Methods for Automatic Schema Matching - A Survey

Nikit Begwani[1] and Rohan Gupta[2]

***Abstract*— A database schema is a blueprint of the database as to how the data is organized in the database. The schema matching problem is a problem of generating correspondence between two or more input schemas. It identifies the semantic relationship between elements of the two schemas. It has important application in many domains like the federated databases, data warehouses, e-commerce etc. A lot of work has been done in this field to develop algorithms and tools for schema matching. In this paper we discuss applications of schema matching, algorithms and techniques developed over time and tools that have been developed for the same.**

I. INTRODUCTION

Work on the matching problem began in early 1970s. Since then a good amount of research has been done in this field. Different techniques are employed for matching schemas at various levels of granularity ranging from matching at structure level to matching based on usage or the queries to the system. Some of the advanced techniques developed use complex models based on graphical structures and machine learning approach. Advanced tools have been developed by the various companies and researchers applying these techniques like the Cupid system developed by Microsoft Research, Agreement Maker developed at University if Illinois etc.

Schema matching can be a very challenging task as it can be applied on complex data models with minimal similarity. Even in models representing same data, there can be huge variation in the structuring and naming of the attributes of the models.

Historically schema matching was done manually, but given the rising complexity of the schemas used, ever increasing number of enterprises catering to databases and the advancement in the networking demanding integration of data from different sources, methods for automatic schema matching have become necessary. The process however hasn't been fully automated till date but the amount of human interaction required has been decreasing as better solutions are proposed and more advanced tools are developed.

In the next section we provide the different use cases of matching schemas. Section 3 introduces a generic match operator for better understanding of different schema matching algorithms proposed overtime as mentioned in section 4. Various tools have been developed based on these algorithms, hence in section 5 we discuss some of the schema matching tools. Table I gives the comparison between these algorithms. Section 6 gives an insight into the trend of development of solutions for this problem and the future expectations. Finally we conclude our discussion in section 7.

II. USECASES- APPLICATION BASED

Schema matching is critical to many applications in the domain of databases. In this section we discuss various systems in which this problem find application.

A. Data Warehouses

A data warehouse of an enterprise is a unified storehouse for all the data that different systems within it collects over a period[16]. They are used for analysis and querying instead of transaction processing. Schema matching find application in these systems as whenever a new data source needs to be integrated into the existing system, a match can be found between the existing transformation from other sources and then the transformations for the common elements can be reused.

[1]Nikit Begwani(130101055), Department of Computer Science and Engineering, IIT Guwahati. nikit.b at iitg.ernet.in

[2]Rohan Gupta(130101066), Department of Computer Science and Engineering, IIT Guwahati. g.rohan at iitg.ernet.in

B. Federated Databases

A federated database system is a type of meta-database management system (DBMS), which transparently maps multiple autonomous database systems into a single federated database [17]. An application querying a federated database can therefore query into multiple databases in a single query instead of querying each database separately and then combining their output. The primary challenge in designing a federated database is problem of matching the input database schemas.

C. E-Commerce

E-commerce connects multiple sellers to various customers spread over a wide geographic area. It deals with providing platform to the customers to buy goods from different sellers online. One of the challenges faced in this is related to the different data models used by sellers for placing orders. The request to order needs to converted to a format supported by that particular seller. This process needs to be done manually but the large pool of online sellers make this task difficult.

Therefore schema matching approaches can be used to get relations between data models used by various sellers to automate this process.

D. Data Integration

Data integration is the process of merging data from distinct sources to provide a combined view to the user of the data. This process finds an application in a wide range of domains ranging from industrial usage to scientific application like merging DNA samples from different sources to study the process of evolution based on the similarity between DNA.[18].

A huge amount of data has been collected in the past few years. Data integration is the first step towards finding some meaning relations in this data. Schema matching algorithms can be employed for this.

III. GENERIC MATCH IMPLEMENTATION

For better understanding of this survey, we introduce an architecture which states an abstract overview of how *schema matching* technique works.The idea of constructing architecture is inspired from [3]. After studying several algorithms, we came to a generic picture of how these algorithms are implemented.

