.....
Georg Gottlob

MASTER THESIS

Schema Matching and Automatic Web Data Extraction

carried out at the

Institute of Information Systems
Database and Artificial Intelligence Group
of the Vienna University of Technology

under the instruction of

Prof. Dr. Georg Gottlob
gottlob@dbai.tuwien.ac.at
and
Dr. Robert Baumgartner
baumgart@dbai.tuwien.ac.at

by

Evaldas Taroza
taroza@gmail.com

Vienna, May 2006

.....
Evaldas Taroza

Abstract

The core of the thesis is *schema matching* in the context of *Web data extraction and integration*. There are many solutions developed for integration purposes, and the schema matching, in particular. They can be grouped into traditional and holistic. The former implies a small scale, while the latter – a large scale integration (matching). As far as *Web data extraction and integration* is concerned, schema matching is needed as a module of a mediator system that integrates extracted Web data. However, choosing the most suitable paradigm, approaches, and techniques for a specific mediator system is difficult. On the one hand the assortment is very wide, and on the other – requirements (e.g., degree of automation, efficiency, etc.) vary from one system to another. The main objective of the thesis is to propose a schema matching solution for the *automatic* Web data extraction. To achieve that, a study, comprising a presentation and thorough comparisons of currently available schema matching paradigms, approaches, techniques, is given. The goal of the study is to summarize and improve the already available comparative studies ([31, 10, 21]) and add other axes of comparison, e.g., traditional vs. holistic paradigm. As a result, the investigation of the schema matching problem in the automatic Web data extraction is conducted. In particular, a use case taken from the *AllRight* project [18] is considered as a specific problem setting. Briefly, it can be described as integrating a bunch of rather simple schemas (together with instance data) into one coherent schema. Since this problem is taken from the currently ongoing project, the thesis is supposed to be beneficial for the project as a basis for a practical application.

Contents

1 Introduction **4**

2 Schema Matching **6**
2.1 Traditional Paradigm 7
2.2 Holistic Paradigm 10

3 Traditional Schema Matching Solutions **12**
3.1 Approaches 12
3.1.1 Schema based matching approach 14
3.1.2 Instance based matching approach 17
3.1.3 Combining different matchers 18
3.2 Representative Solutions 19
3.2.1 LSD 19
3.2.2 CUPID 21
3.2.3 Similarity Flooding 22
3.2.4 Corpus-based Matcher (REVERE) 23
3.2.5 Ontology-based Matcher 25
3.2.6 COMA++ 26
3.2.7 eTuner 27
3.3 Summary of Traditional Matching Techniques 28
3.3.1 Measuring Similarity 28
3.3.2 Auxiliary Information Used in Matching 32
3.3.3 Matching System Architecture 33

4 Holistic Schema Matching Solutions **35**
4.1 The MGS Framework 36
4.1.1 Hypothesis Modeling, Generation and Selection 37
4.2 The DCM Framework 39
4.2.1 Correlation Mining 40
4.3 Clustering-based Matcher 42
4.4 Summary of Holistic Matching Techniques 43
4.4.1 Statistical Techniques 43
4.4.2 Other Techniques 44

5 Software for Schema Matching **44**
5.1 Useful Programming Libraries 47
5.1.1 SimMetrics – String Comparison 47
5.1.2 Weka – Machine-learning 49
5.1.3 Joone – Neural Networks 50
5.1.4 Carrot2 – Search Results Clustering 50
5.1.5 Other 50

6 Comparative Study **51**
6.1 Framework for Making Comparisons 51
6.1.1 Criteria related to the input 52
6.1.2 Measures of matching quality 52
6.1.3 Effort 54
6.2 Schema Matching Paradigms 55
6.3 Traditional Schema Matching Solutions 56
6.4 Holistic Schema Matching Solutions 57

7 A Specific Problem Setting **58**
7.1 The *AllRight* project 58
7.1.1 IR Component 58
7.1.2 IE Component 59
7.1.3 II Component 60
7.2 Definition of the Problem 60
7.3 Analysis of the Problem 61
7.3.1 Feasibility of the Holistic Paradigm 62
7.3.2 Feasibility of the Traditional Paradigm 64
7.4 Solution Proposal for *AllRight* 65
7.4.1 Experiment 67

8 Conclusion **70**

1 Introduction

Schema matching is the problem of resolving schema-level semantic heterogeneity [14]. It stems from the field of databases, which is quite natural, having in mind that a relational database inherently has a relational schema. The need for matching relational schemas can emerge from many situations. For example, a company buys another one, running the same type of business, and decides to merge their databases. Even if the information in both databases represents the same concepts, nevertheless relational schemas are likely to be different, thus semantical mapping must be resolved before performing merging.

Abstracting from databases, a schema can mean any model, for instance, an XML schema, interface definition, semantic network, etc. *Model management* [4] is a research area concentrating on resolution of *mapping*s among models in order to provide high-level operations, like matching, merging, selection, composition, etc., on models as well as on *mapping*s themselves. Motivation for that is the observation that programmers usually deal with some kind of a model and have to develop their own, generally not reusable, libraries, e.g., for mapping an XML schema to a relational database, integrating two source XML schemas into a target one, mapping a conceptual database schema to a relational one to enable queries on the conceptual level, etc.

Intuitively, matching is the operation that produces a mapping between corresponding elements of two (or more) input schemas. Being a fundamental operation in *model management* it became a separate research topic under the title of *schema matching*. It plays a central role in a number of applications, such as electronic commerce, schema integration, schema evolution and migration, data warehousing, database design, and others. Hence the automation of this operation is imperative. However, full automation is not always possible, because the schema matching aims at finding *semantic* correspondences and thus may require human interaction even though advanced reasoning techniques about ontologies, natural language processing, heuristics and statistics are employed.

In the thesis *schema matching* appears especially in the context of *Web data extraction and integration*. This field relies on the Web as a huge database that lacks meaningful querying techniques. One of the ways to query the Web is using *Information Retrieval* (IR) methods. They are highly dependent on statistics; the result of a query is a set of documents that are relevant to keywords in that query. However, such result is not always enough and usually need further processing, at least by a human. Yet another way to query the Web is using *Information Extraction* (IE) methods. These methods allow pulling facts and structured information from the content of large text collections. As the IE can be said to provide information from the inside documents, it is a preferred way to query the Web in enterprise applications, e.g., Business Intelligence, where this information is crucial. The IE is performed by, so called, *wrappers* – procedures that extract content from a particular resource. Apparently, the problem of the wrapper generation is the most significant. For example, Lixto [2] allows for the supervised wrapper generation. One can define a rule set, patterns, domain

ontology and then generate a wrapper that extracts content from structurally similar Web resources. In general, many solutions are proposed to achieve that the wrapper generation, hence the IE, were as much automatized as possible, e.g., machine-learning, wrapper induction.

The second part of *Web data extraction and integration* is about making extracted data coherent. And here *schema matching* comes into play as a part of the mediator system. As depicted in Figure 1, such a system may integrate data also from other sources, not necessarily wrappers. Hence as an input the mediator system gets a set of schemas, and as an output gives a unified view on them.

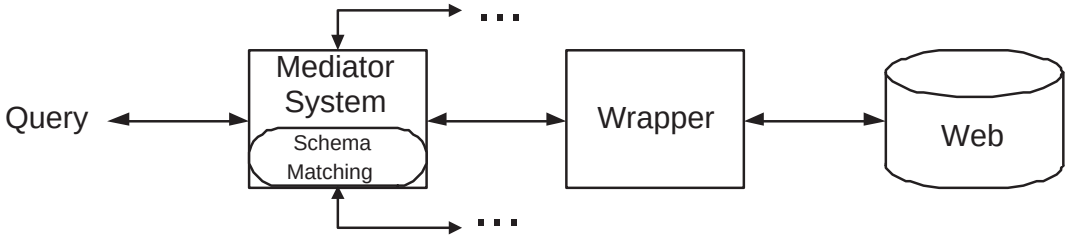

Figure 1: Web data extraction and integration

The purpose of the thesis is to find a solution for the schema matching problem in the automatic Web data extraction. The main challenge in this field is to perform a large scale integration of automatically extracted schemas. Since in the automatic Web data extraction user interference is avoided, the semantical information, necessary for discovering correspondences among schemas, has to be determined automatically, e.g., using data-mining methods. As it is shown later, the holistic schema matching paradigm is feasible for this purpose. Additionally, traditional techniques can fruitfully extend the holistic solution.

Hence the thesis can, roughly speaking, be split into two parts. One part is dedicated to a thorough study of *schema matching*. First, matching related definitions and examples are provided. Then, available paradigms, approaches, techniques and best practices (i.e., the state-of-the-art) are reviewed. Later, a comparative study is given. Worth noticing, that the comparative study is carried out to be beneficial for the specific problem setting introduced in the second part. For example, little attention is paid on matching different types of models, because, as far as the problem is concerned, only XML schemas have to be integrated. In the second part of the thesis the problem, taken from the *AllRight* project [18] (related to automatic Web data extraction), is examined. It is defined and carefully analyzed in the light of the accomplished schema matching study. The intention is to give a good understanding of the schema matching problem in *AllRight* and a basis for determining weaknesses and merits of possible solutions. Additionally, useful programming libraries are discussed. Some of them are used for a simple experiment that justifies the feasibility of the proposed solution.

2 Schema Matching

There are different kinds of schemas, such as relational database schema, XML schema, domain conceptual schema, etc. Moreover, there are various ways to represent them. For instance, XML schema could be defined by such standards as DTD, XML Schema, Relax NG, etc. However, neither variety of schemas nor their representations should be an obstacle for a general discussion. Therefore, below a high-level definition, of what will be considered as a schema, is given.

Definition 2.1
A *schema* is a *set of elements* (element types, attributes, simple types, etc.) connected by some *structure*.

Definition 2.1 implies that schemas under consideration have a graph structure. Sometimes graphs need to be transformed into trees. For instance, in XML schema a shared element type definition can be transformed into many local element type definitions. Even though schemas are graphs, sometimes the structural properties are ignored. For instance, in the holistic paradigm the schemas are usually considered as unstructured sets of elements.

Every node in the graph can contain some meta-data attached, for example, number restrictions, optionality, list of synonyms, etc. This information could be exploited by matching algorithms. In Figure 2, two possible schemas for purchase orders are depicted. The tree structure of schemas should not be mixed with the tree structure of well-formed XML documents, because instance documents would have actual values represented as nodes of the tree.

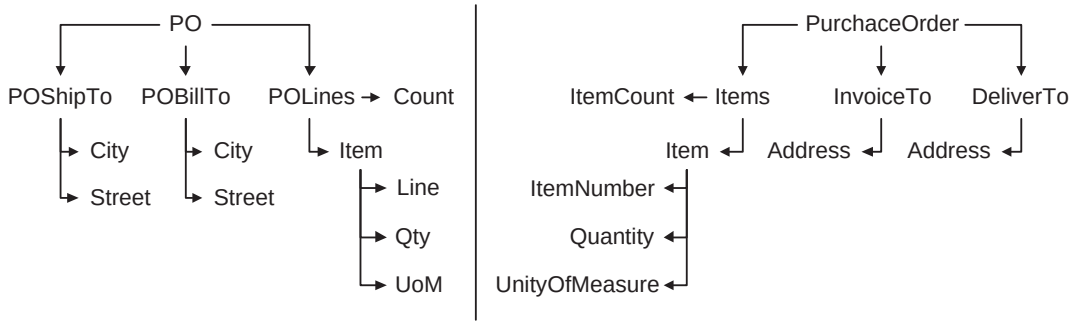

Figure 2: Purchase order schemas S_1 (left) and S_2 (right)

In the literature there are three kinds of schemas mentioned: input, source, and target. Any schema in the matching process can be called *input schema*. the term *source schema* is sometimes used as a synonym for *input schema*, but usually it means that there is a target schema to which the source schema has to be matched. When the matching is not directional the source and the target is not important, because any schema can be either a source or a target.

As it was already mentioned, *schema matching* in *Web data extraction and integration* appears as a part of the mediator system. A general outlook of *schema matching* in such a system is given in Figure 3. Firstly, initial matching

is performed. In comparison, this step requires the most user activity, because it is a crucial step for the whole data integration in the system. After initial matching the system is driven through three more steps. These steps can also be considered as *schema matching*, but in the thesis they will not be profoundly covered. The first step is the normalization of matching results. An input for this step could be a number of mappings that serve for integration purposes. Here an agreement upon final output should be made. The next step is cleaning. Duplicates are removed in this step by some technique, such as record linkage. After there is an agreement upon the final schema and the data is prepared, the final step called fusion is performed. It outputs the actual result of matching as an integrated schema.

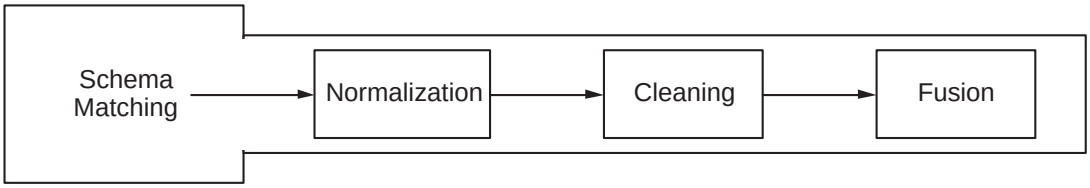

Figure 3: *Schema matching* in a mediator system

In the thesis only the initial step of *schema matching* in the mediator system (as shown in Figure 3) is considered. This means that *schema matching* is actually *mapping discovery* that aims to discover correct mappings with as little user interference as possible. There are two schema matching paradigms to cope with the mapping discovery: *traditional* and *holistic*. They depend on how the matching is performed. The main distinction of the two paradigms is that even with a high number of schemas to be matched the traditional paradigm implies pairwise matching, while the holistic paradigm deals with all the input schemas at once, as depicted in Figure 4. Moreover, there are differences on the definition level, for example, the holistic schema matching has a notion of n-ary match, while in the traditional schema matching it is always binary. Thus both paradigms deserve a separate discussion.

2.1 Traditional Paradigm

The traditional schema matching is built on the idea that there are two (or a small number of) schemas that need to be matched. When the number is greater than two, anyway the matching is performed pairwise. So the main challenge in the traditional schema matching is to develop techniques that allow deriving semantically corresponding elements in two input schemas.

Definition 2.2
A *mapping* is a set of *mapping elements* (matches) each of them indicating that certain elements (domain) of one schema are mapped to certain elements (range) of another one. More formally, given two schemas S_1 and S_2, a *mapping* can be written as $\mathcal{M}_T = \{M_1, M_2, \cdots, M_k\}$, where M_i represents a mapping element as

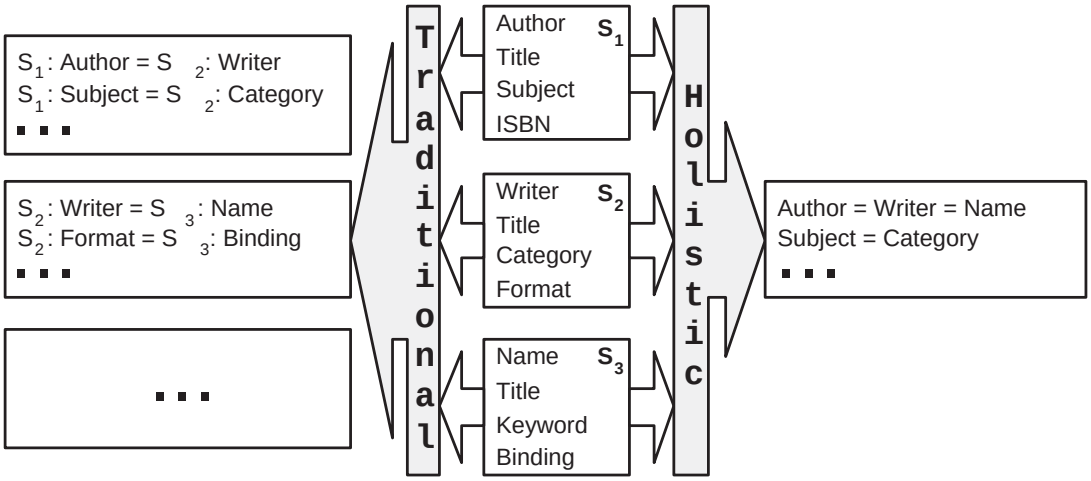

Figure 4: Traditional vs. holistic *schema matching*

a binary predicate, i.e., $G_{i_1} \cong G_{i_2}$, such that $G_{i_1} \subseteq S_1, G_{i_2} \subseteq S_2$ and $|G_{i_j}| \geq 1$. Each mapping element can have a *mapping expression* how mapped elements of the two schemas are related.

The traditional schema matching is characterized by the match operator that takes two schemas as an input and produces a mapping of corresponding elements as an output (see Definition 2.2). A mapping represents semantic relationship between schemas. The semantics (value correspondence) may be encoded into mapping expressions (see Example 2.1) during matching. Assignment of mapping expressions during matching is reasonable only when the result is supposed to be interpreted by a model management system [4], which is not the case in the thesis. This allows ignoring mapping expressions and relieve significant overhead.

Example 2.1
Say that in Figure 2 we have that:

- The element 'POBillTo' in S_1 corresponds to 'InvoiceTo' in S_2
- The elements 'City' and 'Street' in S_1 correspond to 'Address' in S_2

Then respective expressions might be associated with mapping elements:

- 'PO.POBillTo' = 'PurchaceOrder.InvoiceTo'
- 'PO.POBillTo.City' + 'PO.POBillTo.Street' = 'PurchaceOrder.InvoiceTo.Address'

By ignoring mapping expressions the problem of the schema matching is restricted to only producing mappings according to some similarity criteria of two schemas. Hence there is a need to define a *similarity relation* and address only the mapping discovery consistent with this relation. A similarity relation between two schemas represents one mapping element (one match) of the mapping. Thus the key task is to find out criteria and to develop an algorithm which would

"know" when, where and to what extent two schemas are similar. Usually degree of similarity is evaluated by a real number from the interval [0, 1], calculated by a matching algorithm. The closer the value to 1, the more similar are schema elements.

Definition 2.3
The *similarity relation* $\cong_T$ is a binary predicate denoting that elements of schemas match to a certain degree, expressed by a numeric value from the interval [0, 1].

