*Andreas Harth, Katja Hose, Ralf Schenkel*

# *Linked Data Management: Principles and Techniques*

# *List of Figures*

# *List of Tables*

# *Contents*

# Part I

# This is a Part

# Chapter 1

## Linked Data Services

**Sebastian Speiser, Martin Junghans**

*Institute AIFB, Karlsruhe Institute of Technology (KIT)*
*Email: {speiser,junghans}@kit.edu*

**Armin Haller**

*CSIRO Computational Informatics, Australian National University*
*Email: armin.haller@csiro.au*



### 1.1 Introduction

Information services are commonly provided via Web APIs based on Representational State Transfer (REST) principles [6, 17] or via Web Services based on the WS-* technology stack [5, 16]. Currently deployed information services use HTTP as transport protocol, but return data as JSON or XML which requires glue code to combine data from different APIs with information provided as Linked Data. Linked Data interfaces for services have been created, e.g., in form of the book mashup [3] which returns RDF data about books based on Amazon's API, or twitter2foaf which encodes the Twitter follower network of a given user based on the API provided by Twitter. 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 person instances (e.g., described using the

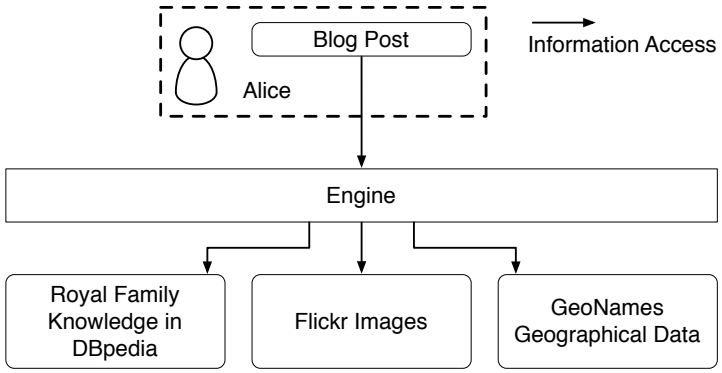


FIGURE 1.1: Scenario: Alice's Blog Post

FOAF vocabulary[1]) and their 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/`.

In this chapter, we present the LInked Data Services (LIDS) approach for creating Linked Data interfaces to information services. The approach incorporates formal service descriptions that enable (semi-)automatic service discovery and integration. Specifically, we present the following components: an access mechanism for LIDS interfaces based on generic Web architecture principles (URIs and HTTP) (Section 1.4); a generic lightweight data service description formalism, instantiated for RDF and SPARQL graph patterns (Section 1.5); and an algorithm for interlinking existing data sets with LIDS (Section 1.6). We discuss related work in Section 1.7. Finally, we evaluate our approach in Section 1.8 and conclude in Section 1.9. Parts of this chapter were previously presented in [20, 21, 19].

## 1.2 Scenario

This section introduces a scenario which we use to motivate, explain, and validate our approach throughout the chapter.

We consider the example of Alice who wants to write a blog post about the English royal family. For this, she needs a complete list of all descendants of Queen Elizabeth II and for each descendant a photo with information about where it was taken. Alice has access to a system that can answer declaratively the specified information needs by accessing information sources and

[1] `http://www.foaf-project.org/`

services, e.g., the Data-Fu system [22]. For Alice's need the engine accesses DBpedia, a database containing facts extracted from Wikipedia, to get the descendants of Queen Elizabeth II; the Flickr API using the names of the descendants to match photos; and the GeoNames API to determine from the photos' geographical locations a name for the place. In Figure 1.1 we visualize the information access that Alice performs to create her blog post.

Linked Data sources cover to some extent the required information for our scenario. Alice can find information about the royal family as Linked Data via DBpedia which includes the names of the family members and also some links to photos. The photos, however, are missing geographical information, so she needs an alternative source of pictures. Flickr supports geographical information for photos, but does not allow arbitrary access to their database. It only allows access via a predefined service interface. Thus, to cover her information needs, we have to integrate the Flickr services with the Linked Data from DBpedia. Furthermore, to check which geographical feature with a human-understandable name is near to the location of a photo, given as latitude and longitude, she needs to invoke another service. She chooses the GeoNames service which relates geographical points to nearby points of interests (e.g., the Buckingham Palace). This relation cannot be fully materialized but must be provided as a service, as there is an infinite number of geographical points which can be given with arbitrary precision.

In this scenario, we find information sources that cannot be materialized as static data sets for various reasons: (i) data is constantly changing, e.g., new photos are frequently uploaded to Flickr; (ii) data is generated depending on input from a possibly infinite domain, e.g., the nearest geographical location can be searched for arbitrary coordinates; (iii) the information provider does not want to give arbitrary access to the information, e.g., Flickr only provides access to individual photos or pre-defined types of queries in order to avoid somebody copying the whole page. We denote such information sources as information services, as they provide a restricted view on a potentially infinite data set.

## 1.3 Information Services

Our notion of information services is as follows:

> Information services return data dynamically at runtime (i.e., during service call time) from the supplied input parameters. Information services neither alter the state of some entity nor modify any data. In other words, information services are free of any side effects. They can be seen as data sources providing information about an entity based on a given input in the form of a set of name/value pairs. The notion of information services

include Web APIs and REST-based services providing output data in XML or JSON.

