6

Overview of matching systems

This chapter is an overview of matching systems which have emerged during the last decade. There have already been some comparisons of matching systems, in particular in [Parent and Spaccapietra, 2000, Rahm and Bernstein, 2001, Do *et al.*, 2002, Kalfoglou and Schorlemmer, 2003b, Noy, 2004a, Doan and Halevy, 2005, Shvaiko and Euzenat, 2005]. Our purpose here is not to compare them in full detail, though we give some comparisons, but rather to show their variety, in order to demonstrate in how many different ways the methods presented in previous chapters have been practically exploited.

We have followed two principles in deciding whether a matching solution should be included in this chapter: it must have an implementation and an archival publication describing it at the time of writing. We have also excluded from consideration the systems which assume that alignments have already been established, and use this assumption as a prerequisite of running the actual system. These approaches include such information integration systems as: Tsimmis [Chawathe *et al.*, 1994], Observer [Mena *et al.*, 1996], SIMS [Arens *et al.*, 1996], InfoSleuth [Fowler *et al.*, 1999, Nodine *et al.*, 2000], Kraft [Preece *et al.*, 2000], Picsel [Goasdoué *et al.*, 2000], DWQ [Calvanese *et al.*, 2002a], AutoMed [Boyd *et al.*, 2004], and InfoMix [Leone *et al.*, 2005]. Even if we have considered around 50 systems and approaches in this chapter, this overview is not exhaustive. An interested reader can find an updated and complete information on the topic at OntologyMatching.org[1], in particular, links to the web sites of the presented systems can be found there. We only mention address of general purpose resources.

We present the matching systems in light of the classifications discussed in Chap. 3. We also point to concrete basic matchers and matching strategies used in the considered systems by referencing to the corresponding subsections of Chap. 4 and Chap. 5. In order to facilitate the presentation we follow two rules. First, the year of the system appearance is considered. Then, if there are some evolutions of the system or very similar systems, these are discussed close to each other. We illus-

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

trate systems where the matching process is of a particular interest with the help of figures. We tried to adopt a unified presentation for these systems. However, some of these are specific enough so that we did not enforce the terminology of Sect. 2.4 but kept that one as used by system designers.

The structure of this chapter is as follows. We first describe systems which mostly focus on schema-level information (§6.1). Secondly, we discuss systems which concentrate on instance-level information (§6.2). Then, we present systems which exploit both schema-level and instance-level information (§6.3). Finally, we overview meta-matching systems (§6.4).

6.1 Schema-based systems

Schema-based systems, according to the classification of Chap. 3, are those which rely mostly on schema-level input information for performing ontology matching.

6.1.1 DELTA (The MITRE Corporation)

DELTA (Data Element Tool-based Analysis) is a system that semi-automatically discovers attribute correspondences among database schemas [Clifton *et al.*, 1997]. It handles relational and extended entity–relationship (EER) schemas. The idea of the approach is to use textual similarities between data element definitions in order to find matching candidates. The system converts available information about an attribute, e.g., attribute name, datatype, narrative description, into a simple text string, called *document*. The documents describing each database attribute constitute a *document base*. Then, DELTA feeds the document base from the first schema into a full-text information retrieval tool, such as Personal Librarian. Matching is viewed as a Personal Librarian query based on the information from the second schema. The query can be a string of disconnected phrases, a full boolean query, a few relevant words, or an entire document. The tool estimates the similarity (by using natural language heuristics, such as considering that rare or repeated words are more important) between a search pattern and contents of a document base (§4.2.1). It is thus exclusively based on string-based techniques. It returns a ranked list of documents that it considers to be similar. The selection of the final alignment is to be performed by users.

6.1.2 Hovy (University of Southern California)

[Hovy, 1998] describes heuristics used to match large-scale ontologies, such as Sensus and Mikrokosmos, in order to combine them in a single reference ontology. In particular, three types of matchers were used based on (i) concept names, (ii) concept definitions, and (iii) taxonomy structure. For example, the name matcher splits composite-word names into separate words (§4.2.2) and then compares substrings in order to produce a similarity score. Specifically, the name matcher score is computed

as the sum of the square of the number of letters matched, plus 20 points if words are exactly equal or 10 points if end of match coincides. For instance, using this strategy, the comparison between Free World and World results in 35 points, while the comparison between cuisine and vine results in 19 points. The definition matcher compares the English definitions of two concepts (§4.2.2). Here, both definitions are first split into individual words. Then, the number and the ratio of shared words in two definitions is computed in order determine the similarity between them. Finally, results of all the matchers are combined based on experimentally obtained formulas. The combined scores between concepts from two ontologies are sorted in descending order and are presented to users for establishing a cutoff value as well as for approving or discarding operations, results of which are saved for later reuse.

6.1.3 TransScm (Tel Aviv University)

TransScm [Milo and Zohar, 1998] provides data translation and conversion mechanisms between input schemas based on schema matching. First, by using rules, the alignment is produced in a semi-automatic way. Then, this alignment is used to translate data instances of the source schema to instances of the target schema. Input schemas are internally encoded as labelled graphs, where some of the nodes may be ordered. Nodes of the graph represent schema elements, while edges stand for the relations between schema elements or their components. Matching is performed between nodes of the graphs in a top-down and one-to-one fashion. Matchers are viewed as rules. For example, according to the *identical* rule, two nodes match if their labels are found to be synonyms based on the built-in thesaurus (§4.2.2); see [Zohar, 1997] for a list of the available rules. The system combines rules sequentially based on their priorities. It tries to find, for the source node, a unique *best* matching target node, or determines a mismatch. In case there are several matching candidates among which the system cannot choose the best one, or if the system cannot match or mismatch a source node to a target node with the given set of rules, user involvement is required. In particular, users with the help of a graphic user interface can add, disable or modify rules to obtain the desired matching result. Then, instances of the source schema are translated to instances of the target schema according to the match rules. For the example of the *identical* rule, translation includes copying the source node components.

6.1.4 DIKE (Università di Reggio Calabria and Università di Calabria)

DIKE (Database Intensional Knowledge Extractor) is a system supporting the semi-automatic construction of cooperative information systems (CISs) from heterogeneous databases [Palopoli *et al.*, 2003b, Palopoli *et al.*, 2003a, Palopoli *et al.*, 1998, Palopoli *et al.*, 2000]. It takes as input a set of databases belonging to the CIS. It builds a kind of mediated schema (called data repository or global structured dictionary) in order to provide a user-friendly integrated access to the available data sources. DIKE focuses on entity-relationship schemas. The matching step is called the extraction of inter-schema knowledge. It is performed in a semi-automatic way.

Some examples of inter-schema properties that DIKE can find are terminological properties, such as synonyms, homonyms among objects, namely entities and relationships, or type conflicts, e.g., similarities between different types of objects, such as entities, attributes, relationships; structural properties, such as object inclusion; subschema similarities, such as similarities between schema fragments. With each kind of property is associated a plausibility coefficient in the $[0\ 1]$ range. The properties with a lower plausibility coefficient than a dynamically derived threshold are discarded, whereas others are accepted. DIKE works by computing sequentially the above mentioned properties. For example, synonyms and homonyms are determined based on information from external resources, such as WordNet (§4.2.2), and by analysing the distances of objects in the neighbourhood of the objects under consideration (§4.3.2). Some weights are also used to produce a final coefficient. Then, type conflicts are analysed and resolved by taking as input the results of synonyms and hyponyms analysis.

6.1.5 SKAT and ONION (Stanford University)

SKAT (Semantic Knowledge Articulation Tool) is a rule-based system that semi-automatically discovers mappings between two ontologies [Mitra *et al.*, 1999]. Internally, input ontologies are encoded as graphs. Rules are provided by domain experts and are encoded in first order logic. In particular, experts specify initially desired matches and mismatches. For example, a rule President $\equiv$ Chancellor, indicates that we want President to be an appropriate match for Chancellor. Apart from declarative rules, experts can specify matching procedures that can be used to generate the new matches. Experts have to approve or reject the automatically suggested matches, thereby producing the resulting alignment. Matching procedures are applied sequentially. Some examples of these procedures are: string-based matching, e.g., two terms match if they are spelled similarly (§4.2.1), and structure matching, e.g., structural graph slices matching, such as considering nodes near the root of the first ontology against nodes near the root of the second ontology (§4.3.2).

ONION (ONtology compositION) is a successor system to SKAT that semi-automatically discovers mappings between multiple ontologies, in order to enable a unified query answering over those ontologies [Mitra *et al.*, 2000]. Input ontologies, RDF files, are internally represented as labelled graphs. The alignment is viewed as a set of *articulation rules*. The semi-automated algorithm for resolving the terminological heterogeneity of [Mitra and Wiederhold, 2002] forms the basis of the *articulation generator*, ArtGen, for the ONION system. ArtGen, in turn, can be viewed as an evolution of the SKAT system with some added matchers. Thus, it executes a set of matchers and suggests articulation rules to users. Users can either accept, modify or delete the suggestions. They can also indicate the new matches that the articulation generator may have missed. ArtGen works sequentially, first by performing linguistic matching (§4.2.2) and then structure-based matching (§4.3). During the linguistic matching phase, concept names are represented as sets of words. The linguistic matcher compares all possible pairs of words from any two concepts of both ontologies and assigns a similarity score in $[0\ 1]$ to each pair. The matcher uses a

word similarity table generated by a thesaurus-based or corpus-based matcher called the *word relator* to determine the similarity between pairs of words (§4.2.2). The similarity score between two concepts is the average of the similarity scores (ignoring scores of zero) of all possible pairs of words in their names. If this score is higher than a given threshold, ArtGen generates a match candidate. Structure-based matching is performed based on the results of the linguistic matching. It looks for structural isomorphism between subgraphs of the ontologies, taking into account some linguistic clues (see Sect. 6.1.11 for a similar technique). The structural matcher tries to match only the unmatched pairs from the linguistic matching, thereby complementing its results.

6.1.6 Artemis (Università di Milano and Università di Modena e Reggio Emilia)

Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) [Castano *et al.*, 2000] was designed as a module of the MOMIS mediator system [Bergamaschi *et al.*, 1999, Bergamaschi *et al.*, 1998] for creating global views. It performs affinity-based analysis and hierarchical clustering of database schema elements. Affinity-based analysis represents the matching step: in a sequential manner it calculates the name, structural and global affinity coefficients exploiting a common thesaurus. The common thesaurus is built with the help of ODB-Tools [Beneventano *et al.*, 1998], WordNet (§4.2.2) or manual input. It represents a set of intensional and a set of extensional relationships which depict intra- and inter-schema knowledge about classes and attributes of the input schemas. Based on global affinity coefficients, a hierarchical clustering technique categorises classes into groups at different levels of affinity. For each cluster it creates a set of global attributes and the global class. Logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table.

6.1.7 H-Match (Università degli Studi di Milano)

H-Match [Castano *et al.*, 2006] is an automated ontology matching system. It has been designed to enable knowledge discovery and sharing in the settings of open networked systems, in particular within the Helios peer-to-peer framework [Castano *et al.*, 2005]. The system handles ontologies specified in OWL. Internally, these are encoded as graphs using the H-model representation [Castano *et al.*, 2005]. H-Match inputs two ontologies and outputs (one-to-one or one-to-many) correspondences between concepts of these ontologies with the same or closest intended meaning. The approach is based on a similarity analysis through affinity metrics, e.g., term to term affinity, datatype compatibility (§4.3.1), and thresholds. H-Match computes two types of affinities (in the $[0\ 1]$ range), namely *linguistic* and *contextual* affinity. These are then combined by using weighting schemas, thus yielding a final measure, called *semantic* affinity. Linguistic affinity builds on top of a thesaurus-based approach of the Artemis system (§6.1.6). In particular, it extends the Artemis approach (i) by building a common thesaurus involving relations among WordNet synsets such

as meronymy and coordinate terms, and (ii) by providing an automatic handler of compound terms, i.e., those composed by more than one token, that are not available from WordNet. Contextual affinity requires consideration of the neighbour concepts, e.g., linked via taxonomical or mereological relations, of the actual concept (§4.3.2).

