Integrating Linked Data and Services with Linked Data Services[⋆]

Sebastian Speiser and Andreas Harth

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

Abstract. A sizable amount of data on the Web is currently available via Web APIs that expose data in formats such as JSON or XML. Combining data from different APIs and data sources requires glue code which is typically not shared and hence not reused. We propose Linked Data Services (LIDS), a general, formalised approach for integrating data-providing services with Linked Data, a popular mechanism for data publishing which facilitates data integration and allows for decentralised publishing. We present conventions for service access interfaces that conform to Linked Data principles, and an abstract lightweight service description formalism. We develop algorithms that use LIDS descriptions to automatically create links between services and existing data sets. To evaluate our approach, we realise LIDS wrappers and LIDS descriptions for existing services and measure performance and effectiveness of an automatic interlinking 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 [2]. which leverage established Web standards such as Uniform Resource Identifiers (URIs), the Hypertext Transfer Protocol (HTTP) and the Resource Description Framework (RDF) [9]. 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 However, there is still a lot of data residing in silos that could be beneficially linked with other data, but will not be published as a fully materialised knowledge base.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 sensor data can have update intervals below one second;

[⋆] This paper is an extension of our previous work [15,16]. We have extended the work with a formal definition of service descriptions, an evaluation of the performance and effectiveness of the proposed methods – including the implementation of several Linked Data Services – and an extensive overview of related work.

[1] http://linkeddata.org/

– data is generated depending on possibly infinite different input data, e.g., the distance between two geographical points 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 APIs or services, in the following also called data or information services, as they provide a restricted view on a possibly implicit data set. APIs are often based on Representational State Transfer (REST) principles [4], use HTTP as transport protocol and pass parameters as name/value pairs in the URI query string. Currently deployed Web APIs return data as JSON or XML, which requires glue code to combine data from different APIs.

There are useful examples for the integration of information services and Linked Data. 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, which encodes a Twitter follower network of a given user based on Twitter's API. 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 FOAF vocabulary[2]) 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/`.

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 graph patterns (Section 4);
– an algorithm for linking existing data sets using LIDS (Section 5)

In Section 6 we describe the creation of LIDS for existing services, and present the results of an experiment measuring performance and effectiveness of the approach. The experiment interlinks the 2010 Billion Triple Challenge data set with a geographic LIDS. We relate our approach to existing work in Section 7 and conclude with Section 8.

2 Preliminaries

In the following we shortly present the basics for our work, namely: data services, and RDF.

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

2.1 Data Services

Our notion of data services is as follows:

> Data services return data dynamically derived (i.e., during service call time) from supplied input parameters. Data services neither alter the state of some entity nor modify data. In other words, data services are free of any side effects. They can be seen as data sources providing information about some entity, when given input in the form of a set of name/value pairs. The notion of data services include Web APIs and REST-based services providing output data in XML or JSON.

Data services are related to Web forms or the "Deep Web" [13], but take and provide data rather than free text or documents. For example, the GeoNames `findNearbyWikipedia` service relates given latitude/longitude parameters to Wikipedia articles describing geographical features that are nearby.

Table 1. Example data-providing services

API	Format	Description
GeoNames	XML, JSON	Functions include besides others: (i) find the nearest GeoNames feature to a given point and (ii) link a geographic point to resources from DBpedia that are nearby URI: `http://www.geonames.org/`
Google GeoCoding API	XML, JSON	Provides latitude and longitude for a given street address. URI: `http://code.google.com/apis/maps/`
Twitter API	XML, JSON, RSS, Atom	Various functions, giving access to Twitter users, follower networks, and tweets. URI: `http://dev.twitter.com/`

Example 1. In Table 1, we list some popular data-providing services. Taking the Google GeoCoding API, to get the geographical coordinates for Karlsruhe, we retrieve the URI `http://maps.googleapis.com/maps/api/geocode/json?address=Karlsruhe&sensor=false`, with the following (abbreviated) result:

```
{ "status": "OK",
  "results": [ {
  ...
  "formatted_address": "Karlsruhe, Germany",
  ...
  "geometry": {
    "location": {
      "lat": 49.0080848,
      "lng": 8.4037563
    },
    ...
  } } ] }
```

