LogMap+: Relational data enrichment and linked data resources matching

Slavko Žitnik, Marko Bajec and Dejan Lavbič
University of Ljubljana
Faculty of computer and information science
Večna pot 113, SI-1000 Ljubljana
{slavko.zitnik, marko.bajec, dejan.lavbic}@fri.uni-lj.si

***Abstract*—Relational database to ontology mapping and ontology matching techniques are mostly addressed separately, even though it is known that the real power of semantic data lies in data interconnection. The latter is especially important when designing a new ontology, which often includes at least some of the concepts that already exist in the linked open data cloud. Thus, in this paper we describe a new end-to-end tool LogMap+ for transformation of relational data into an ontology and matching it against a pre-existent semantic source. Apart from offering the efficient web-based application, the main contributions are the improvements of the domain specific LogMap system. We evaluate our general tool against OAEI 2014 challenge datasets and achieve comparable results to the top performing algorithms and also outperform the domain specific LogMap tool.**

***Keywords*—Ontology mapping, ontology matching, relational database, OAEI.**

I. Introduction

The concept of hyperlinks as we know it from the Internet was first coined in 1945, when Vannevar Bush defined interconnection of distant but semantically connected data in his article "As We May Think" [1]. The World Wide Web today consists of a huge amount of documents, which are indexed by search engine providers but they are still not searchable using semantic queries as Semantic Web documents could be. Organizations also often store their data within the relational databases, which are used by specialized applications and therefore not open (even though some could be) or interconnected to semantically close concepts in other data stores.

On the other hand, there exists a cloud of linked open data, which is easily accessible and can be interconnected with new or existing semantic sources. Thus, we propose an end-to-end tool LogMap+, which offers a mapping of a selected relational database schema into an ontology and then matching it against a pre-existent semantic source.

Ontology matching is an important operation during an ontology design as the whole linked data cloud is well connected and heterogeneus [2]. The principle of connecting multiple ontologies is also practically useful as it allows (1) to design smaller and self-sufficient modules instead of large ontologies, (2) to express relationships between multiple versions of the same ontology, and (3) more efficient design of a general ontology with relationships to more (domain-)specific ontologies.

To evaluate ontology matching techniques, Ontology Alignment Evaluation Initiative (OAEI) organizes regular evaluations since 2004. The goal of the initiative is to define guidelines for development of successful ontology matching systems and algorithms in different domains.

The paper is structured as follows. In the next section we review the related work for the relational database to an ontology mapping and overview ontology matching tools. Then in Sec. III we describe our end-to-end LogMap+ tool and explain the improvements in the part, where we use LogMap. We separately evaluate the whole tool against a use case scenario and the improved version of LogMap against the OAEI 2014 challenge datasets in Sec. IV. We conclude with the discussion of the proposed solution.

II. Related work

To our knowledge, there exist no previous work that would focus on both mapping relational data to an ontology and matching it to existing linked data sources because also in practice the mapping is already a big challenge. Still, there has been a lot of research done for each of the tasks separately.

The task of mapping relational databases to an ontology (RDF/OWL) format is nowadays well understood and widely studied. Some of the RDB-to-RDF/OWL mapping languages and tools are D2RQ [3], Virtuoso RDF Graphs [4], Ultrawrap [5], RDB2OWL Lite [6] and W3C standardized language R2RML [7]. Most of the mapping languages and approaches, however concentrate most on clear mapping structure and efficient implementation with less attention paid to the concise mapping writing possibility suitable both for manual mapping information creation and using of the mapping information as semantic-level documentation of the relational database structure. There are more higher-level mapping definition means in ontop [8], where the mappings are described as separate artifacts to be considered besides both the database and ontology structures. Authors of the RDB2OWL language [9] have shown a principal possibility of reusing both the target ontology and source database structures in the mapping definition via placing the mapping information in a compact textual form into the annotations of the target ontology entities. In our work, we decided to use R2RML because among other features it enables the user to easily adapt the mapping document, it allows the incorporation of definition of multiple logic tables and also a representation of a logic table using an SQL statement.

After we transform the database into an ontology, we

978-1-5090-5476-3/17/$31.00

need to enrich it. The latter can be achieved using ontology matching technique, which is a process of entity alignment among different ontologies [2]. Interconnected ontologies are of great importance in the linked data field, so we build on the top of successful approaches. Rahm and Bernstein have first published a classification of the existing approaches [10], which is still valid. First they separate individual matcher approaches and combining matchers. The individual ones are based on the matching objects, that can be schemas only or instances/contents of the ontology. The combining matchers on the other hand can use a composition different approaches and therefore achieve better performance. The most recent classification (Fig. 1) was provided by Euzenat and Shvaiko [2], who list known techniques that have been used for ontology matching and classify them from the perspective of kind of input and its representation.

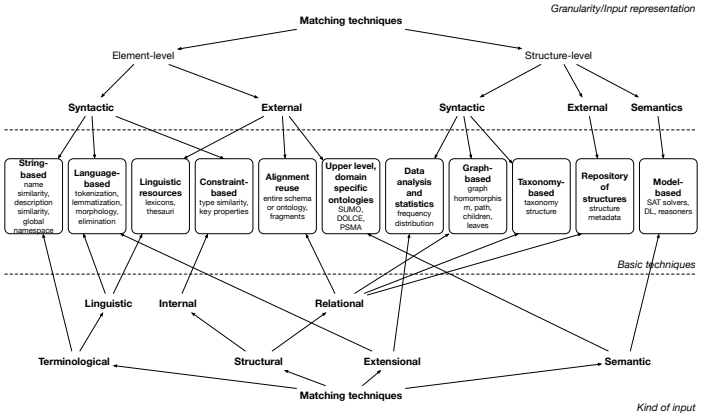

Fig. 1. Classification of basic techniques for ontology matching.

