Integrating Linked Data and Services with LIDS

Sebastian Speiser and Andreas Harth

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany
`lastname@kit.edu`

Abstract. We propose Linked Data Services (LIDS), a general, formalised approach for integrating data-providing services with Linked Data. We present conventions for service access interfaces that conform to Linked Data principles, and an abstract light-weight service description formalism with a mapping to RDF and SPARQL. We develop algorithms that use LIDS descriptions to automatically create links between services and existing data sets, either off-line or during query execution. To evaluate our approach, we realise LIDS wrappers and LIDS descriptions for existing services, and measure performance and effectiveness of the automatic interlinkage algorithm over multiple billions of triples.

1 Introduction

The trend towards publishing data on the Web is gaining momentum, particularly spurred by the Linking Open Data (LOD) project[1] and several government initiatives aimed at publishing public sector data. Data publishers often use Linked Data principles [1]. which leverage established Web standards such as Uniform Resource Identifiers (URIs), the Hypertext Transfer Protocol (HTTP) and the Resource Description Framework (RDF) [2]. Data providers can easily link their data to data from third parties via reuse of URIs. The LOD project proves that the Linked Data approach is, in principle, capable of integrating data from a large number of sources. However, there is still a lot of data residing in silos that could be beneficially linked with other data, but will not be published as a fully materialised knowledge base. Reasons include:

– data is constantly changing, e.g. stock quotes or weather data can have update intervals below one second;
– data is generated depending on possibly infinite different input data, e.g. distance between two geographical points, which can be specified with arbitrary precision;
– the data provider does not want arbitrary access to the data, e.g. prices of flight tickets may be only available for specific requests in order to maintain the possibility for price differentiation.

Such data is commonly provided via Web services, in the following also called data or information services, as they provide a restricted view on a possibly

[1] `http://linkeddata.org/`

implicit data set but are generally stateless and free of side effects, i.e. do not change the world.

Linked Data interfaces for services have been created, e.g., in form of the book mashup [3] which provides RDF about books based on Amazon's API, or twitter2foaf[2], which encodes a Twitter follower network of a given user based on Twitter's API. These are useful examples for the integration of information services and Linked Data. However, the interfaces are not formally described and thus the link between services and data has to be established manually or by service-specific algorithms. For example, to establish a link between a person instance (e.g., described using the FOAF vocabulary[3]) and her Twitter account, one has to hard-code which property relates people to their Twitter username and the fact that the URI of the person's Twitter representation is created by appending the username to `http://twitter2foaf.appspot.com/id/`.

Vast amounts of idle data can be brought to the Semantic Web via a standardised method for creating Linked Data interfaces to services. The method should incorporate formal service descriptions that enable (semi-)automatic service discovery and integration. We present such an approach for what we call LInked Data Services (LIDS). Specifically, we present the following contributions:

– an access mechanism for LIDS interfaces based on generic Web architecture principles (URIs and HTTP) (Section 3);
– a generic lightweight data service description formalism, instantiated for RDF and SPARQL (Section 4);
– algorithms for linking existing data sets using LIDS and algorithms for evaluating queries exploiting LIDS descriptions (Section 5)

In Section 6 we describe the creation of LIDS for existing services, and present the results of two experiments measuring performance and effectiveness of the approach: (i) interlinking the Billion Triple Challenge data set with a geographic LIDS, and (ii) evaluating SPARQL queries with automatic LIDS invocation We relate our approach to existing work in Section 7 and conclude with Section 8.

2 Use Case Scenario

Our use case is a scenario involving the analysis of technology companies. Consider an investor who wants to assess the outlook of a potential investment target. The investor could vet the company by navigating an integrated data set containing basic company data, key personnel, competitors, job openings, IP portfolio and previous VC investments in the company. In addition, the data set could contain Social Media from Twitter and blogs that allows the investor to gauge the media interest in the company.

All required information is available on the Web, but with three major drawbacks:

[2] `http://twitter2foaf.appspot.com/`
[3] `http://xmlns.com/foaf/0.1/`

– The data is accessible via several protocols, e.g., some data is directly accessible via HTTP GET lookups while other data is hidden behind forms requiring HTTP POST and possibly HTTP cookies.
– The data is encoded in heterogeneous data formats, e.g., trademark data from the United States Patent and Trademark Office is available in HTML; patent data from the European Patent Office in Comma Separated Value files; CrunchBase company descriptions, Indeed.com job offers and Twitter messages in JSON; and data from GeoNames services in XML.
– The data is sparsely interlinked, e.g., there exists no link between a company's office location and its GeoNames location, and no link between a company and its trademarks.

