Integrating Linked Data and Services with Linked Data Services

Sebastian Speiser and Andreas Harth

ESWC 2011 – Heraklion, Crete – 02.06.2011

Institute of Applied Informatics and Formal Description Methods (AIFB)

Outline

- **Motivation**
- Exposing Services as Linked Data
- Describing LIDS
- Interlinking Data Sets with LIDS
- Conclusions

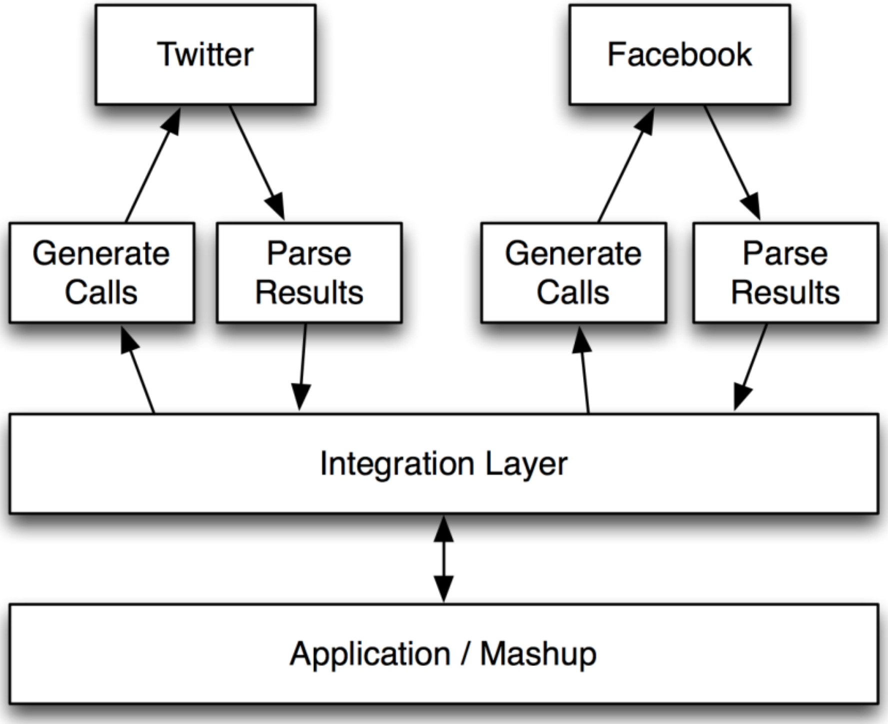
Motivation

- Linked Data (LD) makes a lot of data available in the Web
- Applications typically rely on data from different sources
- With LD, integration of data sources is easy:

  -
    ```
    <http://data.semanticweb.org/conference/eswc/2010>
              foaf:based_near
                        <http://dbpedia.org/resource/Heraklion>
    ```
- There are lot of applications and mashups on the net which do not have this comfort because they rely on Web APIs
  - Typically based on JSON or XML retrieved through a custom URI scheme
    Out of 3274 APIs from ProgrammableWeb, only 37 based on RDF
  - Typically not interlinked
  - Typically the information that users want: Tweets, Facebook friends, eBay auctions, Flickr images and YouTube Videos

Motivation – Example

- Example: Facebook and Twitter API
  - ProgrammableWeb lists 92 mashups using Twitter and Facebook
  - For both APIs: code to generate API call; code to parse JSON results into application’s data model
  - Integration layer that connects information from both sources



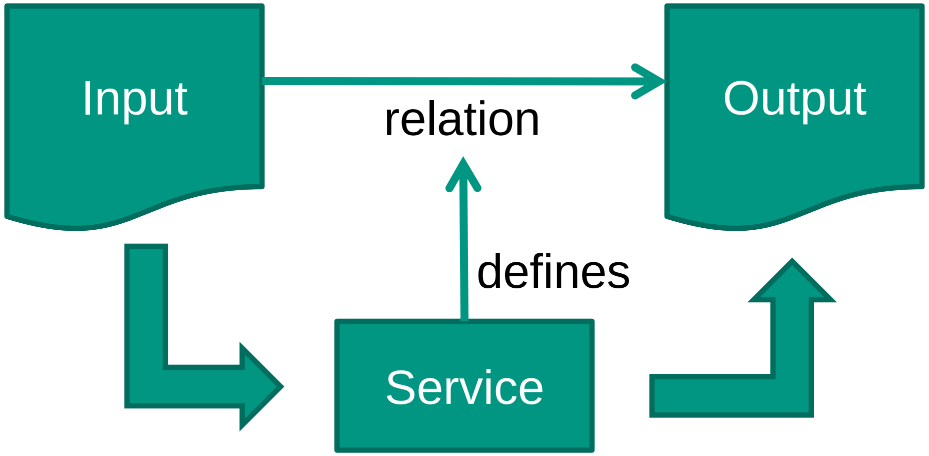
→ 92 times gluing code!

Motivation

- Not all data sources will be published as fully materialised data sets
- Reasons include:
  - Data is changing constantly (e.g. sensor data or stock quotes)
  - Data is calculated based on infinitely many inputs (e.g. route between two geographical locations)
  - Provider does not want arbitrary access (e.g. flight ticket prices, social networks)
- LIDS: Method to publish information services as Linked Data
  - SPARQL patterns describe input and output
  - URIs for service calls can be automatically created
  - Service calls can be directly interlinked with other data

Data Services

- Given input, provide output
- Input and output are related in a service-specific way
- Do not change the state of the world



- E.g. GeoNames findNearbyWikipedia service
  - Input: lat/lon
  - Output: places
  - Relation: output places that are *nearby* input place

Enter LIDS: Linked Data Services

- We’d like to integrate data services with Linked Data

1. LIDS need to adhere to Linked Data principles

- We’d like to use data services in software programs

2. LIDS need machine-readable descriptions of input and output

Outline

- Motivation
- **Exposing Services as Linked Data**
- Describing LIDS
- Interlinking Data Sets with LIDS
- Conclusions

1. Data Services as Linked Data

- Input is given as URI

Service Endpoint