At the time of writing this paper, we found 89 tools for ontology matching. We further classify them into tools that work based on schemas or instances as matching objects or combine both of the approaches (Fig. 2). Among all the tools we decided to use Logic-based and Scalable Ontology Matching (LogMap) [11] because it already features semantic matching based on the ontology structure, returns also the negative matchings, allows for semi-automatic matching, so the user can be actively involved during the runtime and is open-source, so we could easily implement additional features.

III. LogMap+

The end-to-end relational data enrichment and linked data resources tool LogMap+ (Fig. 3) is a novel tool that enables automatic SQL-based data into RDF format transformation and enrichment with existing linked data sources. In the first part we propose an approach of how to efficiently use relational data schema and instances to build a new ontology and then in the second part we improve the existing LogMap tool to allow for a more general ontology matching. LogMap+ therefore introduces the following improvements over LogMap:

- The use of multilinugal lemmatization tool for transformation of strings into a basic form. In the implementation we use Lemmagen[1] tool.

[1]http://lemmatise.ijs.si

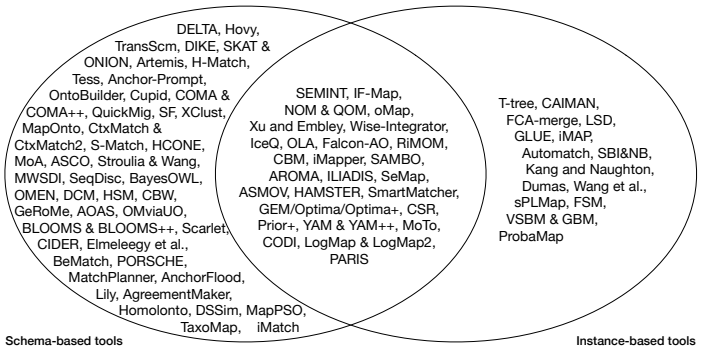

Fig. 2. An overview of existing tools for ontology matching. On the left we show 47 tools that are based on schema matching, on the right we show 15 tools that are based on instance matching and in the middle we list 27 tools that combine both approaches.

- Incorporation of string variantions into the index. We use WordNet to retrieve synonyms, antonyms, etc.
- Implementation of supporting elements property. We use a special property for a representation of neighbouring tables when mapping from an SQL database to an ontology before matching. Supporting classes represent additional context in our new matching step.
- Implementation of matching using context. We define a new relational metric that takes lexical and contextual matching into account.

A. Relational data to linked open data mapping

In short, we design the mapping from a relational data to a new ontology using the following steps: (1) selection of an SQL database to map, (2) definition of appropriate support classes to improve the enrichment in the second part, (2) selection of two representative tables (first and last) that include data to map, (3) selection of an appropriate path between the tables that can contain additional data for a new RDF class, and (4) naming of the concepts in a new ontology.

To map data from a relational database, a user needs first to select data for enrichment. The relational data of interest is defined by the two database tables and a number of paths between them. A user selects one of the possible paths and all the attributes he wants to map into an ontology class. The whole process of mapping a relational data resource to an RDF class is represented in Fig. 4.

After the credentials to a relational database are given, we use SchemaSpy[2] tool to represent an entity-relationship diagram. We transform the diagram into a directed graph $G = (T, F)$, where tables are represented as nodes $T = T_1, T_2, \ldots, T_n$ and foreign key constraints between tables $F = F_1, F_2, \ldots, F_m$ are mapped to directed links. Each foreign key $F_x = (T_a, T_b)$ represents a directed link. We build the graph using the following rules [12]:

- $T_x \in T$ is dependent on table $T_y \in T$ if and only if a walk exists between T_x and T_y,