One of the major characteristics of H-Match is that it can be dynamically configured for adaptation to a particular matching task, because in dynamic settings, the complexity of a matching task is not known in advance. This is achieved by means of four matching models. These are: *surface*, *shallow*, *deep*, and *intensive*, each of which involves different types of constructs of the ontology, see Fig. 6.1. Computation of a linguistic affinity is a common part of all the matching models. In case of the surface model, linguistic affinity is also the final affinity, since this model considers only names of ontology concepts. All the other three models take into account various contextual features and therefore contribute to the contextual affinity. For example, the shallow model takes into account concept properties, whereas the deep and the intensive models extend previous models by including relations and property values, respectively. Each concept involved in a matching task can be processed according to its own model, independently from the models applied to the other concepts within the same task. Finally, by applying thresholds, correspondences with semantic (final) affinity higher than the cut-off threshold value are returned in the final alignment.

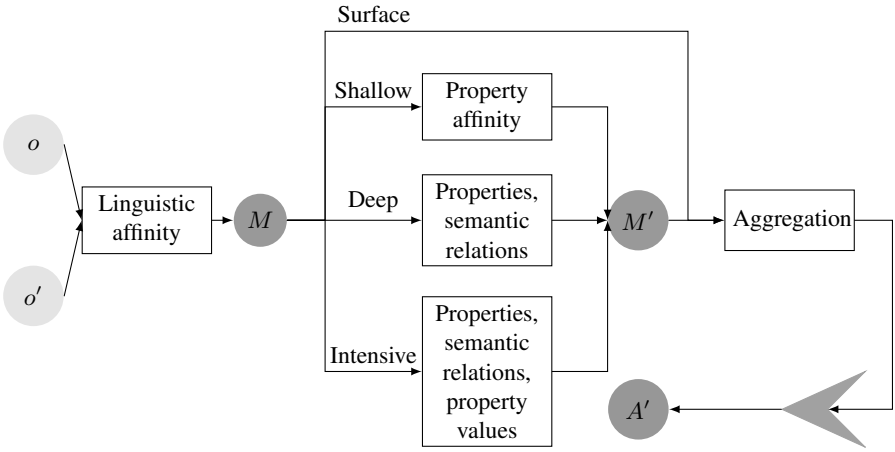

Fig. 6.1. H-Match matching process: H-Match is a conditional system that can use alternatively or in parallel four matching models depending on the resources available.

6.1.8 Tess (University of Massachusetts)

Tess (Type Evolution Software System) is a system to support schema evolution by matching the old and the new versions [Lerner, 2000]. Schemas are viewed as collec-

tions of types. It is assumed (since in the given application scenario changes are typically evolutionary, rather than revolutionary) that input schemas are highly similar. Matching is viewed as generation of *derivation rules* to be applied to data. Tess can operate in modes ranging from fully automated to completely manual. Each derivation rule is associated with a similarity metric, which is meant to measure the impact that applying the derivation rule would have on existing data. By defining a threshold for the similarity metric, the user involvement is determined. Matching is performed in three stages. First, the names of the types of old and new versions are compared (§4.2.1). Second, the structural information is taken into account. In particular, type constructors used by the old and new types and the types of components are analysed (§4.3.1). This provides the ability to handle cases in which, for example, component names have been changed, but their types are unchanged. Third, if everything else fails, matching relies upon some ordering information heuristics. Thus, in this case, Tess will try matching components with different names and different types. Finally, based on the derivation rules a transformer is produced which can update data in a database according to a newer version of the schema. In the simplest case, such as the identity derivation rule case, when type names are identical, as in Sect.6.1.3, the derivation function simply copies existing objects. A more complex transformation may include a join operation to combine two related objects into one.

6.1.9 Anchor-Prompt (Stanford Medical Informatics)

Anchor-Prompt [Noy and Musen, 2001] is an extension of Prompt, also formerly known as SMART. It is an ontology merging and alignment tool with a sophisticated prompt mechanism for possible matching terms [Noy and Musen, 1999]. Prompt handles ontologies expressed in such knowledge representation formalisms as OWL and RDF Schema. Anchor-Prompt is a sequential matching algorithm that takes as input two ontologies, internally represented as graphs and a set of anchors-pairs of related terms, which are identified with the help of string-based techniques, such as edit-distance (§4.2.1), , user-defined distance or another matcher computing linguistic similarity. Then the algorithm refines them by analysing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar positions on similar paths (§4.3.2). Finally, based on the frequencies and user feedback, the algorithm determines matching candidates.

The Prompt and Anchor-Prompt systems have also contributed to the design of other algorithms, such as PromptDiff, which finds differences between two ontologies and provides the editing facility for transforming one ontology into another (see Sect.8.3.2 and [Noy and Musen, 2002b, Noy and Musen, 2003]).

6.1.10 OntoBuilder (Technion Israel Institute of Technology)

OntoBuilder is a system for information seeking on the web [Modica *et al.*, 2001]. A typical situation the system deals with is when users are seeking for a car to be rented. Obviously, they would like to compare prices from multiple providers in order to make an informed decision. OntoBuilder operates in two phases: (i) ontology

creation (the *training* phase) and (ii) ontology adaptation (the *adaptation* phase). During the training phase an initial ontology (in which user data needs are encoded) is created by extracting it from a visited web site of, e.g., some car rental company. The adaptation phase includes on-the-fly match and interactive merge operations of the related ontologies with the actual (initial) ontology. We concentrate below only on the ontology adaptation phase. During the adaptation phase users suggest the web sites they would like to further explore, e.g., the ones of various car rental companies. Each such site goes through the ontology extraction process. This results in a candidate ontology, which is then merged into the actual ontology. To support this, the best match for each existing term in the actual ontology to terms from the candidate ontology is selected. The selection strategy employs thresholds (§5.7.1). The matching algorithm works in a term to term fashion, sequentially executing various matchers. Some examples of the matchers used are: removing noisy characters and stop terms (§4.2.2), substring matching (§4.2.1). If all else fail, thesaurus look-up is performed (§4.2.2). Finally, mismatched terms are presented to users for manual matching. Some further matchers, such as those for precedence matching, were introduced in later work in [Gal *et al.*, 2005b]. Top-k mappings have been proposed in [Gal, 2006] as an alternative for a single best matching, i.e., top-1 category.

6.1.11 Cupid (University of Washington, Microsoft Corporation and University of Leipzig)

Cupid [Madhavan *et al.*, 2001] implements an algorithm comprising linguistic and structural schema matching techniques, and computing similarity coefficients with the assistance of domain specific thesauri. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases (see Fig. 6.2) and operates only with tree-structures, to which non tree cases are reduced.

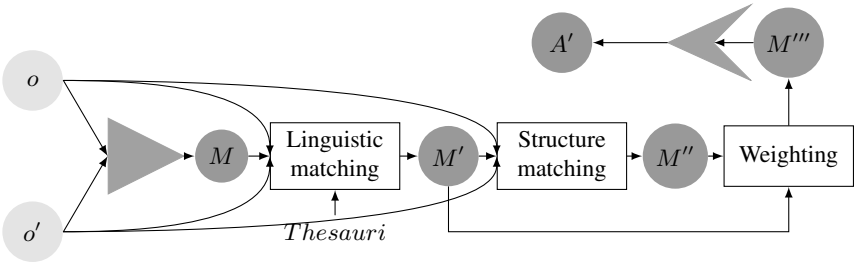

Fig. 6.2. Cupid architecture: it is a very common architecture which mixes parallel and sequential composition. Structure matching takes advantage of the results of linguistic matching, but the results of both of them are taken into consideration for weighting.

The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalisation

(§4.2.2), categorisation, string-based techniques, such as common prefix, suffix tests (§4.2.1), and thesauri look-up (§4.2.2). The second phase (structural matching) computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur (§4.3.2). The third phase (mapping elements generation) aggregates the results of the linguistic and structural matching through a weighted sum (§5.2.2) and generates a final alignment by choosing pairs of schema elements with weighted similarity coefficients, which are higher than a threshold (§5.7.1).

6.1.12 COMA and COMA++ (University of Leipzig)

COMA (COmbination of MAtching algorithms) [Do and Rahm, 2002] is a schema matching tool based on parallel composition of matchers. It provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. As from [Do and Rahm, 2002], COMA contains 6 elementary matchers, 5 hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques, such as affix, n-gram, edit distance (§4.2.1); others share techniques with Cupid, e.g., thesauri look-up. An original component, called *reuse-oriented matcher*, tries to reuse previously obtained results for entire new schemas or for their fragments. Schemas are internally encoded as directed acyclic graphs, where elements are the paths. This aims at capturing contexts in which the elements occur. Distinct features of the COMA tool with respect to Cupid are a more flexible architecture and the possibility of performing iterations in the matching process. It presumes interaction with users who approve obtained matches and mismatches to gradually refine and improve the accuracy of match. COMA++ is built on top of COMA by elaborating in more detail the alignment reuse operation. Also it provides a more efficient implementation of the COMA algorithms and a graphical user interface (§8.2.2).

6.1.13 Similarity flooding (Stanford University and University of Leipzig)

The Similarity flooding approach [Melnik *et al.*, 2002] is based on the ideas of similarity propagation. Schemas are presented as directed labelled graphs grounded on the OIM specification [MDC, 1999]. The algorithm manipulates them in an iterative computation to produce an alignment between the nodes of the input graphs. The technique starts from string-based comparison, such as common prefix, suffix tests (§4.2.1), of the vertices labels to obtain an initial alignment which is refined through iterative computation. The basic concept behind the Similarity flooding algorithm is the similarity spreading from similar nodes to the adjacent neighbours through propagation coefficients. From iteration to iteration the similarity measure is spread to the graph until a fixed point is reached or the computation is stopped. The full process is described in Sect. 5.3.1. The result of this step is a refined alignment which is further filtered to produce the final alignment.

6.1.14 XClust (National University of Singapore)

XClust is a tool for integrating multiple DTDs [Lee *et al.*, 2002]. Its integration strategy is based on clustering. Given multiple DTDs, it clusters them according to their similarity. This aims at facilitating the work of system integrators by allowing them to focus on already similar DTDs of single clusters. Clustering is applied recursively until a manageable number of DTDs is obtained. XClust works in two phases: (i) DTD similarity computation, and (ii) DTD clustering. During the first phase, given a set of DTDs, pairwise similarities between their underlying labelled trees are computed. This is done by using several matchers which exploit schema names as well as some structural information. For example, the *basic similarity* is computed as a weighted sum of a WordNet-based matcher that looks for synonyms among names of schema elements (§4.2.2) and a cardinality constraint matcher that performs a look up in cardinality compatibility table in order to compare cardinalities of schema elements (§4.3.1). Structural similarities exploit previously computed basic similarities and are based on (i) similarity of paths, (ii) similarity of immediate descendants and (iii) similarity of leaves (§4.3.2). For example, similarity of paths is computed as a normalised sum of basic similarities between the sets of elements these paths are composed of, namely elements from the root to the node under consideration (§4.2.1). Structural similarities are aggregated as a weighted sum and then these aggregated similarities are used to choose the best match pairs by applying a threshold. These constitute the alignment for a pair of DTDs. Finally, for two DTDs, best match pairs are summed up and normalised, thereby resulting in a final similarity between these DTDs. The result of the first phase is the similarity matrix of a set of DTDs. During the second phase, based on the DTD similarity matrix, a hierarchical clustering [Everitt, 1993] is applied to group DTDs into clusters.

6.1.15 ToMAS (University of Toronto and IBM Almaden)

ToMAS (Toronto Mapping Adaptation System) is a system that automatically detects and adapts mappings that have become invalid or inconsistent when schemas evolve [Velegrakis *et al.*, 2003, Velegrakis *et al.*, 2004b, Velegrakis *et al.*, 2004a]. It is assumed that (i) the matching step has already been performed, and (ii) correspondences have already been made operational, e.g., by using the Clio system (§6.3.2). Since in open environments, such as the web, schemas can evolve without prior notice, some correspondences may become invalid. This system aims at handling such cases, thereby preserving mapping consistency. In this sense, ToMAS complements the systems dealing with the problems mentioned in items (i) and (ii) above. In particular, it detects mappings affected by structural or constraint changes and it generates automatically the necessary rewritings that are consistent with the updates that have occurred. ToMAS handles relational and XML schemas. It takes two schemas and a set of mappings between them as input. The system works in two phases. First, as a preprocessing step, mappings are analysed and turned into logically valid mappings (if they are not already). During the second step, the result of the previous step is maintained through schema changes. In particular, mappings are

modified one by one independently, as appropriate for each kind of change that may occur to the schemas. Three classes of (primitive) schema changes are addressed: (i) operations that change the schema semantics by adding or removing constraints, (ii) modifications to the schema structure by adding or removing elements, and (iii) modifications that reshape schema structure by moving, copying, or renaming elements. The final result of ToMAS is a set of *adapted* mappings which are consistent with the structure and semantics of the evolved schemas.