```
http://geowrap.openlids.org/findNearbyWikipedia
?lat=37.416&lng=-122.152
#point
```

Parameters

Input Identifier

- Resolving the URI yields RDF:

Output

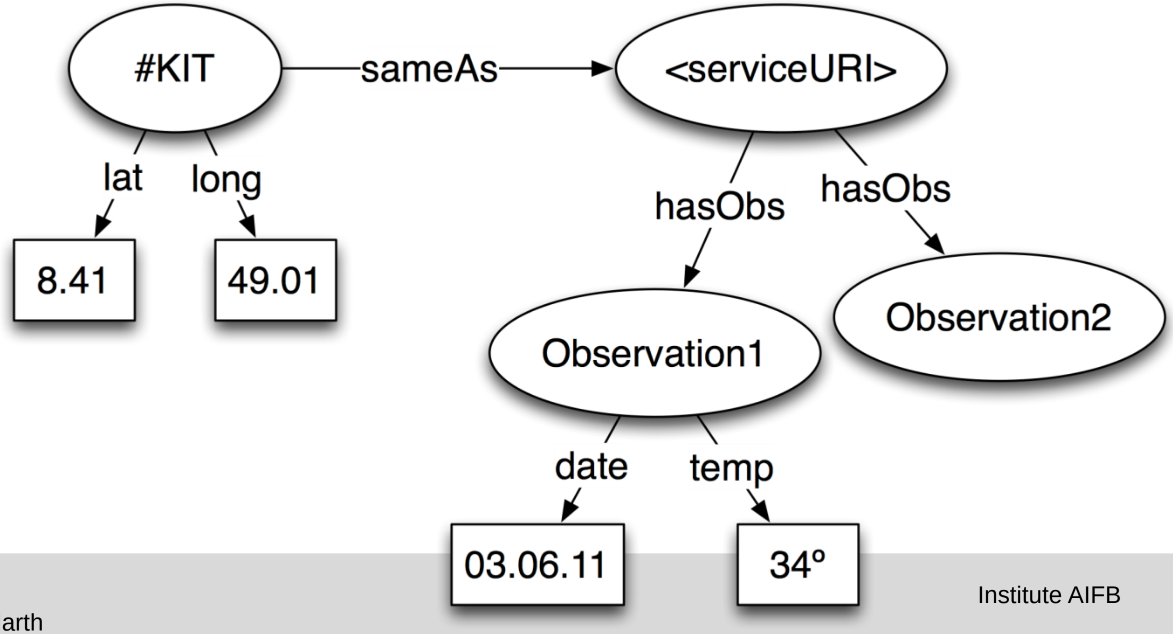
Input

Relation

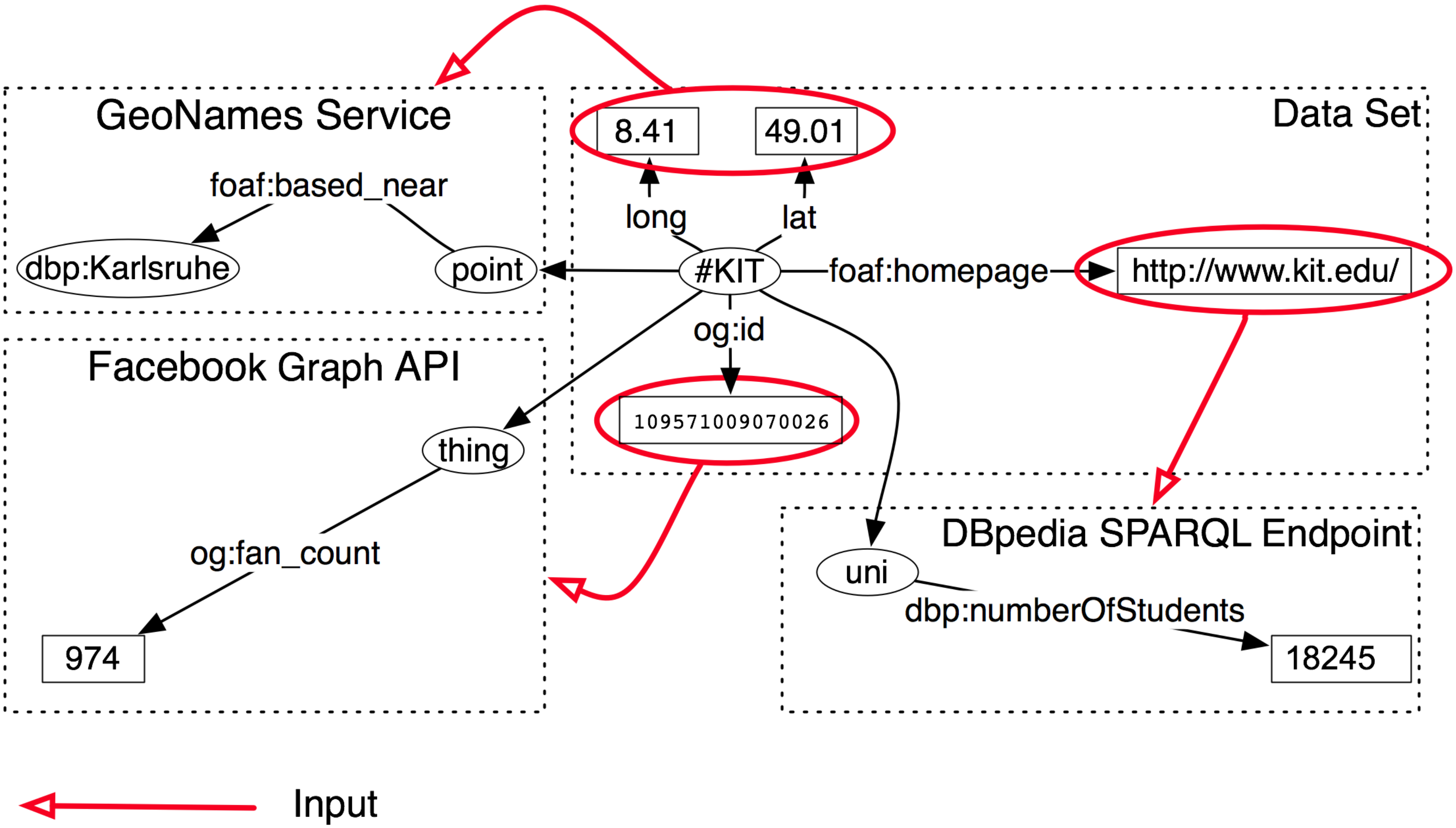
```
@prefix dbp: <http://dbpedia.org/resource/> .
@prefix : <http://geo..Wiki?lat=37.416&lng=-122.152#>
:point
      foaf:based_near dbp:Palo_Alto%2C_California ;
      foaf:based_near dbp:Packard%27s_garage .
```

LIDS Convention Advantages

- Why not just assign URIs to the service outputs?
- Relationship between input and output is explicitly described
- Dynamicity is supported
  - Description can relate input element to current weather resource
  - Older descriptions don’t get wrong / outdated
- Multiple or no output resources can be linked to input
  - E.g. multiple places can be near input



Interlinking Data with Data from Services



Outline

- Motivation
- Exposing Services as Linked Data
- **Describing LIDS**
- Interlinking Data Sets with LIDS
- Conclusions

2. LIDS Descriptions

- LIDS characterised by
  - Endpoint URI ep, which is the base for all input entities
  - Local identifier i of input entity
  - List of parameters X_i
  - Basic graph pattern T_i describing conditions on parameters
  - Basic graph pattern T_o describing minimum output data

- Example:

  ep = **`<http:/geowrap.openlids.org/findNearbyWikipedia>`**
  `i = point`
  X_i `= {?lat, ?lng}`
  T_i `= ?point a Point . ?point geo:lat ?lat .`
  `                             ?point geo:long ?lng`
  T_o `= ?point foaf:based_near ?feature`

Comparing LIDS Descriptions and LaV

- Local as View (LaV): widely used approach for data integration
- `ep($i1,…,$in,o1,…,ok) -> p1(…), …, pm(…) .`
- Source/Service returns tuples
- LaV describes how tuple elements are to be interpreted
- LaV also describes preconditions on inputs ($ vars)

- Example:

```
findNearbyWikipedia(?p,$lat,$lng,?feat) ->
    Point(?p), geo:lat(?p,$lat), geo:long(?p,$lng),
    foaf:based_near(?p,?feat) .
```

- Tuples given $lat = 49.01 and $lng = 8.41:

```
findNearbyWiki(...?lat=49.01&lng=8.41#point,49.01,8.41,dbp:KIT)
findNearbyWiki(...?lat=49.01&lng=8.41#point,49.01,8.41,dbp:Karlsruhe)
```

- Tuples have to be interpreted according to LaV definition

Comparing LIDS Descriptions and LaV

- LIDS:
  - return triples → don’t have to be interpreted
  - Descriptions are easier to understand (separation of input and output)
  - Descriptions are easier to use for algorithms
  - Descriptions can be transformed into/from LaV descriptions

- LaV:
  List of tuples {(val1,val2,val3,…),(val1’,val2’,val3’,…),…}
  and definition how to interpret them in target schema
- LIDS:
  Result directly returned in global schema, i.e., as RDF graph

Generating Links for Service Calls

- For a binding µ of input parameters ($\mathtt{X}_{\mathtt{i}}$) to values, we construct

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

- If only one parameter (x) exists, we also allow

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

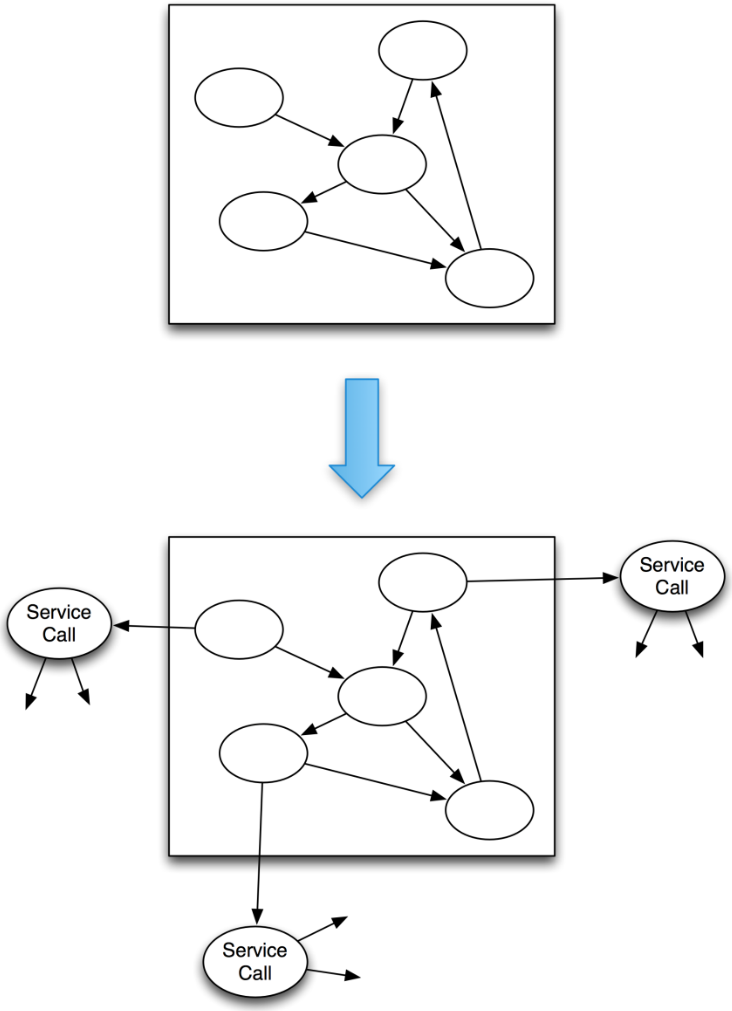
- The input entity is constructed as following

$$uri(ep, X_i, \mu) + {}''\text{\#}'' + i.$$

Outline

- Motivation
- Exposing Services as Linked Data
- Describing LIDS
- **Interlinking Data Sets with LIDS**
- Conclusions

Interlinking Data Sets with LIDS



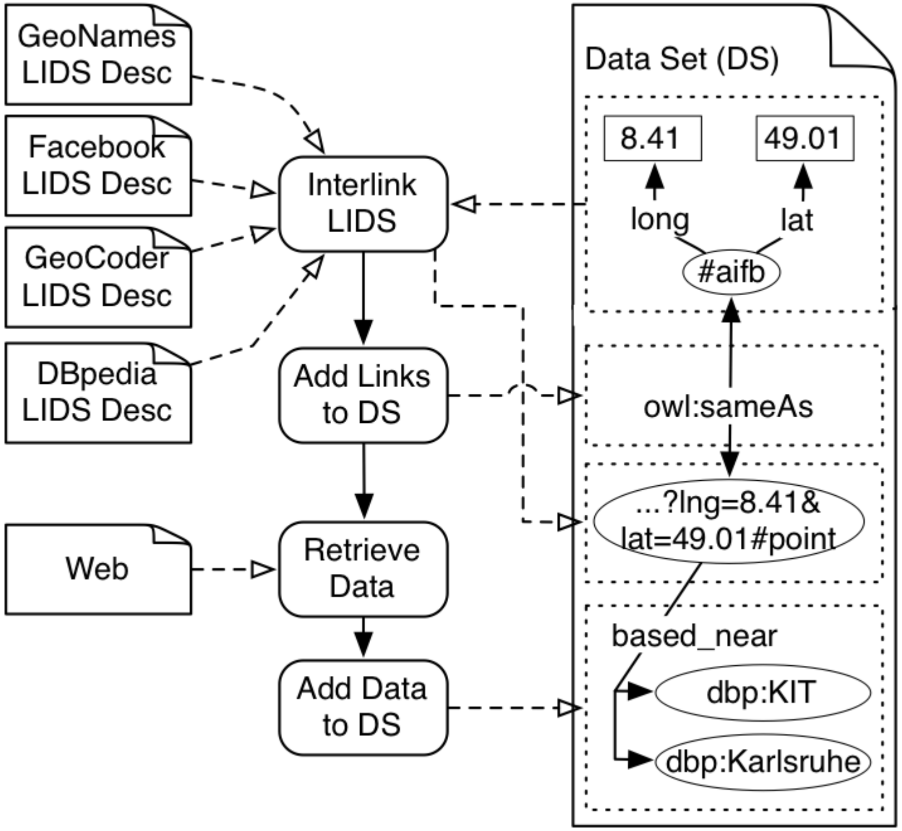
Use Cases:

- Processing of static data set, using new interlinked set for further applications
- Linked Data endpoint, enriching data before returning to client (server-side)
- Linked Data browser, enriching data after retrieving it from server (client-side)

Interlinking Data Sets with LIDS

- Given `ep, i,` X_i, T_i from a service description and a data set D
- Evaluate `select i,` X_i `where` T_i on D
- Result: set of bindings M
- For each µ in M:
  - Equivalence of µ(i) and `inp(ep,` X_i`, µ, i)`
  - → add `owl:sameAs` link between
    - binding for `i` and
    - input entity of service call

Interlink LIDS and Linked Data