**Example 1.** *The Flickr API provides besides other functionality a fulltext search for photos. To search for photos of Prince Charles which are tagged as portraits and have geographic information the following URI can be used (after adding an application-specific API key to the URI that allows Twitter to monitor the API usage by each application):*

```
http://api.flickr.com/services/rest/?method=flickr.photos.
search&text=charles,+prince+of+wales&format=json
```

*with the following (abbreviated) result:*

```
...
{"photos":{"page":1, "pages":12, "perpage":100, "total":"1122",
 "photo":[{"id":"5375098012", "owner":"50667294@N08",
           "secret":"c8583acbbe","server":"5285", "farm":6,
           "title":"The Prince of Wales at
                    Queen Elizabeth Hospital Birmingham"},
          {"id":"2614868465", "owner":"15462799@N00",
           "secret":"50af5f09c9","server":"3149", "farm":4,
           "title":"Prince Charles" ...},
          {"id":"4472414639", "owner":"48399297@N04",
           "secret":"cb8533c199","server":"4025", "farm":5,
           "title":"HRH Prince Charles Visits Troops in Afghanistan"
            ...} ... ] } }
```

*The returned information is given in JSON in a service-specific vocabulary. To retrieve further information about the first photo with the* id *"5375098012", we have to know service-specific rules to build the links to those information sources, e.g., we can construct the URI* `http://farm6.staticflickr.com/5285/5375098012_c8583acbbe.jpg` *according to the Flickr URI construction rules*[2] *to access the JPEG of the actual photo; or we can access the following URI to get further information on the photo:*

```
http://api.flickr.com/services/rest/?method=flickr.photos.
getInfo&photo_id=5375098012&format=json
```

*Retrieving the URI (again with appended API key) gives the following result:*

```
{"photo":{"id":"5375098012", "secret":"c8583acbbe", "server":"5285",
          "farm":6, "license":"6", ...
          "location":{"latitude":52.453616,"longitude":-1.938303,...},
          ... }}
```

[2] `http://www.flickr.com/services/api/misc.urls.html`

*Using the retrieved geographical coordinates, we can build the URI for calling the GeoNames* `findNearbyWikipedia` *service, which relates given latitude/-longitude parameters to Wikipedia articles describing geographical features*[3] *that are nearby. This requires first Flickr-specific knowledge how to extract the latitude and longitude of the image and GeoNames-specific knowledge how to construct the URI for a service call which is:*

```
http://api.geonames.org/findNearbyWikipedia?lat=52.453616&lng=-1.938303
```

*The (abbreviated) result is the following:*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
<entry>
 <lang>en</lang>
 <title>Birmingham Women's Fertility Centre</title>
 ...
 <lat>52.4531</lat>
 <lng>-1.9389</lng>
 <wikipediaUrl>
  http://en.wikipedia.org/wiki/Birmingham_Women%27s_Fertility_Centre
 </wikipediaUrl>
 ...
 <distance>0.0702</distance>
</entry>
...
<entry>
 <lang>en</lang>
 <title>University (Birmingham) railway station</title>
 ...
 <lat>52.451</lat>
 <lng>-1.936</lng>
 <wikipediaUrl>
http://en.wikipedia.org/wiki/University_Birmingham_railway_station
 </wikipediaUrl>
 <distance>0.3301</distance>
</entry>
...
</geonames>
```

This simple example shows that integrating data from several (in this case only two) services is difficult for the following reasons:

- different serialization formats are used (e.g., JSON, XML);
- entities are not represented explicitly, and are thus difficult to identify between different services. For example, the geographical point returned

[3]GeoNames classifies geographical features into nine classes with 645 subcodes. Such features comprise states, roads, and mountains for example.

by the Flickr API does not occur in the output of the GeoNames service. Therefore it is not possible to link the results based on the service outputs alone, but only with service-specific gluing code.

---

## 1.4 LInked Data Services (LIDS)

Linked Data Services provide a Linked Data interface for information services. To make these services adhere to Linked Data principles a number of requirements have to be fulfilled:

- the input for a service invocation with given parameter bindings must be identified by a URI;
- resolving that URI must return a description of the input entity, relating it to the service output data; and
- RDF descriptions must be returned.

We call such services *Linked Data Services (LIDS)*.

**Example 2.** *Inputs for the LIDS version of the* `findNearbyWikipedia` *service are entities representing geographical points given by latitude and longitude, which are encoded in the URI of an input entity. Resolving such an input URI returns a description of the corresponding point which relates it to Wikipedia articles about geographical features which are nearby.*

Defining that the URI of a LIDS call identifies an input entity is an important design decision. Compared to the alternative – directly identifying output entities with service call URIs – identifying input entities has the following advantages:

- the link between input and output data is made explicit;
- one input entity (e.g., a geographical point) can be related to several results (e.g., Wikipedia articles);
- the absence of results can be easily represented by a description without further links;
- the input entity has a constant meaning although data can be dynamic (e.g., the input entity still represents the same point, even though a subsequent service call may relate the input entity to new or updated Wikipedia articles).

More formally we characterize a LIDS by:

- Linked Data Service endpoint HTTP URI $uri_{ep}$.
- Local identifier $i$ for the input entity of the service.
- Inputs $X_i$: names of parameters.

The URI $uri_{X_i}$ of a service call for a parameter assignment $\mu$ (mapping $X_i$ to corresponding values) and a given endpoint URI $uri_{ep}$ is constructed in the following way. We use the Kleene star to notate arbitrary repetition of the group (zero or more times).

$$uri_{X_i} := uri_{ep}\big[\texttt{?}X_i\texttt{=}\mu(X_i)\texttt{&}\big]^*.$$

That is, for each $x \in X_i$, we add a parameter and its value $\mu(x)$ to the URI $uri_{X_i}$. Additionally we introduce an abbreviated URI schema that can be used if there is only one required parameter (i.e. $|X_i| = 1, X_i = \{x\}$):

$$uri_{X_i} := uri_{ep}\texttt{/}\mu(x).$$

Please note that the above definition coincides with typical Linked Data URIs. We define the input entity that is described by the output of a service call as

$$inp_{X_i} = uri_{X_i}\texttt{\#}i.$$

**Example 3.** *We illustrate the principle using the openlids.org wrapper for GeoNames*[4] `findNearbyWikipedia`*. The wrapper is a LIDS, defined by:*

- *endpoint ep* = `gw:findNearbyWikipedia`*;*
- *local identifier* $i$ = `point`*;*
- *inputs* $X_i = \{\texttt{lat}, \texttt{lng}\}$.

*For a binding* $\mu = \{\texttt{lat} \mapsto 52.4536, \texttt{lng} \mapsto -1.9383\}$ *the URI for the service call is* `gw:findNearbyWikipedia?lat=52.4536&lng=-1.9383` *and returns the following description:*

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .

gw:findNearbyWikipedia?lat=52.4536&lng=-1.9383#point
  foaf:based_near dbpedia:Centre_for_Human_Reproductive_Science;
   ...
   foaf:based_near
            dbpedia:University_%28Birmingham%29_railway_station.

dbpedia:Centre_for_Human_Reproductive_Science
   geo:lat "52.453";
```