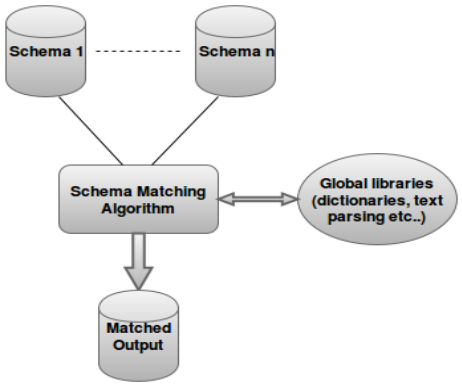

Fig. 1. Generic Implementation of Schema Matcher

Figure 1 shows the generic implementation of schema matcher, it takes different schemas as input, these schema are present in their *native schema representation* (such as XML, SQL, or UML), which leads to inconsistency in type of input parameters. To avoid this, generally input schemas are represented in a *uniform internal representation* (such as graphs, objects, etc) depending on the algorithm used. This internal representation reduces the burden of dealing with individual schema representation.

To smoothen the conversion, the matcher uses import options from the native schema representation to identify different entities (such as nodes for graphs, table name and columns for creating objects). After this the schema matching algorithm is applied. The algorithm might require access to some *global libraries* like dictionaries, string matching, text parsing, or may be some generic formulas like calculating average or range for integer columns. The algorithm then may return either a concrete output or outputs with some similarity coefficients. Depending upon the schema and the algorithm used, the output matching pairs may require to be manual checked for taking decisions to decide which pairs to be considered as matched and which pairs to be rejected. Doing this requires outputs to be again represented in different native representation as demanded by the user.

All the algorithms survey in this paper follows the above method with changes in the method of representation, libraries and the execution of algorithm.

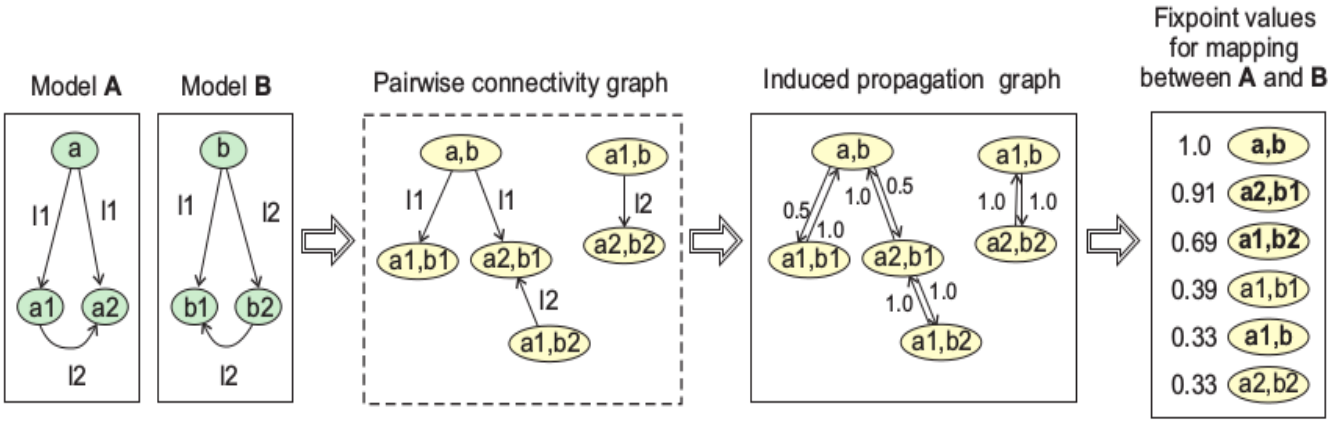

Fig. 2. Depiction of Graph Matching using Similarity Flooding Algorithm.

IV. SCHEMA MATCHING APPROACHES

In this section we discuss major schema matching approaches developed over the years. We discuss briefly about the basic approaches which were known in the late 90's and then we discuss about the recent development in these approaches pertaining to the world of large databases and the world of internet. In a broad sense schema matching approaches can be classified into two types i.e. individual matching algorithms or *individual matchers* and multiple matching algorithms or *multiple matchers*. Over the years of development, we have also seen strategies based on user interaction and feedback in the matching process, strategies using machine learning and artificial intelligence with probabilistic matching, etc.
We first discuss few well studied basic approaches and then move to some advanced approaches:-

- Structure or schema-based matching - Given schemas are said to be similar if they have the same hierarchical graph[1] i.e. if both schemas have similar number and type of tables with similar number and type of columns, we declare a match.
- Constraint-based Matching - We compare the attributes of a schema based on the constraints such as datatype, isNULL, Primary Key, range, etc. We declare a match if both schemas have attributes with same constraints.
- *Instance-based matching* - In this approach we compare the given instances of schemas and declare a schema match if the given instances match above a threshold level. Instance may represent a section of table containing data, output of a query when ran on both the schema, etc.
- Using Auxiliary information - Apart from matching based on the instance level or structure level, we can make a match based on auxiliary information provided, like, we can use dictionaries, linguistic approach, previous-matching decisions, comments or description provided, etc.
- Multiple matchers - We can use more than one matching technique at a time either by matching over multiple criteria by running individual matchers parallely, one on each criteria, or by running sequence of individual matchers one after the other.