Consider data about company offices, which contains latitude and longitude attributes:

```
#usa-palo-alto-hq geo:lat "37.416" .
#usa-palo-alto-hq geo:long "-122.152" .
```

A GeoNames service call to find nearby places described by Wikipedia[4] returns:

```
<geonames>
  <entry>
    <lang>en</lang>
    <title>Stanford, California</title>
    <summary>
       Stanford is a census-designated place (CDP) in Santa Clara County...
    </summary>
    <feature>city</feature>
    <lat>37.4225</lat>
    <lng>-122.1653</lng>
    <wikipediaUrl>
       http://en.wikipedia.org/wiki/Stanford%2C_California
    </wikipediaUrl>
    <distance>1.3774</distance>
  </entry>
  <entry>
    <lang>en</lang>
    <title>Stanford University</title>
    ...
</geonames>
```

Based on the available data one could establish a `foaf:based_near` connection between `#usa-palo-alto-hq` and `http://dbpedia.org/resource/Stanford%2C_California`, however, that step would require specialised code.

Unlocking the data for automated integration and processing requires

– Linked Data interfaces to all services and data sources, so that data can be easily accessed and integrated and

[4] http://ws.geonames.org/findNearbyWikipedia?lat=37.416&lng=-122.152

– formal service descriptions, so that links between data from different sources can be created automatically.

3 Linked Data Services (LIDS) Overview

Data services return data depending on supplied input parameters. For example, the GeoNames `findNearbyWikipedia` service relates given latitude/longitude parameters to DBpedia entities describing things that are nearby. Data services do not alter the state of some entity, nor modify data. They can be seen as data sources providing information about some entity (i.e. the output data), when given some initial description of the entity (i.e. the input data).

Linked Data principles state that a lookup on a URI returns more information related to that URI. Services, however, may return multiple results. The `findNearbyWikipedia` service, for example, returns multiple places that are nearby a given point. Hence, we require that there exists a URI which represents one entity that can be related to multiple outputs. We illustrate the principle using the openlids.org wrapper for GeoNames[5]. The service is based on RDF and Linked Data principles and handles input entities, for example in the form of `gv:findNearbyWikipedia?lat=...&lng=...#point`. Resolving the URI of `gw:findNearbyWikipedia?lat=37.416&lng=-122.152#point` returns the following description:

```
@prefix dbp: <http://dbpedia.org/resource/> .
#point foaf:based_near dbp:Stanford%2C_California ;
       foaf:based_near dbp:Stanford_University ;
       foaf:based_near dbp:Stanford_Stadium ;
       foaf:based_near dbp:Palo_Alto%2C_California ;
       foaf:based_near dbp:Packard%27s_garage .
```

In our model, service results can be directly linked in description formalisms based on URIs, as all needed parameters are encoded in the service URI. For example, we can reuse and link the URI `gw:findNearbyWikipedia?lat=37.416&lng=-122.152#point` from within other Linked Data files. In contrast, if the service would just return the place descriptions of nearby things, there is no way to connect the input data to results from the service.

Informally, a Linked Data Service (LIDS) provides URIs for service inputs that encode parameters as key-value pairs in the query string of a URI. Dereferencing an input URI via an HTTP GET request returns a description of the input that comprises its relation to the service output and the output data itself. A Linked Data Service also requires a machine-readable description to generate service invocations from within software programs.

4 LIDS Descriptions

In this section we define the abstract model of LIDS descriptions, and provide a mapping to RDF and SPARQL.

[5] `http://geowrap.openlids.org/`, abbreviated as `gw`.

4.1 Preliminaries

We assume a semi-structured data model that is based on n-ary predicates identified by IRIs (Internationalized Resource Identifiers) with constant arguments, i.e. either IRIs, literals, or blank nodes. Let I, B, L be disjoint infinite sets of IRIs, blank nodes and literals.

Definition 1 (Data Set). *A data set D is a finite set of tuples, where each tuple $d = (p, d_1, \dots, d_n) \in I \times (I \cup B \cup L)^*$ is written as $p(d_1, \dots, d_n)$.*

The inputs and outputs of a LIDS can vary for each service invocation and are thus modelled as variables. We define V as an infinite set of variables, disjoint with I, B, and L. Relations between variables and constants can be expressed using tuple patterns.