6.1.16 MapOnto (University of Toronto and Rutgers University)

MapOnto is a system for constructing complex mappings between ontologies and relational or XML schemas [An *et al.*, 2005a, An *et al.*, 2005b, An *et al.*, 2006]. This system operates in a similar settings as the Clio tool (§6.3.2). In a sense, this work can be viewed as an extension of Clio when the target schema is an ontology which is treated as a relational schema consisting of unary and binary tables. MapOnto takes as input three arguments: (i) an ontology specified in an ontology representation language, e.g., OWL, (ii) relational or XML schema, and (iii) simple correspondences, e.g., between XML attributes and ontology datatype properties. Input schema and ontology are internally encoded as labelled graphs. Then, the approach looks for 'reasonable' connections among the graphs. The system produces in a semi-automatic way a set of complex mapping formulas expressed in a subset of first-order logic (Horn clauses). The list of logical formulas is also ordered by the tool, thereby suggesting the most reasonable mappings. Finally, users can inspect that list and choose the best ones.

6.1.17 OntoMerge (Yale University and University of Oregon)

OntoMerge [Dou *et al.*, 2005] is a system for ontology translation on the semantic web. Ontology translation refers here to such tasks as (i) dataset translation, that is translating a set of facts expressed in one ontology to another; (ii) generating ontology extensions, that is given two ontologies o and o' and an extension (subontology) o_s of the first one, build the corresponding extension o'_s, and (iii) query answering from multiple ontologies. The main idea of the approach is to perform ontology translation by ontology merging and automated reasoning. Input ontologies are translated from a source knowledge representation formalism, e.g., OWL, to an internal representation, which is Web-PDDL [McDermott and Dou, 2002]. Merging two ontologies is performed by taking the union of the axioms defining them. Bridge axioms or bridge rules are then added to relate the terms in one ontology to the terms in the other. Once the merged ontology is constructed, the ontology translation tasks can be performed fully automatically by mechanised reasoning. In particular, inferences, depending on the task, are conducted either in a demand-driven (backward-chaining) or data-driven (forward chaining) way with the help of a first-order theorem prover, called *OntoEngine*. It is assumed that bridge rules are to be provided by domain experts, or by other matching algorithms, which are able to discover and interpret them with clear semantics. Finally, it is worth noting that OntoMerge supports bridge rules which can be expressed using the full power of predicate calculus.

6.1.18 CtxMatch and CtxMatch2 (University of Trento and ITC-IRST)

CtxMatch [Bouquet *et al.*, 2003c, Bouquet *et al.*, 2003b] uses a semantic matching approach (§4.5.2). It translates the ontology matching problem into the logical validity problem and computes logical relations, such as equivalence, subsumption between concepts and properties. CtxMatch is a sequential system. At the element level it uses only WordNet to find initial matches for classes (§4.2.2). CtxMatch2 [Bouquet *et al.*, 2006] improves on CtxMatch by handling properties. At the structure level, it exploits description logic reasoners, such as Pellet [Sirin *et al.*, 2007] or FaCT [Tsarkov and Horrocks, 2006] to compute the final alignment in a way similar to what is presented in Sect. 4.5.2.

6.1.19 S-Match (University of Trento)

S-Match implements the idea of semantic matching as initially described in [Giunchiglia and Shvaiko, 2003a]. The first version of the S-Match system was a rationalised re-implementation of CtxMatch with a few added functionalities [Giunchiglia *et al.*, 2004]. Later the system has undergone several evolutions, including extensions of libraries of element- and structure-level matchers, adding alignment explanations as well as iterative semantic matching [Giunchiglia and Yatskevich, 2004, Shvaiko *et al.*, 2005, Giunchiglia *et al.*, 2005a, Giunchiglia *et al.*, 2006c, Giunchiglia *et al.*, 2007]. S-Match is limited to tree-like structures and does not consider properties or roles.

S-Match takes as input two graph-like structures, e.g., classifications, XML schemas, ontologies, and returns as output logic relations, e.g., equivalence, subsumption, which are supposed to hold between the nodes of the graphs. The relations are determined by (i) expressing the entities of the ontologies as logical formulas, and (ii) reducing the matching problem to a propositional validity problem. In particular, the entities are translated into propositional formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet. This allows for a translation of the matching problem into a propositional validity problem, which can then be efficiently resolved using (sound and complete) state of the art propositional satisfiability solvers.

S-Match was designed and developed as a platform for semantic matching, namely a modular system with the core of computing semantic relations where single components can be plugged, unplugged or suitably customised. It is a sequential system with a parallel composition at the element level, see Fig. 6.3. The input ontologies (tree-like structures) are codified in a standard internal XML format. The module taking input ontologies performs some preprocessing with the help of oracles which provide the necessary a priori lexical and domain knowledge. Examples of oracles include WordNet (§4.2.2) and UMLS[2]. The output of the module is an enriched tree. These enriched trees are stored in an internal database (PTrees) where they can be browsed, edited and manipulated. The Match manager coordinates the matching

[2] http://www.nlm.nih.gov/research/umls/

process. S-Match libraries contain around 20 basic element-level matchers representing three categories, namely string-based, such as n-gram, edit distance (§4.2.1), WordNet sense-based and WordNet gloss-based matchers (§4.2.2). Structure-level matchers include SAT solvers, e.g., those of SAT4J[3], and ad hoc reasoning methods [Giunchiglia *et al.*, 2005b].

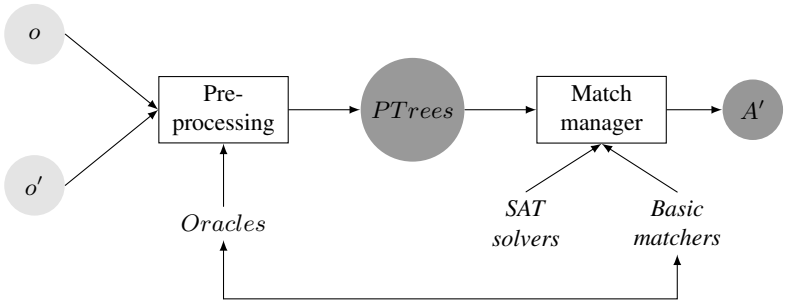

Fig. 6.3. S-Match architecture: ontology entities are converted to logic formulas by using the preprocessor and oracles. The Match manager then uses various basic element-level matchers and logic provers for finding relations between these formulas which, in turn, correspond to relations between the entities.

6.1.20 HCONE (University of the Aegean)

HCONE is an approach to domain ontology matching and merging by exploiting different levels of interaction with users [Kotis *et al.*, 2006, Vouros and Kotis, 2005, Kotis and Vouros, 2004]. First, an alignment between two input ontologies is computed with the help of WordNet (§4.2.2). Then, the alignment is processed straightforwardly by using some merging rules, e.g., renaming, into a new merged ontology. The HCONE basic matching algorithm works in six steps:

1. Chose a concept from one ontology, denoted as c.
2. Obtain all the WordNet senses of c, denoted as $s_1, s_2, \ldots, s_m$. For example, the concept Facility has five senses in WordNet.
3. Obtain hypernyms and hyponyms of all the senses of c (§4.2.2). For example, Police is a hyponym of Facility.
4. Build the $n \times m$ *association matrix*. This relates the n most frequently occurring terms in the *vicinity* of the m senses determined in step 2. The vicinity terms include those from the same synsets of m senses, hypernyms and hyponyms from step 3. In the case of the Facility example this is a 93×5 matrix. For example, the number of occurrences of such a vicinity term as Police is 3.

[3] http://www.sat4j.org/

5. Build a query q by using terms which are subconcepts of c, e.g., TransportationSystem, or which are related to c via domain specific relations in the input ontology. If the terms considered for q also exist among the n terms from step 4, then q memorises that position with the help of flags. Thus, for the Facility concept, q is a 93 position vector, and, since the position of TransportationSystem is at the 35th place the value of $q[35]$ is 1.
6. Taking as input the association matrix computed at step 4 and the query computed at step 5, Latent Semantic Indexing (§4.2.2) is used to compute the grades for what is the correct WordNet sense to be used for the given context (query).

The highest graded sense expresses the most plausible meaning for the concept under consideration. Finally, the relationship between concepts is computed. For instance, equivalence between two concepts holds if the same WordNet sense has been chosen for those concepts based on the procedure described above. The subsumption relation is computed between two concepts if a hypernym relation holds between the WordNet senses corresponding to these concepts. Based on the level at which users are involved in the matching process, HCONE provides three algorithms to ontology matching. These are: fully automated, semi-automated and user-based. Users are involved in order to provide feedback on what is to be the correct WordNet sense on a one by one basis (user-based), or only in some limited number of cases, by exploiting some heuristics (semi-automated).

6.1.21 MoA (Electronics and Telecomunication Research Institute, ETRI)

MoA is an ontology merging and alignment tool [Kim *et al.*, 2005]. It consists of: (i) a library of methods for importing, matching, modifying, merging ontologies, and (ii) a shell for using those methods. MoA handles ontologies specified in OWL-DL. It is able to compute equivalence and subsumption relations between entities (classes, properties) of the input ontologies. The matching approach is based on concept (dis)similarity derived from linguistic clues. The MoA tool is a sequential solution. The preprocessing step includes three phases: (i) names of classes and properties are tokenised (§4.2.1); (ii) tokens of entities are associated with their meaning by using WordNet senses; (iii) meanings of tokens of ancestors of the entity under consideration are also taken into account, thereby extending the local meanings. This step is essentially the same as some part of the preprocessing done within the S-Match system (§6.1.19). Matching itself is based on rules. It is performed in a double loop over all the pairs of entities from two input ontologies. For example, equivalence between two classes or properties holds when there is equivalence between these entities in either step (ii) or (iii). The equivalence, in turn, is decided via relations between the WordNet senses for one of the possible solutions (see Sect. 4.2.2). Thus, for example, author can be found to be equivalent to writer because they belong to the same synset in WordNet.

6.1.22 ASCO (INRIA Sophia-Antipolis)

ASCO is a system that automatically discovers pairs of corresponding elements in two input ontologies [Bach *et al.*, 2004]. ASCO handles ontologies in RDF Schema and computes alignments between classes, relations, and classes and relations. A new version, ASCO2, deals with OWL ontologies [Bach and Dieng-Kuntz, 2005].

The matching is organised sequentially in three phases. During the first phase (linguistic matching) the system normalises terms and expressions, e.g., by punctuation, upper cases, special symbols. Depending on their use in the ontology or if they are bags of words, ASCO uses different string comparison metrics for comparing the terms. Single terms are compared by using Jaro–Winkler, Levenshtein or Monge–Elkan (§4.2.1) and external resources, such as WordNet. Based on token similarities, the similarity between sets of tokens is computed using TFIDF. The obtained values are aggregated through a weighted sum.

The second phase (structure matching), computes similarities between classes and relations by propagating the input of linguistic similarities. The algorithms is an iterative fixed point computation algorithm that propagates similarity to the neighbours (subclasses, superclasses and siblings). Similatities between sets of objects are computed through single linkage. The propagation terminates when the class similarities and the relation similarities do not change after an iteration or a certain number of iterations is reached.

In the third phase, the linguistic and structural similarity are aggregated through a weighted sum and, if the similarities between matching candidates exceed a threshold (§5.7.1), they are selected for the resulting alignment.