Example 2.2
Consider the two schemas in Figure 2. How one would know that:

- The element 'POBillTo' in S_1 corresponds to 'InvoiceTo' in S_2
- The elements 'City' and 'Street' in S_1 correspond to 'Address' in S_2

There can be several answers. Possible respective solutions could be:

- Using some natural language processing techniques derive that billing someone is the same as sending an invoice to someone (linguistic criteria)
- Using an ontology derive that addresses are composed of a city and a street (domain knowledge criteria)

Respective mapping elements from the match result would look like this:

- 'PO.POBillTo' $\cong_T$ 'PurchaceOrder.InvoiceTo'
  with the degree of similarity equal to 1 (as an example)
- 'PO.POBillTo.City' + 'PO.POBillTo.Street' $\cong_T$ 'PurchaceOrder.InvoiceTo.Address'
  with the degree of similarity equal to 0.9 (as an example)

Obviously, it is not an easy task to find good criteria and techniques that are reliable for matching any pair of schemas. Most of the work to date leans on various heuristics and, heavily, on user interaction, because the traditional schema matching is inherently subjective, i.e., several plausible mappings are possible. Thus full automation of the schema matching is not adequate, user interference is needed. Therefore, the traditional schema matching is a procedure comprising the following features:

- A matching algorithm resolves *match candidates* from input schemas.
- A user rejects, accepts or tunes the result, or part of it, obtained by the matching algorithm.
- The final result of the matching algorithm is a mapping of input schemas according to a similarity relation and the user interaction.

The flow of these features depends on the actual implementation. For example, a user can input some parameters iteratively by tuning process. In any case, the aim is to minimize the effort required from the user to produce a mapping.

2.2 Holistic Paradigm

The holistic schema matching challenges the traditional paradigm. It takes into account that pairwise schema matching is not adequate when there are many input schemas. One of the motivations for the holistic schema matching is the *deep Web*, i.e., the need to pose queries for different sources and different query interfaces on the Web, while the information belongs to the same domain. Another motivation, as will be shown later, is automatic Web data extraction and integration.

The main advantage of the holistic paradigm over the traditional is the statistical nature of the matching problem. As [20] states, the statistical (and data-mining) approach is justifiable in the large scale matching for two reasons:

- *Proliferating sources.*
  There are more and more sources of structured information for the same domain on the Web. For example, information on books is available on a wide range of on-line bookshops.

- *Converging vocabularies.*
  The vocabulary used for writing schemas for certain domain tend to stabilize.

In the context of the holistic paradigm, schema elements are called *attributes*. This is due to the fact that in the large scale matching schemas commonly have simple "flat" structure: attribute – value. Calling schema elements as attributes also means that as nodes in the schema they have very little information attached, e.g., name and label, and usually, no instance-level information. Nevertheless, it is not excluded that attributes cannot be hierarchical or verbose.

It is possible to distinguish between *frequent* and *rare* attributes. The attribute frequency is qualified as the number of source schemas in which the attribute occurs. In fact, discriminating low frequency (rare) attributes, at least in the Web query interface matching, leads to faster convergence of vocabularies [20]. So rare attributes can also be considered as the noise which is quite an issue in the holistic schema matching.

Differently from the traditional paradigm, the holistic schema matching produces a mapping on a bunch of schemas at once, not on every pair of them.

Definition 2.4
A *mapping* is a set of n-ary *matches* (mapping elements) each of which indicate that certain elements of any of the input schemas correspond. More formally, given a set of schemas $\mathcal{S} = \{S_1, S_2, \cdots, S_u\}$, a *mapping* can be written as $\mathcal{M}_H = \{M_1, M_2, \cdots, M_v\}$, where M_i is an n-ary match, i.e., $G_{i_1} \cong G_{i_2} \cong \cdots \cong G_{i_n}$, such that $G_{i_j} \subseteq \mathcal{S}$ and $|G_{i_j}| \geq 1$.

Semantically, each $M_i \in \mathcal{M}_H$ represents the similarity relationship of schema element (attribute) groups $G_{i_1}, G_{i_2}, \cdots, G_{i_n}$ and each G_{i_j} represents the grouping relationship of schema elements (attributes) in G_{i_j}.

Definition 2.5
The *similarity relation* $\cong_H$ is an n-ary predicate denoting that (groups of) attributes match to a certain degree, expressed by a numeric value from the interval [0, 1].

Definition 2.6
The *grouping relation* $\mathcal{G}$ is an n-ary predicate denoting that some attributes belong to the same group to a certain degree, expressed by a numeric value from the interval [0, 1]. Every attribute is trivially a group to the degree equal to 1. Usually, instead of the predicate $\mathcal{G}$, the set notation is used.

Definition 2.7
A match M_j *semantically subsumes* a match M_k, denoted by $M_j \succeq M_k$, if all the semantic relationships in M_k are covered in M_j. The semantic relationship is something more general than the similarity relationship, and may vary in different implementations.

Example 2.3
Consider three schemas about books as sets of elements:

- $S_1 = \{\text{fname, lname, btitle}\}$.
- $S_2 = \{\text{author, book}\}$.
- $S_3 = \{\text{full-name, book-title}\}$.

Also consider that there are the following groupings of elements:

- $G_1 = \{\text{fname, lname}\}$, i.e., $\mathcal{G}(\text{fname, lname})$.

Moreover, consider the following matches among the schemas:

- M_1: $\{\text{fname, lname}\} \cong_H \{\text{author}\} \cong_H \{\text{full-name}\}$,
  i.e., $G_1 \cong_H \{\text{author}\} \cong_H \{\text{full-name}\}$ or $\cong_H (G_1, \{\text{author}\}, \{\text{full-name}\})$.
- M_2: $\{\text{lname}\} \cong_H \{\text{author}\}$.
- M_3: $\{\text{author}\} \cong_H \{\text{full-name}\}$.
- M_4: $\{\text{btitle}\} \cong_H \{\text{book}\} \cong_H \{\text{book-title}\}$.

For simplicity let us assume that both groupings and matches are assigned the degree 1, the highest confidence. Then there are the following subsumptions:

- $M_1 \succeq M_2$.
- $M_1 \succeq M_3$.

Note that we cannot write a subsumption between M_2 and M_3, although transitively it would seem that $\{\text{lname}\} \cong_H \{\text{full-name}\}$ to the degree 1. This may vary in different implementations.

The holistic schema matching definitions quite resemble the ones from the traditional paradigm. They can be seen as a generalization from the binary to n-ary case. However, the matching process here is different. Indeed, it reminds of data-mining procedures. From the user interaction perspective, the matching depends only on parameter setting, and not on adjusting match results to make attributes semantically corresponding. So the statistical information extracted from input schemas serve as semantical information, which allows avoiding user interaction.

Consequently, the accuracy of matching algorithms is highly dependent on the data preparation and parameter tuning. As the holistic paradigm has a statistical nature, the larger amount of input schemas, the more precise the results should be. However, the noise data (e.g., rare attributes) has to be taken into account. Also, it may be the case that not all input schemas need to be considered but only a sufficient number of them, which may increase the performance of a matcher. Worth noticing that the holistic paradigm usually presupposes the domain dependency.

3 Traditional Schema Matching Solutions

There are numerous solutions for the traditional schema matching, ranging from general purpose algorithms to programming libraries. All of them help to address different problems, therefore having an overview is indispensable. In this section various solutions are presented aiming at showing the problems they target and in what way. However, one restriction is made: *schema matching* here is considered as the XML schema matching. This restriction helps to concentrate on schema matching solutions in the context of *Web data extraction and integration*.

3.1 Approaches

Here the classification of approaches to the traditional schema matching is presented [31]. It provides with a high level of abstraction in talking about matching algorithms (matchers) and is the basis for choosing a solution for a specific problem depending, for instance, on the application domain. Every approach leans on various techniques, summarized later, which attempt to capture the degree of similarity regarding some criterion.

As it was mentioned earlier, a schema matching procedure has to possess several features. However, here the stress is put on the very algorithm for performing matching. The user interaction is left for the later discussion, when talking about representative solutions. Consequently, it is possible to distinguish levels of generality among matchers. These levels are characterized by the number of criteria a matcher uses to produce a mapping. Hence a matcher which is the least general respects only one criterion, e.g., the linguistic similarity of element names. A combination of matchers leads to a more general matcher.

The classification of approaches, provided by [31], relies on orthogonal axes. They can be briefly summarized as (the whole classification hierarchy is depicted

in Figure 5):

- *Instance vs. schema*: matching approaches can consider instance data (i.e., data contents) or only schema-level information.
- *Element vs. structure*: match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures.
- *Language vs. constraint*: a matcher can use a linguistic based approach (e.g., names of schema elements) or a constraint-based approach (e.g., keys and relationships among schema elements).
- *Match cardinality*: the overall match result may relate one or more elements of one schema to one or more elements of another, yielding four cases: 1:1, 1:n, n:1, n:m.
- *Auxiliary information*: most matchers rely not only on the input schemas S_1 and S_2 but also on auxiliary information, such as dictionaries, global schemas, previous matching decisions, and user input.
- *Individual vs. Combinational*: individual matchers respect only one similarity criterion. Combining matchers produces *hybrid* or *composite* matchers. They differ in the way the combination is used. Hybrid matchers try to calculate similarity according to several criteria. Composite matchers run independent match algorithms on the input schemas and combine the results.

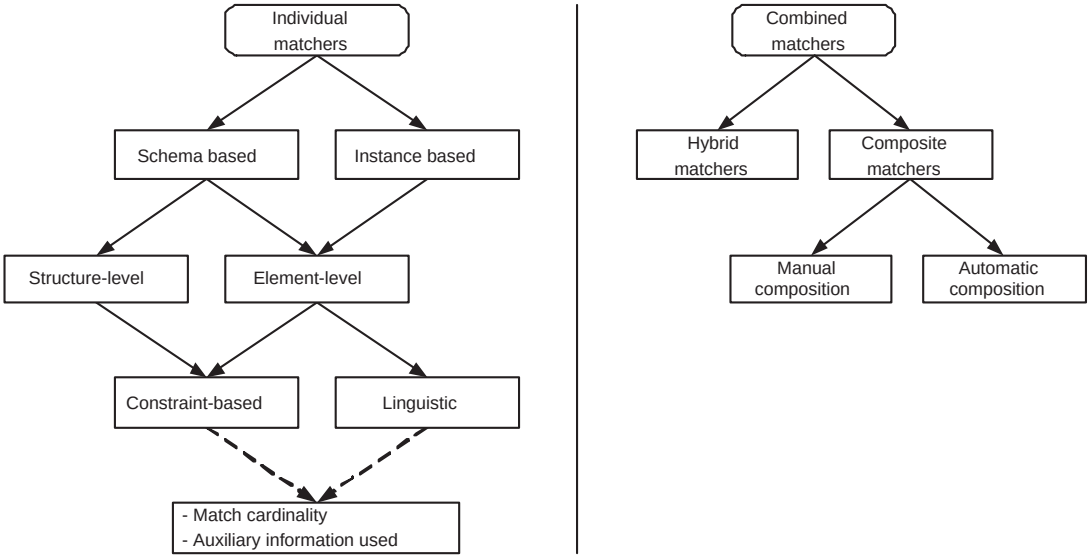

Figure 5: Classification of approaches to the traditional schema matching

Worth noticing that the term *matcher* is used as a synonym for *matching algorithm*. Usually a matcher is meant to compute similarity between schemas

according to one criterion. When talking about composite matchers, the term *matching system* is more appropriate, because this kind of matchers work with respect to many criteria and they have a modular architecture, namely, every combined matcher is called a module.

3.1.1 Schema based matching approach

In the schema based approach only the schema-level information is considered. The available information includes the properties of schema elements, such as name, data type, relationship types, constraints, and schema structure.

Structure-level matching. This kind of matching refers to matching combinations of elements that appear together in a structure. A range of cases is possible, depending on how complete and precise a match of the structure is required, which is usually specified by some kind of a threshold. Ideally, all components of the structures in the two schemas fully match. Alternatively, only a partial structural match is enough.

Example 3.1
Consider the purchase order example in Figure 2. The element 'POLines' in S_1 fully structurally matches the element 'Items' in S_2. However, the element 'PO' in S_1 only partially structurally matches the element 'PurchaseOrder' in S_2.

When a matcher is combined of several matching algorithms, allowing the partial matching can be advantageous because it imposes less constraints, and thus can be more flexible.

The structural matching can be implemented in various ways and usually is used in combined matchers. It helps to acquire contextual information for element-level matching. XPath is heavily used for such purposes.

Element-level matching. For each element of the first schema, the element-level matching determines matching elements in the second input schema. In the simplest case, only the elements at *atomic level*, such as attributes in an XML schema, are considered. However, it may also be applied to a *higher (non-atomic) level* elements, e.g., XML schema element types. In contrast to a structure-level matcher, this kind of approach considers higher-level elements in isolation, ignoring its substructure and components.

Example 3.2
Consider the purchase order example in Figure 2. The element 'POBillTo' in S_1 matches the element 'InvoiceTo' in S_2 at the element level. However, structurally these elements would not necessarily match.

Although in the element-level matching only schema element names is the most obvious criterion, there is a number of others. When dealing with XML schemas it can be, for instance, the data type. Useful criteria will be summarized later on.

Linguistic matching. Language-based or linguistic matchers use names and text (i.e., words or sentences) to find semantically similar schema elements. As mentioned earlier, a schema element can have meta data associated to it. In XML schema case it could be, for instance, a comment for an element type definition. If comments for some elements in input schemas are linguistically similar then it could be derived that the elements themselves are similar. Beside this criterion, one could use names of elements, instead of some associated text. The names are usually compared by considering:

- Equality of names. Equality of names from the same XML namespace bear the same semantics.
- Equality of canonical name representations after stemming and other preprocessing, when it is important to deal with special prefix/suffix symbols.
- Equality of synonyms, e.g., 'car' $\cong_T$ 'automobile'
- Equality of hypernyms. X is a hypernym of Y if Y is a kind of X. For instance, hypernyms of 'oak' include 'tree' and 'plant'.
- Similarity of names based on common substrings, edit distance, pronunciation, soundex (an encoding of names based on howthey sound rather than howthey are spelled), etc., e.g., 'ShipTo' $\cong_T$ 'Ship2'
- User-provided name matches, e.g., 'issue' $\cong_T$ 'bug'.

Obviously employing the linguistic approach usually requires the use of thesauri or dictionaries. General natural language dictionaries may be useful, perhaps even multi-language dictionaries, e.g., English-German, to deal with input schemas of different languages. In addition, name matching can use domain- or enterprise- specific dictionaries and taxonomies containing common names, synonyms, abbreviations, etc.

Constraint-based matching. Schemas often contain constraints to define data types, value ranges, uniqueness, optionality, relationship types, cardinalities, etc. If both input schemas contain such information, it can be used by a matcher as a criterion. For example, similarity can be based on the equivalence of data types and domains or of is-a relationships.

The use of the constraint information alone often leads to imperfect matches, as there may be several elements in a schema with comparable constraints. For instance, when similarity is based on the data type then data type *integer*, which is rather popular, would allow matching 'Count' to 'Quantity' in Figure 2. Still, the approach helps to limit the number of match candidates and may be useful when used in combination with other matchers.

Match cardinality. A schema element can participate in zero, one or many mapping elements of the match result between the two input schemas S_1 and S_2. Moreover, within an individual mapping element, one or more S_1 elements can match one or more S_2 elements. Thus, there are the usual relationship cardinalities (i.e., 1:1, 1:n, n:1, n:m) and they are of two types:

- *Global*, the relationship cardinality is determined on the level of the match result, i.e., the mapping.
- *Local*, the relationship cardinality is determined on the level of a mapping element.

The global cardinality cases with respect to all mapping elements are largely orthogonal to the cases for individual mapping elements.

Example 3.3
Say we have a matcher which is able to deal with addresses only and we input the two schemas from purchase order example in Figure 2. Consider the following situations:

- {'PO.POBillTo.City' + 'PO.POBillTo.Street' $\cong_T$ 'PurchaceOrder.InvoiceTo.Address',
  'PO.POShipTo.City' + 'PO.POShipTo.Street' $\cong_T$ 'PurchaceOrder.DeliverTo.Address'}
  then we have 1:1 global cardinality, since the mapping has two mapping elements and every schema element there is different. However, both mapping elements have n:1 local cardinality, because 'Address' is made up of 'City' and 'Street'.
- {'PO.POBillTo.City' $\cong_T$ 'PurchaceOrder.InvoiceTo.Address',
  'PO.POBillTo.Street' $\cong_T$ 'PurchaceOrder.InvoiceTo.Address',
  'PO.POShipTo.City' $\cong_T$ 'PurchaceOrder.DeliverTo.Address',
  'PO.POShipTo.Street' $\cong_T$ 'PurchaceOrder.DeliverTo.Address'}
  now we have 1:n global cardinality, since among four mapping elements both 'City' and 'Street' are separately treated as 'Address'. However, every mapping element has 1:1 local cardinality.

The element-level matching is typically restricted to local cardinalities of 1:1, n:1, and 1:n. Obtaining n:m mapping elements usually requires considering the structural embedding of the schema elements, and thus requires the structure-level matching. Most existing approaches map each element of one schema to the element of the other schema with highest similarity. This results in local 1:1 and global 1:1 or 1:n mappings. By now little research has been done on generating local and global n:1 and n:m mappings.

Using auxiliary information. Beside dictionaries, thesauri, and the user input, other type of auxiliary information is also used, namely, the reuse of common schema components and previously determined mappings. Reuse-oriented approaches are promising, since many schemas often are very similar to each

other and to previously matched schemas. For example, 'Address' is usually similarly defined as a concatenation of 'Country', 'City' and 'Street'.

The use of names from XML namespaces or specific dictionaries is already reuse-oriented. More generally, entire schema fragments, including such features as data types, keys, and constraints, could be reused. This is especially rewarding for frequently used entities, such as address, customer, employee, purchase order, and invoice, which should be defined and maintained in a schema library.

The reuse of existing mappings has two flavors. On the one hand, previously determined element-level matches may simply be added to the thesaurus. On the other, entire structures could be reused. This is useful when matching similar source schemas to one target schema. Among other things, it is beneficial that manually specified matches could also be reused.