[4] `http://km.aifb.kit.edu/services/geonameswrap/`, abbreviated as `gw`.

```
   geo:long "-1.9388".

dbpedia:University_%28Birmingham%29_railway_station
   geo:lat "52.451";
   geo:long "-1.936".
...
```

## 1.5 Describing Linked Data Services

In this section, we define an abstract model of LIDS descriptions.

**Definition 1.** *A LIDS description consists of a tuple* $(uri_{ep}, CQ_i, T_o, i)$ *where* $uri_{ep}$ *denotes the LIDS endpoint URI,* $CQ_i = (X_i, T_i)$ *a conjunctive query, with* $X_i$ *the input parameters and* $T_i$ *the basic graph pattern specifying the input to the service,* $T_o$ *a basic graph pattern describing the output data of the service, and* $i$ *the local identifier for the input entity.*

We define $X_i$ to be the selected variables of a conjunctive query whose body specifies the required relation between the input parameters. $T_o$ specifies the minimum output that is returned by the service for valid input parameters. More formally:

- $\mu \in \mathcal{M}$[5] is a valid input, if $\mathsf{dom}(\mu) = X_i$;
- for a valid $\mu$, constructing $uri_{X_i}$ returns a graph $D_o$, such that

$$\forall \mu'.\big(\mu' \in [\![T_o]\!]_{D_{impl}} \wedge \mu' \sim \mu\big) \rightarrow \mu'(T_o) \subseteq D_o,$$

  where $D_{impl}$ is the potentially infinite virtual data set representing the information provided by the LIDS.

**Example 4.** *We describe the* `findNearbyWikipedia` *openlids.org wrapper service as* $(uri_{ep}, CQ_i, T_o, i)$ *with:*

$uri_{ep}$ = `gw:findNearbyWikipedia`
$CQ_i$ = ({`lat`, `lng`}, {?`point geo:lat ?lat . ?point geo:long ?lng`})
$T_o$ = {?`point foaf:based_near ?feature`}
$i$ = `point`

[5] $\mathcal{M}$ is the set of all variables bindings.

### 1.5.1 Relation to Source Descriptions in Information Integration Systems

Note that the LIDS descriptions can be transformed to source descriptions with limited access patterns in a Local-as-View (LaV) data integration approach [7]. With LaV, the data accessible through a service is described as a view in terms of a global schema. The variables of a view's head predicate that have to be bound in order to retrieve tuples from the view are prefixed with a \$. For a LIDS description $(uri_{ep}, CQ_i, T_o, i)$, we can construct the LaV description:

$$uri_{ep}(\$I_1, \ldots, \$I_k, O_1 \ldots, O_m) \text{ :- } p_1^i(\ldots), \ldots, p_n^i(\ldots), p_1^o(\ldots), \ldots, p_l^o(\ldots).$$

Where $CQ_i = (X_i, T_i)$, with $X_i = \{I_1, \ldots, I_k\}$ and
$T_i = \{(s_1^i, p_1^i, o_1^i), \ldots, (s_n^i, p_n^i, o_n^i)\}$, $T_o = \{(s_1^o, p_1^o, o_1^o), \ldots, (s_l^o, p_l^o, o_l^o)\}$, and $vars(T_o) \setminus vars(T_i) = \{O_1, \ldots, O_m\}$.

We propose for LIDS descriptions the separation of input and output conditions for three reasons: (i) the output of a LIDS corresponds to an RDF graph as described by the output pattern in contrast to tuples as it is common in LaV approaches, (ii) it is easier to understand for users, and (iii) it is better suited for the interlinking algorithm as shown in Section 1.6.