6.1.23 BayesOWL and BN mapping (University of Maryland)

BayesOWL is a probabilistic framework for modelling uncertainty in the semantic web. It includes the Bayesian Network mapping module (§5.5.1), which is in charge of automatic ontology matching [Pan *et al.*, 2005]. The approach works in three steps. First, two input ontologies are translated into two Bayesian networks. Specifically, classes are translated into nodes in Bayesian network, while edges are created if the corresponding two classes are related by a *predicate* in the input ontologies. During the second step, matching candidates are generated between two Bayesian networks by learning joint probabilities from the web data. In particular, for each concept in an ontology, a group of sample text documents (called *examplars*) is created by querying a search engine. The query contains all the terms, e.g., {product book science} (opposed to a single term, e.g., {science}), in the path from the root to the concept (term) under consideration in the given ontology, thereby enabling some word sense disambiguation (§4.2.2). A text classifier, e.g., Rainbow[4], is trained on the statistical information about examplars from the first ontology. Then, concepts of the second ontology are classified with respect to the concepts of the first ontology by feeding their examplars to the trained classifier. A similarity between two concepts is determined with the help of the Jaccard coefficient

[4] http://www.cs.umass.edu/~mccallum/bow/rainbow/

(§4.4) computed from the joint probabilities. These are used to construct the conditional probability tables. During the third step, the mappings are refined as an update (combination of the Jeffrey rule and Iterative Proportional Fitting Procedure [Jeffrey, 1983, Cramer, 2000]) on probability distributions of concepts in the second Bayesian network, by distributions of concepts in the first Bayesian network, in accordance with the given similarities. By performing Bayesian inference with the updated distribution of the second Bayesian network, the final alignment is produced.

6.1.24 OMEN (The Pennsylvania State University and Stanford University)

OMEN (Ontology Mapping ENhancer [Mitra *et al.*, 2005]) is a semi-automatic probabilistic ontology matching system based on a Bayesian network (§5.5.1). It takes as input two ontologies and an initial probability distribution derived, for instance, from basic (element level) linguistic matchers. In turn, OMEN provides a structure level matching algorithm, thereby deriving the new mappings or discarding the existing false mappings. The approach can be summarised in four logical steps. First, it creates a Bayesian network, where a node stands for a mapping between pairs of classes or properties of the input ontologies. Edges represent the influences between the nodes of the network. This encoding is different from the one described in Sect. 6.1.23. During the second step, OMEN uses a set of *meta-rules* that capture the influence of the structure of input ontologies in the neighbourhood of the input mappings in order to generate conditional probability tables for the given network. An example of a basic meta-rule is as follows. There are two conditions: (i) if the i-th concept from the first ontology, $c_{1,i} \in o_1$, matches the j-th concept from the second ontology, $c_{2,j} \in o_2$; (ii) if there is a relation q between concepts $c_{1,i}$ and $c_{1,k}$ in the first ontology, which matches a relation q' between concepts $c_{2,j}$ and $c_{2,m}$ in the second ontology. Then we can increase the probability of match between concepts $c_{1,k}$ and $c_{2,m}$. Other rules rely more heavily on the semantics of the language in which the input ontologies are encoded. During the third step, inferences are made (OMEN uses Bayesian Network tools in Java (BNJ)[5]) to generate *a posteriori* probabilities for each node. Finally, *a posteriori* probabilities, which are higher than a threshold (§5.7.1), are selected to generate the resulting alignment.

6.1.25 DCM framework (University of Illinois at Urbana-Champaign)

MetaQuerier [Chang *et al.*, 2005] is a middleware system that assists users in finding and quering multiple databases on the web. It exploits the Dual Correlation Mining (DCM) matching framework to facilitate source selection according to user search keywords [He and Chang, 2006]. Unlike other works, the given approach takes as input multiple schemas and returns alignments between all of them. This setting is called *holistic* schema matching. DCM automatically discovers complex correspondences, e.g., {author} corresponds to {first name, last name}, between attributes of the web query interfaces in the same domain of interest, e.g., books. As the name

[5] http://bnj.sourceforge.net

DCM indicates, schema matching is viewed as *correlation mining*. The idea is that co-occurrence patterns often suggest complex matches. That is, *grouping attributes*, such as first name and last name, tend to co-occur in query interfaces. Technically, this means that those attributes are positively correlated. Contrary, attribute names which are synonyms, e.g., quantity and amount, rarely co-occur, thus representing an example of negative correlation between them. Matching is performed in two phases. During the first phase (matching discovery), a set of matching candidates is generated by mining first positive and then negative correlations among attributes and attribute groups. Some thresholds and a specific correlation measure such as the H-measure are also used. During the second phase (matching construction), by applying ranking strategies, e.g., scoring function, rules, and selection, such as iterative greedy selection (§5.7.3), the final alignment is produced.

6.2 Instance-based systems

Instance-based systems are those taking advantage mostly of instances, i.e., of data expressed with regard to the ontology or data indexed by the ontology.

6.2.1 T-tree (INRIA Rhône-Alpes)

T-tree [Euzenat, 1994] is an environment for generating taxonomies and classes from objects (instances). It can, in particular, infer dependencies between classes, called bridges, of different ontologies sharing the same set of instances based only on the extension of classes (§4.4.1). The system, given a set of source taxonomies called viewpoints, and a destination viewpoint, returns all the bridges in a minimal fashion which are satisfied by the available data. That is the set of bridges for which the objects in every source class are indeed in the destination class. The algorithm compares the extension (set of instances) of the presumed destination to the intersection of those of the presumed source classes. If there is no inclusion of the latter in the former, the algorithm is re-iterated on all the sets of source classes which contain at least one class which is a subclass of the tested source classes. If the intersection of the extension of the presumed source classes is included in that of the presumed destination class, a bridge can be established from the latter (and also from any set of subclasses of the source classes) to the former (and also any superclass of the destination class). However, other bridges can also exist on the subclasses of the destination. The algorithm is thus re-iterated on them. It stops when the bridge is trivial, i.e., when the source is empty. Users validate the inferred bridges.

Bridge inference is the search for correlation between two sets of variables. This correlation is particular to a data analysis point of view since it does not need to be valid on the whole set of individuals (the algorithm looks for subsets under which the correlation is valid) and it is based on strict set equality (not similarity). However, even if the bridge inference algorithm has been described with set inclusion, it can be helped by other measurements which will narrow or broaden the search. More

generally, the inclusion and emptiness tests can be replaced by tests based on the similarity of two sets of objects (as is usual in data analysis).

The bridge inference algorithm is not dependent on the instance-based interpretation: it depends on the meaning of the operators $\subseteq$, $\cap$ and $= \emptyset$-test (which are interpreted as their set-theoretic counterpart in the case of the instance-based algorithms). A second version of the system (with the same properties) uses structural comparison: $\subseteq$ is subtyping, $\cap$ is type intersection and $= \emptyset$-test is a subtyping test.

6.2.2 CAIMAN (Technische Universität München and Universität Kaiserslautern)

CAIMAN [Lacher and Groh, 2001] is a system for document exchange, which facilitates retrieval and publishing services among the communities of interest. These services are enabled by using semi-automatic ontology matching. The approach focuses on light-weight ontologies, such as web classifications. The main idea behind matching is to calculate a probability measure between the concepts of two ontologies, by applying machine learning techniques for text classification, e.g., the Rocchio classifier. In particular, based on the documents, a representative feature vector (a word-count, weighted by TFIDF feature vector, §4.2.1) is created for each concept in an ontology. Then, the cosine measure (§4.2.1) is computed for two of those class vectors. Finally, with the help of a threshold, the resulting alignment is produced.

6.2.3 FCA-merge (University of Karlsruhe)

FCA-merge uses formal concept analysis techniques to merge two ontologies sharing the same set of instances [Stumme and Mädche, 2001]. The overall process of merging two ontologies consists of three steps, namely (i) instance extraction, (ii) concept lattice computation, (iii) interactive generation of the final merged ontology. The approach provides, as a first step, methods for extracting instances of classes from documents. The extraction of instances from text documents circumvents the problem that in most applications there are no individuals which are simultaneously instances of the source ontologies and which could be used as a basis for identifying similar concepts. During the second step, the system uses formal concept analysis techniques (§4.4.1) in order to compute the concept lattice involving both ontologies. The last step consists of deriving the merged ontology from the concept lattice. The produced lattice is explored and transformed by users who further simplify it and generate the taxonomy of an ontology.

The result is a merge rather than an alignment. However, the concepts that are merged can be considered as exactly matched and those which are not can be considered in subsumption relation with their ancestors or siblings.

6.2.4 LSD (University of Washington)

Learning Source Descriptions (LSD) is a system for the semi-automatic discovery of one-to-one alignments between the (leaf) elements of source schemas and a mediated

(global) schema in data integration [Doan *et al.*, 2001]. The main idea behind the approach is to learn from the mappings created manually between the mediated schema and some of the source schemas, in order to propose in an automatic manner the mappings for the subsequent source schemas. LSD handles XML schemas. A schema is modelled as a tree, where the nodes are XML tag names. The approach works in two phases. During the first (training) phase, useful objects, such as element names and data values, are extracted from the input schemas. Then, from these objects and manually created alignments, the system trains multiple basic matchers (addressing different features of objects, such as formats, word frequencies, characteristics of value distributions) and a meta-matcher. Some examples of basic matchers are the WHIRL learner (§5.4.2), the naive Bayesian learner (§5.4.1). The meta-matcher combines the predictions of basic matchers. It is trained by using a stacked generalisation (learning) technique (§5.4.5). During the second (matching) phase LSD extracts the necessary objects from the remaining (new) source schemas. Then, by applying the trained basic matchers and the meta-matcher on the new objects (the *classification* operation), LSD obtains a prediction list of matching candidates. Finally, by taking into account integrity constraints and applying some thresholds, the final alignment is extracted.

6.2.5 GLUE (University of Washington)

GLUE [Doan *et al.*, 2004], a successor of LSD, is a system that employs multiple machine learning techniques to semi-automatically discover one-to-one semantic mappings (which are sometimes called 'glue' for interoperability) between two taxonomies. The idea of the approach is to calculate the *joint distributions* of the classes, instead of committing to a particular definition of similarity. Thus, any particular similarity measure can be computed as a function over the joint distributions. As does its predecessor, LSD, GLUE follows a multistrategy learning approach, involving several basic matchers and a meta-matcher. The system works in three steps. First, it learns the joint probability distributions of classes of two taxonomies. In particular, it exploits two basic matchers: the *content learner* (naive Bayes technique, §5.4.1) and the *name learner* (a variation of the previous one). The meta-matcher, in turn, performs a linear combination of the basic matchers. Weights for these matchers are assigned manually. During the second step, the system estimates the similarity between two classes in a user-supplied function of their joint probability distributions. This results in a similarity matrix between terms of two taxonomies. Finally, some domain-dependent, e.g., subsumption, and domain-independent, e.g., if all children of node x match node y, then x also matches y, constraints (heuristics) are applied by using a *relaxation labelling* technique. These are used in order to filter some of the matches out of the similarity matrix and keep only the best ones.

6.2.6 iMAP (University of Illinois and University of Washington)

iMAP [Dhamankar *et al.*, 2004] is a system that semi-automatically discovers one-to-one (e.g., amount $\equiv$ quantity) and, most importantly, complex (e.g., address $\equiv$

concat(city, street)) mappings between relational database schemas. The schema matching problem is reformulated as a search in a match space, which is often, very large or even infinite. To perform the search effectively, iMAP uses multiple basic matchers, called searches, e.g., text, numeric, category, unit conversion, each of which addresses a particular subset of the match space. For example, the text searcher considers the concatenation of text attributes, while the numeric searcher considers combining attributes with arithmetic expressions. The system works in three steps (see Fig. 6.4). First, matching candidates are generated by applying basic matchers (the match generator module). Even if a basic matcher, such as the text searcher, addresses only the space of concatenations, this space can still be very large. To this end, the search strategy is controlled by using the *beam search* technique [Russell and Norvig, 1995]. During the second step, for each target attribute, matching candidates of the source schema are evaluated by exploiting additional types of information, e.g., using the naive Bayes evaluator (§5.4.1), which would be computationally expensive to use during the first step. These yield additional scores. Then, all the scores are combined into a final one (the similarity estimator module). The result of this step is a similarity matrix between $\langle target\ attribute, match\ candidate \rangle$ pairs. Finally, by using a set of domain constraints and mappings from the previous match operations (if applicable and available), the similarity matrix is cleaned up, such that only the best matches for target attributes are returned as the result (the match selector module). The system is also able to explain the results it produces with the help of the explanation module, see for details Chap. 9.

