Silk – A Link Discovery Framework for the Web of Data

Julius Volz
Chemnitz University of Technology
Straße der Nationen 62
D-09107 Chemnitz
volz@hrz.tu-chemnitz.de

Christian Bizer
Freie Universität Berlin
Web-based Systems Group
Garystr. 21
D-14195 Berlin
chris@bizer.de

Martin Gaedke
Chemnitz University of Technology
Straße der Nationen 62
D-09107 Chemnitz
gaedke@cs.tu-chemnitz.de

Georgi Kobilarov
Freie Universität Berlin
Web-based Systems Group
Garystr. 21
D-14195 Berlin
georgi.kobilarov@fu-berlin.de

ABSTRACT

The Web of Data is built upon two simple ideas: Employ the RDF data model to publish structured data on the Web and to set explicit RDF links between entities within different data sources. This paper presents the Silk – Link Discovery Framework, a tool for finding relationships between entities within different data sources. Data publishers can use Silk to set RDF links from their data sources to other data sources on the Web. Silk features a declarative language for specifying which types of RDF links should be discovered between data sources as well as which conditions entities must fulfill in order to be interlinked. Link conditions may be based on various similarity metrics and can take the graph around entities into account, which is addressed using a path-based selector language. Silk accesses data sources over the SPARQL protocol and can thus be used without having to replicate datasets locally.

Categories and Subject Descriptors

H.2.3 [Database Management]: Languages

General Terms

Measurement, Languages

Keywords

Linked data, link discovery, record linkage, similarity, RDF

1. INTRODUCTION

The Web of Data [1] has grown significantly over the last two years and has started to span data sources from a wide range of domains such as geographic information, people, companies, music, life-science data, books, and scientific publications.

While there are more and more tools available for publishing Linked Data on the Web [2], there is still a lack of tools that support data publishers in setting RDF links to other data sources on the Web. The *Silk - Link Discovery Framework* contributes to filling this gap. Using the declarative *Silk - Link Specification Language* (Silk-LSL), data publishers can specify which types of RDF links should be discovered between data sources as well as which conditions data items must fulfill in order to be interlinked. These link conditions can apply different similarity metrics to multiple properties of an entity or related entities which are addressed using a path-based selector language. The resulting similarity scores can be weighted and combined using various similarity aggregation functions. Silk accesses data sources via the SPARQL protocol and can thus be used to discover links between local and remote data sources.

LDOW 2009, April 20, 2009, Madrid, Spain.

The main features of the Silk framework are:

- it supports the generation of owl:sameAs links as well as other types of RDF links.
- it provides a flexible, declarative language for specifying link conditions.
- it can be employed in distributed environments without having to replicate datasets locally.
- it can be used in situations where terms from different vocabularies are mixed and where no consistent RDFS or OWL schemata exist.
- it implements various caching, indexing and entity pre-selection methods to increase performance and reduce network load.

This paper is structured as follows: Section 2 gives an overview of the Silk - Link Specification Language along a concrete usage example. Section 3 reports the results of applying Silk to discover links between several data sources within the LOD data cloud[1]. We describe the implementation of the Silk framework in Section 4 and review related work in Section 5.

