Web Services, REST

Miroslav Blaško

miroslav.blasko@fel.cvut.cz

November 1, 2017

Contents

1. Web services
2. RESTful web services
3. Linked data

Web services

What is a web service?

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.

— W3C, Web Services Glossary

We can identify two major classes of Web services:

- *REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and*
- *arbitrary Web services, in which the service may expose an arbitrary set of operations.*

— W3C, Web Services Architecture (2004)

Comparison of API protocols and styles (2008-2010)

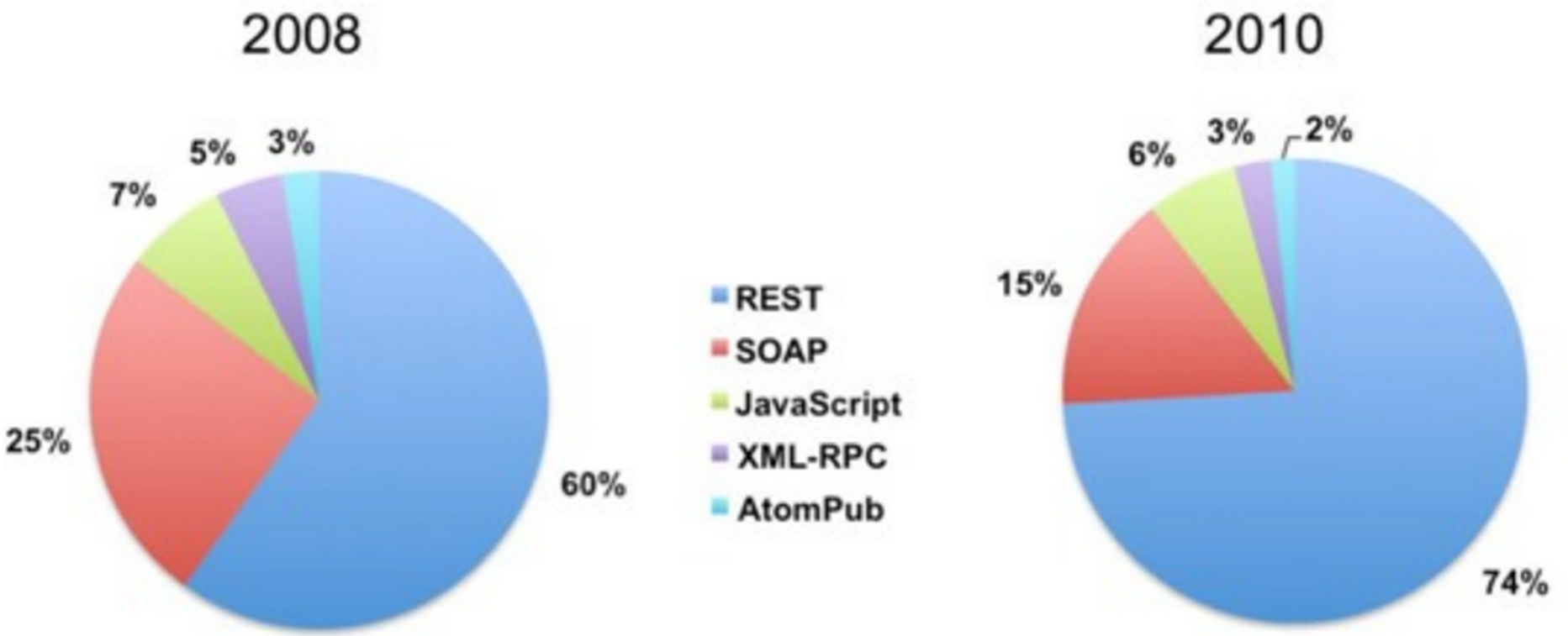


Figure: Distribution of different API protocols and styles based on ProgrammableWeb's directory of more than 2,000 web APIs. Source: `http://royal.pingdom.com/2010/10/15/rest-in-peace-soap/`