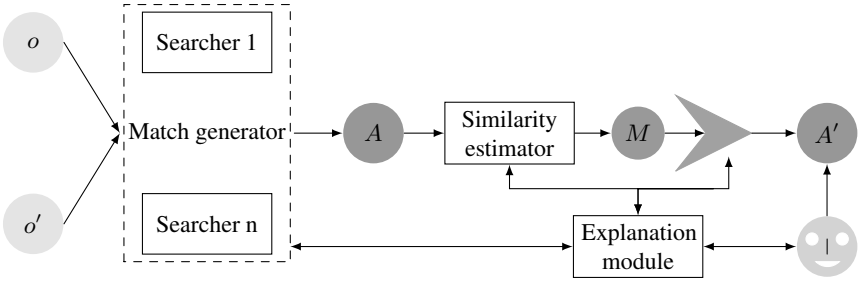

Fig. 6.4. iMAP architecture: several matchers, called searchers, are run in parallel. They provide candidate matches that can be complex. These candidates are further selected by applying the similarity estimator, and then, the final alignment is extracted. Additionally, the explanation module allows users to understand the results and control the process.

6.2.7 Automatch (George Mason University)

Automatch [Berlin and Motro, 2002] is a system for automatic discovery of mappings between the attributes of database schemas. The approach assumes that several schemas from the domain under consideration have already been manually matched

by domain experts. This assumption is a realistic one for a data integration scenario. Then, by using Bayesian learning (§5.4.1), Automatch acquires probabilistic knowledge from the manually matched schemas, and creates the *attribute dictionary* which accumulates the knowledge about each attribute by means of its *possible values* and the *probability estimates* of these values. In order to avoid a rapid growth of the dictionary, the system also uses *statistical feature selection* techniques, such as mutual information, information gain, and likelihood ratio, to learn efficiently, i.e., only from the most informative values, such as 10% of the actually available input training data. A new pair of schemas is matched automatically via the precompiled attribute dictionary. The system first matches each attribute of the input schemas against the attribute dictionary, thereby producing individual match scores (a real number). Then, these individual scores are further combined by taking their sum to produce the scores between the attributes of the input schemas. Finally, the scores between the input schemas, in turn, are combined again, by using a *minimum cost maximum flow* graph algorithm and some thresholds in order to find the overall optimal matching between the input schemas with respect to the sum of the individual match scores.

6.2.8 SBI&NB (The Graduate University for Advanced Studies)

SBI (Similarity-Based Integration) is a system for automatic statistical matching among classifications [Ichise *et al.*, 2003, Ichise *et al.*, 2004]. SBI&NB is the extension of SBI by plugging into the system a naive Bayes classifier (§5.4.1). The idea of SBI is to determine correspondences between classes of two classifications by statistically comparing the membership of the documents to these classes. The pairs of similar classes are determined in a top-down fashion by using the κ-statistic method [Fleiss, 1973]. These pairs are considered to be the final alignment. SBI&NB combines sequentially the SBI and the naive Bayes classifier. The naive Bayes enables hierarchical classification of documents. Thus, the system takes also into account structural information of the input classifications. The exploited classifier is Pachinko Machine naive Bayes from the Rainbow system[4].

6.2.9 Kang and Naughton (University of Wisconsin-Madison)

Kang and Naughton proposed a structural instance-based approach for discovering correspondences among attributes of relational schemas with *opaque* column names [Kang and Naughton, 2003]. By opaque column names are meant names which are hard to interpret, such as A and B instead of Model and Color. The approach works in two phases. During the first phase, two table instances are taken as input and the corresponding (weighted) dependency graphs are constructed based on *mutual information* and *entropy*. The conditional entropy used here describes (with a non negative real number) the uncertainty of values in an attribute given knowledge (probability distribution) of another attribute. Mutual information, in turn, measures (with a non negative real number) the reduction in uncertainty of one attribute due to the knowledge of the other attribute, i.e., the amount of information captured in one attribute

about the other. It is zero when two attributes are independent, and increases as the dependency between the two attributes grows. Mutual information is computed over all pairs of attributes in a table. Thus, in dependency graphs, a weight on an edge stands for mutual information between two adjacent attributes. A weight on a node stands for entropy of the attribute. During the second phase, matching node pairs are discovered between the dependency graphs by running a graph matching algorithm. The quality of matching is assessed by using metrics, such as the Euclidean distance (§4.2.1). The distance is assigned to each potential correspondence between attributes of two schemas and a one-to-one alignment which is a minimum weighted graph matching (§5.7.3) is returned.

6.2.10 Dumas (Technische Universität Berlin and Humboldt-Universität zu Berlin)

Dumas (DUplicate-based MAtching of Schemas) is an approach which identifies one-to-one alignments between attributes by analysing the duplicates in data instances of the relational schemas [Bilke and Naumann, 2005]. Unlike other instance-based approaches which look for similar properties of instances, such as distribution of characters, in columns of schemas under consideration, this approach looks for similar rows or tuples. The system works in two phases: (i) identify objects within databases with opaque schemas, and (ii) derive correspondences from a set of similar duplicates.

For object identification (§4.4.2), in Dumas, tuples are viewed as strings and a string comparison metric, such as cosine measure (§4.2.1), is used to compare two tuples. Specifically, tuples are tokenised and each token is assigned a weight based on TFIDF scheme (§4.2.1). In order to avoid complete pairwise comparison of tuples from two databases, the WHIRL algorithm (§5.4.2) is used. It performs a focused search based on those common values that have high TFIDF score. The algorithm ranks tuple pairs according to their similarity and identifies the k most similar tuple pairs.

During the second phase, based on the k duplicate pairs with highest confidence, correspondences between attributes are derived. The intuition is that if two field values are similar, then their respective attributes match. A field-wise similarity comparison is made for each of the k duplicates, thereby resulting in a similarity matrix. For comparing tuple fields, a variation of a TFIDF-based measure, called soft TFIDF [Cohen *et al.*, 2003a], is used. It allows the consideration of similar terms as opposed to equal terms. The resulting alignment is extracted from the similarity matrix by finding the maximum weight matching. Finally, if based on the maximum matching, multiple alternative matches are possible, therefore the algorithm iterates back to the first phase in order to try to improve the result by discovering more duplicates.

6.2.11 Wang and colleagues (Hong Kong University of Science and Technology and Microsoft Research Asia)

Wang and colleagues propose an instance-based solution for discovering one-to-one alignments among the web databases [Wang *et al.*, 2004] (see also Sect. 6.1.25).

These are query interfaces (HTML forms) and backend databases which dynamically provide information in response to user queries. Authors distinguish between (i) the query interface, which exposes attributes that can be queried in the web database and (ii) the result schema presenting the query results, which exposes attributes that are shown to users. Matching between different query interfaces (inter-site matching) is critical for data integration between web databases. Matching between the interface and result schema of a single web database (intra-site matching), in turn, is useful for automatic data annotation and database content crawling. The approach is based on the following observations (among others):

– The keywords of queries (whose semantics match the semantics of the input element of a query interface) that return results are likely to reappear in attributes of the returned result. For example, such keywords as Logic submitted to the input element title matches its intended use (while it is not the case with the field author which will unlikely produce expected results), and therefore, some results with books about logics will be returned. Moreover, part (Logic) of the value Introduction to logic of the title attribute should reappear in the result schema.
– Based on the work in [He and Chang, 2003], the authors assume the existence and availability of a populated global schema, that is a view capturing common attributes of data, for the web databases of the same domain of interest.

The approach presents a combined schema model that involves five kinds of schema matching for web databases in the same domain of interest: global-interface, global-result, interface-result, interface-interface, and result-result. The approach works in two phases: (i) query probing and (ii) instance-based matching.

The first phase deals with acquiring data instances from web databases by query probing. It exhaustively sends the attribute values of pre-known instances from a global schema and collects results from the web databases under consideration in a *query occurrence cube*. The cube height stands for the number of attributes in the given global schema. The cube width stands for the number of attributes in the interface schema. The cube depth is the number of attributes in the result schema. Finally, each cell in this cube stores an occurrence count associated with the three dimensions. This cube is further projected onto three *query occurrence matrices*, which represent relationships between pairs of three schemas, namely global-interface, global-result, and interface-result.

During the second phase, the re-occurrences of submitted query keywords in the returned results data are analysed. In order to perform intra-site matching, the *mutual information* between pairs of attributes from two schemas is computed (see also Sect. 6.2.9). In order to perform inter-site matching a vector-based similarity is used (§4.2.1). In particular, each attribute of an individual interface or result schema is viewed as a *document* and each attribute of the global schema is view as a *concept*. Each row in the occurrence matrix represents a corresponding document vector. The similarity between attributes from different schemas is computed by using the cosine measure (§4.2.1) between two vectors. Finally, for both intra-site matching and inter-site matching, the matrix element whose value is the largest both in its row and

column represents a final correspondence (this is the greedy alignment extraction presented in Sect. 5.7.3).

6.2.12 sPLMap (University of Duisburg-Essen, and ISTI-CNR)

sPLMap (probabilistic, logic-based mapping between schemas) is a framework which combines logics with probability theory in order to support uncertain schema mappings [Nottelmann and Straccia, 2005, Nottelmann and Straccia, 2006]. In particular, it is a GLAV-like framework [Lenzerini, 2002] where the alignment is defined as uncertain rules in probabilistic Datalog. This allows the support for imprecise matches, e.g., between author and editor attributes and a more general attribute, such as creator, which is often the case in schemas with different levels of granularity. sPLMap matches only attributes of the same concept (typically documents). The system operates in three main steps. First, it evaluates the quality of all possible individual correspondences on the basis of probability distributions (called interpretation). It selects the set of correspondences that maximises probability on the basis of instance data.

Then, for each correspondence, matchers are used as quality estimators: they provide a measure of the plausibility of the correspondence. sPLMap has been tested with the following matchers: (i) same attribute names (§4.2.1), (ii) exact tuples (§4.4), (iii) the k-nearest neighbour classifier, and (iv) the naive Bayesian classifier (§5.4.1). The result of these matchers are aggregated by using linear or logistic functions, or their combinations (§5.2). Coefficients of the normalisation functions are learnt by regression in a system-training phase. Finally, the computed probabilities are transformed in correspondence weights (used as the probability of the corresponding Datalog clause) by using the Bayes theorem.

6.3 Mixed, schema-based and instance-based systems

The following systems take advantage of both schema-level and instance-level input information if they are both available.

6.3.1 SEMINT (Northwestern University, NEC and The MITRE Corporation)

SEMantic INTegrator (SEMINT) is a tool based on neural networks to assist in identifying attribute correspondences in heterogeneous databases [Li and Clifton, 1994, Li and Clifton, 2000]. It supports access to a variety of database systems and utilises both schema- and instance-level information to produce rules for matching corresponding attributes automatically. The approach works as follows. First, it extracts from two databases all the necessary information (features or discriminators) which is potentially available and useful for matching. This includes normalised schema information, e.g., field specifications, such as datatypes, length, constraints, and statistics about data values, e.g., character patterns, such as ratio of numerical characters,

ratio of white spaces, and numerical patterns, such as mean, variance, standard deviation. Second, by using a neural network as a classifier with the self-organising map algorithm (§5.4.3), it groups the attributes based on similarity of the features for a single (the first) database. Then, it uses a back-propagation neural network for learning and recognition. Based on the previously obtained clusters, the learning is performed. Finally, using a trained neural network on the first database features and clusters, the system recognises and computes similarities between the categories of attributes from the first database and the features of attributes from the second database, thus, generating a list of match candidates, which are to be inspected and confirmed or discarded by users.

6.3.2 Clio (IBM Almaden and University of Toronto)

Clio is a system for managing and facilitating data transformation and integration tasks within heterogeneous environments [Miller *et al.*, 2000, Miller *et al.*, 2001, Naumann *et al.*, 2002, Haas *et al.*, 2005], see Fig. 6.5. Clio handles relational and XML schemas. As a first step, the system transforms the input schemas into an internal representation, which is a nested relational model. The Clio approach is focused on making the alignment operational. It is assumed that the matching step, namely, identification of the *value correspondences*, is performed with the help of a schema matching component or manually. The built-in schema matching algorithm of Clio combines in a sequential manner instance-based attribute classification via a variation of a naive Bayes classifier (§5.4.1) and string matching between elements names, e.g., by using an edit distance (§4.2.1). Then, taking the n-m value correspondences (the alignment) together with constraints coming from the input schemas, Clio compiles these into an internal query graph representation. In particular, an interpretation of the input correspondences is given. Thus, a set of logical mappings with formal semantics is produced. To this end, Clio also supports mappings composition [Fagin *et al.*, 2004]. Finally, the query graph can be serialised into different query languages, e.g., SQL, XSLT, XQuery, thus enabling actual data to be moved from a source to a target, or to answer queries. The system, besides trivial transformations, aims at discovering complex ones, such as the generation of keys, references and join conditions.