2. LINK SPECIFICATION LANGUAGE

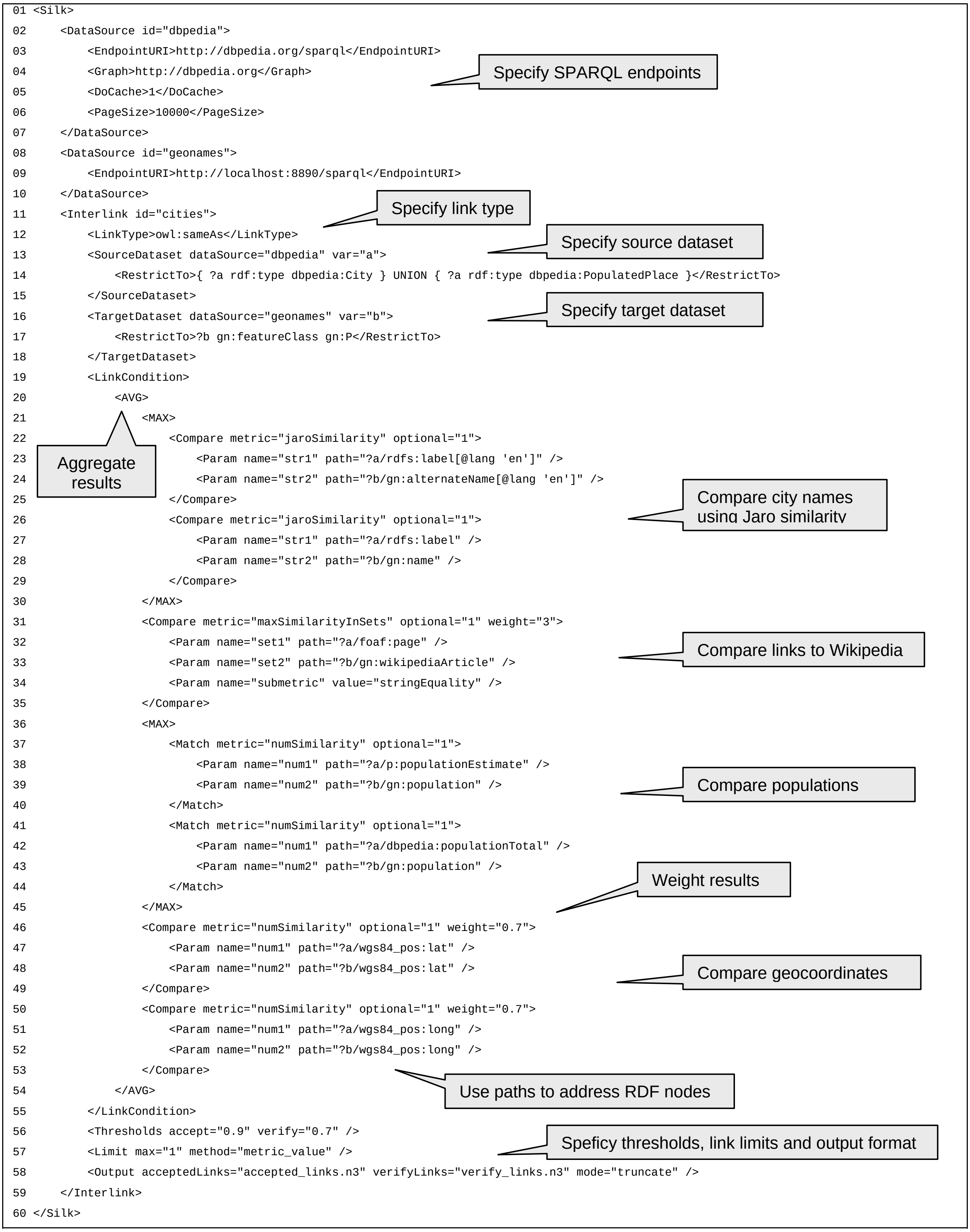
The Silk - Link Specification Language (Silk-LSL) is used to express heuristics for deciding whether a semantic relationship exists between two entities. The language is also used to specify the access parameters for the involved data sources, and to configure the caching, indexing and preselection features of the framework. Link conditions can use different aggregation functions to combine similarity scores. These aggregation functions as well as the implemented similarity metrics and value transformation functions were chosen by abstracting from the link heuristics that were used to establish links between different data sources in the LOD cloud.

Figure 1 contains a complete Silk-LSL example. In this particular use case, we want to discover `owl:SameAs` links between the URIs that are used by DBpedia[2] and by GeoNames[3] to identify cities. In line 12 of the link specification, we thus configure the `<LinkType>` to be `owl:sameAs`.