Using the retrieved coordinates, we can build the URI for calling the GeoNames service to find Wikipedia articles about things, that are nearby Karlsruhe: `http://ws.geonames.org/findNearbyWikipedia?lat=49.0080848&lng=8.4037563`. The (abbreviated) result is the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
  <entry>
    <lang>en</lang>
    <title>Federal Constitutional Court of Germany</title>
    ...
    <lat>49.0125</lat>
    <lng>8.4018</lng>
    <wikipediaUrl>...</wikipediaUrl>
    ...
  </entry>
  <entry>
    ...
  </entry>
</geonames>
```

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

– different serialisation 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 by the GeoCoding 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.

2.2 RDF and Basic Graph Patterns

In contrast to XML or JSON, the Resource Description Framework (RDF) is a graph-based data format which allows for easy integration of data from multiple sources. We now introduce basic RDF notions later reused in the paper; cf. [6].

Let U, B, L, V be disjoint infinite sets of URIs, blank nodes, literals and variables.

Definition 1. *(Triple) A triple $t = (s, p, o)$ is a tuple of length three, $t \in (U \cup B) \times U \times (U \cup B \cup L)$. We often write t as* `s p o`*, where* `s` *is called the subject,* `p` *the predicate and* `o` *the object.*

Definition 2. *(RDF Graph) An RDF graph r is a finite set of triples.*

We often write a set of triples by separating triples by `.` (a dot). To be able to query graphs, we introduce the notion of triple pattern which can include variables.

Definition 3. *(Triple Pattern) A triple pattern* $t \in (U \cup B \cup V) \times (U \cup V) \times (U \cup B \cup L \cup V)$ *abstracts from single triples by allowing variables in every position.*

Definition 4. *(Basic Graph Pattern (BGP) and Conjunctive Query (CQ)) A BGP is a finite set of triple patterns. 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 BGP* T*.*

Let M be the set of all function $\mu : U \cup L \cup V \rightarrow U \cup L$, s.t. μ is the identity for constants, i.e. $\forall a : (a \in U \cup L \rightarrow \mu(a) = a)$. As an abbreviation we also apply a function $\mu \in M$ to a triple pattern $t = p(t_1, \ldots, t_n)$ $(\mu(t) = p(\mu(t_1), \ldots, \mu(t_n)))$, and to a BGP 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 RDF graph* r*, if* $\mu(T) \subseteq r$*. We denote the set of all mappings for a CQ and a graph as* $\mathcal{M}_{CQ}(r) = \{\mu \in M | \mu(T) \subseteq r\}$*.*

3 Linked Data Services

Linked Data Services provide a Linked Data interface for data 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;
- the description must be returned in RDF format.

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 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 an 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 characterise a LIDS by:

– Linked Data Service endpoint: ep, an HTTP URI.
– Local identifier i for the input entity of the service.
– Inputs X_i: names of parameters.

The URI of a service call for a parameter assignment μ (mapping X_i to corresponding values) is constructed in the following way (where addition is understood as string concatenation and subtraction removes the corresponding suffix if it matches):

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

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

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

Please note that the above definition coincides with typical Linked Data URIs. The input entity described by the output of a service call is defined as $inp(ep, X_i, \mu, i) = uri(ep, X_i, \mu) + \text{"\#"} + i$.

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

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

For a binding $\mu = \{lat \mapsto 49.01, lng \mapsto 8.41\}$ the URI for the service call is `gw:findNearbyWikipedia?lat=49.01&lng=8.41` and returns the following description:

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
gw:findNearbyWikipedia?lat=49.01&lng=8.41#point
    foaf:based_near dbpedia:University_of_Karlsruhe_%28TH%29;
    foaf:based_near dbpedia:Federal_Constitutional_Court_of_Germany;
    foaf:based_near dbpedia:Federal_Court_of_Justice_of_Germany;
    foaf:based_near dbpedia:Wildparkstadion;
    foaf:based_near dbpedia:Karlsruhe.
```

4 Describing Linked Data Services

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

[3] `http://km.aifb.kit.edu/services/geowrap/`, abbreviated as `gw`. All other prefixes can be looked up at `http://prefix.cc/`

Definition 6. *(LIDS Description) A LIDS description consists of a tuple* (ep, CQ_i, T_o, i) *where ep denotes the LIDS endpoint,* $CQ_i = (X_i, T_i)$ *a conjunctive query to specify 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.*

The meaning of *ep* and X_i were already explained in the previous section. We define X_i to be the head 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:

– $\mu \in M$ is a valid input, if $\mu \in M_{CQ_i}(r)$, where r is the implicit RDF graph given by all Linked Data;
– for a valid μ, resolving $uri(ep, X_i, \mu)$ returns a graph $D_o \supseteq \{T' \subseteq D_{impl} \mid \exists \mu \in M : \mu(i) = E_s \wedge \mu(T_o) = T'\}$, where D_{impl} is the implicit, potentially infinite data set representing the information provided by the LIDS.

