2.1. Introduction

The main aim of this chapter is to describe the theoretical foundations and relevant background in the domain of ontology mediation. It also brings out the different definitions of ontologies with a justification of the need for ontologies. Finally an overview of ontology matching systems is also presented.

When people or machines have to communicate between themselves, they need a shared understanding of the same concepts. An ontology can be used to solve this problem. Gruber [27] defined an ontology as:

An ontology is a formal, explicit specification of a shared conceptualization.

Due to an increased awareness of potential ontology applications in industry, public administration and academic circles, a growing number of ontologies is created by different organizations and individuals. Although these ontologies are developed for various application purposes and areas, they often contain overlapping information, but these different ontologies cannot easily be used together in a new application.

Furthermore, ontology users or engineers do not only use their own ontologies, but also want to integrate or adapt other ontologies, or even apply multiple ontologies to solve a problem. In this context, it is necessary to find ways to integrate various ontologies and enable cooperation between them.

2.2. Ontologies

In this section, various definitions of ontologies are reviewed and the relationships between them are also discussed. Some definitions are independent of the processes followed to build the ontology and its use in applications, while other definitions are influenced by its development process.
The word ontology was taken from Philosophy; where it means a systematic explanation of being. Guarino and Giaretta [28] propose to use the words ‘Ontology’ (with capital ‘O’) and ‘ontology’ to refer to the philosophical and Knowledge Engineering senses respectively.

Neches and colleagues [29], define an ontology as follows:

An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.

This definition identifies basic terms and relations between terms, identifies rules to combine terms, and provides the definitions of such terms and relations. An ontology includes not only the terms that are explicitly defined in it, but also the knowledge that can be inferred from it.

As stated earlier, Gruber [27] defined ontology as follows:

An ontology is an explicit specification of a conceptualization.

This definition became the most quoted in the literature and by the ontology community. Based on Gruber’s definition, many definitions of what an ontology were proposed.

Borst [30] modified slightly Gruber’s definition as follows:

Ontologies are defined as a formal specification of a shared conceptualization.

Gruber’s and Borst’s definitions have been merged and explained by Studer and colleagues [31] as follows:

An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by group.

In 1995, Guarino and Giaretta [28] collected and analyzed the following seven definitions:
(i) Ontology as a philosophical discipline,

(ii) Ontology as an informal conceptual system,

(iii) Ontology as a formal semantic account,

(iv) Ontology as a specification of a conceptualization,

(v) Ontology as a representation of a conceptual system via a logical theory:
    
    a) Characterized by specific formal properties,
    
    b) Characterized only by its specific purposes.

(vi) Ontology as the vocabulary used by a logical theory,

(vii) Ontology as a (meta-level) specification of a logical theory.

Based on the seven definitions, Guarino and Giaretta proposed to consider an ontology as:

*A logical theory which gives an explicit, partial account of a conceptualization.*

In the above definition, conceptualization is basically the idea of the world that a person or a group of people can have. Guarino and Giaretta [28] formalized the notion of conceptualization and established how to build the ontology by making a logical theory. Hence, this definition would be only applicable to ontologies developed in logic. Guarino and Giaretta’s work has been further refined, and they provided the following definition:

*A set of logical axioms designed to account for the intended meaning of a vocabulary.*

There is another group of definitions based on the process followed to build the ontology. These definitions also include some highlights about the relationship between ontologies and knowledge bases. For example, the definition given by Bernaras and colleagues [32] in the framework of the KACTUS [33] project is:

*An ontology provides the means for describing explicitly the conceptualization behind the knowledge represented in a knowledge base.*

This definition proposes “extracting” the ontology from a knowledge base, which reflects the approach used to build ontologies. In this approach, the ontology is built, following a
bottom-up strategy, on the basis of an application knowledge base by means of an abstraction process. As more applications are built, the ontology becomes more general and therefore, it moves further away from what would be a knowledge base.

Another strategy for building ontologies is to reuse large ontologies like SENSUS [34] to create domain specific ontologies and knowledge bases. Based on this strategy the ontology is defined as follows:

*An ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base.*

According to this definition, the same ontology can be used for building several knowledge bases, which would share the same skeleton or taxonomy. Extensions of the skeleton should be possible at the low level by adding domain-specific sub concepts or at the high level by adding intermediate or upper level concepts that cover new areas. If systems are built with the same ontology, they share a common underlying structure, therefore, merging and sharing their knowledge bases and inference mechanisms will become easier.

The ontology community distinguishes ontologies that are mainly taxonomies from ontologies that model the domain in a deeper way and provide more restrictions on domain semantics. The community calls them lightweight and heavyweight ontologies respectively. Lightweight ontologies include concepts, concept taxonomies, relationships between concepts, and properties that describe concepts. The heavyweight ontologies add axioms and constraints to lightweight ontologies. Axioms and constraints clarify the intended meaning of the terms gathered on the ontology.

Since ontologies are widely used for different purposes (NLP (Natural Language Processing), Knowledge Management, e-commerce, Intelligent Integration of Information, and Semantic Web) in different communities (Knowledge Engineering, Databases and Software Engineering), Uschold and Jasper [35] provided a new definition of the word ontology to popularize it in other disciplines.

Uschold and Jasper [35] defined an ontology as:

*An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication*
of how concepts are inter-related which collectively improve a structure on the domain and constrain the possible interpretations of terms.

A more mathematical definition can be the following by Amann and Fundulaki [36]. An ontology is a triple \( O = (C, S, isa) \) where:

(i) \( C = \{c_1, c_2, \ldots, c_m\} \) is a set of classes, where each class \( c_i \) refers to a set of real world objects (class instances),

(ii) \( S = \{s_1, s_2, \ldots, s_n\} \) is a set of slots, where each slot \( s_i \) could refer to:

a) A property of a class, i.e. a value of a simple type such as Integer, String or Date,

b) A binary typed role, i.e. the representation of a relation between classes.

(iii) \( isa = \{isa_1, isa_2, \ldots, isa_p\} \) is a set of inheritance relationships defined between classes. Inheritance relationships carry subset semantics and define a partial order over classes, organizing classes into one or more tree structures.