[1] http://esw.w3.org/topic/SweoIG/TaskForces/ CommunityProjects/ LinkingOpenData

[2] http://dbpedia.org/About

[3] http://www.geonames.org/ontology/



```
<Silk>
    <DataSource id="dbpedia">
        <EndpointURI>http://dbpedia.org/sparql</EndpointURI>
        <Graph>http://dbpedia.org</Graph>
        <DoCache>1</DoCache>
        <PageSize>10000</PageSize>
    </DataSource>
    <DataSource id="geonames">
        <EndpointURI>http://localhost:8890/sparql</EndpointURI>
    </DataSource>
    <Interlink id="cities">
        <LinkType>owl:sameAs</LinkType>
        <SourceDataset dataSource="dbpedia" var="a">
            <RestrictTo>{ ?a rdf:type dbpedia:City } UNION { ?a rdf:type dbpedia:PopulatedPlace }</RestrictTo>
        </SourceDataset>
        <TargetDataset dataSource="geonames" var="b">
            <RestrictTo>?b gn:featureClass gn:P</RestrictTo>
        </TargetDataset>
        <LinkCondition>
            <AVG>
                <MAX>
                    <Compare metric="jaroSimilarity" optional="1">
                        <Param name="str1" path="?a/rdfs:label[@lang 'en']" />
                        <Param name="str2" path="?b/gn:alternateName[@lang 'en']" />
                    </Compare>
                    <Compare metric="jaroSimilarity" optional="1">
                        <Param name="str1" path="?a/rdfs:label" />
                        <Param name="str2" path="?b/gn:name" />
                    </Compare>
                </MAX>
                <Compare metric="maxSimilarityInSets" optional="1" weight="3">
                    <Param name="set1" path="?a/foaf:page" />
                    <Param name="set2" path="?b/gn:wikipediaArticle" />
                    <Param name="submetric" value="stringEquality" />
                </Compare>
                <MAX>
                    <Match metric="numSimilarity" optional="1">
                        <Param name="num1" path="?a/p:populationEstimate" />
                        <Param name="num2" path="?b/gn:population" />
                    </Match>
                    <Match metric="numSimilarity" optional="1">
                        <Param name="num1" path="?a/dbpedia:populationTotal" />
                        <Param name="num2" path="?b/gn:population" />
                    </Match>
                </MAX>
                <Compare metric="numSimilarity" optional="1" weight="0.7">
                    <Param name="num1" path="?a/wgs84_pos:lat" />
                    <Param name="num2" path="?b/wgs84_pos:lat" />
                </Compare>
                <Compare metric="numSimilarity" optional="1" weight="0.7">
                    <Param name="num1" path="?a/wgs84_pos:long" />
                    <Param name="num2" path="?b/wgs84_pos:long" />
                </Compare>
            </AVG>
        </LinkCondition>
        <Thresholds accept="0.9" verify="0.7" />
        <Limit max="1" method="metric_value" />
        <Output acceptedLinks="accepted_links.n3" verifyLinks="verify_links.n3" mode="truncate" />
    </Interlink>
</Silk>
```

Specify SPARQL endpoints
Specify link type
Specify source dataset
Specify target dataset
Aggregate results
Compare city names using Jaro similarity
Compare links to Wikipedia
Compare populations
Weight results
Compare geocoordinates
Use paths to address RDF nodes
Speficy thresholds, link limits and output format

Figure 1. Example: Interlinking cities in DBpedia and GeoNames

2.1 Data Access

For accessing the source and target datasources, we first configure access parameters to the DBpedia and GeoNames SPARQL endpoints using the `<DataSource>` directive. The only mandatory datasource parameter is the endpoint URI. Besides this, it is possible to define other datasource access options, such as the graph name and to enable the caching of SPARQL query results in memory. In order to restrict the query load on remote SPARQL endpoints, it is possible to set a delay in between subsequent queries using the `<Pause>` parameter, specifying the delay time in milliseconds. For working against SPARQL endpoints that restrict result sets to a certain size, Silk uses a paging mechanism. The maximal result size is configured using the `<PageSize>` parameter. The paging mechanism is implemented via SPARQL `LIMIT` and `OFFSET` queries. Lines 2 to 7 within the example show how the access parameters for the DBpedia datasource are set to select only resources from the named graph `http://dbpedia.org`, enable caching and limit the page size to 10,000 results per query.