[2]http://schemaspy.sourceforge.net

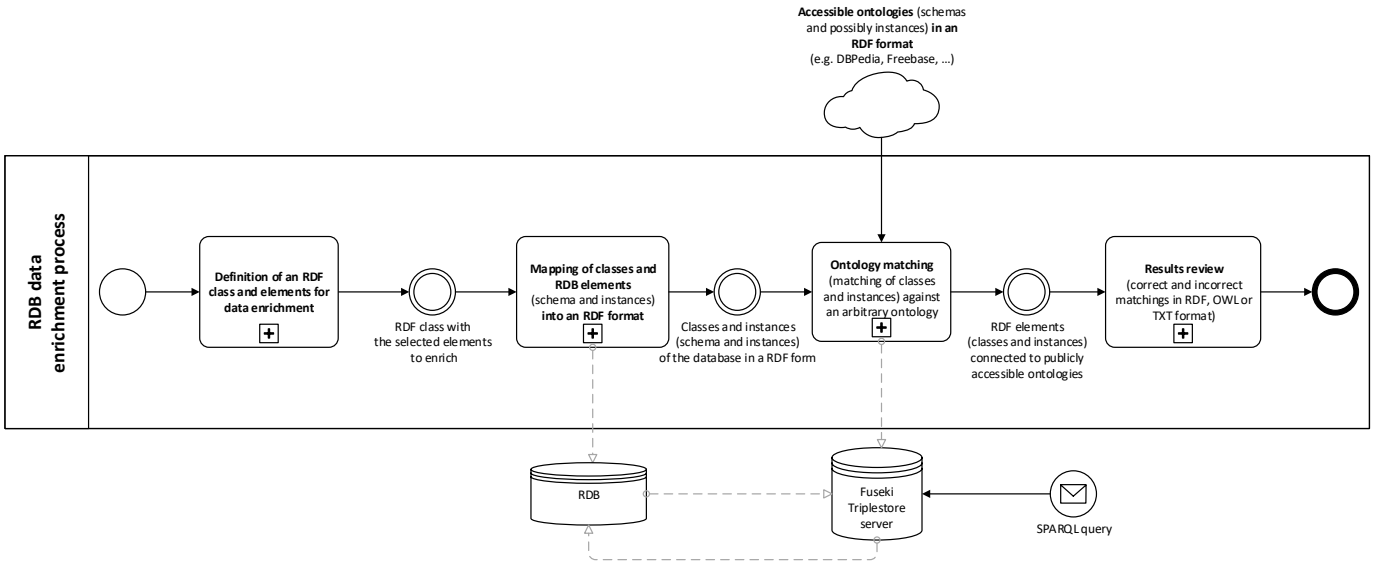

Fig. 3. LogMap+: End-to-end process of relational data mapping and ontology matching.

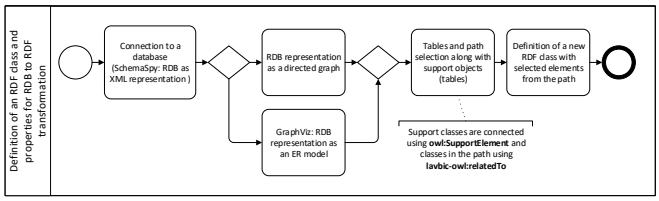

Fig. 4. Process of a relational subschema and a new RDF class definition.

- $T_x \in T$ is directly dependent on table $T_y \in T$ if and only if there exists $F_x \in F$, where $F_x \in (T_x, T_y)$,
- $T_x \in T$ is indepedent on table $T_y \in T$ if and only if a walk does not exist between T_x and T_y,
- $T_x \in T$ is dependent on itself if and only if there exists a directed cycle in the graph G that includes the node T_x.

From a graph G, a user selects two tables, which he would like to map to an RDF format. To retrieve all sensible attributes to a new RDF class, we find all possible paths between the two tables in the graph. We use a variant of breadth-first search algorithm, which returns the paths sorted by their length. We show the algorithm in Alg. (1). After a user selects one of the paths, he can further select all the attributes from the tables on the path to form a new RDF class definition.

Then, we automatically create a mapping document of schema and data from a relational database to RDF triples. The process is shown in Fig. 5. According to the selected relational subschema, we generate a R2RML mapping document, which a user can further manually adapt for special cases if needed. In the next step we use *db2triples*[3] tool, which can convert data

[3]https://github.com/antidot/db2triples

Algorithm 1 Calculate paths in ascending order

Input: G, $T_{start} \in G$, $T_{goal} \in G$
Output: Ordered paths from T_{start} to T_{goal}

```
function BFS(G, T_start, T_goal)
    R ⇐ [ ] //list of found paths
    Q ⇐ [ [T_start] ] //current paths
    while Q is not empty do
        P ⇐ first element in Q
        T_temp ⇐ last element in P
        for all T_n ∈ children(T_temp) do
            if T_n == T_goal then
                R+ = (P++[T_n])
            else
                Q+ = (P++[T_n])
            end if
        end for
    end while
    return R
end function
```

to RDF triples according to a R2RML mapping document. The tool was selected because it can handle the mapping directly using SQL statements that can be manually fine tuned by a user [13]. Relational database schemas include a lot of metadata, which we also map in order to improve data enrichment for ontology matching in further steps. At the end of this step we store triples directly into an RDF database (i.e., Apache Fuseki), which enables SPARQL querying.

B. Ontology matching

In this section we propose an improved algorithm of LogMap for ontology matching, which we use for RDF data enrichment. In the first part we explain the specifics of the LogMap tool and then describe our improvements in the LogMap+ tool. The general idea of ontology matching process

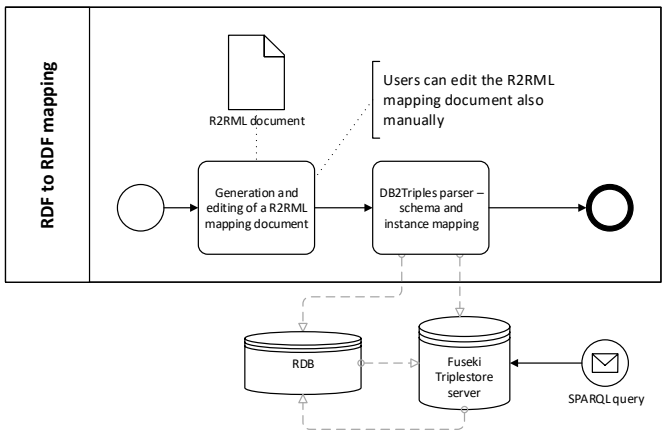

Fig. 5. Process of a relational schema and data mapping to RDF triples.

is represented in Fig. 6. A user needs to select two ontologies for matching. Depending on a matching tool, it can also allow for different parameters setting or specific rules definition.