In order to accommodate the individual instances, this definition can be extended with a fourth element \( I = \{i_1, i_2, \ldots, i_q\} \), where each \( i_w \) is an instance of some class \( c_x \in C \). The instance includes a concrete value for every slot \( s_y \) associated with \( c_x \) or its ancestors as defined by the \( isa \) set.

Thus, ontologies aim to capture consensual knowledge in a generic way, and that they may be reused and shared across software applications and by groups of people. They are usually built cooperatively by different groups of people in different locations.

### 2.3. Why Ontologies?

Ontologies provide a number of useful features for intelligent systems, as well as for knowledge representation in general and for the knowledge engineering process. This subsection summarizes the most important of these features.
2.3.1. Vocabulary

An ontology provides a *vocabulary* for referring to the terms in a subject area. A *controlled vocabulary* provides a finite list of terms together with an unambiguous interpretation of those terms. Every use of a term from a controlled vocabulary will denote exactly the same identifier. A *glossary* provides a list of terms and their meanings, but the meanings are specified in natural language and may often be interpreted differently by different people. Hence they are not unambiguous and thus are not suitable for machine processing. A *thesaurus* provides some additional semantics in the form of synonym relationships between terms, which drastically reduces ambiguity. However, thesauri do not provide explicit term hierarchies [37].

Ontologies are different from such human-oriented vocabularies in that they provide *logical statements* that describe what the terms are, how they are related to each other, and how they can be related to each other. They also specify *rules* for combining the terms and their relations to define extensions to the vocabulary. As Chandrasekaran and colleagues [38] note carefully, it is not the vocabulary as such that qualifies as an ontology, but the conceptualizations that the terms in the vocabulary are intended to capture. An ontology specifies terms with *unambiguous meanings*, with semantics independent of reader and context. Translating the terms in an ontology from one language to another does not change the ontology conceptually. Thus an ontology provides a vocabulary, and a *machine processable common understanding* of the topics that the terms denote. The *meanings* of the terms in an ontology can be communicated between users and applications.

2.3.2. Taxonomy

A *taxonomy* (*concept hierarchy*) is a hierarchical categorization of entities within a domain. It is also a clustering of entities based on common ontological characteristics. The categorization is organized according to a predetermined system. A good taxonomy should separate its corresponding entities into mutually exclusive, unambiguous groups and subgroups that, taken together, include all possibilities. It should also be simple, easy to remember, and easy to use. A good example of the taxonomy is a web site’s taxonomy; it is the way the site organizes its data into categories and subcategories, displayed in a site map.
Every ontology provides the taxonomy in a machine-readable and machine-processable form. However, an ontology is more than its corresponding taxonomy, it is a full specification of a domain. The vocabulary and the taxonomy of an ontology together provide a conceptual framework for discussion, analysis, or information retrieval in a domain.

2.3.3. Content Theory

Since ontologies identify classes of objects, their relations, and concept hierarchies that exist in some domain, they are essentially content theories [38]. Ontologies not only identify those classes, relations, and taxonomies, but also specify them in an elaborate way, using specific ontology representation languages. Classes are specified using frame-based representation principles, i.e., their properties, property values, and possible value restrictions are specified as well. In some ontology representation languages, the value of one property may be expressed as a mathematical equation using values of other properties. Also, some languages allow developers to specify first-order-logic constraints between terms and more detailed relationships such as disjoint classes, disjoint coverings, inverse relationships, and part–whole relationships, etc., [37]. Thus ontologies represent knowledge in a very structured way. Well-structured and well-developed ontologies enable various kinds of consistency checking from applications. They also enable interoperability between different applications.

Being content theories, ontologies clarify the structure of domain knowledge. Developing an ontology requires an effective ontological analysis of the domain whose content the ontology is intended to represent. Ontological analysis reveals the concepts of the domain knowledge, their taxonomies, and the underlying organization. Without such analysis, no knowledge representation for the domain can be well founded. Through ontological analysis, the entire process of knowledge engineering acquires a strong flavor of modeling. The resulting knowledge base does not merely transfer the knowledge extracted from a human expert, but also models the problem domain in the form of the observed behavior of an intelligent agent embedded in its environment [27, 39-41].
2.3.4. Knowledge Sharing and Reuse

The major purpose of ontologies is for knowledge sharing and knowledge reuse by applications. Every ontology provides a description of the concepts and relationships that can exist in a domain and that can be shared and reused among intelligent agents and applications. Moreover, working agents and applications should be able to communicate such ontological knowledge. Shared ontologies allow to build specific knowledge bases that describe specific situations but clearly rely on the same underlying knowledge structure and organization.

According to Neches and colleagues [29], there are many senses in which the work that went into creating a knowledge-based system can be shared and reused; for example:

(i) through the inclusion of source specifications - the content of one module is copied into another one at design time, is then possibly extended and revised, and is finally compiled into a new component,

(ii) through the runtime invocation of external modules or services - one module invokes another, either as a method from a class library or through a Web service, and the like,

(iii) through communication between agents - the messages that intelligent agents send to and receive from each other can have various kinds of knowledge as their content,

(iv) through the exchange of techniques - sharing and reusing not the content, but the approach behind it (in a manner that facilitates reimplementation of the content itself).

These modes of sharing and reuse require shared understanding of the intended interpretations of domain terms, compatibility of the domain models used by different agents and applications, and compliance with the kinds of requests that the external modules/services are prepared to accept.

An ontology captures the intrinsic conceptual structure of the domain [38], and can be used as a basis for developing a rich domain specific knowledge representation language for building knowledge bases in that domain.
In practice, knowledge sharing and reuse is still not easy, even if an ontology is readily available for a given purpose, since there are several different languages for representing ontologies, and knowledge base development tools may not support the language used to develop the ontology. There are also competing approaches and working groups, creating different technologies, traditions, and cultures. There may be several different ontologies that have been developed to describe the same topic or domain. Selecting any one of them may not satisfy all the requirements that the knowledge engineer must fulfill. Combining them may be a solution but the resulting ontology may still be inadequate. On top of all that, there is the problem of knowledge maintenance, since all parts of knowledge evolve over time.