The configured data sources are later referenced in the `<SourceDataset>` and `<TargetDataset>` clauses of the "cities" link specification. Since we only want to match cities, we restrict the sets of examined resources to instances of the classes `dbpedia:City` and `dbpedia:PopulatedPlace` and the GeoNames feature class `gn:P` by supplying SPARQL conditions within the `<RestrictTo>` directives in lines 14 and 17. These statements may contain any valid SPARQL expressions that would usually be found in the `WHERE` clause of a SPARQL query.

2.2 Link Conditions

The `<LinkCondition>` section is the heart of a Silk link specification and defines how similarity metrics are combined in order to calculate a total similarity value for an entity pair.

For comparing property values or sets of entities, Silk provides a number of builtin similarity metrics. Table 1 gives an overview of these metrics. The implemented metrics include string, numeric, data, URI, and set comparison methods as well as a taxonomic matcher that calculates the semantic distance between two concepts within a concept hierarchy using the distance metric proposed by Zhong et al. in [3]. Each metric in Silk evaluates to a similarity value between 0 or 1, with higher values indicating a greater similarity.

Table 1. Available similarity metrics in Silk

Metric	Description
jaroSimilarity	String similarity based on Jaro distance metric
jaroWinklerSimilarity	String similarity based on Jaro-Winkler metric
qGramSimilarity	String similarity based on q-grams
stringEquality	Returns 1 when strings are equal, 0 otherwise
numSimilarity	Percentual numeric similarity
dateSimilarity	Similarity between two date values
uriEquality	Returns 1 if two URIs are equal, 0 otherwise
taxonomicSimilarity	Metric based on the taxonomic distance of two concepts
maxSimilarityInSet	Returns the highest encountered similarity of comparing a single item to all items in a set
setSimilarity	Similarity between two sets of items

These similarity metrics may be combined using the following aggregation functions:

- AVG – weighted average
- MAX – choose the highest value
- MIN – choose the lowest value
- EUCLID – Euclidian distance metric
- PRODUCT – weighted product

To take into account the varying importance of different properties, the metrics grouped inside the AVG, EUCLID and PRODUCT operators may be weighted individually, with higher-weighted metrics having a greater influence on the aggregated result.

In the `<LinkCondition>` section of the example (lines 19 to 55), we compute similarity values for the the labels, Wikipedia links, population counts and geographic coordinates of cities between datasets and calculate a weighted average of these values. Most metrics are configured to be optional since the presence of the respective RDF property values they refer to is not always guaranteed. In cases where alternating properties refer to an equivalent feature (such as `dbpedia:populationEstimate` and `dbpedia:populationTotal`), we choose to perform comparisons for both properties and select the best evaluation by using the `<MAX>` aggregation operator. Weighting of results is used within the metrics comparing the geographical coordinates (lines 46 and 50), with the longitude and latitude similarity weights lowered to 0.7 each.

After specifying the link condition, we finally specify within the `<Thresholds>` clause that resource pairs with a similarity score above 0.9 are to be interlinked, whereas pairs between 0.7 and 0.9 should be written to a separate output file and be reviewed by an expert. The `<Limit>` clause is used to limit the number of outgoing links from a particular entity within the source data set. If several candidate links exist, only the highest evaluated one is chosen and written to the output files as specified by the `<Output>` directive. In this example, we permit only one outgoing `owl:sameAs` link from each resource.

Discovered links are outputted either as simple RDF triples or in reified form together with their creation date, confidence score and the ID of the employed interlinking heuristic.

2.3 Silk Selector Language

Especially for discovering other semantic relationships than entity equality, a flexible way for selecting sets of resources or literals in the RDF graph around a particular resource is needed. For instance, DBpedia and LinkedMDB both contain movies and directors. For generating links between movies in DBpedia and their directors in LinkedMDB, we might want to navigate to the director of a movie in DBpedia and compare her properties with directors in LinkedMDB. In the case of linking musical artists between DBpedia and MusicBrainz[4], an open music database, we might want to compare properties of the albums of the musicians.