6.3.3 IF-Map (University of Southampton and University of Edinburgh)

IF-Map (Information-Flow-based Map) [Kalfoglou and Schorlemmer, 2003a] is an ontology matching system based on the Barwise–Seligman theory of information flow [Barwise and Seligman, 1997]. The basic principle of IF-Map is to match two local ontologies by looking at how these are related to a common reference ontology. It is assumed that such a reference ontology represents an agreed understanding that facilitates the sharing of knowledge. This means that two local ontologies have significant fragments of them that conform to the reference ontology. It is also assumed that the local ontologies are populated with instances, while the reference ontology does not need to.

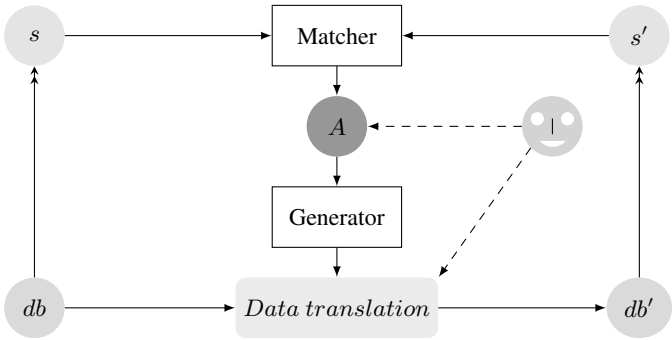

Fig. 6.5. Clio architecture: Clio goes all the way from matching schemas to translating data from one database to another one. It is made up of a classical matcher but also involves users at each step: input, matching control and translation execution.

Matching works as follows. If the reference ontology can be expressed in each of the local ontologies and instances of the local ontologies can be assigned concepts in the reference ontology (or be mapped to equivalent instances in the reference ontology), then IF-Map uses formal concept analysis (§4.4.1) between the three ontologies in order to find the Galois lattice from which it is possible to extract an alignment.

When the mappings are not available, IF-Map generates candidate pairs of mappings (called *infomorphism* in information flow theory) and artificial instances. They are generated through the enforcement of constraints that are induced by the definition of the reference ontology and by heuristics based on string-based (§4.2.1) and structure-based (§4.3) methods.

IF-Map deals with ontologies expressed in KIF or RDF. The IF-MAP method is declaratively specified in Horn logic and is executed with a Prolog interpreter, so the ontologies are translated into Prolog clauses beforehand. IF-Map produces concept-to-concept and relation-to-relation alignments.

6.3.4 NOM and QOM (University of Karlsruhe)

NOM (Naive Ontology Mapping) [Ehrig and Sure, 2004] and QOM (Quick Ontology Mapping) [Ehrig and Staab, 2004] are components of the FOAM framework (§8.2.5).

NOM adopts the idea of parallel composition of matchers from COMA (§6.1.12). Some innovations with respect to COMA are in the set of elementary matchers based on rules, exploiting explicitly codified knowledge in ontologies, such as information about super and subconcepts, super and subproperties, etc. As from [Ehrig and Sure, 2004], the system supports 17 rules related to those of Table 4.6 (p. 100). For example, a rule states that if superconcepts are the same, the actual concepts are similar to each other. These rules are based on various terminological and structural techniques.

QOM (Quick Ontology Mapping) [Ehrig and Staab, 2004] is a variation of the NOM system dedicated to improve the efficiency of the system. The approach is based on the idea that the loss of quality in matching algorithms is marginal (to a standard baseline); however improvement in efficiency can be significant. This fact allows QOM to produce correspondences fast, even for large-size ontologies. QOM is grounded on matching rules of NOM. However, for the purpose of efficiency the use of some rules, e.g., the rules that traverse the taxonomy, has been restricted. QOM avoids the complete pairwise comparison of trees in favour of an incomplete top-down strategy, thereby focusing only on promising matching candidates.

The similarity measures produced by basic matchers (matching rules) are refined by using a sigmoïd function (§5.7.2), thereby emphasising high individual similarities and de-emphasising low individual similarities. They are then aggregated through weighted average (§5.2). Finally, with the help of thresholds, the final alignment is produced.

6.3.5 oMap (CNR Pisa)

oMap [Straccia and Troncy, 2005] is a system for matching OWL ontologies. It is built on top of the Alignment API (§8.2.4) and has been used for distributed information retrieval in [Straccia and Troncy, 2006]. oMap uses several matchers (called classifiers) that are used for giving a plausibility of a correspondence as a function of an input alignment between two ontologies. The matchers include (i) a classifier based on classic string similarity measure over normalised entity names (§4.2.1); (ii) a naive Bayes classifier (§5.4.1) used on instance data, and (iii) a 'semantic' matcher which propagates initial weights through the ontology constructors used in the definitions of ontology entities. This last one starts with an input alignment associating plausibility to correspondences between primitive entities and computes the plausibility of a new alignment by propagating these measures through the definitions of the considered entities. The propagation rules depend on the ontology constructions, e.g., when passing through a conjunction, the plausibility will be minimised. Each matcher has its own threshold and they are ordered among themselves.

There are two ways in which matchers can work: (i) in parallel, in which case their results are aggregated through a weighted average, such that the weights correspond to the credit accorded to each of the classifiers, (ii) in sequence, in which case each matcher only adds new correspondences to the input ontologies. A typical order starts with string similarity, before naive Bayes, and then the 'semantic' matcher is used.

6.3.6 Xu and Embley (Brigham Young University)

Xu and Embley proposed a parallel composition approach to discover, in addition to one-to-one alignments, also one-to-many and many-to-many correspondences between graph-like structures, e.g., XML schemas, classifications [Xu and Embley, 2003, Embley *et al.*, 2004]. Schema matching is performed by a

combination (an average function) of multiple matchers and with the help of external knowledge resources, such as domain ontologies. The basic element level matchers used in the approach include *name matcher* and *value-characteristic matcher*. The name matcher, besides string comparisons (§4.2.1), also performs some linguistic normalisation, such as stemming and removing stop words (§4.2.2). It also detects synonyms among node names with the help of WordNet (§4.2.2). In particular, matching rules are obtained via a C4.5 decision tree generator (§5.4.4) that has been trained over WordNet by using several hundreds synonym names found in the available databases from a domain of interest. The value-characteristic matcher determines where two values of schema elements share similar value characteristics, such as means or variances for numerical data. Similar to the name matcher, matching rules are obtained by training the C4.5 decision tree generator over value characteristics of the available databases from a domain of interest. Structure level matchers are used to suggest new correspondences as well as to confirm correspondences identified by element level matchers, for example, by considering similarities between the neighbour elements computed by element level matchers. Another example of a structural matcher makes use of a domain ontology. In particular, it tries to match both schemas A and B to the structure C, which is an external domain ontology, in order to decide if A corresponds to B.

6.3.7 Wise-Integrator (SUNY at Binghamton, University of Illinois at Chicago and University of Louisiana at Lafayette)

Wise-Integrator is a tool that performs automatic integration of Web Interfaces of Search Engines [He *et al.*, 2004, He *et al.*, 2005]. It provides a unified interface to e-commerce search engines of the same domain of interest, such as books and music. Therefore, users can pose queries by using this interface and the search mediator sends the translated subqueries to each site involved in handling this query and then the results of these sites are reconciled and presented to users. Wise-Integrator consists of two main subsystems: (i) an interface schema extractor, and (ii) an interface schema integrator. The first component, given a set of HTML pages with query interfaces, identifies logical attributes and derives some meta-information about them, e.g., datatype, thereby building an interface schema out of them. For example, the system can derive (guess) that the field Publication Date, is likely to be of date datatype. The second component discovers matching attributes among multiple query interfaces and then merges them, thereby resulting in global attributes. These are used, in turn, to produce a unified search interface.

Attribute matching in Wise-Integrator is based on two types of matches: *positive* and *predictive*. Positive matches are based on the following matching methods: exact name match, look up for synonymy, hypernymy and meronymy in WordNet (§4.2.2), and value-based matchers. When one of the positive matches occurs, the corresponding attributes are considered as matched. Predictive matches are based on the following matching methods: approximate name match, e.g., edit distance (§4.2.1), datatype compatibility (§4.3.1), value pattern matcher (§4.4.3). Predictive

matches have to be strong enough (which is decided based on a threshold) in order to indicate that the attributes under consideration match.

Positive and predictive matches are carried out in two clustering steps: *positive match based clustering* and *predictive match based clustering*. In the first step all the interfaces are taken as input and attributes are grouped into clusters based on the positive matches between attributes. Clustering is done by following some pre-defined rules which govern the order of execution of underlying matchers and how to make groupings based on results of those matchers. For example, the first results of exact name matches are considered and then results of value-based and WordNet-based matchers. Finally, for each cluster a representative attribute name (RAN) is determined. For example, for the cluster with attribute names {Format, Binding type, Format} the RAN is Format. During the second step all local interfaces are reconsidered again. Clustering is done following some pre-defined rules which employ previously determined RANs and a simple weighting scheme over the results of predictive matching methods, if all else fail. When all potentially matching attributes are clustered together, the global attribute for each group of such attributes is generated.

6.3.8 OLA (INRIA Rhône-Alpes and Université de Montréal)

OLA (OWL Lite Aligner) [Euzenat and Valtchev, 2004] is an ontology matching system which is designed with the idea of balancing the contribution of each of the components that compose an ontology, e.g., classes, constraints, data instances. OLA handles ontologies in OWL. It first compiles the input ontologies into graph structures, unveiling all relationships between entities. These graph structures produce the constraints for expressing a similarity between the elements of the ontologies. The similarity between nodes of the graphs follows two principles: (i) it depends on the category of node considered, e.g., class, property, and (ii) it takes into account all the features of this category, e.g., superclasses, properties, as presented in Table 4.6.

The distance between nodes in the graph are expressed as a system of equations based on string-based (§4.2.1), language-based (§4.2.2) and structure-based (§4.3) similarities (as well as taking instances into account whenever necessary). These distances are almost linearly aggregated (they are linearly aggregated modulo local matches of entities). For computing these distances, the algorithm starts with base distance measures computed from labels and concrete datatypes. Then, it iterates a fixed point algorithm until no improvement is produced. From that solution, an alignment is generated which satisfies some additional criterion on the obtained alignment and the distance between matched entities. The algorithm is described in more detail in Sect. 5.3.2. The OLA architecture is typically the one displayed in Fig. 5.8 (p. 127).

6.3.9 Falcon-AO (China Southwest University)

Falcon-AO is a system for matching OWL ontologies. It is made of two components, namely those for performing linguistic and structure matching, see also Fig. 6.6.

LMO is a linguistic matcher. It associates with each ontology entity a bag of words which is built from the entity label, the entity annotations as well as the labels of connected entities. The similarity between entities is based on TFIDF (§4.2.1) [Qu *et al.*, 2006].

GMO is a bipartite graph matcher [Hu *et al.*, 2005]. It starts by considering the RDF representation of the ontologies as a bipartite graph which is represented by its adjacency matrix (A and A'). The distance between the ontologies is represented by a distance matrix (X) and the distance (or update) equations between two entities are simply a linear combination of all entities they are adjacent to, i.e., $X^{t+1} = AX^tA'^T + A^TX^tA'$. This process can be bootstrapped with an initial distance matrix. However, the real process is more complex than described here because it distinguishes between external and internal entities as well as between classes, relations and instances.

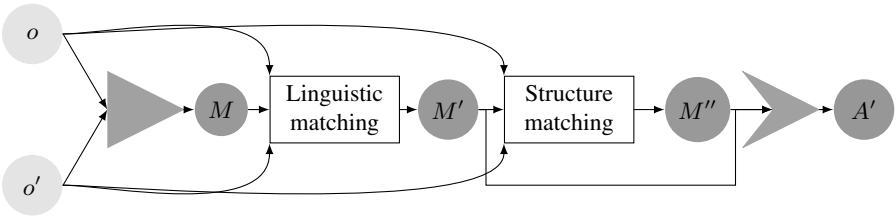

Fig. 6.6. Falcon-AO architecture: it is a sequential composition of two components, but if the output of the linguistic matcher is considered of sufficient quality, then no structure matching is performed.