2.4. Ontology Mediation

On the Semantic Web, data is envisioned to be annotated using ontologies. Ontologies convey background information which enriches the description of the data and which makes the context of the information more explicit. Because ontologies are shared specifications, the same ontologies can be used for the annotation of multiple data sources, not only web pages, but also collections of eXtensible Markup Language documents, relational databases, etc. The use of such shared terminologies enables a certain degree of inter-operation between these data sources. This, however, does not solve the integration problem completely, because it cannot be expected that all individuals and organizations on the Semantic Web will ever agree on using one common terminology or ontology. It can be expected that many different ontologies will appear and, in order to enable inter-operation, differences between these ontologies have to be reconciled. The reconciliation of these differences is called ontology mediation.

Ontology mediation enables reuse of data across applications on the Semantic Web and, in general, cooperation between different organizations. In the context of semantic knowledge management, ontology mediation is especially important to enable sharing of data between heterogeneous knowledge bases and to allow applications to reuse data from different knowledge bases. Another important application area for ontology mediation is Semantic Web Services. In general, it cannot be assumed that the requester and the provider of a service use the same terminology in their communication and thus mediation is required in order to enable communication between heterogeneous business partners.
There are two principle kinds of ontology mediation: ontology mapping and ontology merging. With ontology mapping, the correspondences between two ontologies are stored separately from the ontologies and thus are not part of the ontologies themselves. The correspondences can be used for, querying heterogeneous knowledge bases using a common interface or transforming data between different representations. The semi-automated discovery of such correspondences is called ontology alignment.

When performing ontology merging, a new ontology is created which is the union of the source ontologies. The merged ontology captures all the knowledge from the original ontologies. The challenge in ontology merging is to ensure that all correspondences and differences between the ontologies are reflected in the merged ontology.

Summarizing, ontology mapping is mostly concerned with the representation of correspondences between ontologies. Ontology alignment is concerned with the discovery of these correspondences and ontology merging is concerned with creating the union of ontologies, based on correspondences between the ontologies.

2.4.1. Ontology Matching

Ontologies are considered as an important contribution used for solving the data heterogeneity problem. However, ontologies themselves can also be heterogeneous [42], e.g., the ontology can be expressed in different languages, e.g., Web Ontology Language, Resource Description Framework Schema, OKBC (Open Knowledge Base Connectivity), KIF (Knowledge Interchange Format), etc. Different languages use different syntax, different logical representation, different semantics of primitives, and language expressivity. Using the same ontology language does not solve heterogeneity problems. An ontology on motor-vehicles, for example, may include the concept ”motor-bike”, whereas another ontology on the same subject may ignore it.

Klein [43] categorized possible mismatches of the ontologies heterogeneity by two levels:

(i) Language level:

a) Syntax: Different languages have different syntax. For example, in Resource Description Framework Schema the definition of class Chair is <
rdfs: ClassID = "Chair". In LOOM, the definition of class Chair is (def concept Chair).

b) Logical Representation: Differences in logical representation occur when the statements are syntactically different, but logically equivalent. For example, the way present disjointness in Web Ontology Language Lite is Class (owl: Nothing complete A B), but also valid in Web Ontology Language DL as DisjointClasses (A B).

c) Semantics of Primitives: The same syntactically construct can have different meanings (semantics) in different language. For example, there are several interpretations of A equal to B.

d) Language Expressivity: Some language can express things that the other language cannot. For example, negation can be expressed in one language but not in another.

(ii) Ontology level:

a) Conceptualization: Using the same linguistic terms describe different concepts (e.g., the concept "employee" can have different meaning in the ontologies). Using different modeling conventions and levels of granularity describe concepts (e.g., one ontology model "car" but not "truck", the other ontology model "car" that includes "truck".).

b) Terminological: Using different terms to describe the same concepts (e.g., one ontology uses concept "car", while the other ontology use "automobile"), homonym term (e.g., "conductor" has a different meaning in music than in electrical engineering).

c) Style of Modeling: Using different modeling paradigms to present concepts (e.g., one model uses temporal presentations based on interval logic while another uses a representation based on points).

d) Encoding: Values in the ontologies can be encoded in different formats (e.g., a date represented as dd/mm/yyyy, mm-dd-yyyy and etc.).

As the translation between ontology languages can be considered as an independent issue [44], it is recommendable to translate different ontologies into the same language before
comparing them on ontology level. Currently, most ontology matching systems are focusing on the ontology level.

2.4.2. Terminology

The terms mapping, matching and alignment are frequently used in the research about combining ontologies. Keet [45] summarizes different definitions about these terms. Based on recent studies about combining ontologies, the terms used are defined as follows [45, 46]:

**Ontology Merging:** Combine two ontologies from the same subject area into a new ontology.

**Ontology Integration:** Combine two ontologies from different subject areas into a new ontology.

**Ontology Alignment:** Identify correspondences between the source ontologies.

**Ontology Mapping:** Find equal parts in different source ontologies.

**Ontology Matching:** Find similar parts in the source ontologies or finding translation rules between ontologies.

2.4.3. Usage Categories of Ontology Matching and Applications

Choi [47] categories the usage of ontology matching into three categories:

(i) *Matching between an integrated global ontology and local ontologies:*

In this category, the global ontology provides a shared vocabulary, the matching maps a concept found in one ontology into a view, or query over other ontologies. The main advantage is that it is easier to define mapping and find mapping rules because of the shared vocabulary in the global ontology. However, in this matching, the global ontology is needed which is very difficult to maintain in a highly dynamic environment. The traditional applications (e.g., information integration or schema integration) require determining ontology matching before running the system. For information integration systems, ontology matching
interprets the relationship between the mediated schemas and local schemas [48, 49].

(ii) Matching between local ontologies:

In this category, the matching is transforming the source ontology entities into the target ontology entities based on semantic relations. This approach is more relevant for highly dynamic environments. It is not easier to find mapping between, global ontology and local ontology matching, because of the lack of common vocabularies. The dynamic applications (e.g., agents, peer-to-peer, web services) require running time ontology matching. For example, [50] allows agents in peer-to-peer networks to communicate to other agents based on dynamic ontology mapping. [21] and [51] are web services application examples, where ontology matching is used for finding new services to complete a request.