Silk addresses this requirement by using a simple RDF path selector language for providing parameter values to similarity metrics and transformation functions. A Silk selector language path starts with a variable referring to an RDF resource and may then use one of several operators to navigate the graph surrounding this resource. To simply access a particular property of a resource, the forward operator (/) may be used. For example, the path "`?artist/rdfs:label`" would select the set of label values associated with an artist referred to by the `?artist` variable.

Sometimes, however, we need to navigate backwards along a property edge. For example, musical albums in DBpedia contain a `dbpedia:artist` property pointing to the album's creator. However, there exists no explicit reverse property like `dbpedia:albums` for an artist resource. So if a path begins with an artist and we need to select all of her albums, we may use the backward operator (\ ) to navigate property edges in reverse. Since navigating backwards along the property `dbpedia:artist` would select all of the artist's works, this may not only select albums, but also songs and single releases. This is addressed by a filter operator ([ ]), which allows selected resources to be restricted to match a certain predicate. In this example, we could use the RDF path "`?artist\ dbpedia:artist[rdf:type dbpedia:Album]`" to select only albums amongst the works of a musical artist in DBpedia. The filter operator also supports comparisons of numeric types as predicates. For example, to select songs of an artist with a runtime greater than 200 seconds, the path "`?artist\ dbpedia:artist[dbpedia:runtime > 200]`" can be used.

2.4 Pre-Matching

To compare all pairs of entities of a source dataset S and a target dataset T would result in an unsatisfactory runtime complexity of O(|S|·|T|). Even after using SPARQL restrictions to select suitable subsets of each dataset, the required time and network load to perform all pair comparisons might prove to be impractical in many cases. To avoid this problem, we need a way to quickly find a limited set of target entities that are likely to match a given source entity. Silk supports this by allowing rough index prematching.

When using prematching, all target resources are indexed by one or more specified property values (most commonly, their labels) before any detailed comparisons are performed. During the subsequent resource comparison phase, the previously generated index is used to look up potential matches for a given source resource. This lookup uses the BM25[5] weighting scheme for the ranking of search results and additionally supports spelling corrections of individual words of a query. Only a fixed amount of target resources found in this lookup are considered as candidates for a detailed comparison. An example of such a prematching configuration that could be applied to our city linking example is presented in Figure 2:

```
<PreMatchingDefinition
 sourcePath="?a/rdfs:label"
 hitLimit="10">
    <Index targetPath="?b/gn:name" />
    <Index targetPath="?b/gn:alternateName" />
</PreMatchingDefinition>
```

Figure 2. Pre-Matching

This statement instructs Silk to index the cities in the target dataset by both their `gn:name` and `gn:alternateName` property values. When performing comparisons, the `rdfs:label` of a source resource is used as a search term into the generated indexes and only the first ten target hits found in each index are considered as link candidates for detailed comparisons. If we neglect a slight index insertion and search time dependency on the target dataset size, we now achieve a runtime complexity of O(|S| + |T|), making it feasible to interlink even large datasets under practical time constraints. Note however that this prematching may come at the cost of missing some links during discovery, since it is not guaranteed that a prematching lookup will always find all matching target resources.

3. EXPERIMENTS

During the implementation of Silk, we experimented with linking DBpedia to several other public Linked Data sources. Movies in DBpedia were linked both to their movie counterparts and to their directors in LinkedMDB[6]. Between GeoNames and DBpedia, we created links between cities, as shown in Silk-LSL example above. Finally, clinical drugs from DrugBank[7] were linked with their counterparts in DBpedia. The following section gives a short overview over the employed similarity heuristics as well as the amounts of discovered links.

