LS4LS
Linked Services for Life Science

Sebastian Speiser, KIT, DE
Ruben Verborgh, Ghent University, BE
Linked APIs for Life Science

Linked APIs is the joint effort of independently developed approaches to align Linked Data and “functionalities”
Agenda

- Motivation
- Linked Data
- REST APIs
- Linked APIs
- Describing APIs
- Working with Linked APIs
- Conclusions
Motivating example

• Find potential drugs for disease
  – Get disease and associated genes
  – Get proteins produced by the genes
  – Find similar proteins for which drugs exist
  – Output drugs
Motivating example: sources

• Diseases and associated genes:
• Proteins produced by the genes
• Finding similar proteins
  – E.g., UniProt BLAST service
• Findings drugs targeting given proteins
• Good news: all sources available
Automation by workflows

• Integrating the sources and services by hand?
• No problem for one time procedure: just copy/paste from/to Web sites
• But: diseases affect multiple genes, drugs target multiple proteins, ...
• Workflows formally specify the procedure using machine-accessible services
• Repeatable, documented, automated
The details of Workflows

• Powerful tools with nice UI available, e.g. Taverna

• But: a lot of power wasted for parsing, selecting and transforming data to make services interoperable

• “Gluing code” written again and again
RDF for integration

• Use common data model for data and services
• Take RDF!
• Why not XML, JSON, CSV, ... ?

1. Data and vocabulary terms identified by URIs
2. Graph structure
3. Extendable vocabulary
4. Support for data and vocabulary alignment
   (subClassOf, subPropertyOf, sameAs)
RDF for integration

• Graph merging is easy

```
\[ \text{disease:D1} \quad \text{associatedGene} \quad \text{gene:G1} \]
\[ \text{prot:P1} \quad \text{produces} \quad \text{gene:G1} \]
```
RDF for integration

- Graph merging is easy
Where should the data live?

- Many Life Science datasets available in RDF
- How to work with the data?

Possibility 1: load all datasets into a warehouse
- Many different redundant copies of data exist
- Every copy has to be kept up-to-date
- Lot of storage required even for simple problems
- Not every provider wants to give away all his data
Where should the data live?

- Many Life Science datasets available in RDF
- How to work with the data?
  Possibility 2: access live data over the Web
- Data is always up-to-date
- Only required data is taken
- Pay-as-you-go data integration
- Providers keeps control of their data

Linked Data as a standardized way to access data
Agenda

- Motivation
- **Linked Data**
- REST APIs
- Linked APIs
- Describing APIs
- Working with Linked APIs
- Conclusions
Linked Data

• Data resides at provider/owner
• Access is given in small units
• Standard way to access data given its ID (URI)
• Links to other datasets containing further relevant information

• Lots of data available: Linked Open Data
Linked Open Data

http://linkeddata.org/
Linked Open Data – Life Sciences

http://linkeddata.org/
Linked Data Principles

1. Use **URIs to name things**; not only documents, but also people, locations, concepts, etc.

2. To enable agents (human users and machine agents alike) to look up those names, use **HTTP URIs**

3. When someone looks up a URI we **provide useful information**; with 'useful' in the strict sense we usually mean structured data in RDF.

4. Include **links to other URIs** allowing agents (machines and humans) to **discover more things**

http://www.w3.org/DesignIssues/LinkedData
Principles 1&2: name things with HTTP URIs

- **URIs for information resources:**
  - http://www.uniprot.org
  - http://speiserweb.de/sebastian/foaf.rdf

- **URIs for instances:**
  - http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/1021

- **Prefixes for more readability:**
  - PREFIX diseasesome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/>

- **URIs for classes:**
  - diseasesome:diseases

- **URIs for properties:**
  - diseasesome:associatedGene
Principle 3: provide RDF when URI is dereferenced

• Information resources are documents which can be retrieved directly

• Non-information resources either
  – use local names, e.g.:
    http://www.w3.org/2000/01/rdf-schema#label, or
  – use HTTP redirects to information resources, e.g.:
    diseasesome:diseases/1021