### 1.5.2 Describing LIDS using RDF and SPARQL Graph Patterns

In the following we present how LIDS descriptions can be represented in RDF, thus enabling that LIDS descriptions can be published as Linked Data. The basic format is as follows (unqualified strings consisting only of capital letters are placeholders and explained below):

```
@prefix lids: <http://openlids.org/vocab#>

LIDS a lids:LIDS;
     lids:lids_description [
         lids:endpoint ENDPOINT ;
         lids:service_entity ENTITY ;
         lids:input_bgp INPUT ;
         lids:output_bgp OUTPUT ;
         lids:required_vars VARS ] .
```

The RDF description is related to our abstract description formalism in the following way:

- `LIDS` is a resource representing the described Linked Data service;
- `ENDPOINT` is a URI $uri_{ep}$;
- `ENTITY` is the name of the entity $i$;

- `INPUT` and `OUTPUT` are basic graph patterns encoded as a string using SPARQL syntax. `INPUT` is mapped to $T_i$ and `OUTPUT` is mapped to $T_o$.

- `VARS` is a string of required variables separated by blanks, which is mapped to $X_i$.

From this mapping, we can construct an abstract LIDS description $(uri_{ep}, (X_i, T_i), T_o, i)$ for the service identified by `LIDS`.

**Example 5.** *In the following we show the RDF representation of the formal LIDS description from Example 4:*

```
:GeowrapNearbyWikipedia a lids:LIDS;
  lids:lids_description [
     lids:endpoint
   <http://km.aifb.kit.edu/services/geonameswrap/findNearbyWikipedia>;
     lids:service_entity "point" ;
     lids:input_bgp "?point a Point . ?point geo:lat ?lat .
                                      ?point geo:long ?long" ;
     lids:output_bgp "?point foaf:based_near ?feature" ;
     lids:required_vars "lat long" ] .
```

In the future, we expect a standardized RDF representation of SPARQL which does not rely on string encoding of basic graph patterns. One such candidate is the SPIN SPARQL Syntax[6] which is part of the SPARQL Inferencing Notation (SPIN)[7]. We are planning to re-use such a standardized RDF representation of basic graph patterns and variables in future versions of the LIDS description model.

## 1.6 Interlinking Data with LIDS

In the following, we describe how existing data sets can be automatically enriched with links to LIDS in different settings. Consider for example:

- the processing of a static data set, inserting links to LIDS and storing the new data;

- a Linked Data server that dynamically adds links to LIDS;

- a data browser that augments retrieved data with data retrieved from LIDS.

[6] http://spinrdf.org/sp.html
[7] http://spinrdf.org/

We present a technique that, based on a fixed local dataset, determines and invokes the appropriate LIDS and adds the output to the local dataset.

Given an RDF graph $G$ and a LIDS description $l = (uri_{ep}, CQ_i = (X_i, T_i), T_o, i)$, we obtain the equivalences between $\mu(i)$ and $inp_{X_i} = uri_{X_i}\#i$ for each valid input $\mu \in [\![T_i]\!]_G$. These equivalences can be either used to immediately resolve the LIDS URIs and add the data to $G$, or to make the equivalences explicit in $G$, for example, by adding the following triples to $G$:

**for all** $\mu \in [\![T_i]\!]_G$ **do**
  $G \leftarrow G, \mu(i)$ `owl:sameAs` $inp_{X_i}$
  $G_o \leftarrow$ invoke $uri_{X_i}$
  $G \leftarrow G, G_o$
**end for**

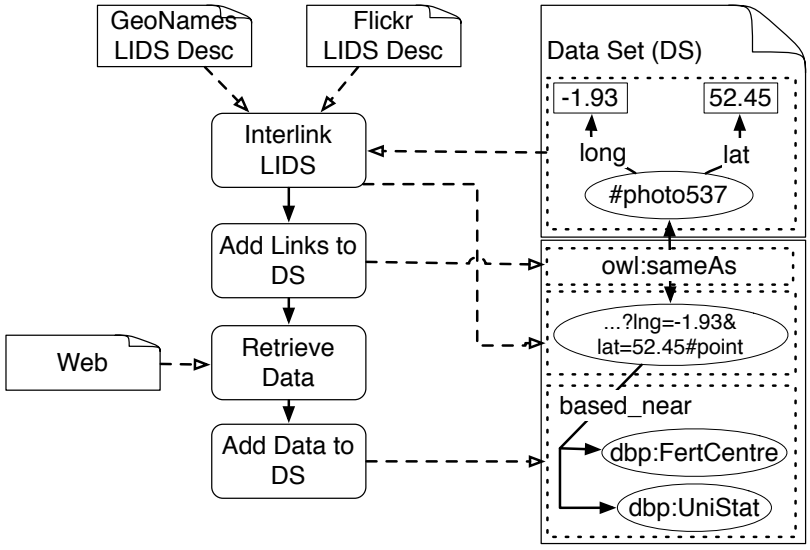


FIGURE 1.2: Interlinking example for GeoNames LIDS

We illustrate the algorithm using LIDS versions of the Flickr API and the GeoNames services. The example and the algorithm are visualized in Figure 1.2. A more formal definition of the algorithm can be found in [19]. Consider a photo `#photo537` for which Flickr returns an RDF graph with latitude and longitude properties:

```
#photo537 rdfs:label "The Prince of Wales ...";
          geo:lat  "52.453616";
          geo:long "-1.938303".
```

In the first step, the data is matched against the available LIDS descriptions (for brevity we assume a static set of LIDS descriptions) and a set of bindings are derived. Further processing uses the GeoNames LIDS which accepts latitude/longitude as input. After constructing a URI which represents the service entity, an equivalence (`owl:sameAs`) link is created between the original entity `#photo537` and the service entity:

```
#photo537 owl:sameAs
    gw:findWikipediaNearby?lat=52.453616&long=-1.938303#point.
```