Definition 2 (Tuple Pattern). *A tuple pattern $t \in I \times (I \cup L \cup V)^*$ abstracts from single tuples by allowing variables as predicate arguments.*

The relation between the input variables that is required for a successful service invocation is modelled as a conjunctive query (CQ) as defined in the following.

Definition 3 (Conjunctive Query). *A conjunctive query $CQ = \{X, T\}$ consists of a head, i.e. a set of variables $X \subset V$, and a body, i.e. a finite set of tuple patterns T.*

We are now able to define LIDS descriptions.

Definition 4 (LIDS Description). *A LIDS description consists of a tuple (ep, CQ_i, T_o) where ep denotes the LIDS endpoint, $CQ_i = (X_i, T_i)$ a conjunctive query to specify the input to the service, and T_o a set of tuple patterns describing the output data of the service.*

In Section 4.2 we describe in more detail the meaning of CQ_i and how it is used to construct service IRIs. The patterns T_o are used in Section 4.3 to define the output data returned by resolving a service IRI.

Example 1. We describe the `findNearbyWikipedia` openlids.org wrapper service as (ep, CQ_i, T_o) with:
$ep =$ `gw:findNearbyWikipedia`
$CQ_i = (\{$ `?lat,?lng` $\}, \{$ `geo:lat(?point,?lat)`,`geo:long(?point,?lng)` $\})$
$T_o = \{$`foaf:based_near(?point,?feature)`$\}$

4.2 Generating Service IRIs from LIDS

Informally, the service IRI I_s is constructed from the service endpoint IRI ep in combination with the variable bindings from CQ_i encoded in the query part of the service IRI. A variable binding relates the head variables of a conjunctive query to constants from the data set by matching the tuple patterns of the body of a conjunctive query.

Let M be the set of all function $\mu : I \cup L \cup V \to I \cup L$, s.t. μ is the identity for constants, i.e. $\forall a : (a \in I \cup L \to \mu(a) = a)$. As an abbreviation we also apply a function $\mu \in M$ to a tuple pattern $t = p(t_1, \ldots, t_n)$ $(\mu(t) = p(\mu(t_1), \ldots, \mu(t_n)))$, and to sets of tuple patterns T $(\mu(T) = \{\mu(t) \mid t \in T\})$.

Definition 5 (Variable Binding). *A function $\mu \in M$ is a variable binding for a conjunctive query $CQ = (X, T)$ and a data set D, if $\mu(T) \subseteq D$. We denote the set of all mappings for a CQ and a data set as $\mathcal{M}_{CQ}(D) = \{\mu \in M \mid \mu(T) \subseteq D\}$.*

Given a variable binding μ, we construct I_s in the following way (addition is understood as string concatenation):

$$uri(ep, X_i, \mu) = ep + \text{"?"} + \sum_{x \in X_i} (x + \text{"="} + \mu(x) + \text{"\&"}) \tag{1}$$

Additionally we introduce an abbreviated IRI schema that can be used if there is only one required parameter (i.e. $|X_i| = 1, X_i = \{x\}$):

$$uri_s(ep, X_i, \mu) = ep + \text{"/"} + \mu(x) \tag{2}$$

We assume that both variable names and bindings are URI-encoded.

Example 2. For a given parameter binding $\mu(lat) = 37.416, \mu(lng) = -122.152$, we can construct the following service IRI:
`<http://geowrap.openlids.org/findNearbyWikipedia?lat=37.416&lng=-122.152>`

4.3 Service Entity IRIs and Service Output

For LIDS we require that one of the variables in the input CQ describes the service entity as a whole.

If a service has several inputs, the inputs define a single object that will then be related to the output(s). If no such natural connecting object exists, the service is probably combining several functionalities in one operation and should be split into multiple parts for each function. In any case, it is possible to define an artificial input concept for the service.

For a complete LIDS description, we are still missing the input variable i, which can be retrieved as the variable appearing both in the input and output description.

Definition 6 (Service Entity). *The service entity of a LIDS is referred to by the variable i that appears both in CQ_i and T_o, i.e. $\{i\} = Var(T_o) \cap Var(T_i)$, where Var is a function mapping a set of tuple patterns to the set of all variables appearing in the patterns.*

The IRI of the service entity E_s is constructed by adding i as fragment identifier to the service IRI I_s.

The service entity IRI E_s consists of a base IRI (ep), input parameters (derived from CQ_i), and the input entity i. The IRI denoting the service entity E_s without the fragment identifier coincides with the IRI for the service call, following Linked Data principles.