Interest over time for major web service APIs

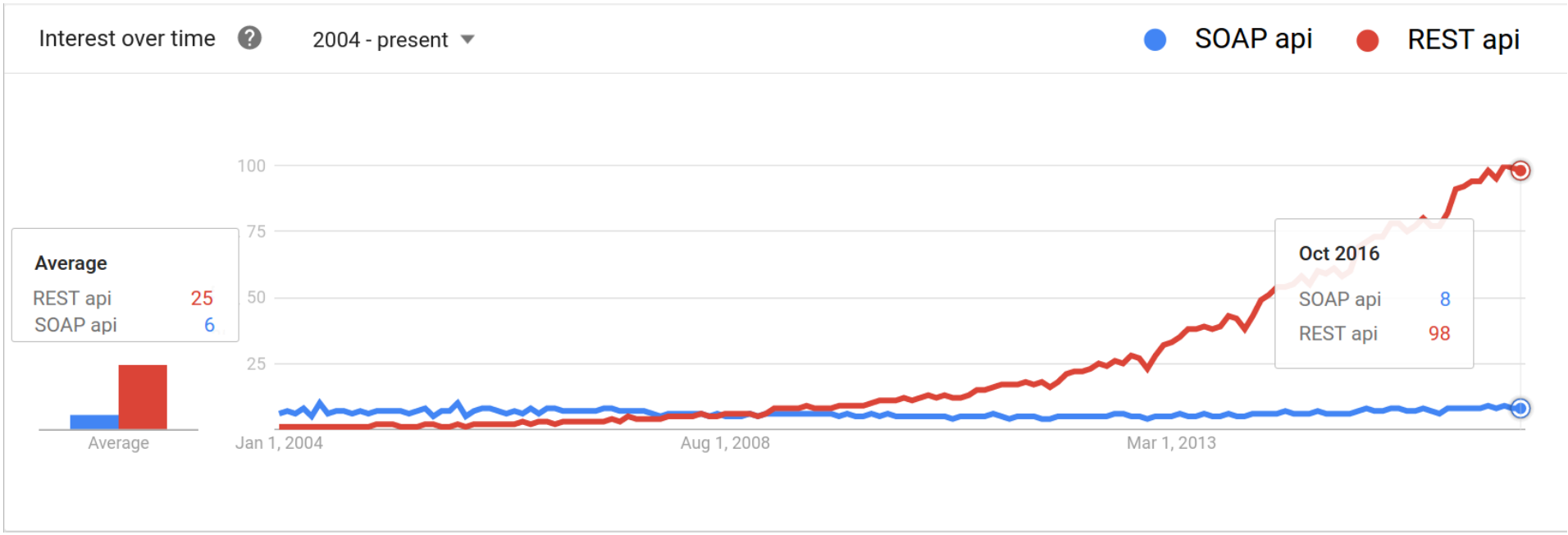


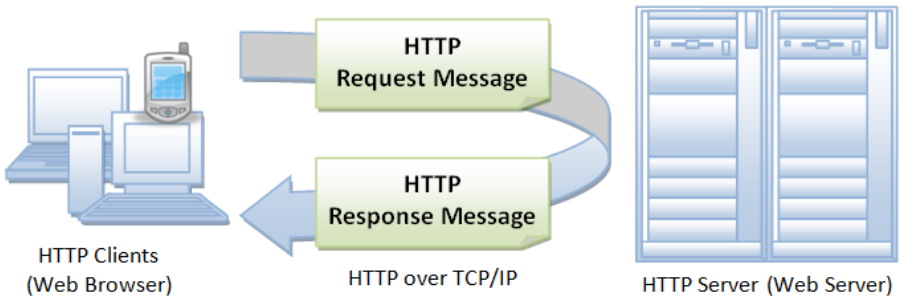
Figure: Interest over time for REST API versus SOAP API based on Google Insights for Search. Source: `https://www.google.com/trends`