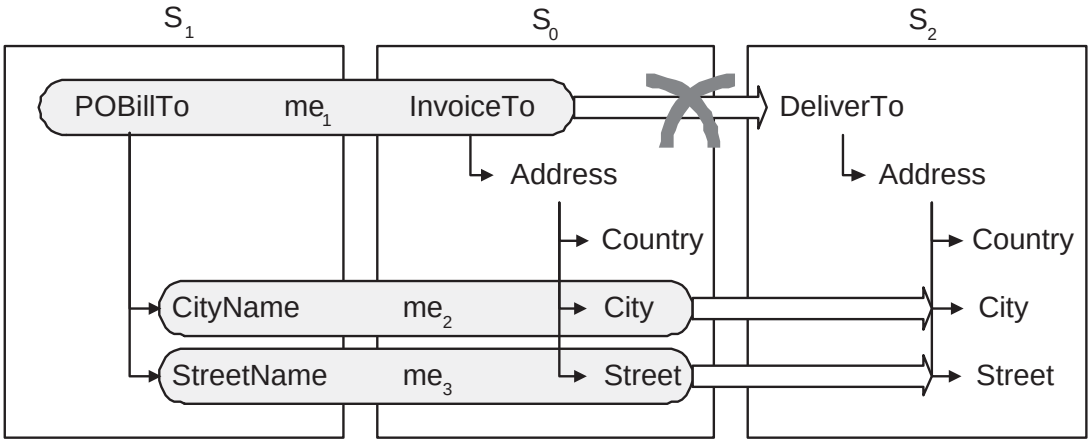

Figure 6: Reusing mapping between S_1 and S_0 for matching S_1 to S_2

Example 3.4
In Figure 6 the problem is to match S_1 to S_2. Supposing S_1 was earlier matched to S_0 and a mapping of three elements, i.e., me_1, me_2, me_3 was produced. As S_0 is quite similar to S_2, mapping elements me_2 and me_3 can be reused, however, as matches are not necessarily transitive, me_1 is not reusable.

3.1.2 Instance based matching approach

Instance-level data can give important insight into the contents and semantics of schema elements. In case of semistructured data, an approximate schema can even be constructed automatically out of an instance document. When substantial schema information is available, the use of the instance-level matching can be valuable to uncover incorrect interpretations of the schema-level information. To put it another way, instance data helps to reveal semantics of schema elements and can be performed on its own or in combination with other matchers.

Almost everything discussed for the schema based matching can be applied to the instance based matching. However, there are some specific details related to the latter. Obviously, one of them is the fact that instance based matching

does not include structure-level matching approaches (see Figure 5). Others are summarized below.

Linguistic matching. Similarity of two text elements is usually obtained using some information retrieval algorithms, for instance, by comparing frequencies of keywords.

Constraint-based matching. For elements with more structured instance data, such as numbers and strings, constraint-based matching can be employed. Instance-level constraints are different from those of the schema level, e.g., numerical value ranges and averages or character patterns. Using such constraints it is possible to recognize phone numbers, zip codes, geographical names, addresses, date entries, money-related entries, etc., but not necessarily the data type.

Match cardinality. In addition to the match cardinalities at the schema level, there may be different match cardinalities at the instance level. Going back to example 3.3, global cardinalities there, would correspond to instance-level cardinalities.

3.1.3 Combining different matchers

Beside the individual matcher approach it is worth considering combination of matchers. An algorithm that uses only one piece of information for matching is unlikely to produce as many good match candidates as the one that combines several of them, e.g., element name, data type, auxiliary information. Combination is possible in two ways: a hybrid matcher that integrates multiple techniques, and a composite matcher that combines the results of independently executed matchers.

Hybrid matchers. These matchers are based on several techniques and employ multiple approaches. Basically such matchers exist because of two reasons:

- They produce better match candidates, than a single matcher, since take more than one criterion into account.
- They give better performance, than executing multiple matchers, by reducing the number of passes over the schema.

However, from the software design point of view, a hybrid matcher is the wrong choice, because it works like a black box. The order of how criteria are evaluated or how the parameter tuning affects the whole algorithm is difficult, if not impossible, to track.

Composite matchers. These matchers are more flexible than the hybrid ones. Although the performance can be lower, they are still supposed to produce better match candidates than individual matchers. From the software design perspective, composite matchers should be the preferred choice. They provide high degree of customization. One can change the order of matchers, make them work sequentially or in parallel, tune them individually, and always plug in a new one. This way the assortment of matchers grows to infinity. Interestingly, even a hybrid matcher can be plugged into a composite matcher, while it is not possible other way round.

As depicted in Figure 5, selection and combination of matchers can be done either automatically, or manually by a human user. An automatic approach can decrease user interference, but there is no general solution for that. Alternatively, a user can be allowed to directly select the matchers for execution, their order and how to combine their results. Such a manual approach leaves more control to the user, thus it is essential that the implementation were usable and comprehensive.

3.2 Representative Solutions

The classification of the traditional schema matching approaches suggests that there should be a range of solutions available. Indeed, many algorithms have been developed, a number of them are implemented. Some are even difficult to position into the above classification.

This section aims at summarizing certain solutions. They are chosen according to superiority among similar ones, popularity in the literature or specificity of the problem they target. The summary should provide with the general outlook of the traditional *schema matching* paradigm by pointing out the representative matchers, their merits and weaknesses as long as the matching of XML schemas is concerned.

3.2.1 LSD

The Learning Source Descriptions (LSD) system [13, 12] uses machine-learning techniques to match a new data source against a previously determined global (mediated) schema, producing a 1:1 mapping between leaf elements. It is a composite matcher which automatically combines match results.

There are several notions used in LSD that need to be translated into *schema matching* terms:

- *Learner*: a matcher that is combined into LSD. It is usually a simple matcher that uses only one piece of information. However, it can also be any hybrid or composite matcher.
- *Data source*: an input schema with instance-level information, also called source schema.
- *Mediated schema*: the schema that is built by the application developer for integration purposes, also called a target schema.

- *Labels*: elements of the mediated schema.
- *Source description*: it is a mapping between the source schema and the target schema.
- *Classification*: the matching process, i.e., assignment of a label to a source schema element.

The main idea of LSD is that after a set of data sources have been manually mapped to the mediated schema, the system should be able to glean significant information from the mappings and to successfully propose mappings for subsequent data sources. Hence, it is a two-phase procedure:

1. *Learining (Training).*
   It is an LSD preparation phase. After the user has specified several semantic mappings, learners are trained with the sample data.
2. *Classification (Matching).*
   In this phase the whole LSD matcher is ready to be executed. Here the actual matching for new source schemas is performed.

Since LSD is a composite matcher, it is easy to distinguish what combination of matchers, called learners, it uses. A learner is not necessarily a machine-learning based matcher. A learner is basically a matcher that can associate a label and an input schema element with some confidence value. There are two kinds of learners:

- *Base learners.*
  These learners produce match candidates by calculating confidence matrix for every label and every input schema element. LSD uses the following important base learners:
  - The name matcher: matches schema elements based on the similarity of their names, allowing synonyms.
  - The content matcher: classifies an input instance based on the labels of its nearest neighbors in the training set [9].
  - The naive Bayesian learner: exploits word frequencies.
  - XML (structure) learner: exploits the hierarchical nature of XML.
- *Meta learner.*
  This learner combines match results from the base learners. It uses a machine-learning method to determine the importance of each base learner to a specific label.

To sum up, LSD employs both the schema based and instance based matching approaches. It combines matchers in a composite way and thus provides high extensibility for additional learners. The authors show that it performs with rather high accuracy. A disadvantage of LSD is that it does not allow saving

its trained state. At least it is not clear how the system could maintain learned knowledge during several runs on schemas from different domains. For example, Automatch [3] uses an *Attribute dictionary* for such purposes.

3.2.2 CUPID

CUPID [27] is a hybrid matcher based on both element-level and structure-level matching. It is intended to be used for any kind of schemas in a *model management* framework (e.g., Microsoft BizTalk Mapper). CUPID has the following distinctive properties:

- Works solely with schema information.
- Includes automated linguistic matching.
- Works both on the element and structure levels.
- Is resistant to variations in the schema structure.
- Assumes that atomic elements (i.e., leaves) bear much schema semantics.
- Exploits keys and referential constraints.
- Makes context-dependent matches of a shared definition.
- Uses auxiliary information, e.g., thesaurus, initial mappings.

The algorithm deduces a mapping by computing similarity coefficients between elements of two input schemas. The coefficients, in the [0,1] range, are calculated in two phases : linguistic and structural. A mapping is created by choosing pairs of schema elements with maximal weighted similarity (a mean of $ssim$ and $lsim$):

$$wsim = w_{struct} \times ssim + (1 - w_{struct}) \times lsim$$

where $wsim$ is the weighted similarity, w_{struct} - the constant between 0 and 1, $ssim$ - the structural similarity coefficient, $lsim$ - the linguistic similarity coefficient.

Linguistic matching phase. This phase is based primarily on schema element names and constitutes of three steps:

1. *Normalization.*
   In this step schema element names are preprocessed with respect to acronyms, punctuations etc. With the usage of thesaurus the following is performed:
   - Tokenization, e.g., 'Product_ID' becomes {'Product', 'ID'}.
   - Expansion (identifying abbreviations and acronyms).
   - Elimination (discarding prepositions, articles, etc.).

2. *Categorization.*
   Schema elements in each schema are separately clustered. This is based on their data types, schema hierarchy and their names. For example, there might be categories for real-valued elements or for money-related elements. This step reduces the number of further comparisons.

3. *Comparison.*
   Linguistic similarity coefficients ($lsim$) are computed between schema elements by comparing the tokens extracted from their names.

The result of this phase is a matrix of $lsim$ coefficients between elements in the two schemas.

Linguistic matching phase in CUPID is pretty fully-fledged. However, because of the natural language ambiguity schema elements may be polysemous. One of the solutions for disambiguation is proposed in [28].

Structural matching phase. The second phase transforms the original schema graph into a tree and then does a bottom-up structure matching, resulting in a structural similarity between pairs of element. The transformation encodes referential constraints into structures that can be matched just like other structures. The intuition for computing structural similarity coefficients ($ssim$) is as follows:

- Leaves in the two trees are similar when they are individually (linguistic and data type) similar, and when elements in their respective vicinities (ancestors and siblings) are similar.

- Non-leaf elements are similar if they are linguistically similar, and the subtrees rooted at the two elements are similar.

- Non-leaf schema elements are structurally similar if their leaf sets are highly similar, even if their immediate children are not. This is due to the assumption that much semantics is in schema leaves.

The result of this phase is a matrix of $ssim$ coefficients between elements in the two schemas.

Although experiments show that CUPID performs better than a couple of other matchers, its structural matching could be improved, like, for instance, in [7].

3.2.3 Similarity Flooding

Similarity Flooding (SF) [29] is a schema matching algorithm that allows the quick development of matchers for a broad spectrum of different scenarios. Its simplicity implies easy implementation of the basic matching. The algorithm is based on the following idea. First, the input schemas are converted into directed labeled graphs. Then, these graphs are used in an iterative fixpoint computation

whose results determine node similarity. The intuition behind is that elements of two distinct schemas are similar when their adjacent elements are similar. In other words, a part of the similarity of two elements propagates to their respective neighbors.

The algorithm is performed in four steps:

1. G_1 = GenerateGraph(S_1), G_2 = GenerateGraph(S_2).
   During this step graphs are generated out of input schemas, which is easy in case of XML schemas, when there are no shared definitions.

2. initialMap = StringMatch(G_1, G_2).
   Here linguistic matching is performed. The algorithm is not restricted to any linguistic techniques and usually the output of this step is a similarity matrix for all nodes in the two graphs.

3. product = SFJoin(G_1, G_2, initialMap).
   This is the step, where the actual "similarity flooding" is carried out. The authors propose an iterative fixpoint computation and as a starting point to use the initial mapping from the previous step. Over a number of iterations the initial similarity of any two nodes propagates through the graphs and stabilizes, i.e., reaches the fixpoint. This step drastically reduces the size of similarity matrix by leaving out only mapping elements with the positive similarity.

4. result = SelectThreshold(product).
   Here the "most plausible" match candidates are selected. This step may require user interference.

Although SF sometimes performs worse than other matchers [7], luckily the algorithm is certainly simple and seems easily extensible. For example, the initial mapping could be produced with respect to more criteria, some semantics could be attached to graph nodes in order to faster reach the fixpoint. Moreover, the matching algorithm is able to capture m:n global mappings.

3.2.4 Corpus-based Matcher (REVERE)

Corpus-based matcher best fits traditional schema matching approaches that highly depend on auxiliary data. The idea reminds Information Retrieval (IR), namely, the statistical analysis of large corpora of text. Worth emphasizing that a corpus is *not* a logically coherent universal database. Rather, it is a collection of various uncoordinated constructions like:

- Schemas.
- Meta-data, i.e., additional information for schema nodes.
- Validated mappings.
- Instance data.

The challenge in corpus-based matching is to find interesting patterns via certain kinds of statistics extracted from corpus contents. This task seems more promising than building a detailed and comprehensive knowledge base [19]. It does not require a careful ontological design, as a knowledge base does, nor a careful control of its contents, thereby removing key bottlenecks of knowledge representation systems.

One corpus-based matcher is proposed in [26]. The general procedure comprises two functions: learning knowledge in the *Mapping Knowledge Base* (MKB) and performing match with respect to the MKB.

Mapping knowledge base. The MKB constitutes of two parts:

- Schemas and mappings that are continually added.
- Learned knowledge that is extracted for every schema element. This knowledge is captured by the use of *base learners* (classifiers):
  - Name learner.
  - Structure learner.
  - Data type learner.
  - Instance learner.
  - Description learner.

  Each base learner captures one matching criterion or use only one piece of information during training. The summary of all base learners is made by the *meta learner*.

Using the MKB for matching. In order to find out if elements of S_1 and S_2 are similar, *interpretation vectors* $\vec{P}_i = \langle p_{im} \rangle$ are calculated, where p_{im} is a similarity degree between e_i from S_1 (or S_2) and an element c_m from the MKB. The similarity degrees are results of the *meta learner*.

The elements e_i and e_j, of schemas S_1 and S_2 respectively, are similar if they cannot be distinguished using their interpretations. Hence, it is enough to compare two vectors $\vec{P}_i$ and $\vec{P}_j$ in order to decide if e_i and e_j are similar. Different measures are possible for doing comparison, such as vector dot product, average weighted difference or average significant weighted difference.

Proposed corpus-based matcher [26], like LSD [13, 12], uses the schema based and instance based approaches. It is a composite matcher that allows making combinations with any kind of other matchers by the use of base learners. However, with the growth of the MKB computational efficiency of this matching algorithm is sacrificed. Moreover, although corpus can contain many schemas, the corpus-based matcher still belongs to the traditional schema matching paradigm.

3.2.5 Ontology-based Matcher

This kind of a matcher works with an incredible increase in matching precision [40]. Beside *direct matches*, an ontology-based matcher is able to reveal *indirect matches*, which are usually left aside in other matching algorithms. However, important is the fact that building an ontology can be a difficult and time-consuming task.

One ontology-based matcher is proposed in [40]. The algorithm uses its own peculiar way for representing both schema elements and conceptual model (light-weight ontology), namely, object-relationship modeling [17]. However, it has little to do with current knowledge representation methods, such as Description Logics (DLs). Therefore inference of *indirect mappings* is actually somewhat different than would be the reasoning in a DL.

In the schema representation elements are either:

- *nonlexical object sets*, meaning non-leaf schema elements, or
- *lexical object sets*, meaning leaf schema elements.

A lexical object set (leaf schema element) has a *data frame* assigned, which can be a thesaurus for the element name or a set of regular expressions. This enables to assign more semantics to the element, but requires additional work on the schema.

The ontology has to be developed for every application domain that input schemas come from. Although it is quite an overhead, using ontologies brings some advantages. For example, indirect matches are easier captured, precision is improved, matches of any cardinality can be obtained. The proposed algorithm allows only very simple ontologies. They are restricted to the following concept roles:

- *Merge/Split*, as shown in Figure 7 (left), the concept 'Address' constitutes of 'Country', 'City' and 'Street' concepts. Choosing merge or split relation depends on the direction of matching.
- *Superset/Subset*, as shown in Figure 7 (right), the concept 'Phone' is a concept 'Home phone', 'Office phone' or 'Cell phone'. Again it depends on the direction of matching which role is used.
- *Property*, as shown in Figure 7 (right), the concept 'Phone' has a property 'Color'.

In order to add more expressiveness to the ontology, concepts are enriched with regular expressions.

The algorithm takes as an input source and target schemas, instance-level information, and an ontology. First, schema elements are matched to ontology concepts. Then, derivation algorithm is executed on those concepts in order to decide similarity of underlying schema elements.

Beside building the ontology, the following disadvantages of this kind of a matcher can be listed:

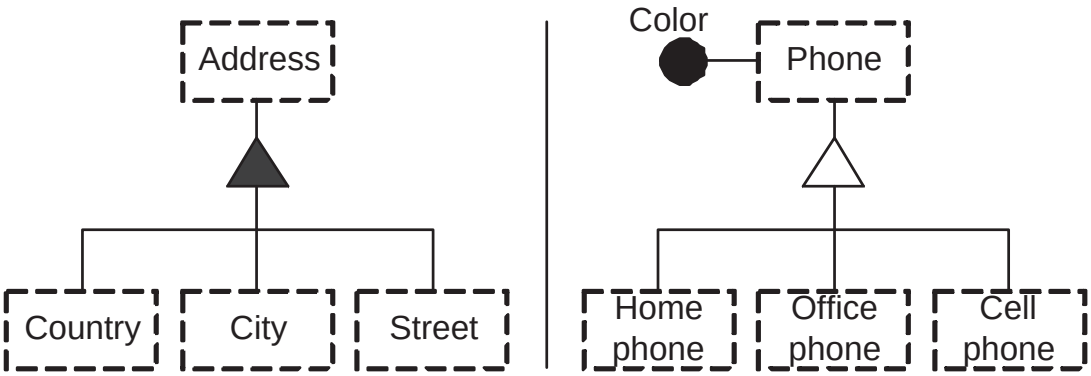

Figure 7: Some application domain partial ontology

- Unclear how it correlates with current knowledge representation formalisms, such as DLs.
- Linguistic part is quite weak.
- Is context insensitive. For example, in Figure 7, 'Address' could be an employee address or a company address even in the same application domain.
- Almost no use of structure-level schema information, e.g., non-leaf elements.

3.2.6 COMA++

COMA [11] is a composite matching system that allows combining matching algorithms in a flexible way. It provides an extensible library of matchers and supports different ways for combining match results. New match algorithms can be included in the library and used in combination with other matchers. COMA thus allows evaluating matching strategies by selecting various match algorithms and their combinations.

This system served as an inspiration for building an industrial-strength schema matching system Protoplasm [5]. Now COMA evolved into a new, powerful version called COMA++ [1]. Many improvements from recent research are present there, such as support for large schemas, ontology matching, GUI, etc.