Next, the data from the service entity URI can be retrieved to obtain the following data:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
gw:findWikipediaNearby?lat=52.453616&lng=-1.938303#point
       foaf:based_near foaf:based_near dbpedia:FertCentre
       foaf:based_near dbpedia:UniStation.
...
```

Please observe that by equating the URI from the input data with the LIDS entity URI, we essentially add the returned `foaf:based_near` statements to `#photo537`. Should the database underlying the service change, a lookup on the LIDS entity URI returns the updated data which can then be integrated. As such, entity URIs can be linked in the same manner as plain Linked Data URIs.

---

## 1.7 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 specialized wrappers, for example the book mashup [3]. We presented the basic concepts of LIDS in [20] and developed the algorithm for interlinking data with LIDS in [21]. Other state-of-the-art data integration systems [24] use wrappers to generate RDF and then publish that RDF online rather than providing access to the services that generate RDF directly. In contrast to these ad-hoc interfaces, we provide a uniform way to construct such interfaces, 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 service integration into existing data sets.

SILK [26] enables the discovery of links between Linked Data from different sources. Using a declarative language, a developer specifies conditions that data from different sources has to fulfill to be merged, optionally using heuristics in case merging rules can lead to ambiguous results. In contrast,

we use Linked Data principles for exposing content of data-providing services, and specify the relationship between existing data and data provided by the service using basic graph patterns. Alternatively, the LIDS approach could also be adapted to use the SILK language for input conditions.

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 [27] and WSMO [18] and aim at providing extensive expressivity in order to formalize every kind of Web Service, including complex business services with state changes and non-trivial choreographies. The expressivity comes at a price: SWS require complex modeling even for simple data services using formalisms that are not familiar to all Semantic Web developers. Considering that, implementations of WSMO [8] did use controlled state Abstract State Machines to procedurally model parts of a choreography [4]. In contrast, our approach focuses on simple information services and their lightweight integration with Linked Data via standard graph patterns.

Closely related to our service description formalism are works on semantic descriptions of stateless services (e.g., [10, 29, 11, 28]). Similar to our approach these solutions define service functionality in terms of input and output conditions. The approaches in [10, 29] employ proprietary description formalisms, [11] uses SAWSDL and SPARQL whereby [28] is assuming the service to be an information service that only adds properties to an input type defined by an ontology. In contrast, our approach relies on standard basic graph patterns. Moreover, our work provides a methodology to provide a Linked Data interface to services.

Norton and Krummenacher propose an alternative approach to integrate Linked Data and services, so-called Linked Open Services (LOS) [15]. LOS descriptions also use basic graph patterns for defining service inputs and outputs. One difference is that our work uses name-value pairs for parameters whereas LOS consume RDF. Thus, in contrast to LOS, the LIDS approach allows that service calls are directly linkable from within Linked Data, as service inputs are encoded in the query string of a URI. The RESTdesc approach semantically describes REST services using N3 [25]. While RESTdesc also uses BGPs as part of N3 for input and output description, the described services are not confined to communicate RDF. Thus, RESTdesc services require additional measures to integrate with Linked Data.

Mediator systems (e.g. Information Manifold [14]) are able to answer queries over heterogeneous data sources, including services on the Web. Information-providing data services are explicitly treated, e.g. in [23, 1]. For an extensive overview of query answering in information integration systems, we refer the interested reader to [7]. All these works have in common that they generate top-down query plans, which is possible because of the completely known schema of the targeted relational databases. In contrast, our proposed approach employs a data-driven approach, where service calls are constructed

when enough input data is found, and services are invoked if they are relevant for the data at hand.

---

## 1.8 Implementation and Evaluation

In Section 1.8.1, we apply our approach in a proof-of-concept solution for the scenario presented in Section 1.2 to show the feasibility of our proposed methods. Besides the feasibility study, we present results of experiments to measure the performance of interlinking of LIDS with existing datasets in Section 1.8.2.

### 1.8.1 Realization of Scenario

For realizing the scenario from Section 1.2, we extended the interlink algorithm to expand the underlying dataset automatically by retrieving dynamically discovered Linked Data URIs similar to Linked Data query processors such as [9, 13]. On the resulting dataset, we support the evaluation of SPARQL queries. The implementation of the extended algorithm resulted in the DataFu Engine [22].

Alice wants to get a list of the descendants of Queen Elizabeth II and for each descendant a picture together with geographical information where it was taken. To gather the information she formulates the following query:

```
prefix vocab: <http://openlids.org/examples/ezII/vocab#>
prefix dbpo: <http://dbpedia.org/ontology/>
prefix dbp: <http://dbpedia.org/resource/>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix flickrlids: <http://openlids.org/flickrlids/vocab#>

SELECT ?n ?p ?f WHERE {
  dbp:Elizabeth_II vocab:hasDescendant ?x .
     ?x foaf:name ?n .
     ?x foaf:depiction ?p . ?p flickrlids:hasLocation ?loc .
         ?loc foaf:based_near ?f
}
```

The relevant triples that the Linked Data query engine can obtain when dereferencing `http://dbpedia.org/resource/Elizabeth_II` are as follows:

```
dbp:Anne,_Princess_Royal           dbpo:parent  dbp:Elizabeth_II .
dbp:Charles,_Prince_of_Wales       dbpo:parent  dbp:Elizabeth_II .
dbp:Prince_Andrew,_Duke_of_York    dbpo:parent  dbp:Elizabeth_II .
dbp:Prince_Edward,_Earl_of_Wessex  dbpo:parent  dbp:Elizabeth_II .
```

We notice that the data contains the `dbpo:parent` property instead of the queried `vocab:hasDescendant` property. We formalize the relation of the properties, i.e., that `vocab:hasDescendant` is the transitive closure of the inverse of `dbpo:parent`, with the following rules:

```
prefix vocab: <http://openlids.org/examples/ezII/vocab#>
prefix dbpo: <http://dbpedia.org/ontology/>

CONSTRUCT { ?x vocab:hasDescendant ?y } WHERE {
     ?y dbpo:parent ?x
}

CONSTRUCT { ?x vocab:hasDescendant ?z } WHERE {
    ?x vocab:hasDescendant ?y .
    ?y vocab:hasDescendant ?z
}
```

Together with this background knowledge, we can derive a list of descendants (bindings for `?x`) and their names (bindings for `?n`). In the following, we list an excerpt of the bindings:

```
?x => dbp:Charles,_Prince_of_Wales          ?n => "Prince Charles"
?x => dbp:Anne,_Princess_Royal             ?n => "Princess Anne"
?x => dbp:Prince_Andrew,_Duke_of_York      ?n => "Prince Andrew"
?x => dbp:Prince_Edward,_Earl_of_Wessex    ?n => "Prince Edward"
?x => dbp:Peter_Phillips                   ?n => "Peter Phillips"
?x => dbp:Zara_Phillips                    ?n => "Zara Phillips"
?x => dbp:Prince_William,_Duke_of_Cambridge ?n => "Prince William"
?x => dbp:Prince_Harry_of_Wales            ?n => "Prince Harry"
?x => dbp:Princess_Beatrice_of_York        ?n => "Princess Beatrice"
?x => dbp:Princess_Eugenie_of_York         ?n => "Princess Eugenie"
?x => dbp:Lady_Louise_Windsor              ?n => "Lady Louise Windsor"
?x => dbp:James,_Viscount_Severn           ?n => "Viscount Severn"
?x => dbp:Laura_Lopes                      ?n => "Laura Lopes"
```

Note that Laura Lopes is returned as a result as her relation to her stepfather Prince Charles is modeled in DBpedia using the `dbpo:parent` property. Further note that the children Savannah Phillips and Isla Elizabeth Phillips of Peter Phillips were not yet represented in the accessed DBpedia version and are thus missing.

While some of the descendants binding to `?x` have associated pictures linked via the `foaf:depiction` property, none of them has geographic information. So, we have to invoke the LIDS version of the Flickr service to retrieve additional photos with geographical information. We wrapped the Flickr API so that it takes the name of a person and returns a list of photos of the person together with their locations. The LIDS description is given as follows:

```
:FlickrLIDS a lids:LIDS;
```

TABLE 1.1: Performance characteristics of Alice's query

| Measurements | live mode | proxy mode |
|---|---|---|
| Number of results | 2,402 | 2,402 |
| Number of retrieved IRIs | 1,411 | 1,411 |
| Run time | 265.66 s | 11.59 s |

```
lids:lids_description [
  lids:endpoint
    <http://km.aifb.kit.edu/services/flickrlids/depictions>;
  lids:service_entity "person" ;
  lids:input_bgp "?person foaf:name ?name";
  lids:output_bgp "?person foaf:depiction ?p .
                   ?p :hasLocation ?loc .
                   ?loc geo:lat ?lat . ?loc geo:long ?long" ;
  lids:required_vars "name" ] .
```

Furthermore the LIDS version of the GeoNames service has to be invoked (as described in Section 1.4) to find nearby located geographical features given the latitude and longitude of a picture. Finally in our experiments we obtained 358 results from which Alice can select one result per descendant. For example the result for Prince Charles is:

```
?n => "Charles, Prince of Wales"
?p => <http://farm6.staticflickr.com/5285/5375098012_c8583acbbe.jpg>
?f => dbp:Centre_for_Human_Reproductive_Science
```

**Efficiency of Answering Alice's Query**

In the following, we present performance characteristics of executing Alice's query about Queen Elizabeth II's descendants and pictures of them with geographical information. The experiment was run on a virtual machine with 4 CPU cores of the Intel x64 architecture, each running with 2.26 GHz and total RAM of 8 GB. The virtual machine was hosted in KIT's Open Nebula cloud. The experiment was either run *live* accessing the actual Linked Data on the Web, or in *proxy* mode, where data is cached in a local instance of the Cumulus RDF store [12]. Cumulus RDF is an HTTP proxy that serves Linked Data stored in a Cassandra backend.

The Flickr LIDS and GeoNames LIDS were hosted locally on the machine performing the query in live mode, but the LIDS had to access the wrapped services (i.e., the Flickr API and the GeoNames services) on the Web. The retrieved quads were loaded into a Cumulus RDF instance and the query repeated in proxy mode. The measurements of both runs are shown in Table 1.1.

Not surprisingly, both live and proxy mode retrieved the same number of information sources and yielded the same results, as the proxy mode uses exactly the data that was retrieved by the live run. The run time of proxy mode is naturally much lower than that of live mode and shows that the query

TABLE 1.2: Performance characteristics of Alice's query for other Persons

| **Persons** | **Number of results** | **Number of IRIs** | **Run time** |
|---|---|---|---|
| Bill Clinton | 204 | 114 | 31.15 s |
| Mohamed Al-Fayed | 135 | 115 | 36.72 s |
| Cher | 55 | 75 | 25.33 s |
| Nigel Lawson | 180 | 101 | 26.17 s |
| Queen Silvia of Sweden | 398 | 351 | 69.52 s |
| Barbara Bush | 159 | 137 | 26.00 s |
| Princess Irene (Netherlands) | 50 | 77 | 25.11 s |
| Prince Edward | 116 | 124 | 39.06 s |
| Nancy Pelosi | 35 | 73 | 27.64 s |
| Constantine II of Greece | 74 | 122 | 30.09 s |
| Diana Quick | 5 | 39 | 25.06 s |
| Dick Cheney | 265 | 194 | 46.97 s |

execution adds only a little overhead compared to the time for accessing data and services on the Web, which makes up 95.6 % of total execution time.

Queen Elizabeth II has a large number of descendants with many Flickr photos. We thus performed the query for a number of persons to show that the approach can be applied in other situations without any customization. We selected twelve random persons from DBpedia who fulfill three requirements: (i) born on or after January 1st, 1925 (persons born before this date rarely have photos on Flickr); (ii) have at least one child recorded on DBpedia; (iii) are still alive (according to DBpedia). The queries were performed in live mode and the results are recorded in Table 1.2. The results show that our approach facilitates an easy adaption of a query to different information needs.

### 1.8.2 Implementing and Interlinking Linked Data Services

We first present several LIDS services which we have made available, and then cover the evaluation of performance and effectiveness of the presented algorithm for interlinking Linked Data with LIDS. 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[8]. All experiments were conducted on a 2.4 GHz Intel Core2Duo laptop with 4 GB of main memory.

#### 1.8.2.1 Implementing LIDS Services

In this section, we show how we applied the LIDS approach to construct publicly available Linked Data interfaces for selected existing services.

The following services are hosted on Google's App Engine cloud environ-

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

ment. The services are also linked on `http://openlids.org/` together with their formal LIDS descriptions and further information, such as IRIs of example entities.

- GeoNames Wrapper[9] 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.
- GeoCoding Wrapper, returning the geographic coordinates of a street address.
- Twitter Wrapper[10] links Twitter account holders to the messages they post.

The effort to produce a LIDS wrapper is typically low. The interface code that handles the service IRIs and extracts parameters can be realized by standardized 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. For some services it is sufficient to write XSLTs that transform XML to RDF, or simple pieces of procedural code that transform JSON to RDF. 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 description requires a SPARQL query to describe the service.

#### 1.8.2.2 Interlinking Existing Data Sets with LIDS

We implemented a streaming version of the interlinking algorithm shown in Section 1.6 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 IRIs. Our interlinking process added 891 new domains that are now linked to GeoNames via the

[9] `http://km.aifb.kit.edu/services/geowrap/`
[10] `http://km.aifb.kit.edu/services/twitterwrap/`
[11] `http://sw.deri.org/2006/08/nxparser/`
[12] `http://km.aifb.kit.edu/projects/btc-2010/`

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.

## 1.9 Conclusion

A large portion of data on the Web is attainable through a large number of data services with a variety of interfaces that require procedural code for the integration of different data sources. We presented a general method for exposing data services as Linked Data, which enables the integration of different data sources without specialized code. Our method includes an interface convention that allows service inputs to be given as URIs and thus linked from other Linked Data sources. By exposing URIs for service inputs in addition to 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 showed how it can be used for automatically interlinking Linked Data Services with appropriate data sets. We showed how the descriptions can be instantiated in 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 an experiment in which we 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) of new links between entities

In the following, we outline possible next steps that can be based on our work.

**Combination with Statistical and Heuristic Methods**

We consider the methods developed in this chapter as exact in the sense that they clearly and unambiguously specify the expected result. The contact with Linked Data in the real world has shown us limitations of such exact approaches due to the heterogeneity and the lack of quality of the data. In the following, we outline how to overcome the limitations by combining our work with statistical and heuristic methods.

A core feature of Linked Data Services is identity resolution by defining equivalences between entities in the service response with entities in other

[13]Linking data is available online: `http://people.aifb.kit.edu/ssp/geolink.tgz`