Example 4. We describe the `findNearbyWikipedia` openlids.org wrapper service as (ep, CQ_i, T_o, i) with:
$ep =$ `gw:findNearbyWikipedia`
$CQ_i =$ ({lat,lng}, { `?point geo:lat ?lat . ?point geo:long ?lng` })
$T_o =$ {`?point foaf:based_near ?feature`}
$i = point$

4.1 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 [5]. 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 (ep, CQ_i, T_o, i), we can construct the LaV description:

$$ep(\$I_1, \ldots, \$I_k, O_1 \ldots, O_m) \;\texttt{:-}\; p^i_1(\ldots), \ldots, p^i_n(\ldots), p^o_1(\ldots), \ldots, p^o_l(\ldots).$$

Where $CQ_i = (X_i, T_i)$, $X_i = \{I_1, \ldots, I_k\}$, $T_i = \{(s^i_1, p^i_1, o^i_1), \ldots, (s^i_n, p^i_n, o^i_n)\}$, $T_o = \{(s^o_1, p^o_1, o^o_1), \ldots, (s^o_l, p^o_l, o^o_l)\}$, 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, not 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 5.

4.2 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 corresponding to *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 $(ep, (X_i, T_i), T_o, i)$ 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/geowrap/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 future, we expect a standardised RDF representation of SPARQL, which does not rely on string encoding of basic graph patterns. One such candidate is the SPIN SPARQL Syntax[4], which is part of the SPARQL Inferencing Notation (SPIN)[5]. We are planning to reuse such a standardised RDF representation of basic graph patterns and variables in future versions of the LIDS description model.

[4] http://spinrdf.org/sp.html
[5] http://spinrdf.org/

5 Algorithm for Interlinking Data with LIDS

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

- 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.

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

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

The obtained equivalences can be either used to immediately resolve the LIDS URIs and add the data to r, or to make the equivalences explicit in r, for example, by adding the following triples to r:

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

Based on the services shown in Figure 1 together with descriptions, we illustrate the algorithm using the following example: consider as starting point an entity URI (e.g., an entity `#aifb`), which, when visited, returns an RDF graph with latitude and longitude properties:

```
#aifb
    rdfs:label "AIFB - Building 11.40";
    geo:lat "49.01";
    geo:long "8.41".
```

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 `#aifb` and the service entity:

```
#aifb owl:sameAs
      gw:findWikipediaNearby?lat=49.01&long=8.41#point.
```