• Looking up an URI results in a RDF description of the identified thing
Principle 3: provide RDF when URI is dereferenced

Example:

HTTP GET on
http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/1021

Redirect to: http://.../data/diseases/1021 containing:
PREFIX diseasome: <http://.../diseasome/>
diseasome:diseases/1021
diseasome:associatedGene
diseasome:genes/ACE.
diseasome:diseases/1012 rdfs:label
“SARS, progression of”.
Principle 4:
include links to other URIs

• Links enable discovery of further relevant data
• E.g.:
  
  \texttt{diseasome:diseases/1021}
  \texttt{diseasome:associatedGene}
  \texttt{diseasome:genes/ACE}

• Retrieving URI of associated gene returns description of gene with further information and links
• Valuable: links crossing dataset boundaries
Missing from Linked Data: APIs

• Not all data sources will be published as fully materialised data sets
• Reasons include:
  – Data is changing constantly (e.g. sensor data)
  – Data is calculated based on infinitely many inputs (e.g. similarity between two gene sequences)
  – Provider does not want arbitrary access (e.g. flight ticket prices, social networks)
Agenda

• Motivation
• Linked Data
• REST APIs
• Linked APIs
• Describing APIs
• Working with Linked APIs
• Conclusions
HTML and HTTP drive the human Web.
HTML and HTTP drive the human Web.
<h1>Genes associated with Parkinson's</h1>
<p>Associated with Parkinson's:</p>
<ol>
  <li><a href="/genes/47">Gene A</a></li>
  <li><a href="/genes/80">Gene B</a></li>
  <li><a href="/genes/21">Gene C</a></li>
</ol>
<h1>Genes associated with Parkinson’s</h1>
<p>Associated with Parkinson’s:</p>
<ol>
  <li><a href="/genes/47">Gene A</a></li>
  <li><a href="/genes/80">Gene B</a></li>
  <li><a href="/genes/21">Gene C</a></li>
</ol>
<h1>Genes associated with Parkinson’s</h1>
<p>Associated with Parkinson’s:</p>
<ol>
  <li><a href="/genes/47">Gene A</a></li>
  <li><a href="/genes/80">Gene B</a></li>
  <li><a href="/genes/21">Gene C</a></li>
</ol>
Genes associated with Parkinson's:

- Gene A
- Gene B
- Gene C
HTML and HTTP drive the human Web.
HyperText Transfer Protocol

• Simple model
  – resources

• Limited actions
  – GET, POST, PUT, DELETE

• Extensible headers
  – Accept
HTTP works with messages.

GET /genes HTTP/1.1

HTTP/1.1 200 OK
HTTP works with messages.

POST /genes HTTP/1.1

HTTP/1.1 200 OK
This is the Web for humans.
This does not work for machines.

GET /genes HTTP/1.1

HTTP/1.1 200 OK
“Simple” Object Access Protocol

POST /GeneService.php HTTP/1.1

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetGenes xmlns:m="http://www.ex.org/genes" />
  </soap:Body>
</soap:Envelope>
Let’s try things the “simple” way.

GET /genes HTTP/1.1

HTTP/1.1 200 OK
SOAP is not the Web for machines.

- SOAP is not about resources.
  - uses HTTP as a tunneling protocol
- SOAP is programming over the Web.
  - basically doing site.getGenes()
  - but HTTP already has verbs
So suddenly, there were two Webs.
A JavaScript revolution came.

• Dynamic pages require JavaScript.
• JavaScript uses JSON.
JSON has a tree structure.

```json
{
    "disease": "Parkinson's",
    "associatedGenes": [
        { "id": 47, "name": "A" },
        { "id": 80, "name": "B" },
        { "id": 21, "name": "C" }
    ]
}
```
People thought they needed APIs.

• Idea: get things in JSON.
  – /api/getGenes.php?diseasId=32&format=json
  – /api/listDeseases.php?beginLetter=P&format=json

• Again, this is not resource-oriented.
  – Again, it is stuff for programmers.
JSON APIs have serious drawbacks.

- Every API is different.
- No semantics inside the data.
- Client need to be programmed specifically.
There are still two Webs.
And it doesn't end there.

