g., to match web directories or medical ontologies (http://oaei.ontologymatching.org). The systems participating in the OAEI contest have significantly improved over the years but still struggle with larger problems [27]. For schema matching and mapping, a comparable benchmark effort is still missing.

Semi-automatic schema matching is also increasingly supported in commercial middleware tools, in particular for XML schemas or relational database schemas. Systems such as Altova MapForce, IBM Infosphere, Microsoft BizTalk Server and SAP Netweaver provide a GUI-based editor for manual mapping specification with some support for automatic determination of match candidates, e.g., based on approximate name matching. However, most of the more recently proposed match techniques have not yet been incorporated in commercial mapping solutions.

5. USING MATCH RESULTS AS-IS

Even the best schema matching algorithms make many mistakes, especially fully-automatic algorithms where there is no human designer in the loop. Despite these errors, some applications can use schema matching results as-is. This is especially the case when a best-effort matching is satisfactory or when the matches contribute only implicitly to the results of some end-user task. For example, consider the following two scenarios for automatically filling out HTML forms.

First, most of today's browsers offer automatic form-filling, e.g., personal data such as name and address prior to a purchase. This can be modeled as a task where the schema of the underlying web-form is being matched to a model of user data that is stored in the browser. The user expects the browser to make a best-effort attempt at filling in personal details, which the user confirms before submitting the form for processing.

Second, schema mappings have been proposed as a means of accessing the content that lies behind HTML forms [47][61]. A deep-web crawler can work as follows: When the crawler encounters an HTML form, it can identify the domain that the form belongs to, and then match the inputs of the form to elements in the previously-computed mediated schema for that domain (see Figure 3). It can then generate form submissions by constructing URLs using sample values for the inputs (based on known values for the elements in the mediated schema). The resulting pages are added to the index of the search engine. The matching results in this case are intermediate results of a multi-step process. End-users are unlikely to know or care about the quality of the match result, except insofar as it affects how the crawler exploits the underlying website.

6. APPLYING MATCH TO MODEL MANAGEMENT

For most of the applications summarized in Section 1, schema matching is just one step in a multi-step process. That multi-step process involves other operators that manipulate schemas and mappings, such as schema merging and mapping composition. This recognition was actually the starting point for our research into schema matching. In [8] and [9], we proposed a set of such operators under the name “model management”. We then embarked on a systematic study of these operators. Since nothing
much can be done until the first mapping is created, it was logical that we started our investigation of operators with schema matching. In fact, our first algorithmic result about one of the operators was our paper “Generic Schema Matching” [45]. Since then, there has been a lot of progress on other operators in addition to match, which is summarized in [10].

Most data integration and data transformation applications, such as those in Section 1, need to construct executable mappings—ones that represent transformations of instances. Since match algorithms produce correspondences, not semantic relationships, the natural next step is to enrich those correspondences with semantics [54]. Often this is a two-step process (Section 3.1 of [10]). The first step is to generate semantics in the form of constraints that relate parts of the instances of one schema to parts of the instances of another schema. Such constraints may not be functions, in which case they are not executable. In this case, a second step is needed to translate the semantic relationships into functions [51] via the operator TransGen.

Depending on the application, the resulting mapping may need to undergo further manipulation. Suppose we match schemas S and T and then generate a semantic mapping between them. We might want to merge S and T into a single schema that covers both of them, for example, to represent a mediated schema. This can be done by the merge operator, which takes as input two schemas and a mapping between them and returns a merged schema with mappings between the merged schema and the two input schemas [15][59][60][64].

Suppose we are using the mapping between S and T as a data transformation that translates data from S’s format into T’s format. If one of the schemas T in a mapping is modified, generating T’, then we need to update the mapping between S and T to one between S and T’. We can do this by composing the mappings S-T and T-T’ [30][36], yielding a mapping T-T’ between S and T’ [7][28][71].

Other model management operators are Diff (which finds the difference between mappings) and Extract (the complement of Diff) [52], and Invert, which reverses the direction of a uni-directional mapping [28][29].

For most practical applications, all of the model management operators manipulate mappings that have semantics—except for the match operator which has a special role. First the match operator computes correspondences and then, building on these correspondences, the other operators develop and manipulate mappings that have semantics.

7. FUTURE TRENDS

Since 2001, there has been a growing realization that matching is not a one-of task. For example, in data integration, as new data sources become available, they are mapped to a single mediated schema. In e-commerce, message formats of new business sources become available, they are mapped to a single mediated schema. Therefore, it is important to have excellent graphical support for viewing mappings [31]. For example, since large schemas cannot be viewed on a single screen, it is beneficial to partition them into fragments that can be matched independently, to the extent possible. Matching tools also need to offer better support for the mapping process. For example, users need help in remembering which schema elements they have examined during the match process and what was learned by that examination, such as promising and specious candidates.

We see an increasing convergence of schema matching and entity resolution approaches, i.e., matching at the metadata level and matching at the instance level. Most recent schema and ontology matching prototypes include instance-based matchers [61] that derive the similarity of schema elements from the similarity or
The availability of ontologies.

resolution approaches are needed and applicable due to the broad web of linked data [13][56] is an area where such semantic entity catalogs. Link discovery to interconnect sources in the so-called based on a pre-determined ontology mapping between the product catalogs. Link discovery to interconnect sources to corresponding or closely related product categories, the organization of products or product offers within product ontologies and the provision of ontology mappings. For example, can benefit from the semantic categorization of entities within

10. REFERENCES