For interlinking movies between DBpedia and LinkedMDB, we used Jaro string similarity to match movie titles and director names, date similarity for comparing release dates and numeric similarity for runtimes. We used the `Thresholds` directive `<Thresholds accept="0.9" verify="0.7" />` to define similarities of 0.9 as acceptable and similarities between 0.7 to 0.9 to be verified by an expert. The number of movies in the datasets and amounts of discovered links are shown in Table 2.

Table 2. Linking movies between DBpedia and LinkedMDB

Number of movies in DBpedia	34,685
Number of movies in LinkedMDB	38,064
Links above accept threshold	26,059
Links above verify threshold	1,858

Interlinking DBpedia movies to their directors in LinkedMDB is an example of creating links other than `owl:sameAs` links, for which we simply used a Jaro string similarity metric to compare a movie's director name to the label of a director in LinkedMDB. Dataset statistics and linking results for this example are given in Table 3.

[4] http://musicbrainz.org

[5] http://xapian.org/docs/bm25.html

[6] http://www.linkedmdb.org/

[7] http://www4.wiwiss.fu-berlin.de/drugbank/

Table 3. Linking DBpedia movies to directors in LinkedMDB

Number of movies in DBpedia	34,685
Number of directors in LinkedMDB	8,367
Links above accept threshold	1,693
Links above verify threshold	374

For linking cities in DBpedia and GeoNames, we used Jaro similarity between city names, URI equality for links to Wikipedia articles as well as numeric similarity for the population counts and geographic coordinates. The results for this use case are shown in Table 4.

Table 4. Linking cities between DBpedia and GeoNames

Number of cities in DBpedia	40,197
Number of populated places in GeoNames	2,410,855
Links above accept threshold	35,031
Links above verify threshold	9,147

Finally, for generating links between clinical drugs in DrugBank and DBpedia, we compared drug labels via the JaroWinkler similarity, PubChem[8] identifiers via string equality and used numeric similarity for comparing the drugs' molecular weights. Table 5 shows the results for this case.

Table 5. Linking drugs between DBpedia and DrugBank

Number of drugs in DBpedia	3,134
Number of drugs in DrugBank	4,772
Links above accept threshold	1,202
Links above verify threshold	245

The metric compositions, weightings and thresholds in these examples were chosen based on what seemed to produce reasonably valid results in our tests. However, a detailed analysis of the quality of the generated links has not yet been performed. When using Silk in a practical scenario, it is advisable to evaluate the accuracy and completeness of generated links more closely while adjusting the linking specification accordingly.

4. SILK IMPLEMENTATION

Silk is written in Python and is run as a batch process on the command line. The framework may be downloaded from Google Code[9] under the terms of the BSD license. For calculating string similarities, a library from Febrl[10], the Freely Extensible Biomedical Record Linkage toolkit, is used, while Silk's prematching features are achieved with the search engine library Xapian[11]. The Silk system architecture is illustrated in Figure 3:

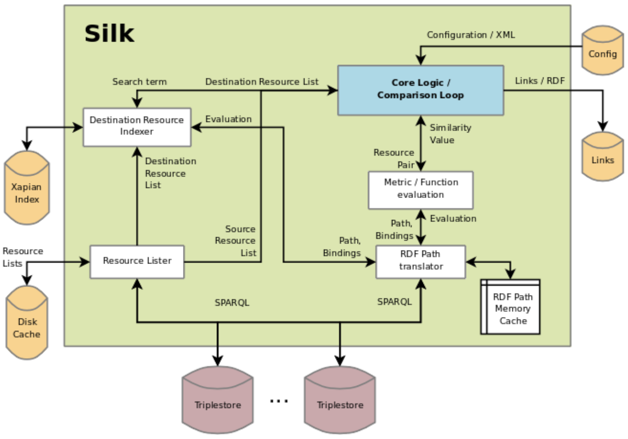


Figure 3. Silk System Architecture

Before executing any comparisons, Silk retrieves the source and target resource lists. The list of source resources is retrieved directly through a resource lister which queries the respective SPARQL endpoint and caches the list on disk for reuse in a later run of Silk. Target resources are first indexed by means of a resource indexer, making them searchable by specific properties or RDF Path evaluations. During comparison processing, a list of target resource candidates for each source resource is looked up in this index, limiting detailed comparisons to index search hits. This prematching of resources is optional, but recommended as it drastically reduces run time and network load.