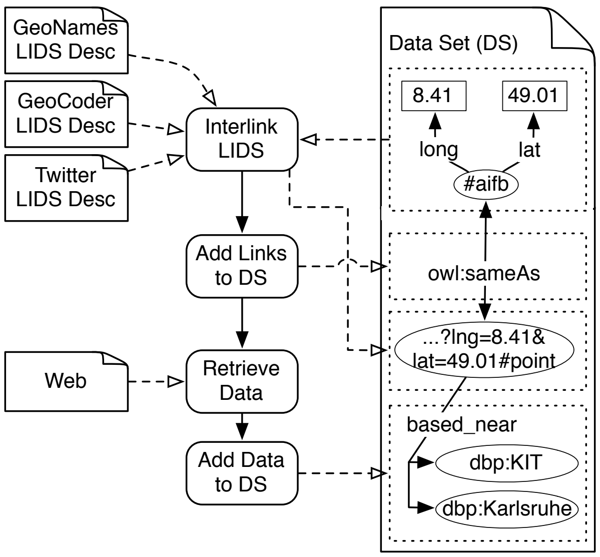

Fig. 1. Interlinking example for GeoNames LIDS

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=49.01&long=8.41#point
       foaf:based_near foaf:based_near dbpedia:Wildparkstadion;
       foaf:based_near dbpedia:Karlsruhe.
...
```

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 `#aifb`. 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.

6 Evaluation of Performance and Effectiveness

We first present several LIDS services which we have made available, and then cover the evaluation of performance and effectiveness of the presented algorithm. 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] `http://code.google.com/p/openlids/`

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 existing services.

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.

- GeoNames Wrapper[7] 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[8] 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 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. 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.

6.2 Interlinking Existing Data Sets with LIDS

We implemented a streaming version of the interlinking algorithm shown in Section 5 based on NxParser[9]. For evaluation of the algorithm's performance and effectiveness we interlinked the Billion Triple Challenge (BTC) 2010 data set[10] 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

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

added[11]. 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.

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]. Other state-of-the-art data integration systems [18] 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 interface generation and service integration into existing data sets.

SILK [19] 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 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 construction of URIs 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 [11] and WSMO [14] 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. 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. 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., [8,7,20]). Similar to our approach these solutions define service functionality in terms of input and output conditions. Most of them, except [8], employ proprietary description formalisms. In contrast, our approach relies on standard SPARQL. Moreover, 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.

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

Norton and Krummenacher propose an alternative approach to integrate Linked Data and services, so-called Linked Open Services (LOS) [12]. LOS descriptions also use basic graph patterns for defining service inputs and outputs. One difference to our work is that LOS consume RDF instead of name-value pairs. With the LIDS approach, service calls are directly linkable from within Linked Data, as service inputs are encoded in the query string of a URI.

Other related work to integrating data comes from the database community, specifically information integration. Mediator systems (e.g., Information Manifold [10]) are able to answer queries over heterogeneous data sources, including services on the Web. Information-providing data services were explicitly treated, e.g., in [17,1]. For an extensive overview of query answering in information integration systems, we refer the reader to [5]. 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 in contrast to our proposed approach that allows data sets to be directly interlinked, independent of a query processor.

8 Conclusions

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 specialised 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. A possible avenue for future work would be to integrate fuzzy matching algorithms, similar to [19], in case the input to a web service is ambiguous, e.g., for services which take keywords as input.

We further plan future work in three main areas:

– improve tool support, so that Semantic Web developers can easily adopt the LIDS method for their applications and services;

– develop approaches for integrating LIDS into SPARQL query processing;
– integrate provenance information and usage policies in the service descriptions, in order to ensure legal compliance and traceability of integrated data sets.

Acknowledgements

The authors acknowledge the support of the European Community's Seventh Framework Programme FP7/2007-2013 (PlanetData, Grant 257641) and of the Deutsche Forschungsgemeinschaft (Information Management and Engineering Graduate School, GRK 895).

References

1. Barhamgi, M., Champin, P.-A., Benslimane, D.: A Framework for Web Services-Based Query Rewriting and Resolution in Loosely Coupled Information Systems (2007)
2. Berners-Lee, T.: Linked Data. Design Issues (2009), `http://www.w3.org/DesignIssues/LinkedData`
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 (2007)
4. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. 2, 115–150 (2002)
5. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal 10, 270–294 (2001)
6. Hayes, P.: RDF Semantics. W3C Recommendation (February 2004), `http://www.w3.org/TR/rdf-mt/`
7. Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., Stevens, R.: Deciding Semantic Matching of Stateless Services. In: AAAI Conference on Artificial Intelligence, AAAI (2006)
8. 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)
9. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Rec. (February 2004), `http://www.w3.org/TR/rdf-concepts/`
10. 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)
11. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services (2004), `http://www.w3.org/Submission/OWL-S/`
12. Norton, B., Krummenacher, R.: Consuming dynamic linked data. In: First International Workshop on Consuming Linked Data, COLD 2010 (2010)
13. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: International Conference on Very Large Data Bases (VLDB), pp. 129–138 (2001)

14. 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), 77–106 (2005)
15. Speiser, S., Harth, A.: Taking the LIDS off Data Silos. In: Triplification Challenge at I-SEMANTICS (2010)
16. Speiser, S., Harth, A.: Towards Linked Data Services. In: The Semantic Web - Posters and Demonstrations, ISWC (2010)
17. 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 (2004)
18. Troncy, R., Fialho, A., Hardman, L., Saathoff, C.: Experiencing events through user-generated media. In: First International Workshop on Consuming Linked Data, COLD 2010 (2010)
19. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 650–665. Springer, Heidelberg (2009)
20. Zhao, W.-F., Chen, J.-L.: Toward Automatic Discovery and Invocation of Information-Providing Web Services. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 474–480. Springer, Heidelberg (2006)