RESTful web services

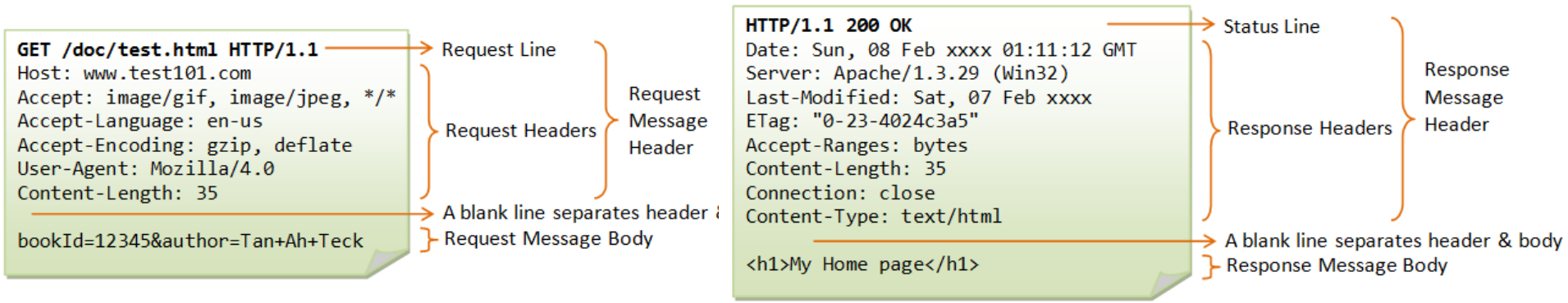
Basic terms

- **Uniform Resource Identifier (URI)** is a string of characters used to identify a resource. (e.g. `http://www.fel.cvut.cz/cz/education/`)
- **The Hypertext Transfer Protocol (HTTP)** is an application *protocol* for distributed, collaborative, hypermedia information systems. It is foundation of data communication for the World Wide Web.
  - initiated by Tim Berners-Lee at CERN in 1989
- **Representational State Transfer (REST)** is *architectural style* for distributed hypermedia systems.
  - defined in 2000 by Roy Fielding in his doctoral dissertation

HTTP protocol basics



- HTTP is a client-server application-level protocol
- typically runs over a TCP/IP connection



Example of HTTP Request Message (left part) and HTTP Response Message(right part). The request accesses URL `http://www.test101.com/doc/test.html?bookId=12345&author=Tan+Ah+Teck`. In addition to the description, the *request line* can be devided into 3 parts: *request method* (i.e. "GET"), *request URI* (i.e. "/doc/test.html") and *HTTP protocol version* (i.e. "HTTP/1.1"). *Request message body* consist of 2 *request parameters* "bookId" and "author". Source: `https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html`

Understanding REST

- REST is architectural style, not standard.
- It was designed for distributed systems to address *architectural properties* such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability.
- REST architectural style is defined by 6 *principles/architectural constraints* (e.g. client-server, stateless).
- System/API that conforms to the constraints of REST can be called *RESTful*.

REST principles

- client-server
- uniform interface
  - resource-based
  - manipulation of resource through representation
  - self-descriptive messages
  - hypermedia as the engine of application state
- stateless interactions
- cacheable
- layered system
- code on demand (optional)

Building RESTful API

- can be build on top of existing web technologies
- reuseing semantics of HTTP 1.1 methods
  - safe and idempotent methods
  - typicaly called HTTP verbs in context of services
  - resource oriented, correspond to CRUD operations
  - satisfies **uniform interface** constraint
- HTTP Headers to describe requests & responses

HTTP verbs – GET

- requests a representation of the specified resource
- should be safe and idempotent
- can have side-effects, but not expected
- can be conditional, or partial (If-Modified-Since, Range)

Example – retrieve user with id 123

GET /users/123

HTTP verbs – POST

- requests to do something with the specified resource
- does not have to be safe or idempotent
- can be used for **create** and **update**

Example – create user

POST /users

```
{ "firstName": "Karel",
  "lastName": "Novak" }
```

HTTP verbs – PUT

- requests to store specified entity at a specified URI
- should be idempotent, but not safe
- can be used for **create** and **update**