The meaning of T_o describing the output is given in the following. The return value of a LIDS for a given service entity E_s is a data set $D_o \supseteq \{T' \subseteq D_{impl} \mid \exists \mu \in M : \mu(i) = E_s \wedge \mu(T_o) = T'\}$, where D_{impl} is the implicit, potentially infinite data set representing the information provided by the LIDS.

4.4 Describing LIDS using SPARQL

In the following we present how LIDS descriptions can be modelled using SPARQL. Please note that other instantiations of the abstract LIDS description model are possible, for example using RIF [5].

We propose to use CONSTRUCT queries with unsafe variables. The use of unsafe variables and the meaning of the endpoint are not completely adhering to the SPARQL standard, but have their intuitive meaning.

A service description is given in the following way:

```
CONSTRUCT { [output] } FROM [endpoint] WHERE { [input] }
```

We restrict `[output]` and `[input]` to basic graph patterns, i.e. conjunctions of triple patterns. We relate a SPARQL query to our abstract description formalism in the following way:

- `[input]`: The condition on input parameters which can be mapped to a pattern set T_i, the body of CQ_i.
- `[endpoint]`: The endpoint uri *ep*.
- `[output]`: The output description, which can be mapped to the output pattern set T_o.

To determine which variables of T_i should be the head variables X_i, and thus are required for a service invocation, we make the following assumption: typically services are not per se interested in individuals or their URIs but in the values of datatype relations, i.e. literals. For example, the `findNearbyWikipedia` service is based on the latitudes and longitudes of the points and not on their IRIs. In such cases, the required literal values are uniquely identifying the input individual from the perspective of the service.

Formally we define the required variables as those that have no outgoing property patterns (except for type declarations):

$$\begin{aligned} X_i = \{o \in V \mid & \exists p \in I, s \in I \cup V \cup B : p(s,o) \in T_i \wedge \\ & \nexists p \in I \setminus \{\texttt{rdf:type}\}, \nexists o' \in I \cup V \cup B \cup L : p(o,o') \in T_i\}. \end{aligned}$$

Example 3. In the following we show the SPARQL representation of the formal LIDS description from Example 1:

```
CONSTRUCT { ?point foaf:based_near ?feature. }
   FROM   <http:/geowrap.openlids.org/findNearbyWikipedia>
   WHERE  { ?point a Point . ?point geo:lat  ?lat .
                             ?point geo:long ?lng }
```

Note that `point` is the input variable i as it appears both in `[input]` and `[output]`. The required variables are `lat` and `lng`, but not `point` as it appears in the subject of two patterns. The formal LIDS description of this service corresponds to the one given in Example 1.

If the actual URI of an individual is needed that is represented by a non-required variable, this can be expressed by using the `log:uri` property, as proposed by Berners-Lee et al. [6]. The property relates an individual to the string representation of its URI, which then is a literal and recognised as a required variable by our formula.

5 Algorithms

In this section we show how LIDS descriptions can be exploited for two problems: (i) automatic interlinking of data sets with LIDS, and (ii) query evaluation with automatic LIDS invocation.

5.1 Interlinking Data with LIDS

In the following, we describe how existing data sets can be automatically enriched with links to LIDS. This can happen in different settings, consider e.g.:

- Processing of a static data set, inserting links to LIDS and storing the new data.
- An endpoint that serves data (e.g. a Linked Data server), and dynamically adds links to LIDS.
- A data browser that locally augments retrieved data with data retrieved from LIDS.

Given a data set D and a LIDS description $l = (ep, CQ_i, T_o)$ the following formula defines a set of entities in D and equivalent entities that are inputs for the LIDS (i is determined from T_i and T_o and $+$ is again string concatenation):

$$equivs_{D,l} = \Big\{ \big(\mu(i), uri(ep, X_i, \mu) + \text{"\#"} + i\big) \mid \mu \in \mathcal{M}_{CQ_i}(D) \} \Big\}.$$

The obtained equivalences can be either used to immediately resolve the LIDS URIs and add the data to D, or to make the equivalences explicit in D given a data model with support for equivalences. For example, for a Linked Data set D, the following information could be added to D:

$$\big\{ \texttt{owl:sameAs}(x_1, x_2) \mid (x_1, x_2) \in equivs_{D,l} \big\}.$$

For example, consider the following dataset:

```
#usa-palo-alto-headquarters;
  cb:description "Headquarters";
  geo:lat "37.416";
  geo:long "-122.152";
```

The data can be directly and automatically interlinked with other LIDS, such as the the `findNearbyWikipedia` service. A service call to `gw:findNearbyWikipedia?lat=37.416&lng=-122.152#point` returns triples associating the URI to points which are `foaf:based_near`. The interlinking algorithm adds the following description:

```
#usa-palo-alto-headquarters owl:sameAs
      gw:findNearbyWikipedia?lat=37.416&lng=-122.152#point .
```

Please observe that by equating the URI from the input data with the LIDS entity IRI, we essentially add the returned `foaf:based_near` statements to `#usa-palo-alto-headquarters`. Should the database underlying the service change, a lookup on the LIDS entity IRI returns the updated data which can be then integrated.