(iii) Matching in ontology merging:

In this category, matching is used to try find similarities and conflicts entities between the ontologies to be merged. Managing and maintaining the different versions of ontologies can also be the applications of ontology matching. Some application examples in these categories are Prompt [52] suit for Protégé editor and Chimera [53] tool for Ontolingua.

2.4.4. Ontology Matching Input and Output

Shvaiko and Euzenat [8] define the matching process has five parameters as in Figure 2.1.

![Figure 2.1. Matching Process](image_url)
The processing is generating a new alignment $A'$ from the input consisting of the ontology $O$ and $O'$. However, other parameters can be involved in the matching process, like the use of input alignment $A$, which is to be completed by the processing, the matching parameters $p$ (e.g., weight, thresholds), and the resources $r$ (e.g., lexicons).

Currently, the matching systems consider the matching of ontologies expressed in the same languages. Different elements of ontologies as input will be analysed in the different approaches, for example, GLUE uses taxonomies and instances [54], OLA uses many elements (e.g., classes, properties, constraints, taxonomy, instances) [55], ASCO uses as much available information in the ontologies as possible [56] (e.g., concepts, relations, structure, even apply TF-IDF (Term Frequencies – Inverse Document Frequencies) to calculate similarity value between descriptions of the concepts or relations). Some systems are schema-based which can be viewed as a special ontology restrained relationship. In these cases, the input is a data model. For example, MOMIS (Mediator Environment for Multiple Information Sources) [57] uses local schemas.

Shvaiko and Euzenat [8] separate different output types of ontology matching:

(i) one-to-one or one-to-many correspondence between ontology entities,

(ii) expression of correspondence between ontology entities can be in the range 0 to 1,

(iii) the relationship between entities in most systems is expressed as equivalence (=). Some systems (e.g., [58]) can provide more expressive result, like equivalence (=), more general ($\supseteq$), less general ($\subseteq$), disjointness ($\perp$)), while the special idk (I do not know) expresses none of the relations.

2.5. Ontology Matching Systems Overview

This section illustrates several ontology matching systems or tools. There are some surveys or comparisons about ontology matching systems in [1, 5, 6, 44, 47, 59, 60]. The systems or tools based on the ontology strategies are discussed for comparison along with their input and output. Finally, the summary of ontology matching systems is presented in table 2.1.
2.5.1. Schema-based Ontology Matching Systems

Schema-based systems are those which rely mostly on schema-level input information for performing ontology matching.

**DELTA (The MITRE Corporation)**

DELTA (Data Element Tool-based Analysis) is a system that semi-automatically discovers attribute correspondences among database schemas [61]. It handles relational and EER (Extended Entity–Relationship) 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, into a simple text string, called document. The documents describing each database attribute constitute a document base. Then, Data Element Tool-based Analysis feeds the document base from the first schema into a full-text information retrieval tool.

Matching is viewed as a 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 between a search pattern and contents of a document base using natural language heuristics. 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.

**Hovy (University of Southern California)**

Hovy [62] describes heuristics used to match large-scale ontologies, such as SENSUS, in order to combine them into single reference ontology. In particular, three types of matchers were used based on:

(i) concept names,

(ii) concept definitions,

(iii) taxonomy structure.

For example, the name matcher splits composite-word names into separate words 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 Information Technology and Technology results in 110 points, while the comparison between Shell and Shell results in 45 points. The definition matcher compares the English definitions of two concepts. 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 to 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.

**TransScm (Tel Aviv University)**

TransScm [63] 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 labeled 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.

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.
DIKE (Universita di Reggio Calabria and Universita di Calabria)

DIKE (Database Intensional Knowledge Extractor) [64] is a system supporting the semi-automatic construction of CIS (Cooperative Information Systems) from heterogeneous databases. It takes as input a set of databases belonging to the Cooperative Information System. It builds a kind of mediated schema in order to provide a user-friendly integrated access to the available data sources. Database Intensional Knowledge Extractor 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 Database Intensional Knowledge Extractor 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.

Each kind of property is associated with 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. Database Intensional Knowledge Extractor works by computing sequentially the above mentioned properties.

SKAT and ONION (Stanford University)

SKAT (Semantic Knowledge Articulation Tool) [65] is a rule-based system that semi-automatically discovers mappings between two ontologies. 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 ≡ Chancellor, indicates that 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, 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.
ONION (ONtology compositION) [66] is a successor system to Semantic Knowledge Articulation Tool that semi-automatically discovers mappings between multiple ontologies, in order to enable a unified query answering over those ontologies. Input ontologies, Resource Description Framework 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 [67] forms the basis of the articulation generator (ArtGen), for the ONtology compositION system. ArtGen, in turn, can be viewed as an evolution of the Semantic Knowledge Articulation Tool 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 and then structure-based matching. 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.

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. The structural matcher tries to match only the unmatched pairs from the linguistic matching, thereby complementing its results.

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

Artemis (Analysis of Requirements: Tool Environment for Multiple Information Systems) [68] was designed as a module of the Mediator Environment for Multiple Information Sources mediator system [69, 70] 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. It represents a set of intentional
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 categories 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.

**H-Match (Università degli Studi di Milano)**

H-Match [71] is an automated ontology matching system. The system handles ontologies specified in Web Ontology Language. Internally, these are encoded as graphs using the H-model representation [72]. 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, data type compatibility, 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 Analysis of Requirements: Tools Environment for Multiple Information Systems [68].

In particular, it extends the Analysis of Requirements: Tools Environment for Multiple Information Systems approach by:

(i) building a common thesaurus involving relations among WordNet synsets such as meronymy and coordinate terms,

(ii) 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.

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. 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.

**Tess (University of Massachusetts)**

Tess [73] (Type Evolution Software System) is a system to support schema evolution by matching the old and the new versions. Schemas are viewed as collections of types. It is assumed that input schemas are highly similar. Matching is viewed as generation of *derivation rules* to be applied to data. Type Evolution Software System 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. 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. 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, Type Evolution Software System 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, the derivation function simply copies existing objects. A more complex transformation may include a join operation to combine two related objects into one.