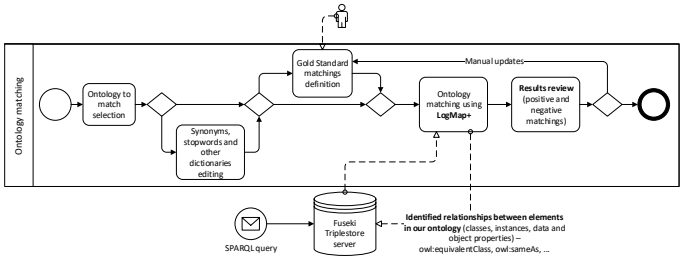

Fig. 6. General representation of an ontology matching tool.

1) LogMap ontology matching: LogMap is a scalable tool for ontology matching, which can operate with semantically rich ontologies consisting of a few millions of RDF triples. It supports also property and instance matching. The incorporated reasoning functionality allows for inconsistency identification within the matched ontology. Fig. 7 shows the LogMap's process for matching ontologies, which takes ontologies O_1 and O_2 as input and returns positive (M) and negative (M') matches.

In the first step LogMap indexes lexical elements and the structure of input ontologies. It indexes data using inverted indexes for classes and instances. For the indexing it uses stems and alternative strings (e.g., synonyms, hyponyms) that it retrieves from UMLS [14] database. For the structure indexing it creates a hierarchy of classes for each input ontology. They are indexed using interval labeling schema [15], which enables simple querying against relationships among them.

In the second step it calculates a set of initial matchings by taking the intersection of inverted indexes from both input ontologies. The matchings are further checked using Stoilos' string similarity metric [16] and local trust metric [11]. The trust metric is used only for classes matching and not for instances, so therefore LogMap does not take context data into account.

The next step is an iterative loop of matchings update & search. The loop executes until there are no new matchings found in an iteration: (1) The update part uses inference and already found matchings to filter the candidate mappings which are not to be matched. The whole structure and also new matchings are represented using Horn clauses. To check their validity, LogMap uses Dowling-Gallier algorithm [17]. (2) The search for new matchings is done in the same way as in the previous step.

The last step of LogMap's ontology matching is the computation of the ontology overlapping score. The user has also an option to approve or reject specific matchings to form the final result of the algorithm.

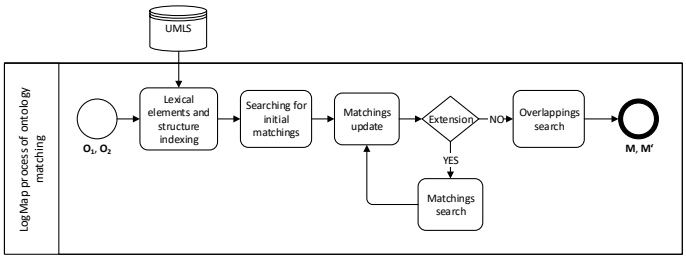

Fig. 7. Process of ontology matching using LogMap tool.

2) LogMap+ ontology matching: In the previous section we can notice the following features LogMap tool is lacking:

- using the context of instances based on their classes,
- comparing the contexts of relational data per instance,
- the use of third-party semantic lexicons and
- the use of stemming only.

In this part we present the improved LogMap+ tool. In Fig. 8 we show the process of the new tool along with the improved functionalities. The tool includes the incorporation of third-party semantic sources and context matching of instances and their relational data.

The Alg. (2) shows all the steps that are needed by the LogMap tool and the improved parts are written in bold. All the specific definitions of the functions can be retrieved from the public source repository [18]. In the indexing of lexical elements step (line 2) we include also string variants as input. First we remove stopwords, then we retrieve lemmas and enrich them with synonyms. We also retrieve their class names and connect them to the same instances within the index.

Then we perform the intersection of indexes (line 3) and based on the result, select only the elements for further processing (line 4). This enables the scalability of the algorithm as we work only on the subsets of the input ontologies - O_1' and O_2'.

Within the structure indexing step (line 5) we build a class hierarchy for each subset ontology. The hierarchy is updated using an additional property "lavbic-owl:relatedTo" (defined in config file), which represents relationships to the tables