5.2 Query Answering using LIDS

LIDS and their descriptions can be employed for query answering in two ways: goal-driven or data-driven.

Goal-driven means that partial conditions in a query are matched against the output description of a LIDS. If there is a match the partial condition is replaced by a union of itself and the retrieval of a LIDS call. In order to determine the input URI of the LIDS, it must be ensured that the required parameters can be bound. This problem is strongly related to answering queries using views with limited access patterns (cf. the survey by Halevy [7]).

The other approach is data-driven, which means that a query is evaluated as usual, with the possibility to call a LIDS based on the incoming query results, whenever the required input data is available. This method leaves room for variation of the following dimensions:

- Evaluation order of the query parts. Depending on the order in which variables are bound, the results of a LIDS can be obtained at different stages of the query evaluation. This influences the results of subsequent query parts, and can also require the repeated evaluation of a previous query part, which then can produce additional results.
- LIDS application strategy: a greedy strategy invokes LIDS calls as soon as the required parameters are available. On the other hand, a lazy strategy only tries to call a LIDS if the original query does not yield any results. These strategies will influence evaluation speed and completeness of results.

Work on production rules (e.g. [8]) provides a rich body of knowledge about different semantics and evaluation strategies for applying data producing rules.

Algorithm 1 `determineRelevantLIDS`

```
Require: Conjunctive query: $cq = (X, T)$
Require: Set of LIDS descriptions: $LIDS$
 1: $availLIDS = LIDS$
 2: $reqLIDS = \emptyset$
 3: $reqPatterns = T$
 4: $checkedPatterns = \emptyset$
 5: while $reqPatterns \neq \emptyset \wedge availLIDS \neq \emptyset$ do
 6:    $curPatterns = reqPatterns$
 7:    $checkedPatterns = checkedPatterns \cup reqPatterns$
 8:    $reqPatterns = \emptyset$
 9:    for all $t \in curPatterns$ do
10:       for all $l = (ep, CQ_i = (X_i, T_i), T_o, i) \in availLIDS$ do
11:          if $\exists t_o \in T_o\ \exists \mu \in M : t = \mu(t_o)$ then
12:             $availLIDS = availLIDS \setminus \{l\}$
13:             $reqLIDS = reqLIDS \cup \{l\}$
14:             $reqPatterns = reqPatterns \cup$
15:                          $\{t_i \in T_i \mid \nexists \mu \in M : \mu(t_i) \in checkedPatterns\}$
16:          end if
17:       end for
18:    end for
19: end while
20: return $reqLIDS$
```

In the following we illustrate an ad-hoc algorithm for evaluating conjunctive queries and integrating LIDS using a combination of both query-driven and data-driven approaches. We do not make assumptions on the concrete data model in use, but adopt the Linked Data principles, especially: (i) things are referred to by URIs, and (ii) dereferencing a URI provides information about the described thing.

The algorithm performs two phases: (i) determination of relevant LIDS using backward chaining, and (ii) query evaluation using forward chaining. The first phase is performed by Algorithm 1, which determines relevant LIDS by searching for LIDS with a tuple pattern in the output description that can be mapped to one of the required tuple patterns (initially the patterns of the query). For each relevant LIDS the patterns in the input description are added to the required patterns and the procedure is repeated until either all LIDS are marked as relevant or all required patterns have been searched.