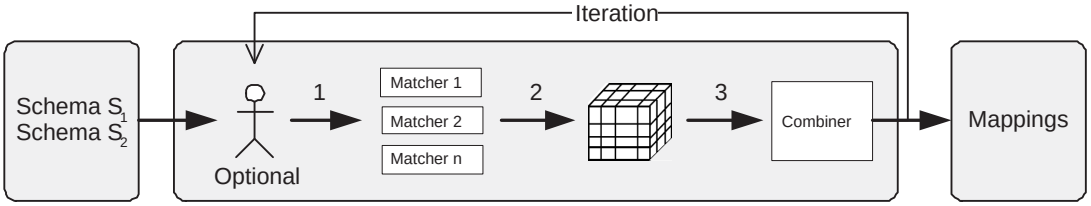

Figure 8: Basic COMA architecture

Figure 8 illustrates match processing in COMA on two input schemas S_1 and S_2. Match processing either takes place in one or multiple iterations depending

on whether an automatic or interactive determination of match candidates is to be performed. Each match iteration consists of three phases:

1. *User feedback phase (optional).*
   In interactive mode, the user can specify the match strategy (selection of matchers, of strategies to combine individual matcher results), define match or mismatch relationships, and accept or reject match candidates proposed in the previous iteration. The interaction is useful to test and compare different match strategies for specific schemas and to continuously refine and improve the match result. In automatic mode, the match process consists of a single match iteration for which a default strategy, or strategy specified by input parameters, is applied.

2. *Execution of different matchers.*
   During this phase, multiple independent matchers are chosen from the matcher library and executed. They exploit different kinds of schema information, such as element names, data types, structural properties, auxiliary information, e.g., as synonym tables and previous match results. Each matcher produces a similarity matrix.

3. *Combination of the individual matcher results.*
   The final phase is to derive the combined match result from the individual matcher results stored in the similarity cube. The produced mapping can be saved for further reuse.

COMA++ has two matching strategies: reuse-oriented and fragment-based. The former heavily relies on the corpus-based matcher [26] covered earlier. The latter is inspired by [32], where matching of large schemas is considered. The main idea of the fragment-based strategy is to first identify similar fragments of schemas and then match them. This way the problem is decomposed into smaller subproblems, which allows increasing the performance.

3.2.7 eTuner

The eTuner [33] solution is proposed with the motivation that tuning of matching algorithms is skill- and time-intensive task. There is barely any matcher that would not require setting some threshold values or a bunch of other coefficients. Therefore, with a wide assortment of matchers the need for the automatic parameter (knob) tuning emerges. As shown experimentally, this contributes to the higher accuracy in matching than using off-the-shelf matchers straightforwardly.

The proposed solution outputs the tuned matching system in two steps:

1. *Workload generation.*
   In this step the source schema S_0 participates in production of *synthetic* matching scenarios. Specifically, a set of common transformation rules is applied to the schema and its instance data in order to generate a collection of synthetic schemas S_1, S_2, ..., S_n. For example, a transformation

rule could look like: "abbreviating a name to the capital letters". After application of such a rule to S_2 from Figure 2, 'PurchaseOrder' would be changed into 'PO', 'UnityOfMeasure' - into 'UOM', etc.

Both synthetic schemas and transformation rules can come as the user input. Having synthetic schemas the set of schema pairs is constructed:

$$\{(S_0, S_1), (S_0, S_2), ..., (S_0, S_n)\}$$

Due to the fact that correct matches in each pair are known, the average accuracy for each knob can be computed.

2. *Staged tuning.*
   During this step an appropriate tuning is selected. The space of possible knob configurations is huge, therefore eTuner uses a greedy, sequential approach.

As experimentally shown by the authors, automatic tuning is feasible. However, the procedure takes about 30 minutes to execute (on examples). Therefore, automatic tuning has to be run off-line.

3.3 Summary of Traditional Matching Techniques

Matching algorithms compute similarity for elements in two schemas. The computation is performed with respect to a certain criterion by employing some technique. For instance, in order to compare two element names, they can be linguistically normalized first, like in CUPID [27].

In this section summary of matching techniques, offered by the research community, is given. This summary should provide with the deeper insight to the traditional schema matching than the general overview of approaches and core solutions. It is built on three groups of techniques [14] according to (1) how the similarity is measured, (2) what information is used during the matching and (3) architecture of the matching system.

3.3.1 Measuring Similarity

Determining semantic similarity of two schemas is the main goal of the traditional schema matching. The similarity is usually outputted as a two dimensional matrix, where the intersection contains the similarity value for the row and column schema elements.

All the matching techniques aim at increasing the precision of the matrix, i.e., the more similar elements are, the closer similarity value is to 1, in contrast, the more dissimilar elements are, the closer similarity value is to 0. Formally, similarity matrix is the relation:

$$SIM : S_1 \times S_2 \rightarrow [0, 1],$$

where SIM is the similarity relation, S_1, S_2 - input schemas (sets of schema elements).

Similarity of schema element names. The techniques that use schema element names for computing SIM mostly attempt to compare the names from the linguistic perspective. However, extracting semantics from element names is a rather difficult task, syntax is not always enough. The following techniques are used to determine similarity of schema element names:

- *XML namespace awareness.*
  Two syntactically equal elements from the same namespace bear the same semantics.

- *Using similarity metrics of strings.*
  This technique assumes that the syntax is the same as the semantics.

- *Preprocessing.*
  It may be beneficial to preprocess element names like, for example, CUPID does tokenization, expansion and elimination.

- *Categorization.*
  Elements in the same category (e.g., according to the data type) are considered more similar than in different ones.

- *Using only atomic level elements.*
  For example, CUPID assumes that it is more important to compare leaf element names, because they bear more semantics.

- *Synonym awareness.*
  Elements are similar if their names are synonyms of one another.

- *Hypernym awareness.*
  Elements are similar if their names are in *is-a* relation with respect to one another.

- *Phonetic awareness.*
  Element names may sound the same rather then look the same, for instance, 'ShipTo' and 'Ship2'. COMA computes phonetic similarity using Soundex (available in SimMetrics [35]).

- *Resolution of homonyms (disambiguation).*
  There are words (strings) that look the same but may mean different things depending on the context. For example, element name 'stud' may mean a student, but also a male horse. On the one hand the context can be resolved using structural information (e.g., 'university.stud', 'farm.stud'), on the other, the user can provide auxiliary information.

- *Using a user provided (mis)match vocabulary.*
  The user interference is crucial for determining semantic matches. For example, an acronym 'ZZZ' may mean something in certain domain, and it is not likely that this meaning can be resolved without the user.

- *Translation.*
  Element names can be in different languages.

- *Learning.*
  An algorithm can be trained to recognize similar names using machine-learning methods like, for instance, in LSD.

- *Derivation using an ontology.*
  An ontology allows detecting indirect matches between elements. For example, it could help to resolve matches according to *has-a* relationship.

Using schema element meta-data and constraints. There are matching techniques that use other information related to schema elements than only element names. This information can be attached as meta-data or specified as constraints. For example, XML schemas allow defining elements as entities of integer type within some interval. The techniques are characterized by the meta-data and constraints they use, however this information may not always be available:

- *XML namespace.*
  The namespace adds semantics for schema elements, because it specifies their context.

- *Data type and value ranges.*
  For instance, a data type *date* will hardly be compatible with negative integers, this helps to discard certain matching candidates.

- *Uniqueness, cardinality, optionality, references.*

- *Descriptions.*
  Performing some natural language processing on descriptions or comments of schema elements may reveal similarity of elements themselves.

- *Regular expressions.*
  Regular expressions can be used as recognizers of element names or contents [40]. For instance, if two expressions accept $ as the first symbol, it may mean that the respective elements deal with currencies.

- *Domain constraints.*
  LSD uses a notion of domain constraints to define rules restricting the mapping from unwanted match candidates.

Similarity at the structure-level. When dealing with XML schemas, structure of the document that the schema defines is a valuable piece of information. For example, it can help to disambiguate element names. The main techniques that exploit structure-level information are:

- *Learning the structure.*
  LSD proposes a machine-learning based algorithm for determining structural similarity in XML schemas.

- *Propagation of the similarity.*
  SF is based on structure-level information, because it adjusts SIM by propagating the similarity of adjacent nodes.

- *Putting stress on schema leaves.*
  CUPID uses the intuition that two non-leaf schema elements are structurally similar if their leaf sets are highly similar.

- *Word sense based.*
  A solution proposed by [28] uses word sense disambiguation achievements in computational linguistics and applies them to decide the similarity of hierarchical structures.

- *Node context metrics.*
  Every node in a schema can be associated with a context: ancestor-context, child-context, leaf-context. Two schema structures can be matched according to some metrics [7], such as *longest common subsequence*, *average positioning*, etc.

- *Schema embedding.*
  As studied by [6] the problem of embedding a source schema into a target schema, while preserving information and conforming to the target schema grammar, is NP-complete but can still be tractable.

Similarity of instance data. Sometimes the schema has to be guessed from an XML document, this way a lot of schema-level information is lost or is incorrect. It may also happen that schema-level information, in general, is not enough for the matching. Then, instance data is used to obtain more semantics which would help producing match candidates. The following kinds of techniques exploit instance-level information:

- *Information Retrieval (IR) based.*
  This kind of techniques are based on various string similarity metrics, i.e., statistics.

- *Value recognition based.*
  Elements can be assigned a recognizer, e.g., regular expression, which would be decisive during the matching. For example, a recognizer may be able to distinguish that the element content is a ZIP code or a phone number.

Bottom-up and top-down matching. A matching algorithm can traverse the hierarchical structure either *top-down* or *bottom-up*. It may be necessary to first transform the XML schema graph into such structure.

A top-down algorithm is usually less expensive than bottom-up, because matches at a high level of the schema structure restrict the choices for matching finer grained structure only to those combinations with matching ancestors. However, a top-down algorithm can be misled if top-level schema structures are

very different, even if finer grained elements match well. By contrast, a bottom-up algorithm compares all combinations of fine grained elements, and therefore finds matches at this level even if intermediate and higher level structures differ considerably.

Other techniques. Other techniques used in the traditional schema matching:

- *Graph-to-tree transformation.*
  XML schemas have a graph structure. CUPID transforms it into a tree taking into account shared types and referential constraints.

- *Pruning leaves.*
  In order to increase performance it may be beneficial to analyze input schemas until some depth and ignore the rest subtrees.

- *Using a corpus.*
  The motivation for this technique is to avoid the overhead of creating a formal ontology and still maintain knowledge in form of a pool of mappings, schemas, precalculated matches, etc.

- *Fragmenting schemas.*
  Usually it is better to divide one match problem into several smaller subproblems [32, 1].

- *Flattening the similarity cube.*
  When a combinational matcher is used, then during execution of the algorithm every matcher produces its own SIM. As the final result of the matcher is only one SIM, a computation resolving (weighted) average similarity must be performed.

- *Using auxiliary information.*
  Auxiliary information is needed to improve matching accuracy, because the information available from a schema, and even its instance, is not enough. For example, user input, thesaurus, dictionary, etc. are considered as auxiliary information.

3.3.2 Auxiliary Information Used in Matching

Auxiliary information, used in addition to input schemas, may improve matching accuracy. There is the following auxiliary information and respective techniques:

- *Information available from linguistic tools.*
  Many matchers count on linguistic information that comes from lists of synonyms, hypernyms, dictionaries, thesauri, etc. Some of them are supposed to be adapted to the domain of interest or a company, e.g., a company uses its own set of acronyms.

- *User provided (mis)matches and user feedback.*
  The input of a user during matching is crucial, because finally the user has to validate the mapping. There is no common strategy for the user's interference but [36] emphasizes that it is more beneficial to ask the user for input in critical points, when no adequate match can be determined automatically, than to allow only pre/post interaction.

- *Previously determined mappings and common schema components.*
  Reusing mappings is useful when separately matching similar schemas to the same target schema, as shown in Figure 6. In addition, it is beneficial to maintain a repository of schemas or schema fragments, which is very rewarding for frequently used entities, such as address, employee, customer, etc.

- *Domain ontology.*
  A domain ontology is valuable for the possibility to determine indirect matches [40].

- *Examples.*
  When a machine-learning technique is implemented it usually needs sample data for training.

- *Schema element meta-data.*
  Schema elements can be assigned some meta-data useful for the matching, for example, a regular expression recognizer for contents of an element.

3.3.3 Matching System Architecture

There are many different individual matchers. The emerging consensus is that matchers should be combined [14]. The most promising way to combine them is using the composite approach, because it provides with the great modularity and easy customizability. A composite matcher is frequently titled as a matching system, and the research community is trying to solve many issues concerning its architecture. Indeed a matching system is no more an isolated algorithm, but rather an entity related to the user and external sources (see Figure 9).

Combining matchers. As discussed earlier, combinational matchers are of two kinds: hybrid and composite. There exists a trade-off between the flexibility of composite matchers and efficiency of hybrid ones. The best combination seems to be a matching system that is combined of as much as possible hybrid matchers, or a set of matchers producing mappings domain-independently and a set of customized ones.

When matchers for the system are chosen automatically, then machine-learning methods usually are feasible, e.g., like in LSD. On the other hand, choosing a combination of matchers manually requires the system to be able to give hints and feedback on performance and accuracy, e.g., like in COMA++. Pre-tuning matchers, e.g., with eTuner, before deciding whether to combine them

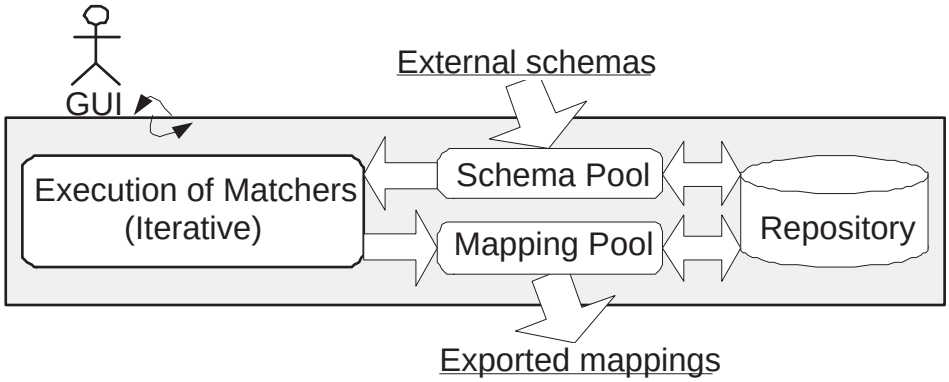

Figure 9: A matching system

into the matching system can also be useful. In general, tailoring a matching system highly depends on the matching problem, as well as on the application domain.

Choosing a strategy. The strategy defines the perspective, or the core feature, of a matching system with respect to a matching problem. When matchers for the system are set up, i.e., combined, customized and tuned to solve the matching problem, the system can be said to maintain some strategy. Possible strategies could be:

- *Reuse oriented*: the matching system heavily relies on common schema components, previous match results, or corpus in general.
- *Fragment oriented*: at the core of the matching system the schema fragmentation (divide-and-conquer) technique is used.
- *Schema-only information oriented*: schema-level information plays the major role in the matching system.
- *Machine-learning driven*: for example, the LSD matching system.

Choosing a strategy basically reduces to choosing the best matching system configuration. The criteria for that can be different, depending on the requirements, but usually range from performance to accuracy limits.

External sources, pools and repositories. As a matching system uses many techniques, it is no more a standalone and isolated procedure. If it complies to some interfaces then schemas and mappings can be imported from external sources. Moreover, working with many schemas and mappings at once requires pooling them and saving or exporting them afterwards.

User interaction. A matching system can not be autonomous, therefore to create a correct mapping it requires user interaction. Beside the usual user interference, when the user needs to specify (mis)matches in advance and accept, reject or validate matches, there is more, for instance, the user may need to configure connections to thesauri, repositories or external sources. Also the user may need to maintain schema and mapping repositories to be up to date. Hence the graphical user interface (GUI) has many requirements which should not be underestimated. Some important requirements are:

- The user must be asked to provide input when the system cannot resolve a match automatically (e.g., when it is ambiguous) [36], and not wait for the next iteration.
- The maintenance process must be driven by hints. For example, when the user tries to change a schema element name she could be provided with a list of "well matchable" values from the repository of schemas.
- As the matching process takes a comparably long time to finish, the user must be able to watch the progress. It is also applicable for other functionality, such as importing or exporting schemas. Moreover the user should be able to safely break long lasting operations at any point. For example, a disadvantage of the COMA++ prototype is that its GUI "freezes" during the matching.

4 Holistic Schema Matching Solutions

The large scale schema matching is a recent topic. There are not so many solutions proposed by the research community and all of them are targeting the *deep Web* problem of integrating query interfaces. The main approaches in the holistic schema matching are:

- *Statistical matching.*
  This approach is purely statistical. According to it there exists a hidden statistical model that can be discovered using input schemas.
- *Matching as mining.*
  This approach is an application of data-mining in *schema matching*. The input schemas are prepared as a database on which, for instance, association mining or clustering is performed.

Beside the listed approaches, there is a solution that targets the removal of the noise in the input schemas. The so called *Ensemble* approach [22] treats some of the input schemas as the noise. As this solution does not exactly concern the matching, it is not covered below, among the holistic paradigm representatives. It is rather considered later as a technique.

4.1 The MGS Framework

The MGS (hypothesis Modeling, Generation and Selection) framework is proposed by [20]. In this framework all the attributes from input schemas are viewed holistically (altogether as a set) and are believed to have a statistically consistent underlying model. The framework is a three-step procedure:

- *Hypothesis modeling.*
  During this step the specification of hypothetical *hidden model* is created. In the *schema matching* context the hidden model could be a mapping, where the similarity relation in matches is defined as a synonym relation.

- *Hypothesis generation.*
  In this step all consistent, with respect to input (observed) schemas, models are generated. In other words, the possible and statistically reasonable mappings are created. The hypothesis generation should not be mixed with the schema generation from the hypothesized model.

- *Hypothesis selection.*
  Here models that have a sufficient statistical consistency with the observed schemas are selected by employing a hypothesis testing method, e.g., $\mathcal{X}^2$. This way a set of candidate mappings is chosen.

This solution is motivated by already mentioned proliferating sources and converging vocabularies that allow hypothesizing the existence of a hidden schema model that probabilistically generates the observed schemas from a finite vocabulary, i.e., from a finite set of attributes. Hence the matching problem can be renamed to the hidden model discovery, more specifically, synonym discovery, using MGS framework.

The framework, with some additional techniques, is implemented in [20] as an algorithm under the title MGS_{sd}. It is exponential with respect to the number of attributes in observed schemas. Nevertheless, experiments show that performance is still reasonable. Hence, with the assumptions that statistical schema matching is performed off-line, vocabularies are converging, and, in addition, empirical results demonstrate very good accuracy, MGS_{sd} seems feasible.