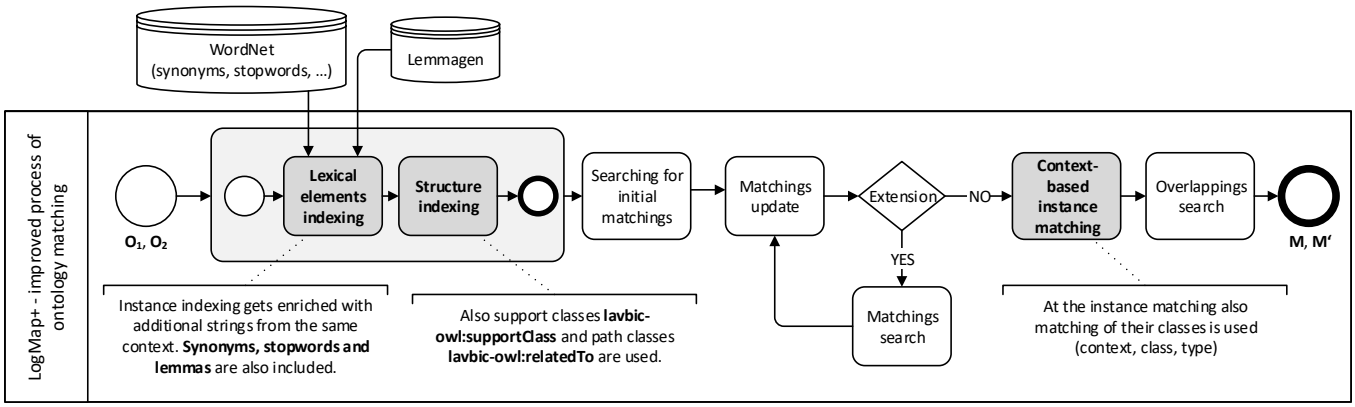

Fig. 8. Process of the LogMap+ tool. The gray parts show the improvements over the existing LogMap tool.

Algorithm 2 LogMap+

Input: O_1, O_2
Output: M, M'

1: **function** LOGMAP+(O_1, O_2)
2: $(LI_1, LI_2) \Leftarrow$ **LexicalIndexes**(O_1, O_2)
3: $M' \Leftarrow$ CandidateMappings(LI_1, LI_2)
4: $(O_1', O_2') \Leftarrow$ Module(O_1, O_2, M')
5: $(H_1, H_2) \Leftarrow$ **StructuralIndex**(O_1', O_2', M)
6: $M \Leftarrow$ ReliableMappings(M', H_1, H_2)
7: $M' \Leftarrow M' \backslash M$
8: $(P_1', P_2') \Leftarrow$ PropEncoding(H_1, H_2, M)
9: $M \Leftarrow$ Diagnosis(P_1', P_2', M)
10: $M' \Leftarrow M' \backslash$ Discarded(LI_1, LI_2, H_1, H_2, M')
11: $M \Leftarrow$ Diagnosis(P_1', P_2', $M \cup M_T$, M)
12: **while** $M_T \Leftarrow$ DiscoverMappings(M, H_1, H_2) **do**
13: $M \Leftarrow$ Diagnosis(P_1', P_2', M_T, M)
14: **end while**
15: $M \Leftarrow$ **InstanceContextMatching**(O_1', O_2', M)
16: **return** M, M'
17: **end function**

along the selected mapping path in the SQL database. Neighbouring tables from the SQL database are of type "lavbic-owl:SupportClass" and are also included into a hierarchy.

The hierarchies and similarity metrics are used to retrieve initial matchings M (line 6). Then we delete all matchings from M', which are marked as reliable (line 7). Now class hierarchies along with the matchings from the previous step are transformed into Horn clauses for detection of incorrect mappings within M (lines 8,9). Further we check the set M' for conflicts in the inverted indexes and class hierarchies. In this step the matchings are re-checked against the alternative strings (lines 10, 11). The last step of LogMap matching is an iterative loop, which is expanding the set of matchings and ends when no new matchings are found in a new loop (lines 12, 13).

At the current stage we have the matchings only based on string similarity between classes. Therefore we added a new step (line 15) in which we use instances and their context to compare the two ontologies. To perform this step efficiently we also propose a relational similarity metric (see Sec. III-D).

C. String similarity metric

To measure similarity between strings we use Stoilos' String Metric for Ontology Alignment (SMOA) [16]. The metric takes into account differences and commonality features between the two input strings s_1 and s_2:

$$SMOA(s_1, s_2) = Comm(s_1, s_2) - Diff(s_1, s_2) + JW(s_1, s_2)$$

$$Comm(s_1, s_2) = \frac{2 \sum_i length(maxCommonSubString_i)}{length(s_1) + length(s_2)}$$

$$Diff(s_1, s_2) = \frac{uLn_{s_1} * uLn_{s_2}}{p + (1-p)(uLn_{s_1} + uLn_{s_2} - uLn_{s_1} * uLn_{s_2})}$$

The function of commonality $Comm(s_1, s_2)$ is defined by the substring string metric. In the substring metric the biggest common substring between two strings is computed. This process is further extended by removing the common substring and by searching again for the next biggest substring until no one can be identified. The difference function $Diff(s_1, s_2)$ is based on the length of the unmatched strings that have resulted from the initial matching step. $uLen$ represents the length of the unmatched string from the initial string. The difference should play a less important role on the computation of the overall similarity, so it is calculated as a parametric triangular norm. Lastly, JaroWinkler (JW) metric [19] is used as a base similarity function. Using parameter p we can tune the weight of string similarity matchings.

The final result of the metric takes values between -1 (non-match) and 1 (full string match).

D. Relational similarity metric

We defined a new relational metric that returns the similarity of an instance matching based also on similarity among their RDF classes. Thus, if string values of instances are similar but classes of these instances are not, a matching is marked as incorrect. The Instance Confidence Factor (ICF) is defined as follows:

$$ICF(n,m) = SMOAAnchor(i_1, i_2) + \frac{\sum_i {}^{n} \sum_j {}^{m} confidence(i,j)}{sizeof(n) * sizeof(m)},$$

where n represents a set of classes of instance i_1 and m a set of classes of instance i_2. The result is on a scale between 0 (instances are completely different) and 2 (complete match). The first part of the metric uses SMOA metric for the two instances and the second part calculates the confidence of matching between the classes of instances:

$$confidence(i,j) = w * SMOAAnchors(i,j) + (1-w) * ScopeAnchors(i,j).$$

The calculation of confidence factor in the first part checks the similarities between string values of classes and then in the second part identifies, whether a pair of classes fall into the same context. During the evaluation we use default weight w of 0.5.