The query evaluation algorithm depends on a Linked Data query engine such as for example [9] or [10]. Such a query engine dereferences URIs appearing in the query and in partial result bindings, to add further data sets to the knowledge base on which the query is evaluated. The result is thus not only a set of tuples but also a new data set. We denote a call to the query engine as $RES, OUTD = queryengine(cq, D)$, where RES are the resulting tuples, $OUTD$ the new data set, cq the query, and D the input data set. Query evaluation with LIDS integration is shown in Algorithm 2. The algorithm lets the

Algorithm 2 `LIDSqueryEvaluation`

```
Require: Conjunctive query: $cq = (X, T)$
Require: Set of relevant LIDS descriptions: $LIDS$
  $ALLRES = \emptyset$
  $D = \emptyset$
  $RES, OUTD = queryengine(cq, D)$
  while $RES \not\subseteq ALLRES$ do
    $ALLRES = ALLRES \cup RES$
    $D = D \cup OUTD$
    for all $l \in LIDS$ do
      $D = D \cup \{x_1 = x_2 \mid (x_1, x_2) \in equivs_{D,l}\}$
    end for
    $RES, OUTD = queryengine(cq, D)$
  end while
  return $ALLRES$
```

Linked Data query engine execute the original query and then interlinks the resulting data set with the relevant LIDS. On the resulting data set the query is again executed, which automatically increases the size of the data set, as the Linked Data engine dereferences the LIDS input links. The process is repeated until no new query results are obtained.

6 Evaluation of Performance and Effectiveness

In this section we first present several LIDS services which we have made available, and evaluate the performance and effectiveness of the presented algorithms. Source code and test data for the implementation of the interlinking algorithm, as well as other general code for handling LIDS and their descriptions can be found online[6]. All experiments were conducted on a 2.4 GHz Intel Core2Duo laptop with 4 GB of main memory.

6.1 Implemented LIDS Services

In this section we show how we applied the LIDS approach to construct publicly available Linked Data interfaces for selected services to realise our use case scenario from Section 2. The following services are hosted on Google's App Engine cloud environment. The services are also linked on `http://openlids.org/` together with their formal LIDS descriptions and further information, such as URIs of example entities.

– CrunchBase Wrapper[7] provides information about tech companies, their funding, founders, top employees, products, and competitors.

[6] `http://code.google.com/p/openlids/`

[7] `http://cbasewrap.ontologycentral.com/`

- GeoNames Wrapper[8] provides three functions:
  - finding the nearest GeoNames feature to a given point,
  - finding the nearest GeoNames populated place to a given point,
  - linking a geographic point to resources from DBpedia that are nearby.
- Twitter Wrapper[9] links Twitter account holders to the messages they post.
- Feedwrapper[10] provides SIOC data about RSS and Atom feeds.

The effort to produce a LIDS wrapper is typically low. The interface code that handles the service URIs and extracts parameters can be realised by standardised code or even generated automatically from a LIDS description. The main effort lies in accessing the service and generating a mapping from the service's native output to a Linked Data representation. While for some services it is sufficient to write a simple JavaScript wrapper that transforms JSON data into RDF/N3, other services require simple Java procedures or XSLTs transforming output XML data to RDF/XML. Effort is higher for services that map Web page sources, as this often requires session and cookie handling and parsing of faulty HTML code. However, the underlying data conversion has to be carried out whether or not LIDS are used. Following the LIDS principles is only a minor overhead in implementation; adding a LIDS descriptions requires a SPARQL query to describe the service.

6.2 Interlinking Existing Data Sets with LIDS

We implemented a streaming version of the interlinking algorithm shown in Section 5.1 based on NxParser[11]. For evaluation of the algorithm's performance and effectiveness we interlinked the Billion Triple Challenge (BTC) 2010 data set[12] with the `findNearby` geowrapper. In total the data set consisted of 3,162,149,151 triples and was annotated in 40,746 seconds (< 12 hours) plus about 12 hours for uncompressing the data set, result cleaning, and statistics gathering. In the cleaning phase we filtered out links to the geowrapper that were redundant, i.e. entities that were already linked to GeoNames (including the GeoNames data set itself). The original BTC data contained 74 different domains that referenced GeoNames URIs. Our interlinking process added 891 new domains that are now linked to GeoNames via the geowrap service. In total 2,448,160 new links were added[13]. Many links referred to the same locations, all in all there were links to ca. 160,000 different geowrap service calls. These results show that even with a very large data set, interlinking based on LIDS descriptions is feasible on commodity hardware. Furthermore the experiment showed that there is much idle potential for links between data sets, which can be uncovered with our approach.