Example – update user with id 123

PUT /users/123

```
{ "firstName": "Karel",
  "lastName": "Novak" }
```

HTTP verbs – DELETE

- **deletes** specified resource
- should be idempotent, but not safe
- deletition does not have to be immediate

Example – delete user with id 123

DELETE /users/123

HTTP Status Codes

- classifies the result of the HTTP request
- main categories of status codes, with most common specific codes are
  - **1xx** - informational
  - **2xx** - success
  - **3xx** - redirection
  - **4xx** - client error
  - **5xx** - server error

Common HTTP status codes indicating error

- **4xx** - client error
  - **400 Bad Request** – malformed syntax, retry with modified request
  - **401 Unauthorized** – authentication is required
  - **403 Forbidden** – server has understood, but refuses request
  - **404 Not Found** – server cannot find a resource by specified URI
  - **409 Conflict** – resource conflicts with client request
- **5xx** - server error
  - **500 Internal Server Error** – server encountered an unexpected condition which prevented it from fulfilling the request

Other common HTTP status codes

- **200 OK** – request has succeeded
- **201 Created** – returns a *Location* header for new resource
- **204 No Content** – server fulfilled request but has nothing to return
- **304 Not Modified** – accessed document was not modified thus cache can be used

Recommended interaction of HTTP methods w.r.t. URIs

HTTP Verb	**CRUD**	**Collection (e.g. /users)**	**Specific Item (e.g. /users/{id})**
POST	Create	201 Created*1	404 Not Found/409 Conflict*3
GET	Read	200 OK, list of users	200 OK, single user/404 Not Found*4
PUT	Update/Replace	404 Not Found*2	200 OK/204 No Content/404 Not Found*4
PATCH	Update/Modify	404 Not Found*2	200 OK/204 No Content/404 Not Found*4
DELETE	Delete	404 Not Found*2	200 OK/404 Not Found*4

Table: Recommended return values of HTTP methods in combination with the resource URIs. (*1) – returns *Location* header with link to /users/{id} containing new ID; (*2) – unless you want to update/replace/modify/delete whole collection; (*3) – if resource already exists; (*4) – if ID not found or invalid.

Naming conventions

- resources should have name as nouns, not as verbs or actions
- plural if possible to apply
- URI should follow a predicatable (i.e. consistent usage), and hierarchical structure (based on structure-relationships of data)

Correct usages

POST /customers/12345/orders/121/lineitems
GET /customers/12345/orders/121/lineitems/3
GET|**PUT**|**DELETE** /customers/12345/configuration

Anti-patterns

GET /services?op=update_customer&id=12345&format=json
PUT /customers/12345/update

