Minimal type inference for Linked Data consumers

and

A descriptive type foundation for RDF Schema

Articles published in Journal of Logical and Algebraic Methods in Programming

Gabriel Ciobanu[(a)] Ross Horne[1 (a)(b)(c)] Vladimiro Sassone[(d)]

(*a*) Romanian Academy, Iaşi, Romania
(*b*) Nanyang Technological University, Singapore
(*c*) Kazakh British Technical University, Almaty, Kazakhstan
(*d*) Electronics and Computer Science, University of Southampton, United Kingdom

26 February 2016

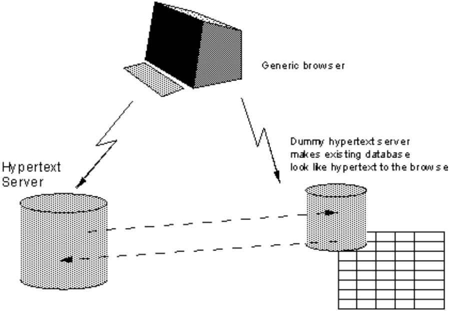
[1] Corresponding author. Email: rhorne@ntu.edu.sg

History of the Web of Linked Data

1989– | The Web of Hypertext

Emphasis on documents interlinked using URIs.

Berners-Lee. Information management: A proposal



2001–2006 | The Semantic Web

Emphasis on deep ontologies classifying everything (we can learn from an AI winter).

Berners-Lee, Lassila & Hendler. The Semantic Web. Scientific american, 284(5):28-37.

Berners-Lee, Hall & Shadbolt. The Semantic Web revisited. Intelligent Systems, 21(3):96-101.

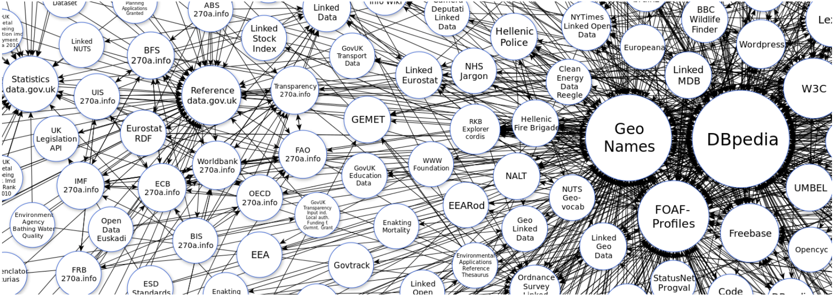
2006– | The Web of Linked Data

Emphasis on raw data interlinked using URIs and delivered by simple data APIs.

Berners-Lee. Linked Data — design issues

Four Principles of Linked Data

- Use URIs to identify resources.
- Use HTTP URIs to identify resources so we can look them up.
- When a URI is looked up, return data about the resource using the standards.
- Include URIs in the data, so they can also be looked up.



Dereferencing the URI *res:Kazakhstan*

```
curl -I -H "Accept:text/n3" http://dbpedia.org/resource/Kazakhstan
```

Request:

```
GET /resource/Kazakhstan HTTP/1.1
Host:  dbpedia.org
Accept:  text/n3
```

Response:

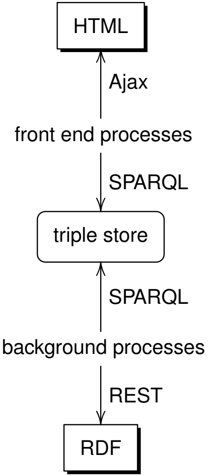
```
HTTP/1.1 303 See Other
Content-Type:  text/n3
Location:  http://dbpedia.org/data/Kazakhstan.n3
```