Although the solution is very abstract and promising, there are some issues:

- It captures only 1:1 mappings and it is unclear how to extend the framework for more complex mappings.

- It could leverage some techniques from traditional paradigm by employing similarity relation instead of synonym relation (i.e., similarity equal to 1).

- There is an assumption that input schemas are from the same domain. This allows, to a certain extent, avoiding issues related to semantical correctness of mappings, e.g., homonyms and hypernyms should be very rare. However, the framework relies on syntactic information and the only semantics it gets is in the form of statistics.

4.1.1 Hypothesis Modeling, Generation and Selection

It was mentioned that the MGS framework is used for the synonym discovery. The details about the framework can be found in [20], here the main ideas of how it works are outlined.

Hypothesis modeling. The schema (hidden) model, in MGS for synonym discovery, describes how to generate schemas from a vocabulary of attributes. A schema is considered as an unstructured set of attributes where the synonymous ones are called concepts.

Definition 4.1
The schema model is a 4-tuple $\mathcal{M} = (\mathcal{V}, \mathcal{C}, P_c, P_a)$, where:

- $\mathcal{V} = \{A_1, A_2, \cdots, A_n\}$ is the *vocabulary* of attributes from input schemas.
- $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$, such that $\mathcal{V} = \bigcup C_k$ and $C_i \cap C_j = \emptyset$, is the *concept partition.*
- P_c is the *concept probability function*, which determines the probability α_i for including concept C_i in schema generation.
- P_a is the *attribute probability function*, which determines the probability β_{j_k} for selecting attribute A_{j_k} , once the concept C_j is included. It also holds that $\sum_{A_l \in C_j} \beta_l = 1$.

According to Definition 4.1, schema generation is the process of selecting a concept and an attribute representing this concept. There are two assumptions for this procedure:

- Concept mutual-independence, i.e., concepts are selected independently.
- Synonym mutual-exclusion, i.e., no two synonyms both are selected.

Hence, the probability of selecting attribute A_j from the model $\mathcal{M}$ is

$$Pr(A_j|\mathcal{M}) = \begin{cases} \alpha_i \times \beta_j & , \exists i : A_j \in C_i \\ 0 & , otherwise \end{cases}$$

Example 4.1
Consider the example provided in [20] and depicted in Figure 10. It visualizes one of the possible schema models $\mathcal{M}_B = (\mathcal{V}_B, \mathcal{C}_B, P_c, P_a)$ for the book domain:

- $\mathcal{V}_B$ ={author, title, ISBN, subject, category}.
- $\mathcal{C}_B = \{C_1, C_2, C_3, C_4\}$, where
  - $C_1 = \{\text{author}\}$,

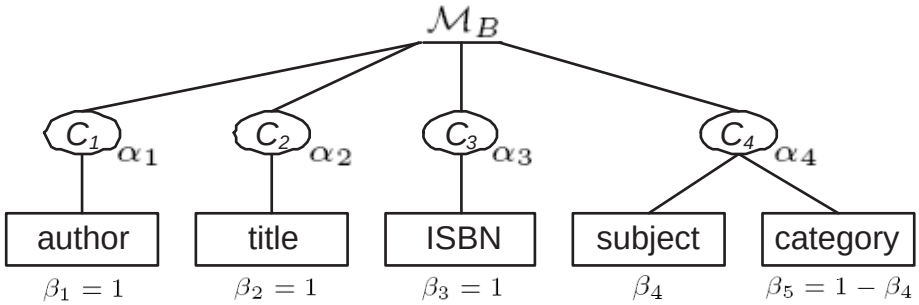

Figure 10: An example of schema model $\mathcal{M}_B$

  - $C_2 = \{\text{title}\}$,
  - $C_3 = \{\text{ISBN}\}$,
  - $C_4 = \{\text{subject, category}\}$.

- $P_c = \alpha_i$, for choosing C_i.
- $P_a = \beta_j$, for choosing A_j.

The above can also be written as $\mathcal{M}_B = \{(\text{author}{:}\beta_1){:}\alpha_1,\ (\text{title}{:}\beta_2){:}\alpha_2,\ (\text{ISBN}{:}\beta_3){:}\alpha_3,\ (\text{subject}{:}\beta_4,\ \text{category}{:}\beta_5){:}\alpha_4\}$. This model could generate such schemas as:

- $I_1 = \{\text{author, title, subject, ISBN}\}$,
- $I_2 = \{\text{title, category, ISBN}\}$,
- other.

The probability with which a schema model generates a schema is determined by P_c and P_a. So the interesting models are those that can generate all the input schemas with a probability more than 0, written as

$$Pr(\mathcal{I}|\mathcal{M}) > 0,$$

where $\mathcal{I}$ is a set of input schemas $\{I_1, I_2, \cdots I_n\}$. The hypothesis generation step of MGS deals with production of such models.

Hypothesis generation. Intuitively, given a set of observations (schemas) $\mathcal{I}$, candidate models, i.e., hypothesis space $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_n$, has to be reconstructed. It must hold that the models are *consistent* with the observed schemas, i.e., $Pr(\mathcal{I}|\mathcal{M}_i) > 0$. Further, since a model vocabulary coincides with the observed schemas (union of sets of attributes), i.e., $\mathcal{V} = \bigcup\{\mathcal{I}\}_i$, the remaining task is to build concept partitions $\mathcal{C}$ and to find out probability functions P_c and P_a for every model in the hypothesis space.

Knowing that only consistent models are needed and that exhaustive vocabulary partitioning is not feasible, since the amount of possible concept partitions

grows faster than exponentially with respect to $|\mathcal{V}|$, one observation is made: there are concept partitions that may have concepts making $Pr(\mathcal{I}|\mathcal{M}) = 0$. Such concept partitions are called inconsistent with $\mathcal{I}$ and should be ignored to prune the search. Two points hold for inconsistent concept partitions:

1. $\exists$ schema $I \in \mathcal{I}$, such that $A_j \in I$ and $A_k \in I$,
2. $\exists$ concept $C_i \in \mathcal{C}$, such that $A_j \in C_i$ and $A_k \in C_i$.

Example 4.2
Consider the model $\mathcal{M}_B$ given in Figure 10. The concept partition of $\mathcal{M}_B$ would be inconsistent with $\mathcal{I}$ if it contained, for example, such schema: $I_i = \{$author, subject, category$\}$, apparently 'subject' and 'category' would mean different things in this case.

As far as probability functions P_c and P_a are concerned, it is straightforward that the greater the probability values the "better" the candidate model. Thus it is basically a maximization problem

$$\max_{P_c,P_a} Pr(\mathcal{I}|\mathcal{M}(\mathcal{V},\mathcal{C},P_c,P_a)),$$

which is a *maximum likelihood estimation* problem for parameters α_i and β_j (Definition 4.1).

Hypothesis selection. At this point the hypothesis space is generated. Every model $\mathcal{M}$, if viewed statistically, gives a distribution of generated schemas. The authors propose to use $\mathcal{X}^2$ method for hypothesis testing and this way select the best models, i.e., the most consistent with the input schemas. The general idea behind $\mathcal{X}^2$ is the following. The input schemas in the set $\mathcal{I}$, are considered as statistically legal observations. Hence they can appear repeatedly in $\mathcal{I}$ with some frequency, i.e., they follow a statistical distribution. The aim is to compare the distribution of generated schemas, provided by $\mathcal{M}$, and the observations distribution, i.e $\mathcal{I}$. The more they are alike, the more $\mathcal{M}$ is consistent with $\mathcal{I}$, and in contrary, the less they are similar, the less $\mathcal{M}$ is consistent with $\mathcal{I}$.

4.2 The DCM Framework

The DCM (dual correlation mining) framework attacks the m:n matching problem in a holistic way. It is developed by [23] as an alternative to the MGS framework. Instead of the purely statistical approach, here matching is treated as a data-mining problem, where attribute co-occurrence patterns are investigated, i.e., association rules among attributes from different input schemas are being discovered. In data-mining the association rules are mined from a set of *transactions*, which in DCM case is a set of schemas. Moreover, a transaction is a set of items, which in DCM case is a set of *attribute entities* derived from a certain schema. By an attribute entity it is meant an entity that contains attribute name, label, type, instance data, etc. Hence discovering correlations of

transaction items is comparable to discovering correlations of schema attributes. Differently from usual correlation mining, in DCM not only the positive correlation matters, but also the negative one. The authors observe that *grouping attributes*, i.e., attributes in grouping relation with each other, e.g., {fname, lname}, tend to be positively correlated (co-present) across input schemas, while *synonym attributes*, i.e., attributes (or attribute groups) in similarity relation with each other, are negatively correlated because they rarely co-appear across input schemas, so, for example, {author} and {fname, lname} should match.

At the core of the DCM framework is the three-step *correlation mining* procedure:

1. *Group discovery.*
   Positively correlated attributes are mined to form potential attribute groups. Singleton groups are those that have only one attribute.

2. *Match discovery.*
   Given the potential groups, negatively correlated attribute groups are mined to form potential n-ary matchings.

3. *Match selection.*
   The most confident and consistent candidate matches are selected to form the final mapping.

In general the accuracy of DCM, as shown empirically [21], is similar to that of MGS. Additionally, the DCM implementation is more efficient than MGS_{sd} and it reveals complex mappings, not only 1:1.

4.2.1 Correlation Mining

Both the positive and negative correlation computation has the same algorithm. Indeed, first and third steps compute correlations, but depending on the input it is either positive or negative:

1. Compute (positive) correlations on schemas, i.e., discover groups,

2. Add a group to the schema when the intersection is nonempty,

3. Compute (negative) correlations on updated schemas, i.e., discover groups that match.

This is due to the observations in [23] on grouping attributes (they have to be positively correlated) and synonymous groups (they have to be negatively correlated). Clearly, the main issue is to find a correlation measure m that is suitable for schema matching purposes. Specifically, m has to be used in correlation mining and enable the "apriori" feature (i.e., downward closure):

$$Corr(\{I_1, I_2, \cdots, I_n\}, m) \leq Corr(\{I_1, I_2, \cdots, I_n\}^*, m),$$

where I_i is any transaction item (i.e., an attribute (group)), and the $*$ denotes any subset. The authors propose to use minimum function for $Corr$ which, indeed, has the "appriori" property:

$$C_{min}(\{I_1, I_2, \cdots, I_n\}, m) = \min m(I_i, I_j), \forall (i \neq j),$$

intuitively, the minimum of a larger set (e.g., min({1, 2, 3})) cannot be higher than any of its subsets (e.g., min({1, 3})).

For defining correlation measures contingency tables are used. Specifically, given a set of schemas and two specified attributes A_p and A_q, there are the following possible combinations of A_p and A_q with respect to a schema S_i:

	$A_p \in S_i$	$A_p \notin S_i$	
$A_q \in S_i$	f_{11}	f_{10}	$f_{1+} = f_{11} + f_{10}$
$A_q \notin S_i$	f_{01}	f_{00}	$f_{0+} = f_{01} + f_{00}$
	$f_{+1} = f_{11} + f_{01}$	$f_{+0} = f_{10} + f_{00}$	$f_{++} = f_{+1} + f_{+0} + f_{1+} + f_{0+}$

,

where each f_{ij} is a number of occurrences of each situation. There are different correlation measures in the literature, for example, a $Lift$ measure in association rule mining is:

$$Lift = \frac{f_{00} \cdot f_{11}}{f_{10} \cdot f_{01}}.$$

However, due to the specificity of the schema matching problem, in the DCM framework a new H measure is developed:

$$H(A_p, A_q) = \frac{f_{01} \cdot f_{10}}{f_{+1} \cdot f_{1+}}.$$

The value of H ranges from 0 to 1. An H value close to 0 denotes a high degree of positive correlation; an H value close to 1 denotes a high degree of negative correlation. So using this measure the corresponding correlation measures used for mapping discovery are defined:

$$m_n(A_p, A_q) = H(A_p, A_q),$$

$$m_p(A_p, A_q) = \begin{cases} 1 - H(A_p, A_q), & \text{when } \frac{f_{11}}{f_{++}} < T_d \\ 0, & \text{otherwise,} \end{cases}$$

where m_p denotes positive, and m_n negative correlation measures. They are used as parameters for the correlation mining algorithm. Notice that m_p has a threshold T_d which helps to filter out false groupings. Beside this threshold, two more, i.e. T_p and T_n, are used in DCM. They are used during the correlation mining like this:

$$C_{min}(\{A_1, A_2, \cdots, A_n\}, m_p) \geq T_p,$$
$$C_{min}(\{G_1, G_2, \cdots, G_m\}, m_n) \geq T_n,$$

where A_i denotes an attribute, and G_j - a grouping.

After the candidate matches are discovered, the selection of the best ones has to be performed. Here the issue is how to determine which matching is better, in other words, there is a need to rank the matchings. The authors propose such a match scoring measure:

$$C_{max}(M_j, m_n) = \max m_n(G_{j_s}, G_{j_t}), \forall(G_{j_s}, G_{j_t} \in M_j), \text{ such that } j_s \neq j_t,$$

where M_i is a candidate match, G_{i_k} - one of its attribute groups (items), m_n is the negative correlation measure, used in the match discovery. Intuitively, the higher the score of a match, the more correlated are its attribute groups, i.e., the stronger is the similarity among its attribute groups. When the scores are the same, the semantical subsumer between two of them is ranked higher.

4.3 Clustering-based Matcher

A clustering-based holistic matcher is proposed by [38]. The solution is able to capture 1:n mappings. However, at the heart of this solution is the clustering algorithm, which allows resolving 1:1 mappings given, as an input, the similarity matrix of the source schemas. The similarity matrix here is somewhat different than in the traditional paradigm, i.e., it takes all the schemas into account:

$$SIM : (S_1 \cup S_2 \cup \cdots S_n) \times (S_1 \cup S_2 \cup \cdots S_n) \rightarrow [0, 1],$$

where SIM is the similarity relation, S_i - input schemas (sets of attributes). The similarity between attributes is computed as the aggregate similarity of:

- *Linguistic similarity*: basically, this similarity is determined with respect to the normalized (tokenized and transformed) attribute names using some string similarity metric.
- *Domain similarity*: here instance-level information is considered, e.g., data types, attribute values.

More formally the aggregate similarity is written as

$$\mathcal{AS}(A_1, A_2) = \lambda_{ls} \cdot \text{lingSim}(A_1, A_2) + \lambda_{ds} \cdot \text{domSim}(A_1, A_2),$$

where λ_{ls} and λ_{ds} are the respective weights for linguistic similarity (lingSim) and domain similarity (domSim).

The clustering algorithm itself is rather easy to implement. The general idea is that the similar attributes fall into the same cluster. Initially, every attribute forms a cluster, then the most similar clusters are iteratively chosen to be merged into a new cluster. The main issue here is the evaluation of similarity of two clusters. The authors propose the following function:

$$Sim(c_1, c_2) = \max \mathcal{AS}(A_i, A_j), A_i \in c_1, A_j \in c_2.$$

Intuitively, the attributes from the cluster c_1 are compared to the attributes in the cluster c_2 and the maximal result is used to represent the similarity of the

two clusters. When all the clusters are compared pairwise, two of them, with the highest similarities, are merged.

The clustering algorithm is not responsible for the complex (1:n) mappings. The authors propose to resolve the hierarchical structure of input schemas first, and then additionally use the clustering algorithm result to produce the final 1:n mapping. The hierarchical structure of input schemas provides with the attribute groupings in advance, therefore its resolution should better be delegated to data preparation tools, e.g., wrappers.

One distinctive feature of the clustering-based matcher proposed by [38] is its semi-automatic nature. The authors employ the supervised machine learning for parameter tuning. Moreover, the user is asked to deal with uncertain situations for determining possible homonyms, synonyms and 1:n mappings.

4.4 Summary of Holistic Matching Techniques

As it was already mentioned, the holistic schema matching is about using statistical information to derive semantical correspondences among many input schemas. Differently from the traditional paradigm, the accent is put not on the schema-level or instance-level information, but rather on the statistical techniques to deal with various discrepancies and phenomenons that appear during matching of real world schemas.

4.4.1 Statistical Techniques

Various statistical techniques are used for detecting and dealing with the real world schemas. Generally, the problems are inherited from the data-mining field, and are tightly related to statistical properties of attributes among input schemas. Therefore, the different techniques employed in the holistic schema matching are related to:

- *Dealing with rare attributes and schemas.*
  Assuming the statistical nature of the holistic paradigm rare attributes and schemas with rare attributes are unwanted because they tend to "confuse" the statistics during the matching. Of course, the larger is the input set of schemas the less significant the rareness is. Hence the general solution is to minimize the significance of the rare attributes. For example, in the MGS framework a threshold parameter is used for removing infrequent attributes, moreover, rare schemas are aggregated into one during hypothesis selection, so that they do not mislead $\mathcal{X}^2$ hypothesis testing method.

- *Dealing with frequent attributes and schemas.*
  Too high frequency may cause holistic matching mistakes. For instance, frequent attributes may co-occur in many schemas because of their popularity among people not because of some interesting statistical properties, e.g., in books domain 'title' and 'author' are very popular, so there would exist an unwanted correlation between them. The DCM framework deals with such a case on the correlation measures definition level.

- *Dealing with noise attributes and schemas.*
  A low level of noise can be considered the same way as the rareness. However, it might be the case that the noise and the rareness is not the same thing. As an example consider that a wrapper automatically extracts a set of input schemas, but is unable to retain only those from the same domain. The *Ensemble* approach [22] tries to tackle the problem of the noise. Its main idea is to execute a holistic matcher on randomly sampled subsets of input schemas and then merge results according to some ranking. The motivation for such a solution comes from the assumption that there are more than enough input schemas, therefore a sample subset is still sufficient for deriving the mapping statistically. Additionally, subsets will contain less noise than the whole set of input schemas altogether.

4.4.2 Other Techniques

Beside the statistical techniques, holistic matchers employ techniques that are encountered in traditional paradigm also. However, the holistic paradigm research community does not concentrate on that. Nevertheless, it is still possible to name a few of such techniques:

- *Measuring linguistic similarity.*
  In the simplest case the linguistic similarity is about synonyms (MGS, DCM). However, the clustering-based matcher uses string similarity metrics. Moreover, the string similarity there is aggregate, determined both from the schema-level and instance-level data.