[8] `http://geowrap.openlids.org/`
[9] `http://twitterwrap.ontologycentral.com/`
[10] `http://feedwrap.openlids.org/`
[11] `http://sw.deri.org/2006/08/nxparser/`
[12] `http://km.aifb.kit.edu/projects/btc-2010/`
[13] Linking data is available online: `http://people.aifb.kit.edu/ssp/geolink.tgz`

6.3 Answering of SPARQL Queries

We implemented a prototype of the query answering algorithm, from Section 5.2 which uses the Semantic Web Client Library[14]. We evaluated the performance and effectiveness of answering three representative queries, searching for agents known by a person and the names of the places which are based near the agents. For answering the queries the openlids.org GeoNames `findNearby` service was relevant. Figure 1 lists the queries used.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geonames: <http://www.geonames.org/ontology#>

(q1) SELECT DISTINCT ?p ?loc WHERE {
          <http://speiserweb.de/sebastian/foaf.rdf#me> foaf:knows ?p .
                    ?p foaf:based_near ?point .  ?p geonames:name ?loc }
(q2) SELECT DISTINCT ?p ?loc WHERE {
          <http://harth.org/andreas/foaf#ah> foaf:knows ?p .
                    ?p foaf:based_near ?point .  ?p geonames:name ?loc }
(q3) SELECT DISTINCT ?p ?loc WHERE {
          <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?p .
                    ?p foaf:based_near ?point .  ?p geonames:name ?loc }
```

Fig. 1. Evaluation queries

Figure 2 shows the results for evaluating the queries. The time overhead for significantly increasing the number of results is relatively small. Please note that the results should be understood as a proof-of-concept demonstrating that LIDS descriptions can be fruitfully integrated in query answering over Linked Data with reasonable time overhead.

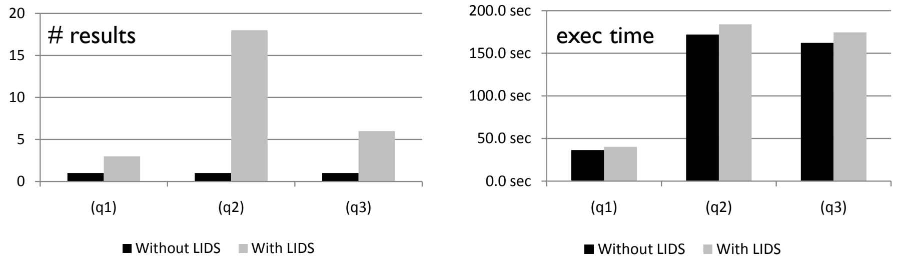

Fig. 2. Results of Answering Queries (q1), (q2), (q3)

[14] `http://www4.wiwiss.fu-berlin.de/bizer/ng4j/semwebclient/`

7 Related Work

Our work provides an approach to open up data silos for the Web of Data. Previous efforts in this direction are confined to specialised wrappers, for example the book mashup [3]. In contrast to these ad-hoc interfaces, we provide a uniform way how such interfaces can be constructed, and thus our work is applicable not only to specific examples but generally to all kinds of data silos. Furthermore we present a method for formal service description that enables the automatic interface generation and service integration into existing data sets.

SILK [11] can be used to discover links between Linked Data from different sources. Using a declarative language, a developer specifies conditions that data from different sources has to fullfil to be merged, optionally using heuristics in case merging rules can lead to ambigious results. In contrast, we use Linked Data principles for exposing content of data-providing services, and specify the construction of IRIs which can be related to already existing data.

There exists extensive literature about semantic descriptions of Web services. We distinguish between two kinds of works: (i) general semantic Web service (SWS) frameworks, and (ii) stateless service descriptions.

General SWS approaches include OWL-S [12] and WSMO [13] and aim at providing extensive expressivity in order to formalise every kind of Web service, including complex business services with state changes and non-trivial choreographies. Their expressivity comes with a price: they require complex modelling even for simple data services using formalisms that are not familiar to all semantic Web developers. In contrast, our approach focuses on simple data services and their lightweight integration with Linked Data.

Most closely related to our service description formalism are works on semantic descriptions of stateless services (e.g. [14–16]). Similar to our approach these solutions define service functionality in terms of input and output conditions. Most of them, except [14], employ proprietary description formalisms. In contrast, our approach relies on standard SPARQL. Furthermore our work provides the following key advantages: (i) a methodology to provide a linked data interface to services, (ii) semi-structured input and output definitions, compared to the static definition of required inputs and outputs in previous approaches.

In [17], the authors of [16] present a system that can answer SPARQL queries over Web services, however it is limited to queries that can be answered by a single service, not in combination with other data sets.

Other related work to query answering with service integration comes from the database community, specifically information integration. Mediator systems (e.g. Information Manifold [18]) are able to answer queries over heterogeneous data sources, including services on the Web. Information-providing data services were explicitly treated, e.g. in [19, 20]. For an extensive overview of query answering in information integration systems, we refer the reader to [7]. All these works have in common that they answer queries using services, but do not provide methods to expose services with a standardised interface and link-able interfaces. Thus information integration is only done at the time of query answering, which is contrast to our proposed approach that allows data sets to be directly interlinked, independent of a query processor.