Dereferencing the URI *res:Kazakhstan*

```
curl -H "Accept:text/n3" http://dbpedia.org/data/Kazakhstan.n3
```

```
@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Medeo dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Zhetysu_Stadium dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Astana_Arena dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Kazakhstan_Sports_Palace dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Munayshy_Stadium dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Aral_Sea dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Aral_Sea dbpedia-owl:country dbpedia:Kazakhstan .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ns4: <http://en.wikipedia.org/wiki/> .
ns4:Kazakhstan foaf:primaryTopic dbpedia:Kazakhstan .
dbpedia:Air_Kokshetau dbpprop:headquarters dbpedia:Kazakhstan .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ns6: <http://data.nytimes.com/> .
ns6:N63032621026086062091 owl:sameAs dbpedia:Kazakhstan .
dbpedia:Rakhimzhan_Qoshqarbaev dbpprop:placeOfBirth dbpedia:Kazakhstan .
@prefix yago-res: <http://mpii.de/yago/resource/> .
yago-res:Kazakhstan owl:sameAs dbpedia:Kazakhstan .
dbpedia:Regina_Kulikova dbpedia-owl:birthPlace dbpedia:Kazakhstan .
dbpedia:Almaty_International_School dbpprop:country dbpedia:Kazakhstan .
dbpedia:The_Gift_to_Stalin dbpedia-owl:country dbpedia:Kazakhstan .
dbpedia:Dmytro_Salamatin dbpedia-owl:birthPlace dbpedia:Kazakhstan .
dbpedia:Dungan_language dbpedia-owl:spokenIn dbpedia:Kazakhstan .
dbpedia:Kazakhstan dbpprop:currencyCode "KZT"@en .
dbpedia:Kazakhstan dbpedia-owl:percentageOfAreaWater "1.7"^^xsd:float .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix yago: <http://dbpedia.org/class/yago/> .
dbpedia:Kazakhstan rdf:type yago:StatesAndTerritoriesEstablishedIn1991 .
@prefix ns10: <http://umbel.org/umbel/rc/> .
dbpedia:Kazakhstan rdf:type ns10:Location_Underspecified .
dbpedia:Kazakhstan rdf:type dbpedia-owl:PopulatedPlace ,
dbpedia:Kazakhstan rdf:type yago:CentralAsianCountries ,
dbpedia:Kazakhstan rdf:type yago:LandlockedCountries .
@prefix ns11: <http://schema.org/> .
dbpedia:Kazakhstan rdf:type ns11:Country ,
dbpedia:Kazakhstan rdf:type dbpedia-owl:Country ,
dbpedia:Kazakhstan rdf:type yago:YagoGeoEntity ,
dbpedia:Kazakhstan rdf:type dbpedia-owl:Place ,
dbpedia:Kazakhstan rdf:type yago:Economy108366753 .
```

An Architecture for Linked Data Consumers



Front end: traditional Web architecture, with SPARQL replacing SQL.

Triples store: graph based query and update, suited to combining data sources so they can be collectively queried.

Back end: pulls raw data from the Web, using RESTful open data APIs.

Our problem: **Make programming the back end easier.**

Comparing SPARQL queries to background scripts.

A *front end* SPARQL query that discovers a URI for the capital of Kazakhstan:

```
select $x
from named res:Kazakhstan
where
    { graph res:Kazakhstan {res:Kazakhstan  dbp:capital  $x} }
limit 1
```

A *background* script that finds the URI for the capital of Kazakhstan, then loads dereferenced data into the triple store in a *named graph* identified by the discovered URI.

```
select $x
from named res:Kazakhstan
where
    graph res:Kazakhstan {res:Kazakhstan  dbp:capital  $x}
from named $x
```

Traditional type systems are Prescriptive.

```
do  select $x: xsd:anyURI, $y: xsd:string
    where
        $x rdfs:label $y
        langMatches($y, en)
        regex($y, ^cyber)
    from $x
```

Types *xsd:anyURI* and *xsd:string* are **prescriptive**. The language and regular expressions only apply to strings, and only URIs can be dereferenced.

Type checking: If programmer **wrongly** writes

```
from $y
```

a type error is thrown since a string cannot be dereferenced.
Given the range of *rdfs:label* is *xsd:string*, if the programmer **wrongly** writes

```
$y rdfs:label $x
```

the type checker will detect the error.

Type inference: Given no type information, a type inference algorithm infers:

- `$y` must be of type *xsd:string* since used in a regular expression,
- `$x` must be of type *xsd:anyURI* since it is dereferenced,
- furthermore, the range of *rdfs:label* must be *xsd:string*.

Programmers make more mistakes for larger scripts

```
from named res:Almaty
select $almalat : xsd:decimal, $almalong : xsd:decimal
where
    graph res:Almaty {res:Almaty  geo:lat  $almalat}
    graph res:Almaty {res:Almaty  geo:long  $almalong}
from named res:Kazakhstan
do   select $loc : xsd:anyURI
     where
         graph res:Kazakhstan {$loc  dbp:location  res:Kazakhstan}
     from named $loc
     select $lat : xsd:decimal, $long : xsd:decimal
     where
         graph $loc {$loc  geo:lat  $lat}
         graph $loc {$loc  geo:long  $long}
         haversine($lat, $long, $almalat, $almalong) < 1000
     do   select $person : xsd:anyURI
          where
              graph $loc {$person  dbp:birthPlace  $loc}
          from named $person
```

Dereference data about people born in places in Kazakhstan within 1000km from Almaty.