A. Graph Matching

This approach follows a series of steps, from conversion of SQL to graph representation to initial mapping using string matching on the graph generated, further *similarity flooding* algorithm is applied and the output is selected by varying the threshold similarity level.

It represents each table or column by node in graph with two attributes associated with each node (type:- column/table ,name:- name of entity). Then a hierarchical graph with edges going from database schema to tables to columns is created for all schema which needs to be matched.

Consider figure 2 (proposed in [1]) for *similarity flooding algorithm*, here first pairwise connec-

tivity graph is generated from hierarchical graph. Connectivity graph combines both the schema as follows - consider tuples (a1,b2) and (a2,b1), it says that if a1 and b2 are of similar type then probably types of b2 and a1 are also same, the evidence for this is provided by edge between them which has same type in both the graphs. After this, induced propagation graph is generated with a new edged attribute known as propagation coefficient which range from 0 to 1. In this, edges are constructed in both the direction and the value is allocated depending upon the number of out-branches of a similar type edge from one node. For example suppose two edges with type l1 branches out so each is allocated 0.5 similarly 0.33 for three and so on and so forth.

Then the fixpoint computation is applied, it defines $\sigma(a, b) \geq 0$ as the similarity measures of nodes a $\in$ A and b $\in$ B, we do it for some predefined multiple iterations and the similarity measure is calculated. Calculation of $\sigma(a, b)$ is very well discussed in [1]. After this various filters and threshold levels are applied to have a more reliable match, at times it might also require manual matching which is decided by *matching accuracy* which states that how much effort it costs the user to modify the obtained output. On an average the results shown in [1] provides a very good manual effort reduction of around 50%.

B. Usage-based Matching

This approach is totally different from other practices of matching based on schema or data instances, instead it declares a match based on the use of schema i.e. based on *query logs*. This technique may be used when information provided regarding schema is incomplete or unreliable.

Techniques have been developed to provide a suitable match even when table or column name are unknown and their layout is not provided [2]. The method proposed in [2] has two main phases namely *feature extraction* and *matching*. The first phase grabs the information from the queries and characterize the attributes based on their roles and inter dependencies with other attributes. Next, the matching phase examines several potential matching pairs and assign a score to each one of them. This phase completes after reporting the pairs with highest scores.

In feature extracting phase , we extract *structure level features* which informs about the usage relationships between attributes of same schema. As proposed in [2],the attributes define role in queries,there are four roles namely answering role (*select*), grouping role (*group by*), filtering role (*where*) and ordering role (*order by*). Depending on these role 16 (4X4) relationship graphs are constructed in which each vertex represents an attribute and we have an edge between two vertices if they participate in that relation in any query. We then find the mappings which gives the highest score for a particular scoring function. There are different scoring methods proposed in [2]. The difficulty of forming an effective scoring function to compare between different mappings has been addressed in [4]. Most of the usage-based matching algorithm is based on this basic method[2].

C. Early Search Space Pruning

This method is based on *multiple matcher*, we first run a fast matching algorithm to eliminate the unlikely matches and then we proceed on to apply more costly or more accurate algorithm on the remaining candidates. The current matching algorithms lack severely in performance when they are used for matching very large schemas, this leads to a tradeoff between speed and accuracy. To avoid this tradeoff and to take advantage of both, we use fast and less accurate mechanisms like linguistic matching or constraint based matching and then we move on to move advanced techniques as *rewrite based optimization technique*[8] or *Query Ontology Mapping*[9] which are comparatively slow but have shown a higher accuracy rate.

D. Fragment-based matching

Schema matching for large databases has been very costly and inaccurate , to improve performance in matching large schemas, one of the method proposed is *fragment-based matching*[10] this method follows divide and conquer approach.