The Richardson Maturity Model

- provides a way to evaluate compliance of API to REST constraints

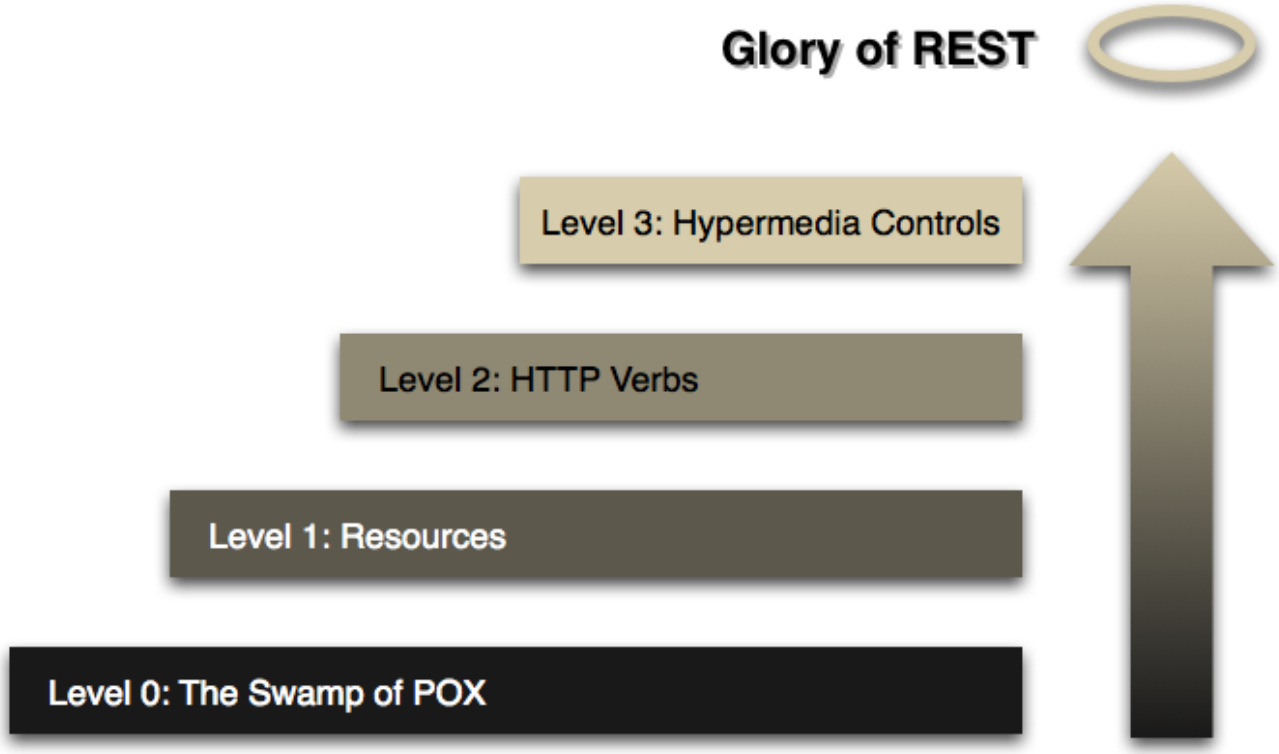


Figure: A model (developed by Leonard Richardson) that breaks down the principal elements of a REST approach into three steps about *resources*, *http verbs*, and *hypermedia controls*. Source: `http://martinfowler.com/articles/richardsonMaturityModel.html`

Linked data

What is Linked data ?

- **Linked Data** is a method of publishing structured data so that it can be interlinked and queried
- it builds upon standard Web technologies to share information in a way that can be read automatically by computers
- there is already a vast amount of data in Linked Data format available on the Web (e.g. Linking Open Data cloud)
- **JSON-LD** (JSON for Linking Data) a lightweight Linked Data format based on JSON