**Anchor-Prompt (Stanford Medical Informatics)**

Anchor-Prompt [74] 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 [75]. Prompt handles ontologies expressed in such knowledge representation formalisms as Web Ontology Language and Resource Description Framework 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, 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. 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 [76, 77].

**OntoBuilder (Technion Israel Institute of Technology)**

OntoBuilder [78] is a system for information seeking on the web. OntoBuilder operates in two phases:

(i) ontology creation (the *training* phase),

(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. The adaptation phase includes running time match and interactive merge operations of the related ontologies with the actual (initial) ontology. During the adaptation phase users suggest the web sites they would like to further explore. 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.

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, substring matching. If all else fail, thesaurus look-up is performed. Finally, mismatched terms are presented to users for manual matching.

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

Cupid [79] 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 as follows:

(i) linguistic matching,

(ii) structural matching,

(iii) mapping elements generation.

The matching algorithm operates only with tree-structures, to which non tree cases are reduced.

The linguistic matching phase computes linguistic similarity coefficients between schema element names (labels) based on morphological normalization, categorization, string-based techniques, such as common prefix, suffix tests, and thesauri look-up. The structural matching phase computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur. The mapping elements generation phase aggregates the results of the linguistic and structural matching through a weighted sum and generates a final alignment by choosing pairs of schema elements with weighted similarity coefficients, which are higher than a threshold.
COMA and COMA++ (University of Leipzig)

COMA (COmbination of MAtching Algorithms) [80] 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. COmbination of Matching Algorithms contains six elementary matchers, five hybrid matchers, and one reuse-oriented matcher. Most of them implement string-based techniques, such as affix, n-gram, edit distance; 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 Combination of Matching Algorithms 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. COmbination of MAtching Algorithms ++ is built on top of COmbination of MAtching Algorithms by elaborating in more detail the alignment reuse operation. Also it provides a more efficient implementation of the COmbination of MAtching Algorithms algorithms and a graphical user interface.

Similarity flooding (Stanford University and University of Leipzig)

The Similarity flooding approach [81] is based on the ideas of similarity propagation. Schemas are presented as directed labelled graphs grounded on the OIM (Ontology Identification Model) specification. 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, 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 result of this step is a refined alignment which is further filtered to produce the final alignment.

**XClust (National University of Singapore)**

XClust is a tool for integrating multiple DTDs (Document Type Definition) [82]. Its integration strategy is based on clustering. Given multiple Document Type Definitions, it clusters them according to their similarity. This aims at facilitating the work of system integrators by allowing them to focus on already similar Document Type Definitions of single clusters. Clustering is applied recursively until a manageable number of Document Type Definitions is obtained.

XClust works in two phases:

1. Document Type Definition similarity computation,
2. Document Type Definition clustering.

During the first phase, given a set of Document Type Definitions, pair wise similarities between their underlying labelled trees are computed. Trees are computed 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 and a cardinality constraint matcher that performs a look up in cardinality compatibility table in order to compare cardinalities of schema elements.

Structural similarities exploit previously computed basic similarities and are based on:

1. similarity of paths,
2. similarity of immediate descendants,
3. similarity of leaves.

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. 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 Document Type Definitions. Finally, for two Document Type Definitions, best match pairs are summed up and normalised, thereby resulting in a final similarity between these Document Type Definitions. The result of the first phase is the similarity matrix of a set of Document Type Definitions. During the second phase, based on the Document Type Definition similarity matrix, a hierarchical clustering is applied to group Document Type Definitions into clusters.

**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 [83]. It is assumed that:

(i) the matching step has already been performed,

(ii) correspondences have already been made operational.

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, Toronto Mapping Adaptation System 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. Toronto Mapping Adaptation System handles relational and eXtensible Markup Language 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. 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,
(iii) modifications that reshape schema structure by moving, copying, or renaming elements.

The final result of Toronto Mapping Adaptation System is a set of adapted mappings which are consistent with the structure and semantics of the evolved schemas.

MapOnto (University of Toronto and Rutgers University)

MapOnto is a system for constructing complex mappings between ontologies and relational or eXtensible Markup Language schemas [84-86]. MapOnto takes as input three arguments:

(i) an ontology specified in an ontology representation language, e.g., Web Ontology Language,

(ii) relational or eXtensible Markup Language schema, and

(iii) simple correspondences, e.g., between eXtensible Markup Language attributes and ontology data type properties.

Input schema and ontology are internally encoded as labeled 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.

OntoMerge (Yale University and University of Oregon)

OntoMerge [87] is a system for ontology translation on the semantic web. Ontology translation refers to tasks, such 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 (sub ontology) OS of the first one, build the corresponding extension OS‘, 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 source knowledge representation formalism, e.g., Web Ontology Language, to an internal representation, which is Web-PDDL (Planning Domain Definition Language) [88]. 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 mechanized 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. OntoMerge supports bridge rules which can be expressed using the full power of predicate calculus.

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

CtxMatch [89, 90] uses a semantic matching approach. 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. CtxMatch2 [91] improves on CtxMatch by handling properties. At the structure level, it exploits description logic reasoners, such as Pellet [92] or FaCT [93] to compute the final alignment.

**S-Match (University of Trento)**

S-Match [94] 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, eXtensible Markup Language schemas, ontologies, and returns as output logic relations, e.g., equivalence, subsumptions, 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. The input ontologies (tree-like structures) are codified in a standard internal eXtensible Markup Language format. The module taking input ontologies performs some preprocessing with the help of oracles which provides the necessary a priori lexical and domain knowledge. Examples of oracles include WordNet. 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.

**HCONE (University of the Aegean)**

HCONE is an approach to domain ontology matching and merging by exploiting different levels of interaction with users [95-97]. First, an alignment between two input ontologies is computed with the help of WordNet. Then, the alignment is processed straightforwardly by using some merging rules, e.g., renaming, into a new merged ontology. The HCOME basic matching algorithm works in six steps:

(i) choose a concept from one ontology, denoted as $c$,

(ii) obtain all the WordNet senses of $c$, denoted as $s_1, s_2, ..., s$.