It is quite possible that many portions of large schema doesn't have a matching counterpart so, it becomes efficient to break schemas into fragments and match those parts which have high similarity index. With this we can achieve not only better running time but a high quality matching. Apart from user method for selecting fragments,

three strategies are proposed in [10] for selecting fragments automatically. The base strategy being considering schema as a single fragment. Another strategy is considering subschemas as fragments. Subschemas represent those portions of database which can be separately taken into account without having dependency on each other.For example - tables involved in a relationship in relational databases may be considered as a subschema. Next strategy is based on selecting fragments based on their usage this category shows a high potential for matches[10].

Once the fragments are decided, both the schemas partitioned into fragments are given as input to fragment matcher if a match is found, it outputs the fragments in pair containing one fragment from each schema. Then these fragments are further passed to *individual matchers* which then run schema matching algorithm on these fragments.

If we consider the time complexity then there are actually tradeoffs depending upon the approach for fragmentation. If we consider fragments based on schema and subschema method then step 1 of obtaining similar fragments will be fast while the second step of matching schemas will be slower. On the other hand if shared scheme is used then step 1 will consume more time compared to step 2. However if number of fragments are increased then step 1 and step 2 both takes longer execution time.

E. Parallel Matching

Parallel matching algorithms are used when different cores or computer nodes are available for computing simultaneously. Broadly parallel matching is classified into two categories *inter-matcher parallelization* and *intra-matcher parallelization*. *Inter-matcher parallelization* helps in parallel execution of independently running matchers to utilize all the resource available while on the other hand *intra-matcher parallelization* helps in parallel execution of a single *individual matcher* which leads to a faster execution of matching algorithms one technique for doing so is by dividing them into fragments[10].

In past years several matching algorithms using divide and conquer have been proposed where a small subset of input schemas are matched with each other. One such method is used in COMA++[6] which divides the schema into fragments and then run parallel matching algorithms on them. Another such method is clustering which forms clusters based on linguistic approach and then the performance of matcher is improved by running it on the minimized sample space. One very popular method based on probability measures is proposed in [9] it uses heuristics to reduce the number of possible matchers which then reduces the number of comparisons.

V. Tools For Schema Matching

In this section we provide an insight to into the architecture and techniques used in some of the schema matching tools that have been developed by researchers and companies. Table I gives a comparative study of these tools and hence also summarizes the functionality offered by them and techniques used.

A. Cupid

Cupid is schema matching algorithm developed at Microsoft Research [5]. Motivation behind development was to generate a general purpose schema matcher. It achieves robust functionality through combination of multiple schema matching approaches. It combines the element based matching along with structure based matching. It is a schema-based matcher and does take data instances into consideration.

Matching is performed in three phases; first phase is linguistic matching, second is structural matching followed by generation of mapping.

In linguistic matching, names of elements in the schema are used for matching. The names are tokenized, for eg ROLLNo goes to {ROLL, No} followed by categorization using various parameters like data type, class container etc. Name similarity coefficient is calculated using the dictionary and finally the linguistic similarity coefficient is calculated using the categorized parameters and name similarity.

Next, for each pair of elements of the two schema, structural similarity is calculated. The input schema are converted into a hierarchical tree and matching between elements is performed starting from the bottom.

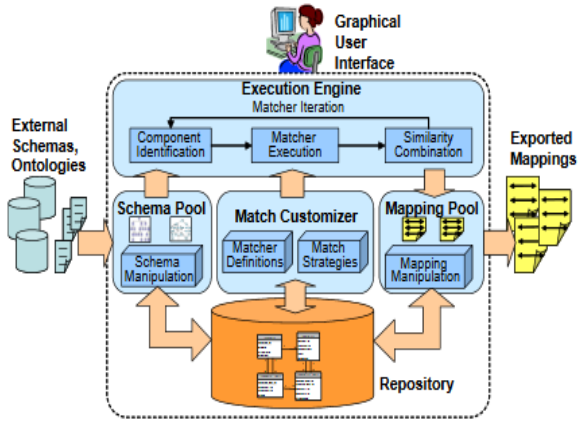

Fig. 3. Architecture of COMA++

The mapping between elements is generated using the linguistic and structural similarities. This step is application dependent and hence different techniques are employed based on requirements of the module.

B. COMA++

It is an extension of previously developed prototype COMA [6]. It employs a combination of various match features to perform matching on generic data models and offers comprehensive UI support for its users.

Figure 3 shows the architecture of COMA++. The repository stores all the meta-data related to the match operation: input schemas, synonym tables, configurations of matcher, mappings generated etc.