Linked open cloud

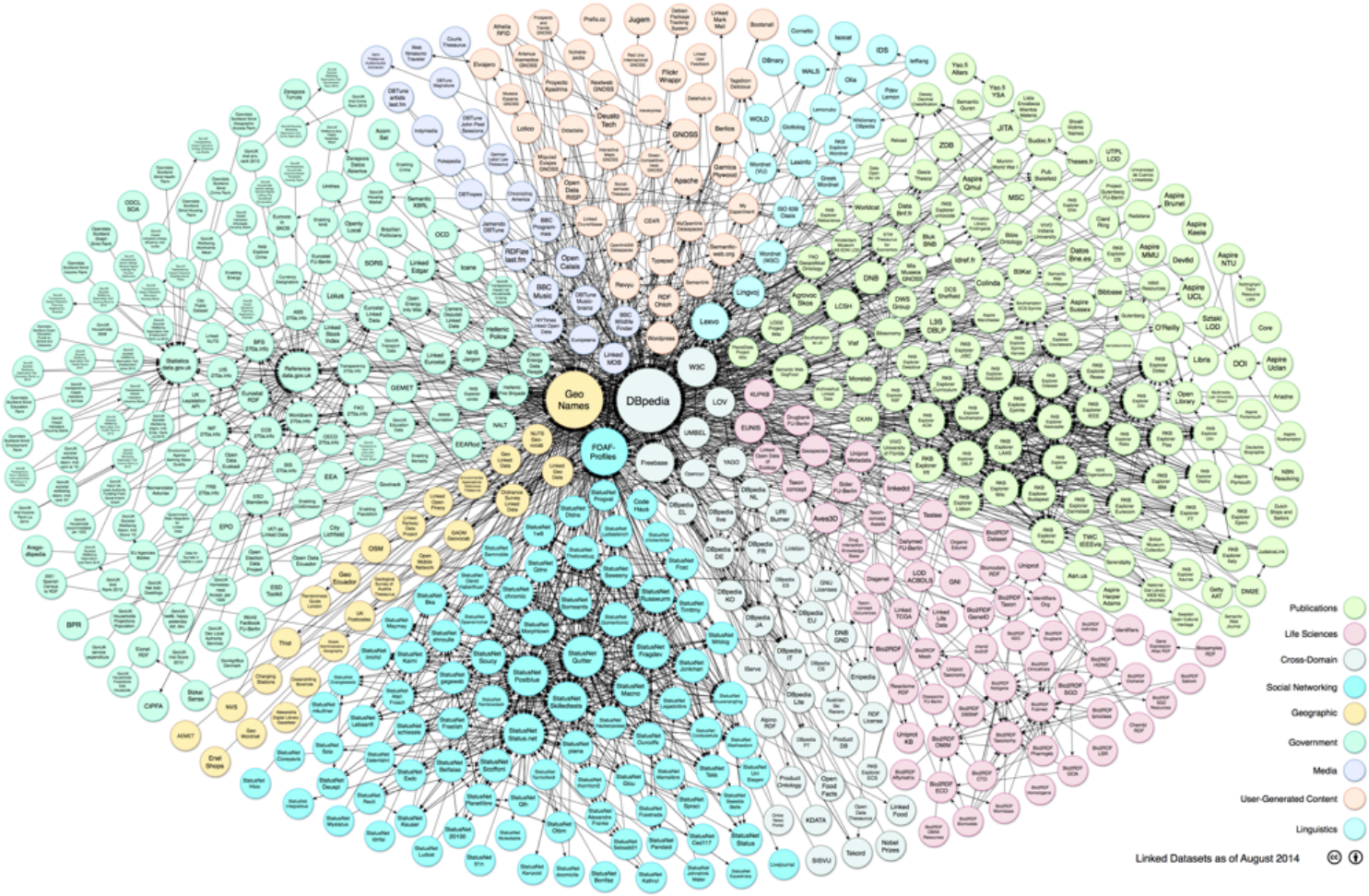


Figure: Linked Open Data cloud. Each buble represent a dataset, while edges represent links across datasets. There are about 10^{11} statements about resources within all datasets of the cloud. Source: `http://lod-cloud.net`

Hydra: Hypermedia-Driven Web APIs (1)

- Hydra is an effort to combine Linked Data principles to publish data and REST principles for web services
- REST services are used with JSON-LD format instead of plain JSON

Hydra: Hypermedia-Driven Web APIs (2)

Example – retrieve user with id 123

GET /user/123

Response is in JSON-LD

```
{ "@context": "http://schema.org/",
  "@type": "Person"
  "@id": "/user/123"
  "givenName": "Karel"
  "familyName": "Novak" }
```

- Type of the resource (i.e. "http://schema.org/Person") as well as specific properties (i.e. "http://schema.org/givenName", "http://schema.org/familyName") are dereferencable. It is used to describe semantics of the schema in human-readable as well as machine-readable way.

The End

Thank You

Resources

- Fielding, R.T., 2000. Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine),
- Fowler, M., 2010. Richardson Maturity Model: steps toward the glory of REST. Online at `http://martinfowler.com/articles/richardsonMaturityModel.html`.
- Lanthaler, M. and Gütl, C., 2012, April. On using JSON-LD to create evolvable RESTful services. In Proceedings of the Third International Workshop on RESTful Design (pp. 25-32). ACM.
- `https://spring.io/understanding/REST`
- `http://www.restapitutorial.com`