IV. Results and discussion

We verify our approach using two types of evaluations. In the first part we start with Sakila model database [4], transform it to a RDF format and then match it against the public DBPedia ontology. Then we evaluate only the matching process explicitly (i.e. LogMap+) against Ontology Alignment Evaluation Initiative (OAEI) [20].

The web based tool LogMap+ that supports transformation of an SQL database to an ontology and further the ontology matching to other linked data sources is available in the public source repository [18].

A. End-to-end process evaluation

To support an end-to-end process of transforming relational data into an enriched ontology, we developed a modern web-based application that consists of a six guided steps to achieve the goal.

Step 1: A user selects a relational database type and credentials for the tool to access the database. The database schema is shown to the user as a dynamic or static image of entity-relationship diagram.

Step 2: The selected database is represented as a directed graph (see Sec. III-A) The user selects the tables he would like to include in an ontology and also the first and the last table. The tool then calculates and shows all possible paths between the tables. Now, the user needs to select a path that he would like to use for the mapping.

Step 3: The user defines a name of a new RDF class along with the attributes. The attributes can be selected from all the tables that were found between the first and the last table in the previous step.

Step 4: The tool prepares a R2RML mapping document, which can be fine-tuned by the user. The document defines that database tables to map to RDF classes and database rows to the ontology instances. The new classes are connected using "lavbic-owl:relatedTo" relationships. The neighbouring tables of the first and the last selected table are included into the mapping to improve the accuracy of the new ontology matching. We name these classes the supporting classes and they are connected to the other ones by the "lavbic-owl:supportClass" object property.

Step 5: The user defines the new ontology namespace and export format along with a location (a file or an ontology data store). The DB2Triples tool is then used to execute the transformation from the relational database to the selected ontology format.

Step 6: The user selects a public linked data source or an ontology to map the new ontology against. Next, the user can set the matching parameters: correct matchings as a golden standard, additional lexicons to use (e.g. synonyms, stopwords, WordNet), thresholds, constraints (e.g. maximum number of sysnonyms to take into account), new URIs definition. As a result, the tool generates matchings in a selected format and also a list of matchings that were marked as incorrect. After the user confirms the result of a matching, the new RDF statements are written into the matched ontology.

To validate the process above, we use the Sakila model database, from which we selected a path between the tables "Country" and "Staff" (i.e., *country, city, address, store, staff*). We map the selected path to an ontology and match it with DBPedia[5], which is one of the largest RDF data sources. To validate the whole process, we use three different system setups:

- **Setup 1:** The use of LogMap without our improvements. Therefore no context from support classes is used and the default UMLS thesaurus is set.
- **Setup 2:** LogMap with all of the our improvements without lemmatization (stemming is still used) and third-party contextual sources (WordNet or UMLS).
- **Setup 3:** LogMap with all our features and improvements.

During the evaluation we execute each scenario with different threshold values that define the validity of the matchings. The value of a matching between classes is defined by a measure of confidence and a string similarity metric. On the level of instances we use a context metric. Prior to the evaluation we manually defined the correct matchings (i.e., golden standard matchings) between Sakila database and DBPedia.

In Fig. 9 we show the number of identified matchings for a specific threshold level. We observe that only Setup

[4]https://dev.mysql.com/doc/index-other.html

[5]http://wiki.dbpedia.org/

3, which includes all our improvements, achieves the same number of matchings as defined by the golden standard. The curves of setup 1 and 3 seems similar as they are based on the same principles, except setup 3 is improved in context matching also. The constant number of matchings for setup 2 is a result of the context matching step, which excludes the matchings that do not have the same context values. The setup therefore returns the same number of matchings regarding of the threshold level. In comparison the the setups 2 and 3 we achieve more matchings as the system cannot disambiguate between the correct and incorrect matchings due to lists of possible context words. On the other hand, for higher threshold values they improve the matchings that are lexicographically different but contextually equivalent (e.g. *Staff* and *Person*). The result of setup 1 clearly shows that it is impossible to exclude the identified matchings without taking into account their context as they are lexicographically very similar. Due to the threshold-based results we further show only the results achieved at the threshold level 0.95.

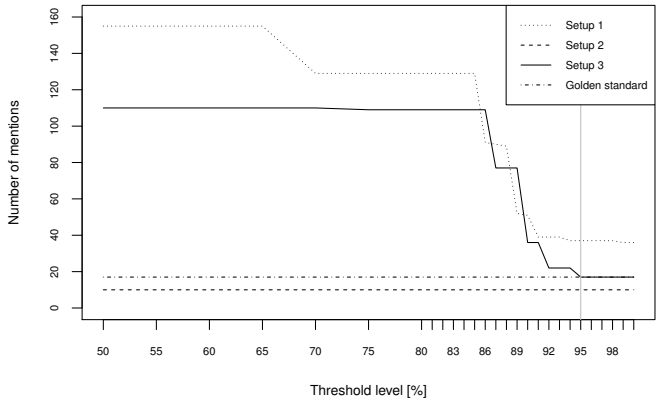

Fig. 9. Number of matchings based on a matching threshold level for each of the system setups.