-
  ```
  SELECT ?point ?lng ?lat
  WHERE {
    ?point geo:long ?lng;
           geo:lat  ?lat }
  ```

-
  ```
  μ = {?point -> #aifb,
       ?lng -> 8.41,
       ?lat -> 49.01}
  ```

-
  ```
  inp(…,{?lng,?lat},μ ,point) =
  ...?lng=8.41&lng=49.01#point
  ```

Scale-Up Experiment: Link BTC to GeoNames

- 3 billion triples from the Billion Triple Challenge (BTC) 2010 data set:
- Annotate with LIDS wrapper of GeoNames findNearby service
- Annotation time: < 12 hours on laptop!
- ~ 12 hours for uncompressing the data set, cleaning results, and gather statistics

- Original BTC data: 74 different domains that linked to GeoNames URIs
- Interlinking process added 891 new now linked to LIDS geowrap
- In total 2,448,160 new links were added

Outline

- Motivation
- Exposing Services as Linked Data
- Describing LIDS
- Interlinking Data Sets with LIDS
- **Conclusions**

Conclusion

- LIDS provide convention to expose Data Services as Linked Data
- Descriptions based on RDF and SPARQL patterns

- LIDS useful for
  - Inserting links to LIDS into static RDF data sets
  - Linked Data endpoints that dynamically add links from their data to LIDS
  - LD browsers that augment retrieved data with data retrieved from LIDS
  - Integrating LIDS into SPARQL query processing

http://openlids.org/