- *User interaction.*
  Although the holistic paradigm tries to replace the user with statistical information, still there is a need for setting parameters before running a matcher. Even more, the clustering-based matcher, as mentioned earlier, allows the user to deal with uncertain situations, instead of applying any statistical techniques.

- *Machine learning.*
  This technique is usually employed for parameter (threshold) learning.

5 Software for Schema Matching

Most of the matching algorithms and techniques discussed above are implemented. However, there is not any API available, which would standardize the development of new matchers and allow easily composing them when needed. Thus every new implementation is basically a reimplementation of what has already been done. Of course there exist programming libraries that facilitate certain tasks, such as implementation of machine-learning approaches, doing string comparison, etc., but still current matching systems and frameworks exist either as prototypes or as parts of certain proprietary systems (see the Table 1).

Name	Research	Notes
CUPID [27]	Microsoft	A hybrid matcher integrated into MS BizTalk Mapper. Performs better than DIKE and MOMIS (with ARTEMIS).
Clio [41]	IBM	A user-assisted matching system.
LSD [13, 12]	academic	A matching system based on machine-learning.
SF [29]	academic	A simple matcher that is used in a number of matching systems.
COMA++ [11, 1]	academic	A matching system (tool) that allows matching XML schemas and OWL ontologies. The implementation is publicly available.
eTuner [33]	academic	An approach for tuning the parameters of a matching system.
Protoplasm [5]	Microsoft	An attempt, inspired by COMA, to build an industrial-strength matching system.
Rondo [30]	academic	The first complete prototype of a generic model management system.
WHIRL [9]	AT & T	An extension of relational databases that allows "soft joins" based on the similarity of text identifiers. Uses various string similarity metrics.
MGS [20]	academic	A holistic matcher that solves synonym discovery.
DCM [23]	academic	A holistic matcher that exploits correlation mining for the mapping discovery.
Clustering-based matcher [38]	academic	A holistic matcher that uses clustering algorithm for finding similar attributes.

Table 1: Matchers, matching systems and frameworks

One of the most important prototypes today in the traditional schema matching research is COMA++. As it was previously mentioned, this solution is a matching system (composite matcher), where nearly all significant advancements of the traditional paradigm are present. It is freely available for non-commercial purposes as a packaged tool written in JAVA. Nevertheless COMA++ is still a prototype and there is much space for improvement. The main advantages and disadvantages of COMA++ are listed below:

- Advantages
  - Clear and natural architecture [1].
  - Many advancements of the traditional paradigm are implemented.
  - Much important functionality.
  - Good help for the GUI.
- Disadvantages
  - The API is not published, although the tool is free to use.
  - The GUI is not very intuitive, neither for matching nor configuration purposes.
  - Installation of MySQL DBMS is required.
  - Progress of a task is unavailable for longer tasks.

There are two possibilities for developing traditional schema matching tools. First possibility is to extend or adapt available implementations. However, this is not always appropriate, because current matching systems are poorly documented, basically incompatible with each other and, in general, do not represent any commonly accepted API. For example, although LSD is a composite matcher it is still not clear how to plug in a new matching algorithm into it without harming system logic. As a second possibility one could choose to develop a new matching system from the scratch. In this case architecture should be somewhat similar to the one of COMA++. So the golden mean between two possibilities seems to be the following:

1. As a starting point use COMA++.
2. Decouple it by defining clear interfaces (if they are not available) for matchers, e.g., *Tunable*, *MatchResultProvider*, *TreeSchemaConsumer*, etc.
3. Develop new matching algorithms that implement those interfaces and extend the matching system.

The development of a holistic matcher seems to be easier because the algorithms, as discussed previously, are quite simple. The main obstacle here is to prepare the input data and to tune parameters well. Also, differently from the traditional paradigm in the holistic schema matching there is no need for

composing matchers to get better accuracy. Therefore, the architecture becomes much simpler, and developing a new holistic schema matching tool from the scratch seems quite feasible because the issues of code reuse, decoupling, etc. are minimal. Of course, this does not mean that all the necessary machine-learning or data-mining algorithms should be reimplemented.

5.1 Useful Programming Libraries

A summary of several open source (JAVA) programming libraries, useful for developing new matching algorithms, is given below.

5.1.1 SimMetrics – String Comparison

It is an open source library of similarity metrics [35], dissimilarity metrics, string metrics, using edit distance, Levenshtein, Gotoh, Soundex and other metrics for similarity measures. This library has been developed to provide a consistent interface layer to similarity measures that act in a normalized manner enabling comparison and composition of metrics (not like SecondString [34]), whilst still allowing usage of the basic algorithms original output.

All the metrics can work on a simple basis whereby they take two strings and return a similarity measure from the interval [0, 1], where 0 being entirely different, 1 being identical. The metrics developed have been optimized for fast processing time and include methods that provide timing estimates. Some of them are summarized below.

Levenshtein distance. This is the basic edit distance function where the distance is given simply as the minimum edit distance which transforms s_1 into s_2. Edit Operations are the following:

- Copy character from s_1 over to s_2 (cost = 0).
- Delete a character in s_1 (cost = 1).
- Insert a character in s_2 (cost = 1).
- Substitute one character for another (cost = 1).

The recursive edit distance function is

$$D(i,j) = min \begin{cases} D(i-1,j-1) + d(s_1[i], s_2[j]), & \text{substitute or copy} \\ D(i-1,j) + 1, & \text{insert} \\ D(i,j-1) + 1, & \text{delete} \end{cases}$$

where i and j are the indexes of characters in strings s_1 and s_2 respectively, and the function d(x,y) is defined as

$$d(x,y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{otherwise.} \end{cases}$$

For example, distance between two strings 'sam chapman' and 'sam john chapman' is equal to 5.

Jaro distance. The Jaro distance metric takes into account typical spelling deviations. Briefly, for two strings s_1 and s_2, let

1. $ch_1^{s_2}$ be the characters in s_1 that are "common with" s_2,
2. $ch_2^{s_1}$ be the characters in s_2 that are "common with" s_1.

Roughly speaking, a character $x \in s_1$ is "in common" with s_2, if the same character x appears in about the same place in s_2. Then the Jaro similarity metric for s_1 and s_2 is

$$Jaro(s_1, s_2) = \frac{1}{3} \cdot \left(\frac{|ch_1^{s_2}|}{|s_1|} + \frac{|ch_2^{s_1}|}{|s_2|} + \frac{|ch_1^{s_2}| - T_{ch_1^{s_2}, ch_2^{s_1}}}{2 \cdot |ch_1^{s_2}|} \right),$$

where $T_{ch_1^{s_2}, ch_2^{s_1}}$ is the number of transpositions of characters in $ch_1^{s_2}$ relative to $ch_2^{s_1}$.

Soundex. It is a coarse phonetic indexing scheme. Soundex allows phonetic misspellings to be easily evaluated, for instance the names 'John', 'Johne' and 'Jon' can be treated as being the same.

The main idea of Soundex is to encode two strings according to a phonetic scheme and then compare them. For example, lets say there is such a scheme of consonants: 1 = J, 2 = N. Then we could encode 'John' = 12 and 'Jon' = 12, which would mean that they are the same.

Obviously, the main challenge is to find an appropriate phonetic scheme, therefore the one offered by SimMetrics may be insufficient for the schema matching.

Overlap Coefficient. This is a measure whereby if a set of characters X is a subset of Y or the converse, then the similarity coefficient is a full match:

$$overlap(X, Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}.$$

Q-grams distance. Q-grams are typically used in approximate string matching by "sliding" a window of length q over the characters of that string in order to create a number of q length grams. A match is then rated as a number of q-gram matches within the second string over all possible q-grams.

The intuition behind the use of q-grams as a foundation for approximate string processing is the following. When two strings s_1 and s_2 are within a small edit distance of each other, they share a large number of q-grams in common. For example, the positional q-grams of the length $q = 3$ for the string "web" are:

(1, __w), (2, _we), (3, web), (4, eb_), (5, b__).

5.1.2 Weka – Machine-learning

Weka [37] is a collection of machine-learning algorithms for data-mining tasks. The algorithms can either be applied directly to a dataset using the GUI or called from the JAVA code. The general view of Weka is given in Figure 11. First, in the preprocessing step a training dataset has to be loaded, then it can be customized or automatically filtered (modified). Afterward, the training process starts. There are three types of machine-learning algorithms in Weka:

1. *Classifiers*: wrappers for classification algorithms from data-mining.
2. *Clusterers*: wrappers for clustering algorithms from data-mining.
3. *Associators*: wrappers for association rules algorithms from data-mining.

The algorithms are trained using the input training data and can be tested on the specified test data.

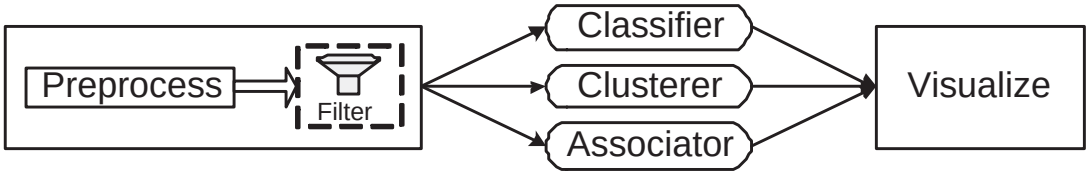

Figure 11: General view of Weka

Weka is not intended to specifically support the schema matching. Nevertheless, it can be relatively easily extended to mimic, for instance, LSD. In particular, schema-level information should be prepared in a special way, so that it could be fed to various classifiers, clusterers or associators. For example, this can be achieved by implementing a certain filter, similarly to the class

`weka.filters.unsupervised.attribute.StringToWordVector`.

One of the possible ways to approach the traditional schema matching problems using Weka consists of the following steps:

- Decide upon matching criteria that needs to be learned by Weka. For example, LSD uses instance-level information as a criterion for matching schema elements.
- Develop filters for each of the criteria, which would prepare the training data. For example, use some linguistic procedures to obtain numeric representations of the input.
- Develop converters for each of the criteria, in order to obtain test (and training) data.

- Follow the usual procedures provided by Weka. For example, take an XML file as an input, then apply a filter on it, train a classifier with the obtained training data and evaluate it on the test data.

On the other hand, the holistic schema matching can sometimes be tackled even easier. For example, Weka contains the Apriori algorithm which is the core of the DCM framework. So it may be the case that high quality data preparation is sufficient.

5.1.3 Joone – Neural Networks

Joone (Java object oriented neural engine) [24] is a neural net framework. It is composed by a core engine, a GUI editor and a distributed training environment and can be extended with new modules implementing new algorithms or architectures. In the context of schema matching, Joone can be used similarly to Weka, i.e., for parameter learning. However, it does not provide any interfaces for implementing data-mining tasks.

5.1.4 Carrot2 – Search Results Clustering

Carrot2 [8] is a framework for clustering query results from various data sources, such as Web search engines, RSS feeds, etc. It is highly extensible due to the loosely-coupled, XML-based architecture and contains many pre-built clustering algorithms. This software can be useful for automatic Web data extraction and holistic schema matching. For example, it may be the case that clustered Web search results are easier to process and integrate.

5.1.5 Other

In the XML schema matching, as the name suggests, there is a need to process XML documents. A wide variety of APIs and libraries are available for that.

XML-related libraries. The most powerful and popular XML-related programming libraries are the following:

- *DOM4j* [15].
  An open source XML framework optimized for JAVA. It allows reading, writing, navigating, creating and modifying XML documents. DOM4j supports such widely used APIs like JAXP, SAX, DOM and is considered as a better alternative than JDOM, because of being optimized for large XML documents and streams.

- *Xerces* [39].
  An open source XML parser that supports DOM levels 1,2 and 3 as well as XML 1.1 version.

- *Saxon* [25].
  A collection of XML transformation and query tools. It supports XSLT 2.0, XPath 2.0, XQuery 1.0 and conforms to JAXP API.

Eclipse platform. A vast collection of tools are available with the Eclipse platform [16]. They help to reduce a great deal of programming effort. For instance, the Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for building tools and other applications based on a structured data model; XML Schema Infoset Model (XSD) is a library that provides an API for manipulating the components of an XML schema as described by the *W3C XML Schema* specifications, as well as an API for manipulating the DOM-accessible representation of XML schemas.

6 Comparative Study

Comparing schema matching solutions is crucial to be able to choose the most suitable one. It is important not only to know what approaches and techniques they employ but also what is their empirical performance. All the schema matching prototypes up to date present the implementation and give some experimental results, however, to grasp the overall picture is nearly impossible. One reason is that various matchers use different evaluation means for experiments, another reason is the diversity of matchers. For instance, composite matchers can be combined of hundreds of matchers, so it is unclear how to treat them.

In this section the traditional and holistic paradigms with respective solutions will be summarized and compared regarding techniques and approaches they use, as well as their capacity shown experimentally. The comparative study enables one to objectively choose the most suitable solution in a certain context, which in case of the thesis, is the automatic Web data extraction.

6.1 Framework for Making Comparisons

Choosing attributes for making objective comparisons is a tedious task. On the one hand it is difficult to select an attribute that would be meaningful for every component, on the other - it is easy to miss an important attribute. However, there are attempts to challenge this, e.g., survey of *schema matching* approaches [31], at least for the traditional schema matching paradigm. One of the most important attempts to summarize experimental results of traditional matching systems is made by [10]. The authors use four axes for assessing matchers:

- *Input*: the kind of input data used during matching, e.g., schema-level information, instance-level data, auxiliary information, etc.

- *Output*: the information included with the match result. The less information the systems provide as output, the lower the probability of making errors but the higher the post-processing effort may be.

- *Quality*: the metrics chosen to quantify the accuracy and completeness of the match result.
- *Effort*: the amount of saved manual effort.

Each axis can be detailed according to interesting criteria (reviewed below with some comments on holistic paradigm). These criteria facilitate the task of comparing matchers, therefore later in the thesis some of them are used as the basis for choosing significant characteristics, i.e., attributes.

6.1.1 Criteria related to the input

The following information is important in matching problems:

- *Schema language.*
  In this thesis only XML schemas are considered.
- *Number of schemas and match tasks.*
  The higher the number of different match tasks, the more realistic match behavior is expected. The problem itself is also important, e.g., matching many independent schemas with each other vs. matching source schemas to a single global schema.
- *Schema information.*
  Most important is the number of schema elements. The bigger the input schemas are, the larger the search space for match candidates is, which often leads to lower match quality. Additionally, matchers exploiting specific facets perform better.
- *Schema similarity.*
  The more dissimilar (noisyer) the input schemas are, the "harder" the matching problem.
- *Auxiliary information used.*
  Examples are dictionaries, thesauri, constraints, etc. Availability of such information can greatly improve the result quality.

6.1.2 Measures of matching quality

To provide the basis for evaluating the quality of automated matching, the correct mapping should be available. This allows choosing metrics with respect to correct matches, incorrect matches, etc. Figure 12 shows the schema under which the metrics can be calculated:

- A: false negatives.
  The set of automatically unidentified matches.
- B: true positives.
  The set of correctly derived matches.

- C: false positives.
  The set of incorrectly derived matches.
- D: true negatives.
  The set of correctly discarded matches.

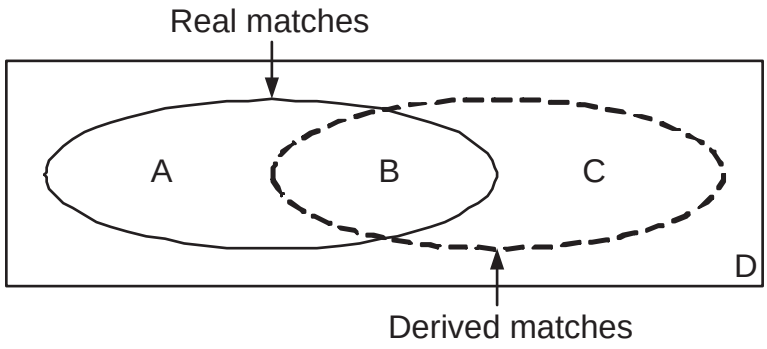

Figure 12: Measuring matching quality

There are two commonly accepted measures, *Precision* and *Recall*, based on the cardinality of the sets in Figure 12. These measures come from information retrieval. *Precision* reflects the share of real correspondences among all found ones, whereas *Recall* is the measure of the share of real correspondences that is found.

$$Precision = \frac{|B|}{|B|+|C|}, \quad Recall = \frac{|B|}{|A|+|B|}.$$

In the ideal case, when A, B and C coincide, then $Precision = Recall = 1$. However, neither *Precision* nor *Recall* alone can accurately measure the match quality. For example, it might be the case that $Precision = 1$, though $Recall \approx 0$.

Hence it is necessary to consider both measures in combination. There are several proposed solutions, the most important ones are: *F-Measure* and *Overall*. The former stems from information retrieval and intends to attach weights to *Precision* and *Recall*:

$$F - Measure(\alpha) = \frac{Precision \cdot Recall}{(1-\alpha) \cdot Precision + \alpha \cdot Recall},$$

where α is from the interval [0, 1]. When $\alpha = 0.5$, this measure is called the harmonic mean of *Precision* and *Recall*:

$$F - Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}.$$

The other solution for combining *Precision* and *Recall* is called *Overall* or *Accuracy*. This measure was developed specifically in the traditional schema matching context and embodies the idea to quantify the post-match effort needed for adding false negatives and removing false positives.

$$Overall = Recall \cdot \left(2 - \frac{1}{Precision}\right).$$

As far as the holistic paradigm is concerned, there precision and recall metrics are also used. Although they may differ in what is meant by A, B, C and D from Figure 12, but, in general, semantics of the metrics stay the same as in the traditional paradigm. For example, in the MGS framework the hidden models are important, so the precision and recall are calculated as the share of correctly revealed models. Also, the two metrics, although under the same title, slightly vary among the holistic paradigm solutions themselves. For instance, in the DCM framework the precision and recall are calculated only on the attributes which appear with a certain frequency (20% and 10%). Nevertheless, assuming the objectiveness of experiments conducted for holistic matchers, the metrics can be comparable to each other.

6.1.3 Effort

As the main purpose of automatic schema matching is to reduce the amount of manual work, estimating the required user effort is very important. In order to assess the manual effort one should consider both the *pre-match effort* required before an automatic matcher execution as well as the *post-match effort* to add the false negatives to and remove the false positives from the final match result. For example, pre-match effort includes:

- Training of the machine learning-based matchers.
- Configuration of the various parameters of the match algorithms, e.g.,, setting different threshold and weight values.
- Specification of auxiliary information, such as, domain synonyms and constraints.

Unfortunately, the effort associated with manual pre-match and post-match operations varies heavily with the background knowledge and cognitive abilities of users, their familiarity and the usability of tools, making it difficult to capture the cost in a generic way.