In Table I we show the matching results for all the three setups at threshold level set to 0.95. We observe that setups 2 and 3 achieve better results than setup 1, which a LogMap tool without improvements. It is important to notice that having a precision of 1.0 means that all the predicted matchings are correct. Further, setup 3 improves the second one by having also higher recall because in finds correct matching also using context. For example, to match the following pairs correctly - *"lavbic-owl:Paris" = "dbpedia:Paris_Hilton"* or *"lavbic-owl:Woodridge" = "dbpedia:Woodridge_Western_Australia"*, only the context can be of help.

TABLE I. SAKILA DATABASE TRANSFORMATION TO RDF AND MATCHING TO DBPEDIA RESULTS.

	Precision	Recall	F-score
Setup 1	0.19	0.54	0.28
Setup 2	**1.0**	0.66	0.80
Setup 3	**1.0**	**1.0**	**1.0**

B. Ontology matching evaluation

Our end-to-end relational database to ontology enrichment tool features improvements in the LogMap tool. To show the real contribution, we evaluate the LogMap+ against two standard tests prepared by OAEI - *Large Biomedical Ontologies FMA-NCI matching problem* and *Instance Matching* problem.

1) Large biomedical ontologies matching problem: focuses on finding matchings between big ontologies such as Foundational Model of Anatomy (FMA) and Nacional cancer institute thesaurus (NCI). These ontologies consist of rich semantic features and contain more ten thousands of instances. The ontologies FMA and NCI have 78989 and 66724 RDF classes. The problem is divided into two parts - the small part, which contains 5% data from FMA and 10% of data from NCI, and the large part, which includes all the data. We show the results in Table II and Table III, respectively.

Some of the proposed approaches are also based on the LogMap tool, while other are specifically build for the challenge. Other approaches employ also different techniques of semantic elevation such as information extraction, similarity detection, rules definition, etc. [21].

TABLE II. LARGE BIOMEDICAL ONTOLOGIES FMA-NCI MATCHING PROBLEM. RESULTS FOR THE SMALL PROBLEM PART OF 5% FMA (3696 CLASSES) AND 10% NCI (6488 CLASSES) MATCHING.

Tool	Time [s]	# of matchings	Precision	Recall	F-score
OMReasoner	36369	1403	**0.96**	0.47	0.63
LogMapLite	44	3467	0.68	0.82	0.74
LogMap-C	289	2124	0.88	0.65	0.75
XMap	144	2571	0.84	0.75	0.79
LogMap-Bio	1226	3412	0.72	**0.87**	0.79
AML	112	2931	0.83	0.86	**0.84**
Average	*142*	*749*	*0.81*	*0.75*	*0.76*
LogMap	235	3146	0.73	0.76	0.74
LogMap+	**323**	**2694**	**0.87**	**0.77**	**0.82**

The results show that LogMap+ achieves similar results as the top performing systems in the field. The Agreement Maker Light (AML) system outperforms our system by a little, but we must take into account that our system does not use any specific knowledge about biomedical domain, while all other systems use specialized techniques or dictionaries. For both problems we improve the performance of the basic LogMap tool, especially on the small part, where LogMap achieves a score below the average.

TABLE III. LARGE BIOMEDICAL ONTOLOGIES FMA-NCI MATCHING PROBLEM. RESULTS FOR THE LARGE PROBLEM PART OF THE WHOLE FMA AND WHOLE NCI MATCHING.

Tool	Time [s]	# of matchings	Precision	Recall	F-score
RSDLWB	2216	728	0.96	0.24	0.38
AOTL	20908	790	0.90	0.24	0.38
OMReasoner	82000	1362	1.0	0.47	0.64
AOT	9341	3696	0.66	0.86	0.75
MaasMatch	1460	2981	0.81	0.84	0.82
LogMap-C	81	2153	0.96	0.72	0.83
XMap	17	2657	0.93	0.85	0.89
LogMapLite	5	2479	**0.97**	0.82	0.89
LogMap-Bio	975	2892	0.91	0.92	0.92
AML	27	269	0.96	**0.90**	**0.93**
Average	*101,992*	*2178*	*0.91*	*0.73*	*0.78*
LogMap	37	2816	0.92	0.87	0.89
LogMap+	**46**	**2716**	**0.96**	**0.86**	**0.91**

2) Instance matching problem: deals with the identification of similarities between different RDF or OWL objects. The OAEI's instance matching problem defines two sets of ontology-based instances to match. The first set consists of 1330 instances within four classes, five data properties and one label property. On the other hand, the second set is constituted of 2649 instances within four classes, four data properties, one object property and one label property.

In Table IV we show results of the instance matching task. LogMap+ achieves poor performance in recall, while best score in precision. Therefore, we can correctly skip poor matchings. The recall is poor because the task was multilingual between Italian and English, so our matching technique could not identify similarities without having a dictionary. Still, we achieve second result overall.

TABLE IV. INSTANCE MATCHING PROBLEM AGAINST OAEI'S 2014 DATASET.