(iii) obtain hypernyms and hyponyms of all the senses of $c$,

(iv) 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 (ii). The vicinity terms
include those from the same synsets of $m$ senses, hypernyms and hyponyms from step (iii),

(v) build a query $q$ by using terms which are sub concepts of $c$. If the terms considered for $q$ also exist among the $n$ terms from step (iv), then $q$ memorises that position with the help of flags,

(vi) taking as input the association matrix computed at step (iv) and the query computed at step (v), Latent Semantic Indexing 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, HCOME 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).

**MoA (Electronics and Telecommunication Research Institute, ETRI)**

MoA [98] is an ontology merging and alignment tool. 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 (Web Ontology Language – Description Logic). 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 tokenized,
(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 SMatch system. 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. Thus, for example, author can be found to be equivalent to writer because they belong to the same synset in WordNet.

ASCO (INRIA Sophia-Antipolis)

ASCO is a system that automatically discovers pairs of corresponding elements in two input ontologies [56]. ASCO handles ontologies in Resource Description Framework Schema and computes alignments between classes and relations. A new version, ASCO2, deals with Web Ontology Language ontologies [99]. 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 and external resources, such as WordNet. Based on token similarities, the similarity between sets of tokens is computed using Term Frequencies – Inverse Document Frequencies. 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 algorithm is an iterative fixed point computation algorithm that propagates similarity to the neighbours (subclasses, superclasses and siblings). Similarities between sets of objects are computed through single linkage. The propagation terminates when the class similarities and the relation similarities do not change after iteration or a certain number of iterations is reached. In the third phase, the linguistic and structural similarity is aggregated through a weighted sum and, if the similarities between matching candidates exceed a threshold, they are selected for the resulting alignment.
BayesOWL and BN mapping (University of Maryland)

BayesOWL is a probabilistic framework for modeling uncertainty in the semantic web. It is a Bayesian Network mapping approach for ontology matching [100]. 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 exemplars) is created by querying a search engine. The query contains all the terms, in the path from the root to the concept (term) under consideration in the given ontology, thereby enabling some word sense disambiguation. A text classifier, is trained on the statistical information about exemplars from the first ontology. Then, concepts of the second ontology are classified with respect to the concepts of the first ontology by feeding their exemplars to the trained classifier.

A similarity between two concepts is determined with the help of the Jaccard coefficient 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 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.

OMEN (The Pennsylvania State University and Stanford University)

OMEN (Ontology Mapping Enhancer) [101] is a semi-automatic probabilistic ontology matching system based on a Bayesian network. It takes as input two ontologies and an initial probability distribution derived, for instance, from basic (element level) linguistic matchers. In turn, Ontology Mapping Enhancer 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. During the
second step, Ontology Mapping Enhancer uses a set of meta-rules that capture the influence of the structure of input ontologies in the neighborhood 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, $C_{1i} \in O_1$, matches the $j$-th concept from the second ontology, $C_{2j} \in O_2$.

(ii) if there is a relation $q$ between concepts $C_{1i}$ and $C_{1k}$ in the first ontology, which matches a relation $q'$ between concepts $C_{2j}$ and $C_{2m}$ in the second ontology.

Then the probability of match can be increased between concepts $C_{1k}$ and $C_{2m}$. 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 to generate a posteriori probabilities for each node. Finally, a posteriori probabilities, which are higher than a threshold, are selected to generate the resulting alignment.

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

MetaQuerier [102] is a middleware system that assists users in finding and querying multiple databases on the web. It exploits the DCM (Dual Correlation Mining) matching framework to facilitate source selection according to user search keywords [103]. 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.

Dual Correlation Mining 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 Dual Correlation Mining 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, the final alignment is produced.

2.5.2. Instance-based Ontology Matching Systems

Instance-based systems are those which rely mostly on instance-level input information for performing ontology matching. Instance based systems are those taking advantage of instances.

T-tree (INRIA Rhône-Alpes)

T-tree [104] 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. 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 super class 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 sub typing, $\cap$ is type intersection and $=\emptyset$-test is a sub typing test.

CAIMAN (Technische Universit¨at M¨unchen and Universit¨at Kaiserslautern)

CAIMAN [105] 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 Term Frequencies – Inverse Document Frequencies feature vector), is created for each concept in an ontology. Then, the cosine measure is computed for two of those class vectors. Finally, with the help of a threshold, the resulting alignment is produced.

FCA-merge (University of Karlsruhe)

FCA-merge uses formal concept analysis techniques to merge two ontologies sharing the same set of instances [106]. 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 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.

**LSD (University of Washington)**

LSD (Learning Source Descriptions) [107] 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. 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. Learning Source Descriptions handles eXtensible Markup Language schemas. A schema is modeled as a tree, where the nodes are eXtensible Markup Language tag names.

Learning Source Descriptions approach consists of two phases: Training phase and Matching phase. During 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 (Word-based Heterogeneous Information Representation Language) learner, the Naive Bayesian learner. The meta-matcher combines the predictions of basic matchers. It is trained by using a stacked generalization (learning) technique. During the matching phase Learning Source
Descriptions 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), Learning Source Descriptions obtains a prediction list of matching candidates. Finally, by taking into account integrity constraints and applying some thresholds, the final alignment is extracted.

**GLUE (University of Washington)**

GLUE [108], a successor of Learning Source Descriptions, is a system that employs multiple machine learning techniques to semi-automatically discover one-to-one semantic mappings (which are 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.

GLUE follows a multi strategy 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) 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 node $x$ also matches node $y$, 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.

**iMAP (University of Illinois and University of Washington)**

iMAP [109] 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. 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 [110]. 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, 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 \(<\text{target attribute}, \text{match candidate}>\) 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.

**Automatch (George Mason University)**

Automatch [111] 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, 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.

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.

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

SBI (Similarity-Based Integration) is a system for automatic statistical matching among classifications [112, 113]. SBI & NB (Similarity-Based Integration and Naive Bayes classifier) is an extension of Similarity-Based Integration by plugging into the system a Naive Bayes classifier. The idea of Similarity-Based Integration 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 $\kappa$-statistic method [114]. These pairs are considered to be the final alignment. Similarity-Based Integration and Naive Bayes classifier combines sequentially the Similarity-Based Integration 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.