6.2 Schema Matching Paradigms

A summary of the traditional and holistic schema matching paradigms is given in Table 2. It can be observed that the two paradigms are orthogonal, which implies that each of them has to be applied only for a particular type of problems. For example, the traditional schema matching should be used when there are 2 schemas to be matched, and the matching process is driven by the user. Meanwhile, in the holistic paradigm more than two schemas are required for mining tasks to perform well.

	Traditional	**Holistic**
Historical roots	Databases (old)	The deep Web (recent)
Number of schemas	≥ 2 (small scale)	$\gg 2$ (large scale)
How it works	Creates a binary mapping for (each pair of) input schemas	Creates one n-ary mapping for all input schemas
Schema information	The more information related to schemas is available, the more accurate mappings are produced	Schemas are considered as unstructured sets of attributes and usually in the beginning are "flat"
Same domain assumption	Not necessarily	Usually yes
Approaches	As summarized in Figure 5	Statistical matching; matching as mining
Automation	Inherently semi-automatic	Statistics instead of users
Good accuracy prerequisites	User interaction; composition of matchers; as much as possible information related to the input schemas, i.e., employment of many techniques	Input schemas should be preprocessed to have no side effects on statistics, or techniques should be employed to deal with unwanted cases, e.g., noise attributes; well tuned input parameters; abundance in schemas
General performance	Slow	Fast
Remarks	-	-

Table 2: Comparative study of *schema matching* paradigms

6.3 Traditional Schema Matching Solutions

A comparison of the main solutions related to the traditional schema matching is given in Table 3. Worth noticing that COMA is treated in union with COMA++, because there are no experimental results available for the latter. As the table suggests, COMA seems to overcome others in many aspects, such as flexibility, extensibility and accuracy. However, COMA may bring much overhead, when a simple solution is needed. In this case SF or CUPID should be considered.

	COMA[11, 1]	**LSD**[13, 12]	**CUPID**[27]	**SF**[29]
Combination of matchers	Composite	Composite	Hybrid	Hybrid
Matching approaches	Any, depending on combined matchers; possible to choose a strategy; pre-built: schema based ↪element-level, schema based ↪structure-level	Schema based ↪element-level ↪linguistic; schema based ↪structure-level; instance based ↪element-level; instance based ↪element-level ↪constraint-based	Schema based ↪element-level ↪linguistic; schema based ↪structure-level; schema based ↪structure-level ↪constraint-based; schema based ↪element-level ↪constraint-based	Schema based ↪element-level ↪linguistic; schema based ↪structure-level
Matching techniques	Any, depending on combined matchers; pre-built: graph-to-tree transformation, fragmenting schemas, flattening the similarity cube	Name equality, synonyms; XML classifier for matching non-leaf elements; learning, Whirl, Bayesian learners; list of valid domain values	Name equality, synonyms, hypernyms, homonyms, abbreviations; data type and domain compatibility, referential constraints; matching subtrees, weighted by leaves; graph-to-tree transformation	Name equality; propagation of the similarity in the graph
Local/global cardinality	1:1/m:n	1:1/n:1	1:1/n:1	1:1/m:n
Reuse/auxiliary information used	A special matcher for reusing mappings; thesauri; glossaries; user interference at any point	Comparison with training matches; lookup for valid domain values	Thesauri; glossaries	none
Pre-match effort	Specifying domain synonyms, acronyms	Training; specifying domain synonyms, constraints	Specifying domain synonyms	none
Precision/Recall	0.93/0.89	$\sim 0.8/\sim 0.8$	n/a	n/a
F-Measure	0.90	~ 0.8	n/a	n/a
Overall	0.82	~ 0.6	n/a	~ 0.6
Remarks	Easily extensible	-	-	Simple to implement

Table 3: Comparative study of traditional matchers

6.4 Holistic Schema Matching Solutions

A comparison of the main solutions related to the holistic schema matching is given in Table 4. Among three solutions in the table, DCM seems to be the best. It is fast, mostly automatic, allows for the n:m mapping cardinality, and is very accurate. However, there are two points that should be taken into account when choosing DCM. First, it needs data preparation before performing correlation mining. Second, experimental results are given for certain attribute frequencies, which means that in practice the DCM framework may perform worse.

	MGS[20]	**DCM**[23]	**Clustering-based**[38]
Approach	Statistical matching (global evaluation)	Matching as mining (local evaluation)	Matching as mining (local evaluation)
Core statistics	Hidden model discovery (the hypothesis testing performed using $\mathcal{X}^2$)	Correlation mining (similar to the association rules mining), positive correlation for group discovery, negative correlation for match discovery	Clustering of attributes according to the similarity matrix, each cluster contains similar (synonymous) attributes
Techniques employed	Statistical: attribute selection, rare schema smoothing, consensus projection	Statistical (built in the correlation measures): dealing with sparseness, rare attributes and negative correlation, frequent attributes and positive correlation	Workaround for capturing 1:n mappings; measuring the linguistic similarity; user interaction; machine learning
Schema information	Not restricted, but implemented for attribute names	Not restricted, but implemented for attribute names	Not restricted, but implemented for attribute names, labels and domains
Cardinality captured	n-ary 1:1 mappings, i.e., 1:1: ... :1	n-ary m:n mappings, i.e., m:n: ... :z	1:n mappings, i.e., sets of 1:n matches (clustering algorithm captures only n-ary 1:1)
User effort	Parameter tuning	Parameter tuning	Parameter training; handling uncertain situations
Precision/Recall	$\geq 0.95/ \geq 0.95$, on the average for 4 domains	$\geq 0.95/ \geq 0.95$, on the average for 8 domains and attribute frequencies $\geq 10\%$ and $\geq 20\%$	$\sim 0.88/ \sim 0.91$, on the average for 5 domains (automatic); $\sim 0.95/ \sim 0.88$, on the average for 5 domains with learned parameters; $\sim 0.96/ \sim 0.94$, on the average for 5 domains with all user interactions
General performance	Slow (exponential)	Fast	Fast
Remarks	Synonym discovery	-	Easy to implement

Table 4: Comparative study of holistic matchers

7 A Specific Problem Setting

As it was mentioned in the introduction, the aim of the thesis is to propose a solution for automatic Web data extraction. For this purpose the *AllRight* project [18] is discussed below and the point where it needs schema matching is identified. Then, the matching problem is defined and carefully analyzed. Further, a solution is proposed, which straightforwardly flows from the earlier given comparative study. Worth noticing that although the specific problem is targeted, it is not meant to be thoroughly solved in the *AllRight* project from the software engineering point of view.

7.1 The *AllRight* project

The goal of this project is to automatically extract data/information from the Web. Differently from using wrappers, *AllRight* aims to implement procedures for fully automatic extraction from the Web (although it may use wrappers as submodules). Generally, the final system should have three components, as depicted in Figure 13:

1. *Information retrieval (IR)* locates relevant web pages.
2. *Information extraction (IE)* extracts content from the relevant web pages.
3. *Information integration (II)* merges extracted information into one coherent view.

Having such a system enables querying the Web for integrated structured information. Therefore, for example, a user buying a camera is saved from the long lasting browsing of the Web.

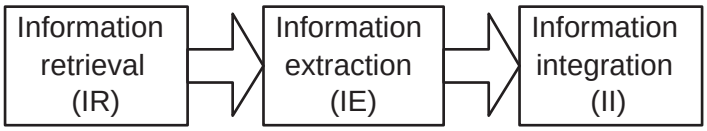

Figure 13: General view of *AllRight* project

7.1.1 IR Component

This component locates relevant web pages with an assumption that the Web is a huge database, where there is a sufficient amount of simply structured document conforming to the query. *AllRight* uses a three-step procedure to locate interesting pages:

1. *Initial search.*
   The initial search is performed using standard web search engines. However, having a user query usually is not enough for getting good search

results. Therefore, a domain knowledge base (RDF ontology) is used to derive high-quality keywords, and machine learning – to gather the ones that give good results. In other words, before performing the initial search, the user query is preprocessed (transformed) to improve the accuracy of the search.

2. *Filtering search results.*
   Apparently, the initial search can lead to enormous number of results. According to the above assumption, it should be possible to ignore part of them, and concentrate only on those that can be easily processed by the IE component.

3. *Crawling for similar pages.*
   Here a heuristic, that web pages usually link to similar pages, is used. So the set of relevant web pages is enriched with the ones crawled from the filtered intial search results. The crawled pages have to be filtered as well, in order to be consumable by the IE component.

Hence the result of the IR component is a set of web pages relevant to the initial user query. Moreover they have quite simple (i.e., "flat") structure.

7.1.2 IE Component

Beside helping the IR component in filtering initial search results, this component is responsible for the actual data extraction. The difficult part here is that the extraction has to be automatic. This is the reason why IR component has to retrieve only the documents having a certain ("simple") structure. In particular, the relevant information in the web page should be present in the tabular form. More specifically, the whole extraction process leans on the following properties of web pages:

- Visual appearance.
- Structure.
- Contents.

These properties are utilized by certain techniques, in order to extract information as a set of Subject–Predicate–Object (S–P–O) triples. For example, consider that the set of relevant web pages is about the camera Canon Ixsus IIs, then one triple could look like 'Canon Ixsus IIs'–'resolution'–'2048x1536'.

Without going too much into technical details, the outcome of the IE component is a set of S–P–O table structures (SPOTS) extracted from respective web pages. To put it even simpler it is a bunch of tables as depicted in Figure 14. The figure shows what is a subject, predicate and object, moreover, in brackets a translation into terms of *schema matching* is given.

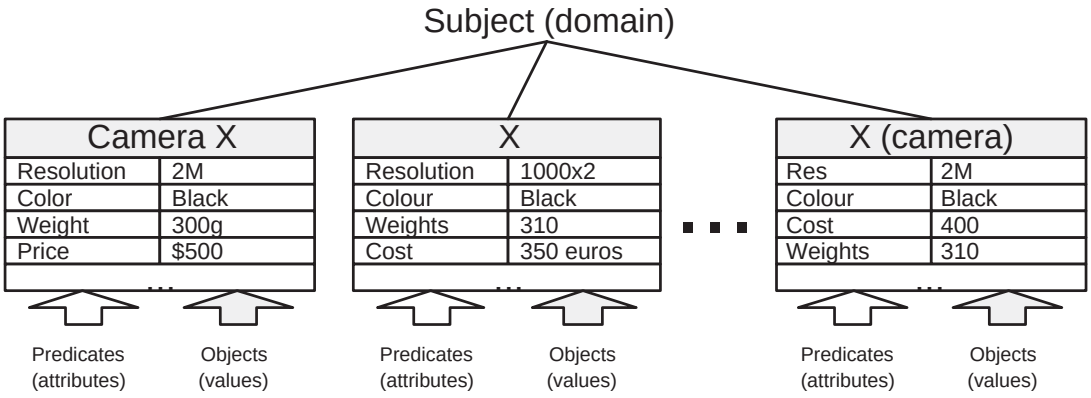

Figure 14: The output of the IE component

7.1.3 II Component

The goal of the last component is to integrate all the SPOTS into one coherent view. Technically, they are encoded into XML documents that are valid under some XML schema $\mathcal{S}$. So each $\mathcal{S}$ instance represents the extracted content (a table) from a respective web page, and the task of the II component is to merge them. Considering that every extracted tabular information was differently presented, e.g., Figure 14, the integration problem is actually the *schema matching* problem as depicted in Figure 3. Worth noticing that the matching is performed on the data stored in $\mathcal{S}$ instances, i.e., attributes.

7.2 Definition of the Problem

The schema matching problem in *AllRight* appears in the information integration component. This component can be seen as a mediator system described earlier and in the thesis the initial step, prior to normalization, cleaning and fusion, is investigated. Therefore, the problem to be solved is the following: *given a set of XML documents that represents a set of SPOTS, derive a mapping among the schemas they store.* This problem is depicted in Figure 15 and further is referred as SPOTS-to-MAP.

Beside the problem definition, it is crucial to identify what is given and what assumptions are made. In the light of the *AllRight* project the following hold:

- As Figure 15 suggests, it is straightforward to obtain schemas for matching from XML documents containing SPOTS, for instance, by applying some transformation.
- The majority of input (schemas to be matched), that the II component operates on, is from the same domain of interest, e.g., cameras. Moreover, they are about the same subject (from S–P–O triples), e.g., specific camera. The rest of the input can be considered as the noise, with which the IE component was unable to deal.

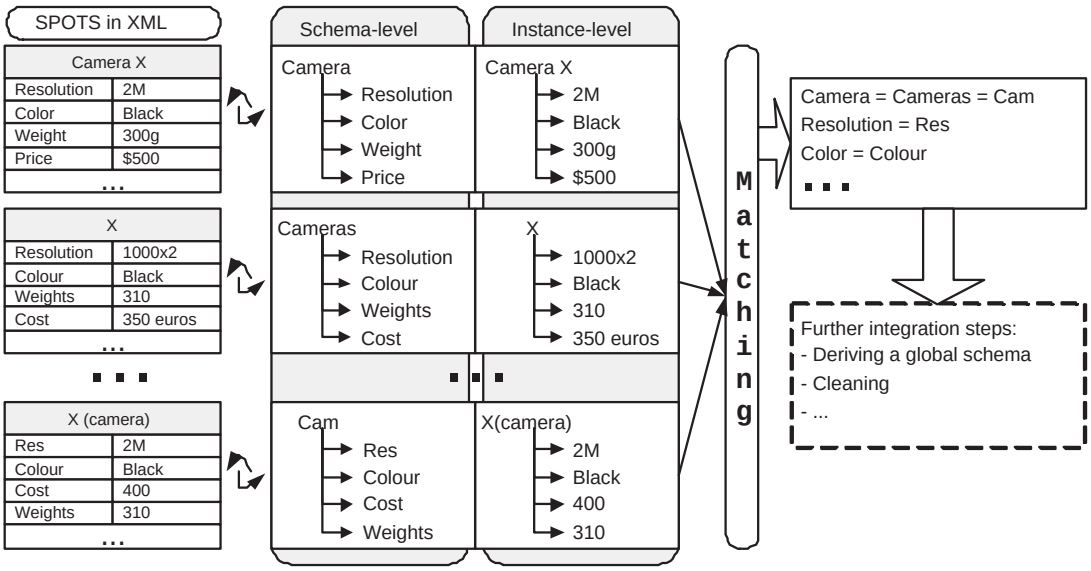

Figure 15: Problem depiction

- There are $\gg 2$ schemas to be matched.
- The schemas have a "flat" structure. Although they are represented as trees, it is easy to notice that their depth is 1. However, the IE component in *AllRight* is able to extract a more nested structure from web pages, as long as it was presented in the tabular form. This hierarchical information does not prohibit to treat the schemas as "flat" (though some techniques may take advantage of it).
- Both schema-level and instance-level information is available for schemas. This information depends on the IE component. For instance, hierarchical information can be appended to schema element names, or types can be guessed for attribute values.
- There is no prescribed target schema used during the matching. However, there is a domain ontology (also used in the IR component) which can be helpful for deriving the final schema.
- It is preferred that the schema matching is automatic in a sense that the user is allowed to set input parameters, but not interfere with the matching process and results.

7.3 Analysis of the Problem

Although understanding the problem is already half of the solution, nevertheless it should be examined with respect to the study presented earlier. At the first sight SPOTS-to-MAP needs to be tackled using holistic paradigm, however, it still may be enhanced by the traditional schema matching approaches and

techniques. Therefore, further the problem is viewed from the two paradigms perspectives.

7.3.1 Feasibility of the Holistic Paradigm

It was already mentioned that the holistic schema matching paradigm emerged due to the need to integrate Web query interfaces. Therefore, it is basically enough to show that the matching problem in the deep Web, i.e., matching of Web query interfaces, and the SPOTS-to-MAP problem are highly overlapping. This would enable the *AllRight* project to use the great deal of available research. Moreover, the deep Web is only one motivation for doing a large scale schema matching, that is mentioned in the literature. Thus pointing SPOTS-to-MAP as another one can lead to additional interest and branching of the research as far as the holistic schema matching is concerned.

The notion of the deep Web is related to the information accessibility on the Web. It has been observed that a large number of data sources on the Web are hidden behind query interfaces. Simply, from a user perspective, she has to fill out some form in a web page and submit it, in order to get the requested information. Therefore, in the deep Web the main issue is to merge all the query interfaces so that the user could query many sources, about the same domain of interest, at once. The integration of query interfaces can be divided into two steps:

1. *Mapping discovery*: semantic correspondences of fields over different query interfaces are identified.

2. *Unification of source query interfaces*: the interfaces are unified based on the mapping produced in the first step.

Clearly, the first step is actually the schema matching problem that during research led to the holistic paradigm. So this step is of interest here, and in the following, the matching problem in the deep Web stands for mapping discovery of query interfaces.

The comparison of SPOTS-to-MAP and the matching problem in the deep Web is given in the Table 5. Although the comparison is not very systematic and detail, still it gives the impression that SPOTS-to-MAP can be seen similar to the deep Web matching problem. Moreover, it seems that the former is a subproblem of the latter. Of course, such a statement needs a proof, but intuitively, it is difficult to find aspects of SPOTS-to-MAP that are not considered in the deep Web mapping discovery.

The holistic schema matching solutions are fragile with respect to statistical properties of the input schemas. Special techniques, summarized earlier, are employed for dealing with noise, rare and frequent attributes and schemas, because they may distort matching results and make the accuracy poorer. In case of SPOTS-to-MAP these issues may have to be dealt differently. Consider that in the deep Web, after matching schemas, query interfaces have to be unified in a way that the query result from the unified interface is the same as the union

	***AllRight* problem**	**Deep Web problem**	**Related?**
Source of schemas	Output of the IE component	Query interfaces, e.g., web forms, can be considered as schemas	No
Number of schemas	$\gg 2$, but should be reasonable with respect to the performance	$\gg 2$, but the more, the better in order for statistics to work	Yes
Domain of schemas	Assumed to be the same	Usually considered as the same	Yes
Structure of schemas	Usually "flat", i.e., attribute – value	Usually "flat", i.e., attribute – value domain	Yes
Information available	Anything that comes from the IE component: schema-level and instance-level	There are solutions that use: attribute names and labels, hierarchical structure of schemas, instance-level data. But they can be easily extended	Yes
Mapping discovery	Basically attribute synonym discovery is needed (because the same domain is assumed for all the input schemas), but can rely on some threshold of attribute similarity	There are solutions that do synonym discovery and more	Partial
Mapping cardinality	Usually n-ary 1:1 is enough, whenever it is semantically correct	There are solutions for n-ary 1:1, m:n and 1:n	Yes
Target schema	Not known, but in some way has to be consistent with the knowledge base used in the IR component	The "union" of all the input schemas	No
Dealing with the noise	Rare attributes/schemas may be removed depending on the wanted result	Statistical techniques are employed	Partial
Automation	User interaction is undesired, maybe only for parameter tuning	Statistical information is used to replace the user	Yes

Table 5: Can holistic paradigm solve SPOTS-to-MAP?

of results acquired from each of the source query interface. In other words, the unified query interface has to combine sufficiently similar attributes from the input interfaces, retain attributes specific to some interface, and make the layout preserve the ordering of the attributes and the structure of the source query interfaces as much as possible [38]. Similar conditions are not required in the *AllRight* project, therefore, it is problematic to decide how rich, in a sense of the matched attributes and cardinality, the mapping should be. For example, consider that the IE component extracts 50 SPOTS about the camera 'X'. Among the 50 schemas there is one that subsumes all the information in the rest 49. So it seems that the matching in the example is not needed at all. Actually, the issue here is due to the assumption that there is no target schema for SPOTS-to-MAP. In comparison, in the deep Web it is assumed that the target schema is a coherent "union" of input schemas.