COMA++ provides support for multiple formats like XSD, XDR, OWL and relational schemas. The schema pool has various methods for importing external schemas and load and save operations from the repository.

The matching operation is done in the execution engine in three steps as shown in the figure. In the identification step, schema components to be used for matching are used followed by the matcher which performs the matching operation based on various matching techniques, Match Customizer contains the pool of match strategies that can be applied on the imported schema. These steps are repeated multiple times to generate multiple matchings which are later used to get the final matching.

Matching Pool maintains all the generated matchings and offers various functions to further manipulate them.

C. U-Map

U-Map is a schema matching tool for usage-based matching [11]. U-Map is particularly useful in cases where the correspondence between the two input schemas is low in terms of their structure. The use of query logs instead of the schema structure, can give a decent match. Moreover their use can also capture associations that may have been missed by other schema based matchers.

Matching is done in two main steps: Correspondence generation and matching generation. In correspondence generation step, features about attribute use are derived from the query logs. Two levels of features are used, structure level are used for get relationship between attributes based in usage and element level to characterize individual attributes.

Next U-Map tries to identify relations having IS-A relationship between them using the query logs. For example is relations might be bound in IS-A relationship if they have references to the same foreign keys.

Next step is generation of a mapping function. A scoring function is computed for every pair of attributes reflecting the level of similarity for that pair.

D. Clio

Clio is semi-automatic schema integration tool developed by IBM. It provide support for matching XML and SQL schemas. This evaluates the schemas in three steps.

The first step is the concept extraction step. The software recasts each source schema, with its constraints and nesting, into a higher-level concept hierarchy (with no constraints or nesting) [13].

The next step is concept matching and merging step. Each pair of attributes or concepts are matched, either into one integrated concept or left unmatched. This step generates a number of possible sets consisting of integrated attributes.

The final step is integrated schema refinement. This step is manual step. The user interacts in this steps with a comprehensive UI to evaluate various sets generated in previous step.

TABLE I

COMPARISON OF SCHEMA MATCHING TOOLS

	Cupid	COMA++	U-MAP	Clio	Agreement Maker
Linguistic Match	✓	✓	✓	✓	✓
Ontologies	-	✓	-	-	✓
Relational Database	✓	✓	✓	✓	-
XML Database	✓	✓	-	✓	✓
Parallel Matching	-	-	-	-	-
Schema Partitioning	-	✓	-	✓	-
Use of External Dictionaries	✓	✓	✓	✓	✓

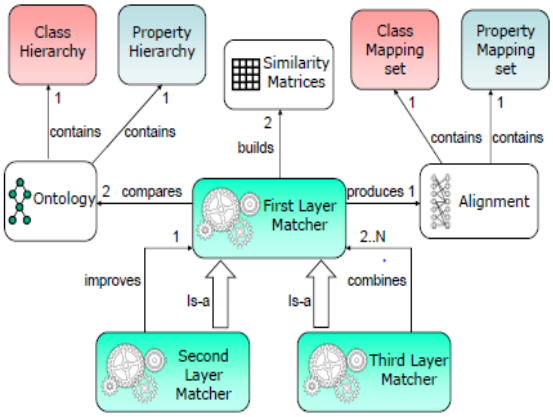

Fig. 4. Architecture of Agreement Maker [12]

It uses a hybrid approach for generation of sets in step 2. String matching is used for element names and Bayes algorithm is used for instance data.

E. Agreement Maker

Agreement maker is a system for matching real world schemas and Ontologies, which may consist of thousands of concepts [12]. This system has been developed for domain experts and offers many advanced features.

Some of the unique features offered by this system are that it supports input in multiple formats, it features a wide variety of schema matching approaches and also provides support for easily accommodating new techniques, this tool can also be used to compare the performance of various matching techniques. It provides comprehensive interface for the end user.

Figure 4 shows the architecture of the agreement maker system. The process of matching can be broadly divided into two steps, first step consist of evaluation of similarity between all pairs of attributes of the input schemas generating a similarity matrix. In the next steps matching method are used to compute a match using the similarity matrix. Several matching techniques are used in this step.

The similarity matrix is computed by the first level matcher, it uses various algorithms for lexical and syntactical comparison. Second level matcher improves the result of first level matcher using the structural properties and use algorithms like the descendant similarity inheritance and the sibling similarity contribution. Third level matcher combines the result of multiple iterations of previous level matchers to compute final matching.