• What if we also need semantics?
• No problem, just create a new API.
  – /diseases/parkinsons/associated-genes.html
  – /api/getGenes.php?diseaseId=32&format=json
  – /api/getGenes.php?diseaseId=32&format=rdf
Now there are three Webs.

And whose are they anyway?
We need only one Web.
Every new service claims to be REST. 

Raise your hand if you’ve heard about any of:

– REST.
– RESTful API.
– More RESTful API.
– RESTful Web service.
– Representational State Transfer.
– RESTful practices.
– This is not REST.
– This is REST.
– REST!!!
REST is just proper HTTP.
Representational State Transfer

- Resources are the essential elements.
- Resources are *your* building blocks:
  - genes /genes/37
  - diseases /diseases/32
  - proteines /proteines/45
Resources have representations.

• The server cannot serve resources.
  – How would you send a gene across a cable?

• The server sends representations.
  – HTML
  – JSON
  – RDF
  – JPEG
  – ...

• The client negotiates about the representation.
Humans understand text.
Humans understand images.

GET /genes/37 HTTP/1.1
Accept: image/*

HTTP/1.1 200 OK
Content-type: image/jpeg
Machines interpret JSON.

GET /genes/37 HTTP/1.1
Accept: text/json

HTTP/1.1 200 OK
Content-type: text/json
Machines interpret RDF.

GET /genes/37 HTTP/1.1
Accept: text/turtle
HTTP/1.1 200 OK
Content-type: text/turtle
REST is driven by hyperlinks.

- Hypermedia constraint: serve representation and controls.
- Typed hypermedia links.

`/genes/37`

associatedWith `/diseases/parkinsons`
associatedWith `/diseases/dementia`
produces `/proteines/78`
REST

No services.
One resource.
Different representations with hypermedia links.
Agenda

• Motivation
• Linked Data
• REST APIs
• **Linked APIs**
• Describing APIs
• Working with Linked APIs
• Conclusions
What can be Linked in APIs?

Descriptions of APIs can be exposed as Linked Data

- linked from API directory
- linked from schema elements to relevant APIs
  - e.g., property description of gene sequence could link to API
- linked from instance data
  - e.g., description of organism could link to database search API
What can be Linked in APIs?

API calls can be given URI, and be linkable from other data sets, e.g.

- description of person links to API call returning its geographic coordinates
- API description can link to another API call returning the load factor of the hosting server
- list of genes links for every gene to an API call returning list of gene databases that have information about the gene available (two steps)
What can be Linked in APIs?

Output of API can contain links to
• ... other data sets, e.g., diseases to DBpedia concepts
• ... other API calls, e.g., search in other databases
• ... other API descriptions, e.g., specialised API directory

Links help to
discover APIs,
interpret their output, and
integrate with data from others sources.
Agenda

• Motivation
• Linked Data
• REST APIs
• Linked APIs
• Describing APIs
• Working with Linked APIs
• Conclusions
Linked API principles

• Restrict to data APIs (today)
• Describe APIs as Linked Data prosumers
  – No grounding: just communicate RDF
• Make relation between input and output explicit
Data APIs

• Given input, provide output
• API-specific relation btw. input and output

• Example:
  – input gene with sequence, output: list of genes
  – relation: “have similar sequences”
Linked APIs as LD Prosumers

- Linked APIs consume and produce RDF
- Standard way to describe subgraphs for RDF: Basic Graph Patterns (BGPs) as in SPARQL

Example:

- Input:
  ?prot1 lals:proteinSequence ?seq

- Output:
  ?prot1 lals:similar_to ?prot2
Explicit relation between Input and Output

- API invocation can be regarded as rule applications: \([\text{input}] \Rightarrow [\text{output}]\)
- Output contains unbound variables (not occurring in input)
- Unbound variables are bound by API
- At least one bound variable in output makes relation between input and output explicit

Example:

\[
?\text{prot1} \ \text{lals:proteinSequence} \ ?\text{seq} \\
\Rightarrow \ ?\text{prot1} \ \text{lals:similar_to} \ ?\text{prot2}
\]
Describing API Call Construction