As far as the specific holistic paradigm solutions are concerned, all the three of them, discussed earlier, are suitable for SPOTS-to-MAP. However, the MGS_{sd} algorithm should be considered carefully because it is exponential, and moreover, the study about proliferating sources and converging vocabularies is done only in the context of the deep Web. Ignoring the MGS framework is not a big loss having in mind that experiments show similar accuracy to the DCM framework [21]. Consequently, only mining as matching solutions left, which is a favorable situation from implementation point of view, because there are open source libraries for data-mining tasks (Weka).

7.3.2 Feasibility of the Traditional Paradigm

The traditional schema matching paradigm alone is not suitable for SPOTS-to-MAP. In the context of the *AllRight* project, it does not seem feasible to develop a user-driven slow matching system that would pairwise match a bunch of schemas. However, it can be beneficial to employ certain approaches and techniques in conjunction with the holistic paradigm.

The task of the traditional schema matching is to find the best combination of matching techniques and criteria to obtain the best accuracy in search for semantic correspondences. The most problematic is dealing with ambiguities. Many techniques are proposed for disambiguation, but in SPOTS-to-MAP this is not an issue due to the assumption of the same domain. Therefore, it is possible to state that in SPOTS-to-MAP it is enough to deal on the synonym discovery and syntactic similarity levels.

The choice of techniques for SPOTS-to-MAP is determined by the available information from the IE component. Hence the earlier summarized techniques for measuring schema element similarity in the traditional paradigm can be chosen according to the following list:

- Schema-level information:
  - Attribute names.
  - Hierarchical structure, if any (attribute groups, subgroups, etc.).

- Instance-level information:
  - Attribute values.
  - Guessed value data types.

Moreover, the traditional schema matching has a collection of techniques that exploit auxiliary information. For instance, synonym and acronym lists, a domain ontology, etc. are valuable for SPOTS-to-MAP.

The IE component in *AllRight* is trilateral. Beside the structure and contents, visual information that it uses for Web data extraction should not be underestimated. Although it is not encountered in any of the techniques available in the traditional paradigm, obviously it can provide with some interesting and valuable metrics. For example, the size of the table in a relevant web page can mean that it contains more information than a smaller one.

The most obvious way how to employ traditional schema matching techniques can be observed in the clustering-based holistic matcher. This matcher uses similarity matrix as an input for the clustering algorithm. After applying multiple techniques the similarity matrix should become more precise, therefore, clustering should become more accurate.

7.4 Solution Proposal for *AllRight*

On the whole the solution for SPOTS-to-MAP is to use the holistic schema matching and try to improve accuracy with traditional methods. There are two, out of three, solutions in the holistic paradigm to choose from (since MGS framework was discarded earlier). As a matter of fact, both the clustering-based matcher and the DCM framework seem suitable. The reason is that they both use the matching as mining approach. However, there are some points that need to be considered.

Although, DCM performs better than the clustering-based matcher (see Table 4), such results are achieved only by assuming sufficient observations. Therefore, prior to starting actual matching, DCM has a data preparation step. In this step source schemas are cleaned to increase attribute frequencies (see Example 7.1). Without it the DCM framework would obviously perform worse, because there would be too few correlations.

Example 7.1
Consider four schemas from the flights domain:

- S_1 ={'From City', 'To City', $\cdots$},
- S_2 ={'From:', 'To:', $\cdots$},
- S_3 ={'Departure', 'Destination', $\cdots$},
- S_4 ={'Departure City', 'Destination City', $\cdots$}.

After data preparation in DCM the schemas respectively can become:

- S_1 ={from, to, $\cdots$},
- S_2 ={from, to, $\cdots$},
- S_3 ={from, destination, $\cdots$},
- S_4 ={from, destination, $\cdots$}.

Clearly, the preparation step in DCM is the point where traditional schema matching techniques can be exploited. In the clustering-based matcher those techniques are needed for computation of the similarity matrix. Thus, both of the holistic paradigm solutions could be tweaked for SPOTS-to-MAP with traditional paradigm techniques.

One of the points that is not addressed among holistic schema matching solutions is the corpus for future reuse. The traditional paradigm offers techniques for building and using corpus of schemas, mappings, etc., however, the holistic paradigm treats all the input schemas as a corpus. In case of the *AllRight* project, it would be beneficial to keep a corpus together with the knowledge base used in the IR component. Such a corpus could at least serve the clustering-based matcher to avoid user interactions while solving uncertainties. On the other hand, a corpus can be valuable for providing statistical stability. In particular, the holistic paradigm solutions have a property that statistics are more accurate, when there are more observations available. For example, running a matcher on 50 input schemas can give worse results, than running it on them plus other 50 (i.e., on 100) from a corpus.

Further, the absence of the target schema is a problem. As it was already mentioned, the research on deep Web strives for a unified schema on all of the input schemas. However, in *AllRight* the target schema is not available. Nevertheless, in the next phase, i.e., after initial matching, the integration process continues and there will have to be an agreement upon the final (integrated) schema. So there are three options:

1. The final schema is determined and static for a specific domain. This would make it available before the matching starts and may increase the performance of holistic matchers, because only mappings covering this schema would be needed. Even more, in this case the exploitation of traditional schema matching solutions, instead of holistic ones, should be favored.

2. The final schema is dynamic and has to be provided as a unified view on all the input schemas (like in the deep Web). In this case, the use of the holistic paradigm is reasonable, and unification of schemas is a completely separate problem.

3. The final schema is dynamic and has to cover some set of attributes. This means that not necessarily all the matches have to be discovered. In this case, the knowledge base used in the IR component can be decisive for what attributes are important to be matched.

Having a target schema, or at least a set of attributes of the target schema is advantageous in the way that it can be included among input schemas for a holistic matcher. This way it is straightforward to have the integrated schema out of the produced mapping.

The general picture of the discussed points above is provided in Figure 16. Moreover, the figure suggests an early design draft. To sum up, the system gets input schemas (transformed SPOTS), then, depending on the implementation, either the DCM or clustering-based algorithm is performed. The holistic matchers can be integrated into Weka, in order to take advantage of the machine-learning framework for data-mining tasks. The user would interact only in the parameter tuning and, when needed, preparation of auxiliary information (e.g., a corpus, an ontology), which can be used together with traditional schema matching techniques. After matching the system passes the mapping to further steps of integration.

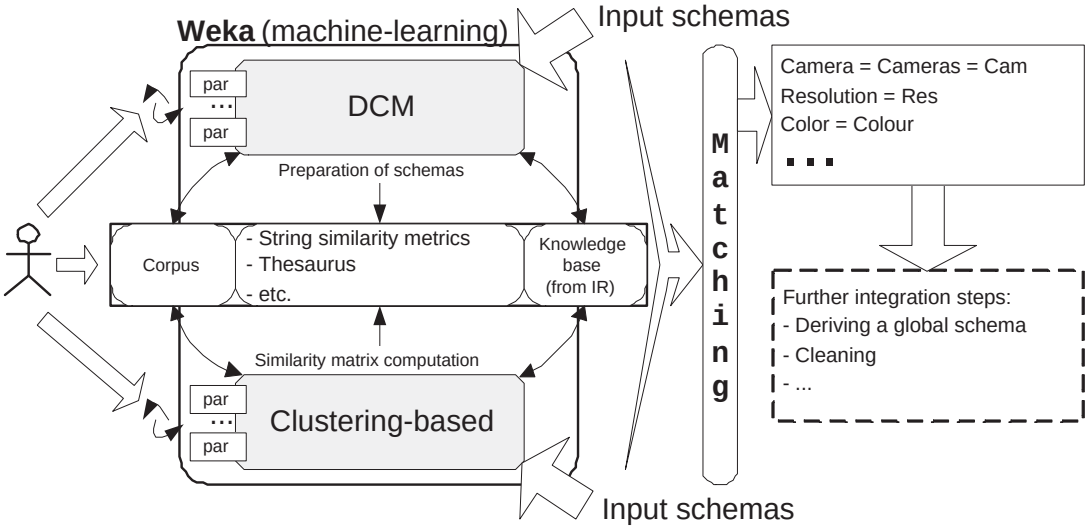

Figure 16: General picture of the solution proposal

7.4.1 Experiment

As a justification of the proposed solution and the proof of the possible simplicity of the implementation, a black box experiment is conducted. The goal is to check if the mining as matching approach really works (without implementing any of the holistic matchers). For this purpose Weka and SimMetrics software is used.

The whole experiment can be summarized into the following steps:

1. *Defining requirements for the experiment.*
   The experiment has to test the clustering-based matching approach. Clustering has to be performed on 6 schemas, each having 40 attributes.

2. *Preparing data.*
   Initially one SPOTS about a camera is taken. It is extracted by the IE

component implemented in *AllRight*. Schema attributes from the SPOTS (XML document) are pulled out using the following XPath expression:

```
//Attribute/name/text()
```

Not going deep into technical details, this query gives all the names of the extracted schema attributes. Then, randomly 40 of them are chosen, as required for the experiment.

5 more schemas are artificially created by randomly fuzzifying the names of the first schema attributes, for example, an attribute 'External flash', becomes something like 'E-xter-n-al fl-ash-' in one schema, and 'E—xternal flash' in another. So altogether $6 \times 40 = 240$ attributes participate in clustering.

The crucial part of this step is the computation of the similarity matrix. For this purpose a simple JAVA program that uses Levenshtein string similarity metric (provided in SimMetrics) is written. It creates a 240×240 similarity matrix according to that metric and encodes it into an ".arff" file, acceptable by Weka. Intuitively, an ".arff" file can be compared to a ".csv" (comma separated values) file.

3. *Running a clustering algorithm.*
   Weka GUI is used for performing clustering of the similarity matrix. **EM** is chosen as a clustering algorithm. The step-by-step procedure is as follows:

   (a) Run Weka and open the **Explorer**.

   (b) In the **Preprocess** tab load the ".arff" file containing the similarity matrix.

   (c) In the **Cluster** tab choose the **EM** clusterer and set the parameters as:

   - debug = 'false'
   - maxIterations = '100'
   - minStdDev = '0.1'
   - numClusters = '40'
   - seed = '100'

   The parameters for the **EM** clusterer are chosen accidentally, except for *numClusters*. The value of the parameter is 40 because there are 40 attributes in each of the 6 input schemas and they map 1:1.

   (d) Start the clustering.

4. *Checking the results.*
   The expected outcome is 40 clusters, each containing matching attributes from the 6 input schemas. In other words, every cluster should contain 6 elements, i.e., 1 of them is real and 5 are synthetic, for example, one cluster could be such set of items:

Nr.	Attribute name	Quality
1.	‘External flash’	Real
2.	‘E-xter-n-al fl-ash-’	Synthetic
3.	‘E-xter-n-al flash’	Synthetic
4.	‘External-flash’	Synthetic
5.	‘Extern-al fl–ash’	Synthetic
6.	‘-External flash-’	Synthetic

Indeed, the results are positive. Weka finds 38 clusters, where only 6 of them contain < 6 items, and 6 of them contain > 6 (including the correct ones). Hence a general result is $38 - 6 = 32$ “correct” clusters out of 40.

Worth noticing that this experiment does not aim at showing any advantages or disadvantages of the matching as mining approach. It is rather a quick proof of concepts stated throughout the thesis. Therefore results may vary depending on the data preparation, the amount of input schemas, and the choice of a clustering algorithm.

8 Conclusion

In the thesis a solution for the schema matching problem in the *AllRight* project was proposed. It was observed that in the automatic Web data extraction, and thus *AllRight*, a large scale schema matching was needed. Therefore, the research from the deep Web, i.e., matching of Web query interfaces, could be adopted. Specifically, it was argued that the DCM framework and the clustering-based matching were suitable for the schema matching in *AllRight*. Moreover, points, where the holistic matching as mining approach could be extended with traditional schema matching techniques, were identified. In particular, the clustering-based matcher [38] can be improved by enhancing calculation of the similarity matrix, and the DCM framework [23] – by better preparing data before doing correlation mining.

Beside the proposed solution, another contribution of the thesis is the study of *schema matching*. There were two schema matching paradigms presented and discussed. The traditional schema matching paradigm comprises a number of approaches and techniques. This paradigm is comparably old and stems from the field of databases. The main idea of the traditional schema matching is to discover semantic correspondences between the elements of two input schemas. Therefore, the central research challenge here is to develop robust techniques that could capture the similarity of elements accurately. Another paradigm is called holistic. Its general idea is to take a lot of input schemas from the same domain and discover the mapping for all of them at once. The key challenge in the holistic schema matching is to develop statistical techniques that would capture the similarity of schema elements. Generally, this paradigm is quite recent and roots from the research in the deep Web area, namely, matching of Web query interfaces.

In addition to the presentation of the paradigms, both of them were detailed with respective solutions available in the literature. Moreover, a thorough comparative study was conducted. First, the paradigms were compared against each other, then representative solutions from each paradigm were evaluated among themselves.

As a practical contribution of the thesis, useful software for *schema matching* was discussed. Moreover, two packages, Weka and SimMetrics, were used in the experiment justifying the feasibility of the proposed solution. The simplicity of the experiment implies that the implementation of the solution in *AllRight* should be fairly cheap.

References

[1] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. SIGMOD'05 accepted demo.

[2] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *VLDB*, pages 119–128, 2001.

[3] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, pages 452–466, 2002.

[4] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A vision for management of complex models. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):55–63, 2000.

[5] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38–43, 2004.

[6] P. Bohannon, W. Fan, M. Flaster, and P. P. S. Narayan. Information preserving XML schema embedding. In *VLDB*, pages 85–96, 2005.

[7] A. Boukottaya and C. Vanoirbeek. Schema matching for transforming structured documents. In *ACM Symposium on Document Engineering*, pages 101–110. ACM, 2005.

[8] Carrot2. An open-source search results clustering framework. http://sourceforge.net/projects/carrot2.

[9] W. W. Cohen and H. Hirsh. Joins that generalize: text classification using Whirl. In R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of KDD-98, 4th International Conference on Knowledge Discovery and Data Mining*, pages 169–173, New York, US, 1998. AAAI Press, Menlo Park, US.

[10] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, pages 221–237, 2002.

[11] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[12] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[13] A. Doan, P. Domingos, and A. Y. Levy. Learning source description for data integration. In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[14] A. Doan, N. F. Noy, and A. Y. Halevy. Introduction to the special issue on semantic integration. *SIGMOD Record*, 33(4):11–13, 2004.

[15] DOM4j. An open-source library for working with XML, XPath and XSLT on the Java platform.
http://www.dom4j.org.

[16] Eclipse.

[17] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data Knowledge Engineering*, 31(3):227–251, 1999.

[18] W. Gatterbauer, B. Krüpl, W. Holzinger, and M. Herzog. Web information extraction using eupeptic data in Web tables.

[19] A. Y. Halevy, J. Madhavan, and P. A. Bernstein. Discovering structure in a corpus of schemas. *IEEE Data Eng. Bull.*, 26(3):26–33, 2003.

[20] B. He and K. C.-C. Chang. Statistical schema matching across Web query interfaces. In *SIGMOD Conference*, pages 217–228. ACM, 2003.

[21] B. He and K. C.-C. Chang. A holistic paradigm for large scale schema matching. *SIGMOD Record*, 33(4):20–25, 2004.

[22] B. He and K. C.-C. Chang. Making holistic schema matching robust: an Ensemble approach. In *KDD*, pages 429–438. ACM, 2005.

[23] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across Web query interfaces: a correlation mining approach. In *KDD*, pages 148–157. ACM, 2004.

[24] Joone. An open-source neural net framework written in java.
http://sourceforge.net/projects/joone.

[25] M. Kay. Saxon - the XSLT and XQuery processor.
http://saxon.sourceforge.net.

[26] J. Madhavan, P. A. Bernstein, K. Chen, A. Y. Halevy, and P. Shenoy. Corpus-based schema matching. In *IIWeb*, pages 59–63, 2003.

[27] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *The VLDB Journal*, pages 49–58, 2001.

[28] F. Mandreoli, R. Martoglia, and E. Ronchetti. Versatile structural disambiguation for semantic-aware applications. In *CIKM*, pages 209–216. ACM, 2005.

[29] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[30] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *SIGMOD Conference*, pages 193–204, 2003.

[31] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[32] E. Rahm, H. H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.

[33] M. Sayyadian, Y. Lee, A. Doan, and A. Rosenthal. Tuning schema matching software using synthetic scenarios. In *VLDB*, pages 994–1005. ACM, 2005.

[34] SecondString. An open-source Java-based package of approximate string-matching techniques.
http://secondstring.sourceforge.net.

[35] SimMetrics. An open-source extensible library of string similarity and distance metrics.
http://sourceforge.net/projects/simmetrics.

[36] G. Wang, J. A. Goguen, Y.-K. Nam, and K. Lin. Critical points for interactive schema matching. In *APWeb*, pages 654–664, 2004.

[37] Weka. A collection of machine learning algorithms for solving real world data-mining problems.
http://sourceforge.net/projects/weka.

[38] W. Wu, C. T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep Web. In *SIGMOD Conference*, pages 95–106. ACM, 2004.

[39] Xerces. XML parsers, http://xml.apache.org.

[40] L. Xu and D. W. Embley. Discovering direct and indirect matches for schema elements. In *DASFAA*, pages 39–46, 2003.

[41] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):485–496, 2001.