VI. FUTURE TRENDS

Since the *dot-com boom*, people have realized the importance of having huge data and extracting maximum information out of it. Schema matching has been playing an important role in this area.

We have seen a decent amount of work in this field pertaining to application domain. Seeing the importance of this problem, we believe that there is scope for doing a more quantitative intense research on schema matching which can help better decide that which method produces a better result under given circumstances.

We believe that with the importance of huge data being realized, schema matching can help in various other streams apart from computer science. For example, schema matching can help in maintaining a global big data for diseases by gathering and combining data from different medical

institutions. The above principle can be applied to various other fields as well like gathering data for understanding plant growth throughout the world, providing price list of various products being offered globally, etc. The more we move towards optimization in this field, the more accurate its applications become.

In recent times, there has been a lot of machine learning techniques[14] getting incorporated into schema matching which has shown improvements in running time. These algorithms make a match depending upon the results of already matched outputs, we can expect a better similarity coefficient as an addition benefit in this approach.

Till now, manual effort has not been reduced to zero, to have a better and escalated manual processing, people have tried developing visual tools[15] which provide a graphical representation of mappings.

VII. CONCLUSIONS

In this paper, we briefly discussed about the existing schema matching techniques developed so far, we covered some details of the well known algorithms and discussed about a generic implementation of them. We also provided a summarized information about the tools which have been developed till now.

Schema matching is still an open ended problem with scope for improving accuracy, getting better similarity coefficients, improving speed and reducing the manual effort required. With the growing rate of usage of internet, we see a great increase in amount of data being collected each day from users. Such a large data can be clustered together by using schema matching on different schemas obtained from different data collection centers and can be used to provide different types of services to users.

There is always availability of new information which can be incorporated into improving available techniques or combining them in a clever way. Thus essentially, schema matching is still a young and vibrant topic with opportunities for developers and researchers to make and optimize tools.

REFERENCES

[1] Melnik, S., H. Garcia-Molina, and E. Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. Proc. ICDE, 117-128, 2002.

[2] Elmeleegy, H., M. Ouzzani, and A.K. Elmagarmid: Usage-Based Schema Matching. Proc. ICDE, 20-29, 2008

[3] Rahm, E. and P.A. Bernstein: A Survey of Approaches to Automatic Schema Matching. VLDB J. 10(4), 334-350, 2001.

[4] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In ICDE, 2005.

[5] Madhavan, Jayant, Philip A. Bernstein, and Erhard Rahm. "Generic schema matching with cupid." VLDB. Vol. 1. 2001.

[6] Aumueller, David, et al. "Schema and ontology matching with COMA++." Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005.

[7] Cruz, Isabel F., Flavio Palandri Antonelli, and Cosmin Stroe. "AgreementMaker: efficient matching for large real-world schemas and ontologies." Proceedings of the VLDB Endowment 2.2 (2009): 1586-1589.

[8] Peukert, E., H. Berthold, and E. Rahm: Rewrite Techniques for Performance Optimization of Schema Matching Processes. Proc. EDBT, 433-464, 2010

[9] Ehrig M., and S. Staab: Quick ontology matching. Proc. Int. Conf. Semantic Web (ICSW), Springer LNCS 3298, 683-697, 2004.

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

[11] Elmeleegy, Hazem, et al. "U-MAP: a system for usage-based schema matching and mapping." Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011.

[12] Cruz, Isabel F., Flavio Palandri Antonelli, and Cosmin Stroe. "AgreementMaker: efficient matching for large real-world schemas and ontologies." Proceedings of the VLDB Endowment 2.2 (2009): 1586-1589.

[13] Chiticariu, Laura, et al. "Semi-automatic schema integration in clio." Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007.

[14] Doan, A-H., P. Domingos, and A.Y. Halevy: Reconciling the Schemas of Disparate Data Sources: A MachineLearning Approach. Proc. SIGMOD, 509-520, 2001.

[15] Falconer, S.M. and N.F. Noy: Interactive Techniques to Support Ontology Matching. In: Z. Bellahsene, A.Bonifati, E. Rahm (eds), Schema Matching and Mapping,Springer, 29-52, 2011

[16] Data warehouse definition. Retrieved April 19, 2016, from http://searchsqlserver.techtarget.com/definition/data-warehouse

[17] Federates database definition. Retrieved April 19, 2016, from http://en.wikipedia.org/wiki/Federated_database_system

[18] Data Integration. Retrieved April 19, 2016, from https://en.wikipedia.org/wiki/Dataintegration