**Kang and Naughton (University of Wisconsin-Madison)**

Kang and Naughton [115] proposed a structural instance-based approach for discovering correspondences among attributes of relational schemas with opaque column names. 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 for 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. 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 is returned.

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

Dumas (DUPLICATE-based MAAtching of Schemas) [116] is an approach which identifies one-to-one alignments between attributes by analysing the duplicates in data instances of the relational schemas. 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, in DUPLICATE-based MAAtching of Schemas, tuples are viewed as strings and a string comparison metric, such as cosine measure, is used to compare two tuples. Specifically, tuples are tokenised and each token is assigned a weight based on Term Frequencies – Inverse Document Frequencies scheme. In order to avoid complete pairwise comparison of tuples from two databases, the Word-based Heterogeneous Information Representation Language algorithm is used. It performs a focused search based on those common values that have high Term Frequencies – Inverse Document Frequencies 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 Term Frequencies – Inverse Document Frequencies -based measure, called soft Term Frequencies – Inverse Document Frequencies [117], 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.

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

Wang and colleagues [118] propose an instance-based solution for discovering one-to-one alignments among the web databases. These are query interfaces (Hyper Text Markup Language 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:
(i) 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.

(ii) It is assumed that 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: query probing and 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.

In order to perform inter-site matching a vector-based similarity is used. In particular, each attribute of an individual interface or result schema is viewed as a document and each attribute of the global schema is viewed 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 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.
**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 [119,120]. In particular, it is a Global-Local-As-View like framework [121] 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. Probabilistic, Logic-based Mapping between schemas 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.

Probabilistic, Logic-based Mapping between schemas has been tested with the following matchers:

(i) same attribute names,

(ii) exact tuples,

(iii) the $k$-nearest neighbor classifier, and

(iv) the Naive Bayesian classifier.

The result of these matchers is aggregated by using linear or logistic functions, or their combinations. Coefficients of the normalization 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.

**2.5.3. Summary**

In summary, the following can be observed concerning the ontology matching systems:
(i) Schema-based matching solutions have been so far investigated more intensively than the instance-based solutions.

(ii) Most of the systems under consideration focus on specific application domains and with particular ontology types, few systems aim at being general, suit various application domains, and handle multiple types of ontologies.

(iii) A pair of ontologies is taken as input in most of the systems; while only few systems take input as multiple ontologies.

(iv) These systems mostly follow the tree-like structures, while few other systems handle graphs.

(v) Ontologies focus on one-to-one alignments for the problems, and only few systems have tried to address the problem of discovering more complex correspondences, such as one-to-many and many-to-many.

(vi) A good number of ontology 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.

Table 2.1 summarises the matching systems based on the matching techniques used by the different matching systems and on the requirements of a matching system in general.

2.6. Evaluations of the Ontology Matching Systems

Currently, there exist a number of ontology matching systems. However, these systems are developed for various purposes and using different strategies. Noy and Musen [122] propose a framework for evaluating ontology-mapping tools based on a variety in underlying assumptions and requirements, in order to compare different systems, surveys (e.g., [47], [48]) summarize and evaluate several tools. However, this evaluation focuses on functionality, user interaction and mapping strategies, but does not deal with matching quality. To evaluate the increasing number of ontology matching methods and their qualities, OAEI (Ontology Alignment Evaluation Initiative) have suggested evaluation measures.
<table>
<thead>
<tr>
<th><strong>Ontology Matching System</strong></th>
<th><strong>Element - Level</strong></th>
<th><strong>Techniques</strong></th>
<th><strong>Structure - Level</strong></th>
<th><strong>Semantic</strong></th>
<th><strong>Requirements</strong></th>
<th><strong>Input</strong></th>
<th><strong>Output</strong></th>
<th><strong>Operation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Syntactic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELTA</td>
<td>String-based</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Relational</td>
<td>User</td>
<td>Alignment</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Schema, EER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hovy</td>
<td>String-based</td>
<td>-</td>
<td>Taxonomic Structure</td>
<td>-</td>
<td>Ontology</td>
<td>Semi</td>
<td>Alignment</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Language-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transcm</td>
<td>String-based</td>
<td>Built-in Thesaurus</td>
<td>Taxonomic Structure Matching of neighbourhood</td>
<td>-</td>
<td>SGML, OO</td>
<td>Semi</td>
<td>Translator</td>
<td>Data Translation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIKE</td>
<td>String-based, Domain Compatibility</td>
<td>WordNet</td>
<td>Matching of neighbourhood</td>
<td>ER</td>
<td>ER</td>
<td>Semi</td>
<td>Merge</td>
<td>Query Mediation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SKAT &amp; ONION</td>
<td>String-based</td>
<td>Auxiliary Thesaurus, Corpus-based</td>
<td>Taxonomic structure, Matching of neighbourhood</td>
<td>RDF</td>
<td>RDF</td>
<td>Semi</td>
<td>Bridge Rules</td>
<td>Data Translation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artemis</td>
<td>Domain Compatibility, Language-based</td>
<td>Common Thesaurus</td>
<td>Matching of neighbours via Thesaurus, Clustering</td>
<td>-</td>
<td>Relational Schema, OO, ER</td>
<td>Auto</td>
<td>Views</td>
<td>Query Mediation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H-Match</td>
<td>Domain Compatibility, Language-based, Domains and Ranges</td>
<td>Common Thesaurus</td>
<td>Matching of neighbours via Thesaurus, Relations</td>
<td>-</td>
<td>OWL</td>
<td>Auto</td>
<td>Alignment</td>
<td>P2P Query Mediation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain Compatibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2.1. Ontology Matching Systems