First LMO is used for assessing the similarity between ontology entities on the basis of their name and text annotations. If the result has a high confidence, then it is directly returned for extracting an alignment. Otherwise, the result is used as input for the GMO matcher which tries to find an alignment on the basis of the relationships between entities [Jian *et al.*, 2005].

6.3.10 RiMOM (Tsinghua University)

The RiMOM (Risk Minimisation based Ontology Mapping) approach, being inspired by Bayesian decision theory, formalises ontology matching as a decision making problem [Tang *et al.*, 2006]. Given two ontologies, it aims at an optimal and automatic discovery of alignments which can be complex (such as including concatenation operators). The approach first searches for concept-to-concept correspondences and then for property-to-property correspondences. The RiMOM matching process is organised into the following phases [Li *et al.*, 2006]:

1. Select matchers to use. This task can be performed either automatically or manually. The basic idea of automatic strategy selection is if two ontologies have high

label similarity factor, then RiMOM will rely more on linguistic based strategies; while if the two ontologies have a high structure similarity factor, RiMOM will exploit similarity-propagation based strategies on them.
2. Execute multiple independent matchers, given the input ontologies and, optionally, user input. Examples of matchers include linguistic normalisation of labels, such as tokenisation, expansion of abbreviations and acronyms (§4.2.2) based on GATE tools[6], edit-distance, matchers that look for label similarity based on WordNet (§4.2.2), k-nearest neighbours statistical learning, naive Bayes matcher (§5.4.1), as well as some other heuristics for data type similarity and taxonomic structure similarity. This results in a cube of similarity values in $[0\ 1]$ for each pair of entities from the two ontologies (see also Sect. 6.1.12).
3. Combine the results by aggregating the values produced during the previous step into a single value. This is performed by using a linear-interpolation.
4. Similarity propagation. If the two ontologies have high structure similarity factor, RiMOM employs an algorithm called similarity propagation to refine the found alignments and to find new alignments that cannot be discovered using the other strategies. Similarity propagation makes use of structure information.
5. Extract alignment for a pair of ontologies based on thresholds (§5.7.1) and some refinement heuristics to eliminate unreasonable correspondences, e.g., use concept-to-concept correspondences to refine property-to-property correspondences.
6. Iterate the above described process by taking the output of one iteration as input into the next iteration until no new correspondences are produced. At each iteration, users can select matchers, and approve and discard correspondences from the returned alignment

RiMOM offers three possible structural propagation strategies: concept-to-concept propagation strategy (CCP), property-to-property propagation strategy (PPP), and concept-to-property propagation strategy (CPP). For choosing between them, RiMOM uses heuristic rules. For example, if the structure similarity factor is lower than some threshold then RiMOM does not use the CCP and PPP strategies, only CPP is used.

6.3.11 Corpus-based matching (University of Washington, Microsoft Research and University of Illinois)

Madhavan and colleagues [Madhavan *et al.*, 2005] proposed an approach to schema matching which, besides input information available from schemas under consideration, also exploits some domain specific knowledge via an external corpus of schemas and mappings. The approach is inspired from the use of corpus in information retrieval, where similarity between queries and concepts is determined based on analysing large corpora of text. In schema matching, such a corpus can be initialised with a small number of schemas obtained, for example, by using available

[6] http://gate.ac.uk/

standard schemas in the domain of interest, and should eventually evolve in time with new matching tasks.

Since the corpus is intended to have different representations of each concept in the domain, it should facilitate learning these variations in the elements and their properties. The corpus is exploited in two ways. First, to obtain an additional evidence about each element being matched by including evidence from similar elements in the corpus. Second, in the corpus, similar elements are clustered and some statistics for clusters are computed, such as neighbourhood and ordering of elements. These statistics are ultimately used to build constraints that facilitate selection of the correspondences in the resulting alignment.

The approach handles web forms and relational schemas and focuses on one-to-one alignments. It works in two phases. Firstly, schemas under consideration are matched against the corpus, thereby augmenting these with possible variations of their elements based on knowledge available from the corpus. Secondly, augmented schemas are matched against each other. In both cases the same set of matchers is applied. In particular, basic matchers, called learners, include (i) a *name learner*, (ii) a *text learner*, (iii) a *data instance learner*, and (iv) a *context learner*. These matchers mostly follow the ideas of techniques used in LSD (§6.2.4) and Cupid (§6.1.11). For example, the *name learner* exploits names of elements. It applies tokenisation and n-grams (§4.2.1) to the names in order to create training examples. The matcher itself is a text classifier, such as naive Bayes (§5.4.1). In addition, the name learner, uses edit distance (§4.2.1), in order to determine similarity between element names string. The *data instance learner* determines whether the values of instances share common patterns, same words, etc. A matcher, called *meta-learner*, combines the results produced by basic matchers. It uses *logistic regression* with the help of the stacking technique (§5.4.5) in order to learn its parameters. Finally, by using constraints based on the statistics obtained from the corpus, candidate correspondences are filtered in order to produce the final alignment.

6.4 Meta-matching systems

Meta-matching systems are systems whose originality is in the way they use and combine other matching systems rather than in the matchers themselves.

6.4.1 APFEL (University of Karlsruhe and University of Koblenz-Landau)

APFEL (Alignment Process Feature Estimation and Learning) is a machine learning approach that explores user validation of initial alignments for optimising automatically the configuration parameters of some of the matching strategies of the system, e.g., weights, thresholds, for the given matching task [Ehrig *et al.*, 2005]. It is a component of the FOAM framework (§8.2.5). The overall architecture of APFEL is given in Fig. 6.7.

APFEL parameterises the FOAM steps by using declarative representations of the (i) features engineered, Q_F; (ii) similarities assessed, Q_S; (iii) weight schemas,

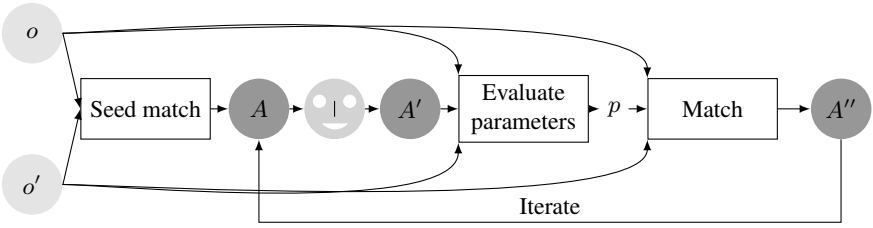

Fig. 6.7. APFEL architecture (adapted from [Ehrig, 2007]): it generates alignments and asks users for feedback. Then it adjusts methods and aggregation parameters in order to minimise the error and iterate, if necessary.

e.g., for similarity aggregation, Q_W; and (iv) thresholds, Q_T. For that purpose, the interfaces of matching systems are unified as Parameterisable Alignment Methods (PAM), which accept these parameters. First, given a matching system, for example QOM (§6.3.4) or Prompt (§6.1.9), a PAM is initialised with it, e.g., PAM(QOM). Then, once an initial alignment is obtained, this alignment is validated by users. Finally, by analysing the validated alignment and the above parameters, with the help of machine learning techniques (§5.4), e.g., decision tree learner, neural networks, support vector machines of the WEKA machine learning environment[7], a tuned weighting scheme and thresholds are produced for the given matching task. This process can be iterated.

6.4.2 eTuner (University of Illinois and The MITRE Corporation)

eTuner [Sayyadian *et al.*, 2005] is a system which, given a particular matching task, automatically tunes a schema matching system (computing one-to-one alignments). For that purpose, it chooses the most effective basic matchers, and the best parameters to be used, e.g., thresholds. eTuner models a matching system as a triple: $\langle L, G, K \rangle$, such that:

- L is a library of matching components, including basic matchers, e.g., edit distance, n-gram, combiners, e.g., modules taking average, minimum and maximum of the results produced by basic matchers, constraint enforcers, e.g., pre-defined domain constraints or heuristics which are computationally expensive to be used as basic matchers, and match selectors, e.g., modules applying thresholds for determining the final alignment.
- G is a directed graph which encodes the execution flow among the components of the given matching system.
- K is a set of *knobs* to be set (and named *knob configuration*). Matching components are viewed as black boxes which expose a set of adjustable knobs, such as thresholds, weights, or coefficients.

[7] http://www.cs.waikato.ac.nz/ml/weka/

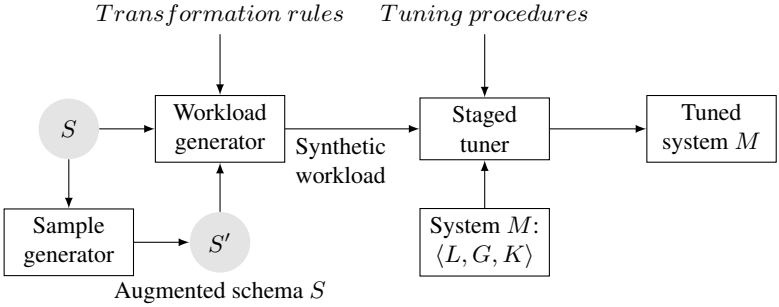

Fig. 6.8. eTuner architecture: eTuner generates a set of schemas to match with an initial schema. Then, it generates a plan for learning parameters. Finally, it tunes the method parameters and aggregation parameters.

The system works in two phases (see Fig. 6.8). During the first phase, in which the workload is synthesised with a known ground truth, given a single schema S, the system synthetises several schemas (S_1, S_2,...,S_n) out of S by altering it (for instance by modifying names of attributes, e.g., authors becomes aut). Thus, by taking a set of pairs $\{\langle S, S_1\rangle, \langle S, S_2\rangle, \ldots \langle S, S_n\rangle\}$ together with the reference correspondences available for free by construction of the synthetic schemas, the F-measure (§7.3) can be computed over any knob configuration. The second phase consists of searching the best parameters. Since the space of knob configurations can be large, the system uses a sequential, greedy approach, called *staged tuning*. In particular, by using the synthetic workload, it first tunes each of the basic matchers in isolation. Then, it tunes the combination of the basic matchers and the combiner, having the knobs of the basic matchers fixed, and so on and so forth. Once the entire system is tuned, it can be applied to match schema S with any subsequent schemas.

6.5 Summary

The panorama of systems considered in this chapter has multiplied the diversity of basic techniques of Chap. 4 by the variety of strategies for combining them introduced in Chap. 5. Moreover, usually each individual system innovates on a particular aspect. However, there are several constant features that are shared by the majority of systems. In summary, the following can be observed concerning the presented systems:

– Based on the number of systems considered in the various sections of this chapter, we can conclude that schema-based matching solutions have been so far investigated more intensively than the instance-based solutions. We believe that this is an objective trend, since we have striven to cover state of the art systems without bias towards any particular kind of solutions.

– Most of the systems under consideration focus on specific application domains, such as books and music, as well as on dealing with particular ontology types, such as DTDs, relational schemas and OWL ontologies. Only few systems aim at being general, i.e., suit various application domains, and generic, i.e., handle multiple types of ontologies. Some examples of the latter include Cupid (§6.1.11), COMA and COMA++ (§6.1.12), Similarity flooding (§6.1.13), and S-Match (§6.1.19).
– Most of the approaches take as input a pair of ontologies, while only few systems take as input multiple ontologies. Some examples of the latter include DCM (§6.1.25) and Wise-Integrator (§6.3.7).
– Most of the approaches handle only tree-like structures, while only few systems handle graphs. Some examples of the latter include Cupid (§6.1.11), COMA and COMA++ (§6.1.12), and OLA (§6.3.8).
– Most of the systems focus on discovery of one-to-one alignments, while only few systems have tried to address the problem of discovering more complex correspondences, such as one-to-many and many-to-many, e.g., iMAP (§6.2.6) and DCM (§6.1.25).
– Most of the systems focus on computing confidence measures in the $[0\ 1]$ range, most often standing for the fact that the equivalence relation holds between ontology entities. Only few systems compute logical relations between ontology entities, such as equivalence, subsumption. Some examples of the latter include CtxMatch (§6.1.18) and S-Match (§6.1.19).