During each detailed resource pair comparison, the user-specificed metric aggregation tree is evaluated. Function or metric parameters passed as RDF Path values are transformed to SPARQL queries by an RDF Path translator and sent to the respective SPARQL endpoint for evaluation. Query results are cached in memory during Silk runtime.

If a metric aggregation for a pair of resources results in a value above the specified linking thresholds, a candidate link is saved in memory. After completing all comparisons for a link specification, a link limit may be applied to limit the maximum number of outgoing links from a single resource. Only a specified count of highest-rated links are kept, lower-valued links are discarded. The remaining links are written to the output file in the format specified by the user (Turtle, CSV, reified format together with meta-information such as confidence score and creation date).

5. RELATED WORK

There is a large body of related work on record linkage [5] and duplicate detection [4] within the database community as well as on ontology matching [6] in the knowledge representation community. Silk builds on this work by implementing similarity metrics and aggregation functions that proved successful within other scenarios. What distinguishes Silk from this work is its focus on the Linked Data scenario where different types of

[8] http://pubchem.ncbi.nlm.nih.gov

[9] http://silk.googlecode.com

[10] http://sourceforge.net/projects/febrl

[11] http://xapian.org

semantic links should be discovered between Web data sources that often mix terms from different vocabularies and where no consistent RDFS or OWL schemata spanning the data sources exist.

Related work that also focuses on Linked Data includes Raimond et al. [7] who propose a link discovery algorithm that takes into account both the similarities of web resources and of their neighbors. The algorithm is implemented within the GNAT tool and has been evaluated for interlinking music-related data sets. In [8], Hassanzadeh et al. describe a framework for the discovery of semantic links over relational data which also introduces a declarative language for specifying link conditions. A main difference between LinQL and Silk-LSL is the underlying data model and Silk's ability to more flexibly combine metrics through aggregation functions. A framework that deals with instance coreferencing as part of the larger process of fusing Web data is the KnoFuss Architecture proposed in [9]. In contrast to Silk, KnoFuss assumes that instance data is represented according to consistent OWL ontologies.

6. CONCLUSIONS

We presented the Silk framework, a flexible tool for discovering links between entities within different Web data sources. We introduced the Silk-LSL link specification language and demonstrated its applicability within different link discovery scenarios.

The value of the Web of Data rises and falls with the amount and the quality of links between data sources. We hope that Silk and other similar tools will help to strengthen the linkage between data sources and therefore contribute to the overall utility of the network.

The complete Silk- LSL language specification and further Silk usage examples are found on the Silk project website at http://www4.wiwiss.fu-berlin.de/bizer/silk/.

7. REFERENCES

[1] Berners-Lee, T.: Linked Data - Design Issues. http://www.w3.org/DesignIssues/LinkedData.html

[2] Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web. http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/

[3] Zhong, J., et al.: Conceptual Graph Matching for Semantic Search. The 2002 International Conference on Computational Science (ICCS2002), Amsterdam, April 2002.

[4] Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering 19(1), 1–16 (2007).

[5] Winkler, W.: Overview of Record Linkage and Current Research Directions. Bureau of the Census, Technical Report, 2006.

[6] Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg, 2007.

[7] Raimond, Y., Sutton, C., Sandler, M.: Automatic Interlinking of Music Datasets on the Semantic Web. In: Linked Data on the Web Workshop (LDOW2008), 2008.

[8] Hassanzadeh, O., et al.: A Declarative Framework for Semantic Link Discovery over Relational Data. Poster at 18th World Wide Web Conference (WWW2009), 2009.

[9] Nikolov, A., et al.: Integration of Semantically Annotated Data by the KnoFuss Architecture. In: 16th International Conference on Knowledge Engineering and Knowledge Management, 265-274, 2008.