• Besides describing input, output and their relation, one needs to know which URIs to call
• Several possibilities:
  – Constant URI to which RDF of input is sent
  – Convention how to construct URI from parameters (bindings to input pattern)
  – Rule-based URI generation
Concrete Linked API approaches

• Several independently developed approaches:
  – Linked Data Services by Sebastian Speiser, Andreas Harth
  – RESTdesc by Ruben Verborgh, Thomas Steiner
  – Linked Open Services by Barry Norton, Reto Krummenacher, Steffen Stadtmüller

• Currently in the process of aligning them
Linked Data Services (LIDS)

• Descriptions:
  – Input and output BGPs
  – Endpoint (base URI)
  – Service entity (a bound variable in the output BGP)
• Convention to create URIs for service calls:
  – Endpoint + parameter bindings + # + service entity
• Parameter binding can be LD URI for further lookup
• Dereferencing service call URI results in description of service entity that relates to output data
Example:
Input:  ?prot lals:proteinSequence ?seq
Output: ?prot lals:similar_to ?prot2
Endpoint: http://localhost:8080/LALS/simProts

Service entity: prot
• Binding for ?seq = “MRVVLAAALTALAART ...”
• Construct service call:
  http://...simProts?seq=MRV...#prot
• Service call URI identifies the protein with the given sequence
• Dereferencing the URI returns description:
  <http://...simProts?seq=MRV...#prot>
    lals:similar_to
    <http://.../protXYZ> .
REST describes hyperlinks.

- REST APIs have hypermedia links.
- Machines should browse the Web like humans.
- What does it mean to follow a link?
  - explain to machines in RDF
  - using if/then construct
A client receives unknown links.

/proteins/53

producedBy /genes/37
producedBy /genes/54
similarTo /proteins/4
similarTo /proteins/6
hasSequenceSimilarTo /sequences/9
RESTdesc describes this link.

protein A hasSequenceSimilarTo sequence B

protein B similarTo protein B

hasSequence
RESTdesc describes this link.

```json
{
  ?proteinA lals:hasSequenceSimilarTo ?sequenceB.
}
=>
{
  _:req http:methodName "GET";
  http:requestURI ?sequenceB;
  http:resp ?sequenceB.

  ?proteinA lals:similarTo _:proteinB.
  _:proteinB lals:hasSequence ?sequenceB.
}
```
If you have this link...

```json
{
  ?proteinA lals:hasSequenceSimilarTo ?sequenceB.
}
=>
{
  _:req http:methodName "GET";
  http:requestURI ?sequenceB;
  http:resp ?sequenceB.

  ?proteinA lals:similarTo _:proteinB.
  _:proteinB lals:hasSequence ?sequenceB.
}
```
...then you can make a request...

{  
  ?proteinA lals:hasSequenceSimilarTo ?sequenceB.
}  
=>
{  
   _:req http:methodName "GET";
   http:requestURI ?sequenceB;
   http:resp ?sequenceB.

   ?proteinA lals:similarTo _:proteinB.
   _:proteinB lals:hasSequence ?sequenceB.
}

...and the response will mean this.

```
{
  ?proteinA lals:hasSequenceSimilarTo ?sequenceB.
}
=>
{
  _:req http:methodName "GET";
  http:requestURI ?sequenceB;
  http:resp ?sequenceB.

  ?proteinA lals:similarTo _:proteinB.
  _:proteinB lals:hasSequence ?sequenceB.
}
```
RESTdesc: towards a shared Web.