Table 6.1 summarises how the matching systems cover the solution space in terms of the classifications of Chap. 3. For example, S-Match (§6.1.19) exploits string-based element-level matchers, external matchers based on WordNet, propositional satisfiability techniques, etc. OLA (§6.3.8), in turn, exploits, besides string-based element-level matchers, also a matcher based on WordNet, iterative fixed point computation, etc. Table 6.1 also testifies that ontology matching research so far was mainly focused on syntactic and external techniques. In fact, many systems rely on the same string-based techniques. Similar observation can be also made concerning the use of WordNet as an external resource of common knowledge. In turn, semantic techniques have rarely been exploited, this is only done by CtxMatch (§6.1.18), S-Match (§6.1.19) and OntoMerge (§6.1.17). Concerning instance-based system, techniques based on naive Bayes classifier and common value patterns are the most prominent.

Table 6.1. Basic matchers used by the different systems.

	Element-level		**Structure-level**	
	Syntactic	**External**	**Syntactic**	**Semantic**
DELTA §6.1.1	String-based	-	-	-
Hovy §6.1.2	String-based, Language-based	-	Taxonomic structure	-
TranScm	String-based	Built-in thesaurus	Taxonomic structure,	-

Table 6.1. Basic matchers used by the different systems (continued).

	Element-level Syntactic	External	Structure-level Syntactic	Semantic
§6.1.3			Matching of neighbourhood	
DIKE §6.1.4	String-based, Domain compatibility	WordNet	Matching of neighbourhood	-
SKAT §6.1.5	String-based	Auxiliary thesaurus, Corpus-based	Taxonomic structure, Matching of neighbourhood	-
Artemis §6.1.6	Domain compatibility, Language-based	Common thesaurus	Matching of neighbours via thesaurus, Clustering	-
H-Match §6.1.7	Domain compatibility, Language-based, Domains and ranges	Common thesaurus	Matching of neighbours via thesaurus, Relations	-
Tess §6.1.8	String-based, domain compatibility	-	Matching of neighbours	-
Anchor-Prompt §6.1.9	String-based, Domains and ranges	-	Bounded paths matching: (arbitrary links), Taxonomic structure	-
OntoBuilder §6.1.10	String-based, Language-based	Thesaurus look up	-	-
Cupid §6.1.11	String-based, Language-based, Datatypes, Key properties	Auxiliary thesauri	Tree matching weighted by leaves	-
COMA & COMA++ §6.1.12	String-based, Language-based, Datatypes	Auxiliary thesauri, Alignment reuse, Repository of structures	DAG (tree) matching with a bias towards various structures, e.g., leaves	-
Similarity flooding §6.1.13	String-based, Datatypes, Key properties	-	Iterative fixed point computation	-
XClust §6.1.14	Cardinality constraints	WordNet	Paths, Children, Leaves, Clustering	-
ToMAS §6.1.15	-	External alignments	Preserving consistency, Structure comparison	-
MapOnto §6.1.16	-	External alignments	Structure comparison	-
OntoMerge §6.1.17	-	External alignments	-	-
CtxMatch §6.1.18	String-based, Language-based	WordNet	-	Based on description logics
S-Match §6.1.19	String-based, Language-based	WordNet	-	Propositional SAT
HCONE §6.1.20	Language-based (LSI)	WordNet	-	-
MoA §6.1.21	Language-based	WordNet	-	-
ASCO §6.1.22	String-based, Language-based	WordNet	Iterative similarity propagation	-
BayesOWL §6.1.23	Text classifier	Google	Bayesian inference	-
OMEN §6.1.24	-	External alignment	Bayesian inference, Meta-rules	-
DCM §6.1.25	-	-	Correlation mining, Statistics	-

Table 6.1. Basic matchers used by the different systems (continued).

	Element-level Syntactic	External	Structure-level Syntactic	Semantic
T-tree §6.2.1	-	-	Correlation mining	-
CAIMAN §6.2.2	String-based (Rocchio classifier)	-	-	-
FCA-merge §6.2.3	-	-	Formal concept analysis	-
LSD/GLUE/ iMAP §6.2.4-6.2.6	WHIRL, Naive Bayes	Domain constraints	Hierarchical structure	-
Automatch §6.2.7	Naive Bayes	-	Internal structure, Statistics	-
SBI&NB §6.2.8	Statistics, Naive Bayes	-	Pachinko Machine naive Bayes	-
Kang & Naughton §6.2.9	Information entropy	-	Mutual information, Dependency graph matching	-
Dumas §6.2.10	String-based WHIRL	-	Instance identification	-
Wang & al. §6.2.11	Language-based	-	Mutual information	-
sPLMap §6.2.12	Naive Bayes, kNN classifier, String-based	-	-	-
SEMINT §6.3.1	Neural network, Datatypes, Value patterns	-	-	-
Clio §6.3.2	String-based, Language-based, Naive Bayes	-	Structure comparison	-
IF-Map §6.3.3	String-based	-	Formal concept analysis	-
NOM & QOM §6.3.4	String-based, Domains and ranges	Application-specific vocabulary	Matching of neighbours, Taxonomic structure	-
oMap §6.3.5	Naive Bayes, String-based	-	Similarity propagation	-
Xu & al. §6.3.6	String-based, Language-based	WordNet, Domain ontology	Decision trees	-
Wise-Integrator §6.3.7	String-based, Language-based, Datatypes, Value patterns	WordNet	Clustering	-
OLA §6.3.8	String-based, Language-based, Datatypes	WordNet	Iterative fixed point computation, Matching of neighbours, Taxonomic structure	-
Falcon-AO §6.3.9	String-based	WordNet	Structural affinity	-
RiMOM §6.3.10	String-based, Naive Bayes	WordNet	Taxonomic structure, Similarity propagation	-
Corpus-based matching §6.3.11	String-based, Language-based, Naive Bayes, Value patterns	Corpus schemas, Domain constraints	-	-

Table 6.2 summarises the position of these systems with regard to some of the requirements of Sect. 1.7 (namely those requirements that can be given in the specification of the system rather than being measured). In Table 6.2, the *Input* column stands for the input taken by the systems. In particular, it mentions the languages that the systems are able to handle (if this information was not available form the articles describing the corresponding systems we used general terms, such as database schema and ontology instead). This is, of course, very important for someone who has a certain type of ontology to match and is looking for a system. The *Needs* column stands for the resources that must be available for the system to work. This covers the automatic aspect of Sect. 1.7, which is here denoted as *user* when user feedback is required, *semi* when the system can take advantage of user feedback but can operate without it and *auto* when the system works without user intervention (of course, users can influence the system by providing the initial input or evaluating the results afterwards, but this is not taken into account here). Similarly, the *instances* value specifies that the system requires data instances to work. In addition, some systems may require *training* before the actual matching as well as *alignment* to be improved. The *Output* column denotes the form of the results given by the system: *Alignment* means that the system returns a set of correspondences, *merge* that it merges the input ontologies or schemas, *axioms* or *rules* that it provides rules for querying or completing the ontologies, etc.

Table 6.2. Position of the presented systems with regard to the requirements of Chap. 1.

System	Input	Needs	Output	Operation
DELTA §6.1.1	Relational schema, EER	User	Alignment	-
Hovy §6.1.2	Ontology	Semi	Alignment	-
TranScm §6.1.3	SGML, OO	Semi	Translator	Data translation
DIKE §6.1.4	ER	Semi	Merge	Query mediation
SKAT §6.1.5	RDF	Semi	Bridge rules	Data translation
Artemis §6.1.6	Relational schema, OO, ER	Auto	Views	Query mediation
H-Match §6.1.7	OWL	Auto	Alignment	P2P query mediation
Tess §6.1.8	Database schema	Auto	Rules	Version matching
Anchor-Prompt §6.1.9	OWL, RDF	User	Axioms (OWL/RDF)	Ontology merging
OntoBuilder §6.1.10	Web form, XML schema	User	Mediator	Query mediation
Cupid §6.1.11	XML schema, Relational schema	Auto	Alignment	-
COMA & COMA++ §6.1.12	Relational schema, XML schema, OWL	User	Alignment	Data translation

Table 6.2. Position of these systems with regard to the requirements of Chap. 1 (continued).

System	Input	Needs	Output	Operation
Similarity flooding §6.1.13	XML schema, Relational schema	User	Alignment	-
XClust §6.1.14	DTD	Auto	Alignment	-
ToMAS §6.1.15	Relational schema, XML schema	Query, Alignment	Query, Alignment	Data transformation
MapOnto §6.1.16	Relational schema, XML schema, OWL	Alignment	Rules	Data translation
OntoMerge §6.1.17	OWL	Alignment	Ontology	Ontology merging
CtxMatch/CtxMatch2 §6.1.18	Classification, OWL	User	Alignment	-
S-Match §6.1.19	Classification, XML schema, OWL	Auto	Alignment	-
HCONE §6.1.20	OWL	Auto, Semi, User	Ontology	Ontology merging
MoA §6.1.21	OWL	Auto	Axioms, OWL	-
ASCO §6.1.22	RDFS, OWL	Auto	Alignment	-
BayesOWL §6.1.23	Classification, OWL	Auto	Alignment	-
OMEN §6.1.24	OWL	Auto, Alignment	Alignment	-
DCM §6.1.25	Web form	Auto	Alignment	Data integration
T-tree §6.2.1	Ontology	Auto, Instances	Alignment	-
CAIMAN §6.2.2	Classification	Semi, Instances, Training	Alignment	-
FCA-merge §6.2.3	Ontology	User, Instances	Ontology	Ontology merging
LSD/GLUE §6.2.4,§6.2.5	Relational schema, XML schema, Taxonomy	Auto, Instances, Training	Alignment	-
iMAP §6.2.6	Relational schema	Auto, Instances, Training	Alignment	-
Automatch §6.2.7	Relational schema	Auto, Instances, Training	Alignment	-
SBI&NB §6.2.8	Classification	Auto, Instances, Training	Alignment	-
Kang & Naughton §6.2.9	Relational schema	Instances	Alignment	-
Dumas §6.2.10	Relational schema	Instances	Alignment	-
Wang & al. §6.2.11	Web form	Instances	Alignment	Data integration
sPLMap §6.2.12	Database schema	Auto, Instances, Training	Rules	Data translation
SEMINT §6.3.1	Relational schema	Auto, Instances (opt.), Training	Alignment	-
Clio	Relational schema,	Semi,	Query	Data

Table 6.2. Position of these systems with regard to the requirements of Chap. 1 (continued).

System	Input	Needs	Output	Operation
§6.3.2	XML schema	Instances (opt.)	transformation	translation
IF-Map §6.3.3	KIF, RDF	Auto, Instances, Common reference	Alignment	-
NOM & QOM §6.3.4	RDF, OWL	Auto, Instances (opt.)	Alignment	-
oMap §6.3.5	OWL	Auto, Instances (opt.), Training	Alignment	Query answering
Xu & al. §6.3.6	XML schema, Taxonomy	Auto, Instances (opt.), Training	Alignment	-
Wise-Integrator §6.3.7	Web form	Auto	Mediator	Data integration
OLA §6.3.8	RDF, OWL	Auto, Instances (opt.)	Alignment	-
Falcon-AO §6.3.9	RDF, OWL	Auto Instances (opt.)	Alignment	-
RiMOM §6.3.10	OWL	Auto Instances (opt.)	Alignment	-
Corpus-based matching §6.3.11	Relational schema, Web form	Text corpora, Instances, Training	Alignment	-
APFEL §6.4.1	RDF, OWL	User	Alignment	-
eTuner §6.4.2	Relational schema, Taxonomy	Auto	Alignment	-

The *Output* delivered by a system is very important because it shows the capability of the system to be used for some applications, e.g., a system delivering views and data translators cannot be used for merging ontologies as is. It is remarkable that many systems deliver alignments. As such, they are not fully committed to any kind of operation to be performed and can be used in a variety of applications. This could be viewed as a sign of possible interoperability between systems. However, due to lack of a common alignment format, each system uses its own way to deliver alignments (as lists of URIs, tables, etc.). Finally, the *Operation* column describes the ways in which a system can process alignments.

Not all the requirements are addressed in Table 6.2. Indeed, completeness, correctness, run time should not be judged from the claims of system developers. No meaningful system can be proved to be complete, correct or as fast as possible in a task like ontology matching. Therefore, the degree of fulfillment of these requirements must be evaluated and compared across systems. Moreover, different applications have different priorities regarding these requirements, hence, they may need different systems. Thus, this evaluation depends on an application in which the system is to be used.

It is difficult to evaluate and compare systems without commonly agreed testbenchs, principles and available implementations. The next chapter presents methods for empirical evaluation and comparison of matching systems.