Tool	Precision	Recall	F-score
InsMT	0.001	0.78	0.002
InsMTL	0.001	0.78	0.002
LogMap-C	0.64	0.04	0.08
RiMOM-IM	0.65	0.49	**0.56**
LogMap	0.60	0.05	0.10
LogMap+	**0.85**	**0.06**	**0.11**

V. CONCLUSIONS

In real world scenarios it is hard to improve the quality or automatically enrich the data in relational databases. It has already been shown that semantically rich databases such as ontologies can be used to interconnect schemas. Also, there exist a lot of general (e.g. DBPedia, FreeBase) or domain specific (e.g. ChemBl, GeoNames) public semantic sources within the Linked Open Data cloud that can be of use. As no end-to-end approach for the transformation of relational data to an ontology and matching to the existing ontologies exist, we proposed a new LogMap+ tool.

The LogMap+ is therefore available as a web application that consists of the two parts. In the first part a user selects a relational subschema and defines the properties of the new RDF classes. Then he can set the parameters for further ontology matching, after which the final result is returned as a new enriched semantic source. We used some existing tools to automate the process and proposed an improved approach for a general ontology mapping task. These include lexical data indexing, use of an arbitrary general thesaurus, data cleaning and multi-lingual lemmatization.

A user needs to select relational database entities to map to an ontology and after that, the process can be automated. Still, a user has possibilities to set up the system with additional corpora and guide the system with accepting or rejecting some of the initial matchings.

We evaluated our end-to-end tool against an example Sakila relational database and DBPedia ontology. Further, we separately evaluated the improved LogMap tool against the official Ontology Alignment Evaluation Initiative 2014 challenge datasets. They are based on biomedical domain but our general tool achieves the second score without employing domain specific thesaurus, such as UMLS. As LogMap+ also relies on a context, it achieves far better results when enriching semantically rich ontologies with many supporting classes.

ACKNOWLEDGEMENTS

The work has been supported by the Slovene Research Agency (P2-0359). We would also like to thank to Damir Balija, who implemented the proposed approach during his master thesis [22] preparation.

REFERENCES

[1] V. Bush *et al.*, "As we may think," *The atlantic monthly*, vol. 176, no. 1, pp. 101–108, 1945.

[2] J. Euzenat, P. Shvaiko *et al.*, *Ontology matching: Second edition*. Springer, 2013.

[3] R. Oldakowski, "D2rq platform–treating non-rdf databases as virtual rdf graphs," 2011.

[4] C. Blakeley, "Rdf views of sql data (declarative sql schema to rdf mapping)," *OpenLink Software*, 2007.

[5] J. F. Sequeda, R. Depena, and D. P. Miranker, "Ultrawrap: Using sql views for rdb2rdf," *Proc. of ISWC2009*, 2009.

[6] K. Čerāns and G. Būmans, "Database to ontology mapping patterns in rdb2owl lite," in *International Baltic Conference on Databases and Information Systems*. Springer, 2016, pp. 35–49.

[7] S. Das, S. Sundara, and R. Cyganiak. (2017, Feb.) R2rml: Rdb to rdf mapping language. [Online]. Available: https://www.w3.org/TR/r2rml/

[8] A. Soylu, M. Giese, E. Jimenez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks, "Optiquevqs: towards an ontology-based visual query system for big data," in *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*. ACM, 2013, pp. 119–126.

[9] K. Čerāns and G. Būmans, "Rdb2owl: a rdb-to-rdf/owl mapping specification language," *Databases and Information Systems VI*, pp. 139–152, 2011.

[10] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *the VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[11] E. Jiménez-Ruiz and B. Cuenca Grau, *LogMap: Logic-Based and Scalable Ontology Matching*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 273–288. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25073-6_18

[12] R. Radev, "Representing a relational database as a directed graph and some applications." in *BCI (Local), CEUR Workshop Proceedings*, 2013, pp. 1–8.

[13] B. Villazón-Terrazas and M. Hausenblas. (2017, Jan.) Rdb2rdf implementation report. [Online]. Available: https://www.w3.org/TR/rdb2rdf-implementations/

[14] U.S. National Library of Medicine. (2017, Jan.) Unified medical language system. [Online]. Available: http://www.nlm.nih.gov/research/umls

[15] V. Nebot and R. Berlanga, "Efficient retrieval of ontology fragments using an interval labeling scheme," *Inf. Sci.*, vol. 179, no. 24, pp. 4151–4173, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2009.08.012

[16] M. Cheatham and P. Hitzler, *String Similarity Metrics for Ontology Alignment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 294–309.

[17] W. F. Dowling and J. H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional horn formulae," *The Journal of Logic Programming*, vol. 1, no. 3, pp. 267 – 284, 1984.

[18] D. Lavbič and D. Balija. (2017, Jan.) LodMapFRI - RDB Data Enrichment Tool. [Online]. Available: https://bitbucket.org/dlavbic/lodmapfri/wiki/Home

[19] W. E. Winkler, "String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage." in *Proceedings of the Section on Survey Research*, 1990, pp. 354–359.

[20] J. Euzenat. (2017, Jan.) Ontology alignment evaluation initiative. [Online]. Available: http://oaei.ontologymatching.org/

[21] Z. Dragisic, K. Eckert, J. Euzenat, D. Faria, A. Ferrara, R. Granada, V. Ivanova, E. Jiménez-Ruiz, A. O. Kempf, P. Lambrix *et al.*, "Results of the ontology alignment evaluation initiative 2014," in *Proceedings of the 9th International Conference on Ontology Matching-Volume 1317*. CEUR-WS. org, 2014, pp. 61–104.

[22] D. Balija, "Enrichment of data in relational database with Linked Data resources," Master's thesis, University of Ljubljana, Faculty for computer and information science, Slovenia, 2015.