information sources. We built the resolution on basic graph patterns, which provide exact descriptions of entities. An interesting idea would be to replace the basic graph patterns by patterns in the SILK language [26], which supports heuristic conditions such as thresholds on the editing distance of labels. Furthermore, it would be interesting to experiment with (semi-)automatic schema alignment methods when processing Linked Data and services instead of the static rule-based alignments that we currently use.

**Alignment of Efforts for Aligning of Linked Data and Services**
Our Linked Data Services approach was developed in parallel to other independent efforts to align Linked Data and services, most notably are Linked Open Services [15] and RESTdesc [25]. We are currently in the process of aligning the different efforts under the label of Linked APIs. We already have organized events together, including tutorials at international conferences and the Linked APIs workshop in conjunction with the Extended Semantic Web Conference 2012. For the future, it would be interesting to bring our approaches and experiences into standardization activities such as the Linked Data Platform Working Group organized by the W3C.

# *Bibliography*


[1] Mahmoud Barhamgi, Pierre-Antoine Champin, and Djamal Benslimane. A Framework for Web Services-based Query Rewriting and Resolution in Loosely Coupled Information Systems, 2007.

[2] Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors. *Proceedings of the 8th International Semantic Web Conference (ISWC'09)*, volume 5823 of *LNCS*, Washington DC, USA, 2009. Springer.

[3] Christian Bizer, Richard Cyganiak, and Tobias Gauss. The RDF Book Mashup: From Web APIs to a Web of Data. In *Proceedings of the Workshop on Scripting for the Semantic Web (SFSW'07) in conjunction with the 4th European Semantic Web Conference (ESWC'07)*, Innsbruck, Austria, 2007.

[4] Emilia Cimpian and Adrian Mocan. Wsmx process mediation based on choreographies. In *Proceedings of the Third international conference on Business Process Management*, BPM'05, pages 130–143, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, 2004.

[6] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[7] Alon Y Halevy. Answering Queries using Views: A Survey. *The VLDB Journal*, 10(4):270 – 294, 2001.

[8] Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Christoph Bussler. WSMX – A Semantic Service-Oriented Architecture. In *International Conference on Web Services*, pages 321 – 328, Orlando, FL, USA, july 2005. IEEE Computer Society.

[9] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing SPARQL Queries over the Web of Linked Data. In Bernstein et al. [2], pages 293–309.

[10] Duncan Hull, Evgeny Zolin, Andrey Bovykin, Ian Horrocks, Ulrike Sattler, and Robert Stevens. Deciding Semantic Matching of Stateless Services. In Anthony Cohn, editor, *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06) - Volume 2*, pages 1319–1324, Boston, MA, USA, 2006. AAAI Press.

[11] Kashif Iqbal, Marco Luca Sbodio, Vassilios Peristeras, and Giovanni Giuliani. Semantic Service Discovery using SAWSDL and SPARQL. In *Proceedings of the 4th International Conference on Semantics, Knowledge and Grid (SKG'08)*, pages 205–212, Beijing, China, 2008.

[12] Günter Ladwig and Andreas Harth. CumulusRDF: Linked Data Management on Nested Key-Value Stores. In *Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'11) in conjunction with the 10th International Semantic Web Conference (ISWC'11)*, Bonn, Germany, 2011.

[13] Günter Ladwig and Thanh Tran. Linked Data Query Processing Strategies. In Peter F. Patel-Schneider, Yue Pan, Birte Glimm, Pascal Hitzler, Peter Mika, Jeff Pan, and Ian Horrocks, editors, *Proceedings of the 9th International Semantic Web Conference (ISWC'10) Part I*, volume 6496 of *LNCS*, pages 453–469. Springer, Shanghai, China, 2010.

[14] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources using Source Descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, pages 251–262, Mumbai (Bombay), India, 1996. Morgan Kaufmann.

[15] Barry Norton and Reto Krummenacher. Consuming Dynamic Linked Data. In *Proceedings of the 1st International Workshop on Consuming Linked Data (COLD'10) in conjunction with the 9th International Semantic Web Conference (ISWC'10)*, Shanghai, China, 2010.

[16] Mike P. Papazoglou. *Web Services: Principles and Technology*. Pearson – Hall, 2007.

[17] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, May 2007.

[18] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[19] Sebastian Speiser. *Usage Policies for Decentralised Information Processing*. PhD thesis, Karlsruhe Institut für Technologie, Fakultät für Wirtschaftswissenschaften, 2013.

[20] Sebastian Speiser and Andreas Harth. Taking the LIDS off Data Silos. In Adrian Paschke, Nicola Henze, and Tassilo Pellegrini, editors, *Proceedings the 6th International Conference on Semantic Systems (I-SEMANTICS'10)*, Graz, Austria, 2010. ACM.

[21] Sebastian Speiser and Andreas Harth. Integrating Linked Data and Services with Linked Data Services. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *Proceedings of the 8th Extended Semantic Web Conference (ESWC'11) Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 170–184, Heraklion, Crete, Greece, 2011. Springer.

[22] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In *Proceedings of the 22nd International Conference on World Wide Web (WWW'13)*, Rio de Janeiro, Brazil, 2013.

[23] Snehal Thakkar, José Luis Ambite, and Craig A Knoblock. A Data Integration Approach to Automatically Composing and Optimizing Web Services. In *Proceedings of Workshop on Planning and Scheduling for Web and Grid Services at International Conference on Automated Planning and Scheduling (ICAPS'04)*, Whistler, British Columbia, Canada, 2004.

[24] Raphael Troncy, Andre Fialho, Lynda Hardman, and Carsten Saathoff. Experiencing Events through User-Generated Media. In *Proceedings of the 1st International Workshop on Consuming Linked Data (COLD'10) in conjunction with the 9th International Semantic Web Conference (ISWC'10)*, Shanghai, China, 2010.

[25] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Rik Van de Walle, and Joaquim Gabarr Valls. Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices (NWeSP'11)*, Salamanca, Spain, 2011.

[26] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and Maintaining Links on the Web of Data. In Bernstein et al. [2], pages 650–665.

[27] W3C. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, 2004. Available at `http://www.w3.org/Submission/OWL-S/`.

[28] MD. Wilkinson, B. Vandervalk, and McCarthy L. The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation. *Journal Biomed Semantics*, 2, 2011.

[29] Wen-Feng Zhao and Jun-Liang Chen. Toward Automatic Discovery and Invocation of Information-providing Web Services. In Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia, editors, *Proceedings of the 1st Asian Semantic Web Conference (ASWC'06)*, number 4185 in Lecture Notes in Computer Science, pages 474–480, Beijing, China, 2006. Springer.