<table>
<thead>
<tr>
<th>Ontology Matching System</th>
<th>Techniques</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Element - Level Syntactic</td>
<td>Structure - Level Syntactic</td>
</tr>
<tr>
<td>Anchor-Prompt</td>
<td>String based Domains and Ranges</td>
<td>Bounded paths matching: (arbitrary links), Taxonomic Structure</td>
</tr>
<tr>
<td>OntoBuilder</td>
<td>String-based, Language-based Thesaurus Lookup</td>
<td>-</td>
</tr>
<tr>
<td>Cupid</td>
<td>String-based, Language-based, Datatypes, Key Properties Auxiliary Thesauri</td>
<td>Tree Matching weighted by Leaves</td>
</tr>
<tr>
<td>COMA, COMA++</td>
<td>String-based, Language-based, Datatypes Auxiliary Thesauri, Alignment Reuse, Repository of Structures</td>
<td>Direct Acyclic Graph(tree) Matching with a bias towards various Structures, e.g., Leaves</td>
</tr>
<tr>
<td>Similarity Flooding</td>
<td>String-based, Datatypes, Key Properties</td>
<td>Iterative fixed point computation</td>
</tr>
<tr>
<td>Xclust</td>
<td>Cardinality Constraints Word Net</td>
<td>Paths, Children, Leaves, Clustering</td>
</tr>
<tr>
<td>ToMAS</td>
<td>-</td>
<td>External Alignments Preserving Consistency, Structure Comparison</td>
</tr>
</tbody>
</table>

53
Table 2.1. Ontology Matching Systems

<table>
<thead>
<tr>
<th>Ontology Matching System</th>
<th>Techniques</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Element - Level Syntactic</td>
<td>Structure - Level Syntactic</td>
</tr>
<tr>
<td>MapOnto</td>
<td>External Alignments</td>
<td>Structure Comparison</td>
</tr>
<tr>
<td>OntoMerge</td>
<td>External Alignments</td>
<td>-</td>
</tr>
<tr>
<td>CtxMatch &amp; CtxMatch2</td>
<td>String-based, Language-based</td>
<td>Word Net</td>
</tr>
<tr>
<td>S-Match</td>
<td>String-based, Language-based</td>
<td>Word Net</td>
</tr>
<tr>
<td>HCOME</td>
<td>Language-based (LSI)</td>
<td>Word Net</td>
</tr>
<tr>
<td>MoA</td>
<td>Language-based</td>
<td>Word Net</td>
</tr>
<tr>
<td>ASCO</td>
<td>String-based, Language-based</td>
<td>Word Net</td>
</tr>
<tr>
<td>Bayes OWL &amp; BN Mapping</td>
<td>Text Classifier</td>
<td>Google</td>
</tr>
<tr>
<td>OMEN</td>
<td>External Alignment</td>
<td>Bayesian Inference, Meta-rules</td>
</tr>
<tr>
<td>DCM</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>T-tree</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ontology Matching System</td>
<td>Techniques</td>
<td>Requirements</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>CAIMAN</td>
<td>String-based (Rocchio Classifier)</td>
<td>Classification</td>
</tr>
<tr>
<td>FCA-Merge</td>
<td>-</td>
<td>Ontology</td>
</tr>
<tr>
<td>LSD/GLUE</td>
<td>WHIRL, Naive Bayes, Domain Constraints, Hierarchical Structures</td>
<td>Relational Schema, XML Schema, Taxonomy</td>
</tr>
<tr>
<td>iMAP</td>
<td>WHIRL, Naive Bayes, Domain Constraints, Hierarchical Structures</td>
<td>Relational Schema</td>
</tr>
<tr>
<td>Automatch</td>
<td>Naive Bayes, Internal Structure, Statistics</td>
<td>Relational Schema</td>
</tr>
<tr>
<td>SBI &amp; NB</td>
<td>Statistics, Naive Bayes, Pachinko Machine Naive Bayes</td>
<td>Classification</td>
</tr>
<tr>
<td>Kang &amp; Naughton</td>
<td>Information Entropy, Mutual Information, Dependency Graph Matching</td>
<td>Relational Schema</td>
</tr>
<tr>
<td>Dumas</td>
<td>String-based, WHIRL, Instances Identification</td>
<td>Relational Schema</td>
</tr>
<tr>
<td>Wang &amp; Colleagues</td>
<td>Language-based, Mutual Information</td>
<td>Web Form</td>
</tr>
<tr>
<td>sPLMAP</td>
<td>Navie Bayes, KNN Classifier, String-based</td>
<td>Database Schema</td>
</tr>
</tbody>
</table>

Table 2.1. Ontology Matching Systems
Evaluation Input

The inputs of evaluation are two ontologies written in the Web Ontology Language - Description Logic language. The different elements of ontologies, e.g., concepts, instance and relations can be aligned. The output is a \(*:*\) equivalence alignment of named entities. For example, one entity of one ontology can (e.g., injective, total, one-to-one) map to entity/entities of the other ontology. That means non constraint on the alignment.

Two benchmarks are proposed to be evaluated: a competence benchmark and a comparison benchmark [42]. Competence benchmarks aim to distinguish the performance of a special system regarding a set of well-defined tasks which are isolated special characteristics. Competence benchmarks help the system designers to evaluate their systems to localize with the stable system. Comparison benchmarks aim to compare the performance of different systems on a defined task or application. It aims to improve the whole field instead of individual systems.

How to Measure Evaluation Results

Ontology Alignment Evaluation Initiative proposed different evaluation measures, from machine-focused (e.g., compliance and performance measures) to user-focused, from general to task-specific measures. However, user-focused measures need interaction of users which is not easy to get any objective evaluations. Task-specific measures need to set up different task compare profiles with respect for certain tasks. It is difficult to determine the evaluation value of the alignment process independently, so the current evaluations focus more on compliance and performance measures.

Compliance Measures

Compliance measures aim to evaluate the quality of the output provided by a system compared to a reference output. The compliance measures consist of Hamming distance, Precision, Recall, Fallout, F-measure, Overall, etc. [1, 42].

Performance Measures

Performance measures compare important features of the algorithms (e.g., speed, memory scalability and user related measures). However, performance measures depend on the
evaluation environment and ontology management system. It is difficult to get objective evaluations.

2.7. Summary

The concept of ontology matching is a quite new area but developing faster. It involves a large number of fields, e.g., machine learning, database schema, linguistics, etc. Different strategies used in ontology matching are based on these fields. However, the matching systems still need be improved. Until now, there are a few ontology evaluation tools available. Ontology Alignment Evaluation Initiative proposes different measures to evaluate ontology matching. However, it does not support to evaluate the combination of matchers.