8 Conclusions

We presented a general method to integrate data services and Linked Data. The method includes an interface convention that allows service inputs to be given as URIs and thus linked from other data sources. By exposing URIs for service inputs instead of service outputs, the model neatly integrates with existing data, can handle multiple outputs for one input and makes the relation between input and output data explicit. Furthermore we proposed a lightweight description formalism and used it to define algorithms for automatically interlinking Linked Data Services (LIDS) with appropriate data sets, and for evaluating conjunctive queries with automatic invocation of LIDS. We showed how the method can be implemented using SPARQL. We applied our method to create LIDS for existing real-world service, thus contributing new data to the Web. The approach was evaluated for performance and effectiveness in two experiments. The first was about evaluating SPARQL queries on Linked Data with and without the integration of a LIDS wrapping GeoNames. The results showed that with relatively low time overhead it was possible to produce more answers to the given test queries. The second experiment interlinked the Billion Triple Challenge (BTC) 2010 data set with the GeoNames LIDS wrapper. We showed that the algorithm scales even to this very large data set and produces large numbers (around 2.5 million) new links between entities. A possible avenue for future work would be to integrate fuzzy matching algorithms, similar to [11], in case the input to a web service is ambigious, e.g. in the case services which take keywords as input.

References

1. Berners-Lee, T.: Linked Data . Design Issues (2009) `http://www.w3.org/DesignIssues/LinkedData`.
2. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium, Recommendation (February 2004) `http://www.w3.org/TR/rdf-concepts/`.
3. Bizer, C., Cyganiak, R., Gauss, T.: The RDF Book Mashup: From Web APIs to a Web of Data. In: Workshop on Scripting for the Semantic Web at European Semantic Web Conference (ESWC). (2007)
4. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. World Wide Web Consortium, Recommendation (2008) `http://www.w3.org/TR/rdf-sparql-query/`.
5. Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A., Reynolds, D.: RIF Core Dialect. World Wide Web Consortium, Recommendation (June 2010) `http://www.w3.org/TR/rif-core/`.
6. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming **8**(03) (2008)

7. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal **10** (2001) 270 – 294
8. Widom, J., Finkelstein, S.J.: A syntax and semantics for set-oriented production rules in relational database systems. SIGMOD Record **18**(3) (1989) 36–45
9. Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL Queries over the Web of Linked Data. In: 8th International Semantic Web Conference (ISWC). (2009)
10. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: Proceedings of the 19th International Conference on World Wide Web (WWW). (2010)
11. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: 8th International Semantic Web Conference (ISWC). (2009) 650–665
12. W3C: OWL-S: Semantic Markup for Web Services. W3C Member Submission (2004) `http://www.w3.org/Submission/OWL-S/`.
13. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Applied Ontology **1**(1) (2005) 77–106
14. Iqbal, K., Sbodio, M.L., Peristeras, V., Giuliani, G.: Semantic Service Discovery using SAWSDL and SPARQL. In: International Conference on Semantics, Knowledge and Grid (SKG). (2008)
15. Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., Stevens, R.: Deciding Semantic Matching of Stateless Services. AAAI Conference on Artificial Intelligence (AAAI) (2006)
16. Zhao, W.F., Chen, J.L.: Toward Automatic Discovery and Invocation of Information-Providing Web Services. In: Asian Semantic Web Conference (ASWC). (2006)
17. Zhao, W.F., Meng, X., Chen, J.L., Liu, C.: Unify the description of Web services and RDF data sources towards uniform data integration. In: China-Ireland International Conference on Information and Communications Technologies (CIICT). (2008)
18. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: International Conference on Very Large Data Bases (VLDB). (1996)
19. Thakkar, S., Ambite, J.L., Knoblock, C.A.: A Data Integration Approach to Automatically Composing and Optimizing Web Services. In: Workshop on Planning and Scheduling for Web and Grid Services at International Conference on Automated Planning and Scheduling (ICAPS). (2004)
20. Barhamgi, M., Champin, P.A., Benslimane, D.: A Framework for Web Services-Based Query Rewriting and Resolution in Loosely Coupled Information Systems (2007) Technical Report RR-LIRIS-2007-001; Laboratoire d'InfoRmatique en Image et Systmes d'information (LIRIS). Available at `http://liris.cnrs.fr/Documents/Liris-2647.pdf`.