- Humans and machines
  - same Web
  - same resources
    - multiple representations
    - various hypermedia links
- Explaining hypermedia links
- http://restdesc.org/
Agenda

• Motivation
• Linked Data
• REST APIs
• Linked APIs
• Describing APIs
• Working with Linked APIs
• Conclusions
Executing a Workflow

- LD and LAPIs produce RDF
- LAPIs consume RDF
- Query for information need consumes RDF
- Execute workflows on triple spaces (similar to blackboard approach)
  - Create an empty triple space (RDF graph)
  - Load triples from LD lookups
  - Send subgraphs to services, add resulting triples to space
  - Evaluate target query over triple space
Executing a Workflow

• Remember example:
  find potential drugs for disease: \texttt{d:1021}

• Information need as SPARQL:

  \begin{verbatim}
  SELECT ?diseaselabel ?druglabel WHERE {
    d:1021 diseasome:associatedGene ?g .
    ?p lals:similar_to ?simprot .
    d:1021 rdfs:label ?diseaselabel }
  \end{verbatim}
Executing a Workflow

• Step 1: find associated genes for d:1021

• Add to triple space: LD lookup for d:1021

  d:1012  rdf:type  diseasome:diseases;
  d:associatedGene  d:genes/ACE;
  d:class  d:Immunological;
  rdfs:label  “SARS, progression of”;  

• Lookup all associated genes, i.e., all \(?g\), s.t.

  SELECT  ?g  WHERE  \{d:1012  d:associatedGene  ?g\}

• i.e., lookup d:genes/ACE and add triples
Executing a Workflow

• Step 2: find proteins produced by genes
• Apply lookup to LAPI wrapping UniProt:
  
  >     ?gene rdfs:label ?name
  >     ?prot lals:sequence ?seq .
  > Endpoint: http://.../LALS/genes
  > Service entity: gene
• Service call: http://.../LALS/genes?name=ACE#gene
• Adds to triple space:

  > PREFIX : <http://.../LALS/genes?name=ACE#> .
  > ...


Executing a Workflow

• Step 3: find similar proteins
• Apply lookup to LAPI searching for similar proteins
  Input: ?prot lals:sequence ?seq
  Endpoint: http://.../LALS/simProt
  Service entity: prot
• Service call: http://.../LALS/simProt?seq=XYZ#prot for every found protein sequence
• Adds to triple space:
  PREFIX : <http://.../LALS/simProt?seq=XYZ#> .
  :prot lals:similar_to drugbank:target143 .
  ...

Executing a Workflow

• Step 4: find drugs for similar proteins
• Output of previous service described found proteins as DrugBank targets (URIs)
• Lookup all target URIs, adds for each target to triple space:
  - drugbank:target143
  - drugbank:proteinSequence "..."
  - drugbank:drugs/DB00492
  - drugbank:target drugbank:target143
• Lookup all drug URIs
Executing a Workflow

• Step 5: execute information need query
• Triple space contains information about disease, associated genes, produced and similar proteins, and applicable drugs
• SELECT ?diseaselabel ?druglabel WHERE ... 
• Results:

<table>
<thead>
<tr>
<th>?diseaselabel</th>
<th>?druglabel</th>
</tr>
</thead>
<tbody>
<tr>
<td>SARS</td>
<td>Deserpidine</td>
</tr>
<tr>
<td>SARS</td>
<td>Captopril</td>
</tr>
<tr>
<td>SARS</td>
<td>Enalapril</td>
</tr>
<tr>
<td>SARS</td>
<td>...</td>
</tr>
</tbody>
</table>
Observations

• No manual connecting of inputs and outputs => automatically matched
• Selecting URIs for lookups could be automated => LD query processors lookup partial bindings
• Vocabulary alignments by rules/ontologies can be integrated
Agenda

• Motivation
• Linked Data
• REST APIs
• Linked APIs
• Describing APIs
• Working with Linked APIs
• Conclusions
Linked APIs the story so far ... 

• REST principles make APIs first-class Web citizens

• Descriptions enable:
  – Discovery of APIs
  – Understanding APIs
  – Integration of APIs into workflows
  – Invocation of APIs
Linked APIs for Life Scientists

• Already a lot of Life Science LD available
• Linked APIs enables dynamic data to blend in
• And more:
  – Functions
  – Real-world actions
  – ...

How you can contribute

• Integrate Linked APIs in your favorite workflow tool
• Publish your API as a Linked API
• Tell us about your use cases

Ruben Verborgh
ruben.verborgh@ugent.be – @RubenVerborgh

Sebastian Speiser
speiser@kit.edu

Mailing list:
linkeddataandservices@googlegroups.com