Descriptive Types for RDF Schema: simple entailment

Data:

res:Vitali_Klitschko dbp:boxerCategory res:Heavyweight

RDF Schema:

dbp:boxerCategory rdfs:domain dbp:Boxer
dbp:boxerCategory rdfs:range dbp:BoxingCategory

Rule:

$$\frac{uri_1 \;\; \text{rdfs:domain} \;\; type \qquad uri_0 \;\; uri_1 \;\; uri_2}{uri_0 \;\; a \;\; type} \;(\text{rdfs2})$$

Simply entailed type:

res:Vitali_Klitschko a dbp:Boxer

Descriptive Types for RDF Schema: simple entailment

More data:

res:Vitali_Klitschko dbp:boxerCategory res:Heavyweight

res:Vitali_Klitschko dbp:birthPlace res:Kyrgyz_SSR

More RDF Schema:

dbp:boxerCategory rdfs:domain dbp:Boxer
dbp:boxerCategory rdfs:range dbp:BoxingCategory

res:Vitali_Klitschko a dbp:Boxer

dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place

Simply entailed types:

res:Vitali_Klitschko a dbp:Boxer
res:Vitali_Klitschko a dbp:Person

or equivalently

res:Vitali_Klitschko a owl:intersectionOf(dbp:Boxer, dbp:Person)

Descriptive Types for RDF Schema: schema inference

More data:

res:Vitali_Klitschko dbp:boxerCategory res:Heavyweight

res:Vitali_Klitschko dbp:birthPlace res:Kyrgyz_SSR

More RDF Schema:

dbp:boxerCategory rdfs:domain dbp:Boxer
dbp:boxerCategory rdfs:range dbp:BoxingCategory

res:Vitali_Klitschko a dbp:Boxer

dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place

Inferred subclass assumption:

dbp:Boxer rdfs:subClassOf dbp:Person

Descriptive Types for RDF Schema: well-typed data

Data about Vitali's brother Wladimir:

res:Wladimir_Klitschko dbp:birthPlace dbp:Kazakh_SSR

RDF Schema information:

res:Wladimir_Klitschko a dbp:Boxer
dbp:Kazakh_SSR a dbp:Place
dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place
dbp:Boxer rdfs:subClassOf dbp:Person

The following simply entailed type is *redundant*.

res:Wladimir_Klitschko a dbp:Person

We already know this from:

dbp:Boxer rdfs:subClassOf dbp:Person
res:Wladimir_Klitschko a dbp:Boxer

Typing: for well-typed systems, no new inferences need be applied.

Descriptive Type for RDF Schema: ask an expert

More data about Vitali:

res:Vitali_Klitschko free:government/politician/party free:m/0j1b9hc

More RDF Schema information:

free:government/politician/party rdfs:domain free:government/politician
free:government/politician/party rdfs:range free:government/political_party

Options presented to the expert:

1. Use simple entailment:

   res:Vitali_Klitschko a owl:intersectionOf(dbp:Boxer, free:government/politician)

2. Use inferred subclass assumption:

   dbp:Boxer rdfs:subClassOf free:government/politician

3. Ignore warning and resolve later.

Subjectivity: Human experts know not every boxer is a politician. Machines don't.

Example of Scripting Problem: want Andrei Ershov the scientist

- Initial script:

```
from res:Andrei_Yershov
select $book: free:book
where res:Andrei_Yershov free:book.author.works_written $book
```

- Initial data including:

free:book.author.works_written rdfs:domain free:book.author
free:book.author.works_written rdfs:range free:book

Warning with options:

1. Refine type of *res:Andrei_Yershov* to *free:book.author*.
2. Relax domain of *free:book.author.works_written* to *xsd:anyURI*.
3. Ignore warning and resolve later.

Example of Scripting Problem: got Andrei Ershov the sportsman

- Second state of script:

  select $book: *free:book*
  where *res:Andrei_Yershov free:book.author.works_written* $book

- Data including triples from dereferencing *res:Andrei_Yershov*:

  res:Andrei_Yershov a dbp:IceHockeyPlayer
  free:book.author.works_written rdfs:domain free:book.author
  free:book.author.works_written rdfs:range free:book

Warning with options (**None make sense, suggesting a mistake!**):

1. Refine type of *res:Andrei_Yershov* :

   res:Andrei_Yershov a owl:intersectionOf(yago:IceHockeyPlayer,
   free:book.author)

2. Relax domain of *free:book.author.works_written* :

   free:book.author.works_written rdfs:domain owl:unionOf(free:book.author,
   dbp:IceHockeyPlayer)

3. Refine subtype assumptions such that:

   dbp:IceHockeyPlayer rdfs:subClassOf free:book.author

Conclusion: Prescriptive and Descriptive Types

Prescriptive types:

- Traditional type system aware of simple datatypes, such as integers and strings.
- Helps avoid writing scripts that can never work.
- Type inference makes programmer's life easier, by automatically calculating types.

Descriptive types:

- Novel aspects of RDF Schema types, concerning URIs.
- Helps make sense of subjective knowledge consumed from the Web.
- Throws warnings with menu of options rather than errors.
- RDF Schema simple entailment is one of several inference modes.
- Accommodates more data publishing models than W3C standards.
- Philosophically, narrows gap between type theory and semiotics.

Implementation:

- Mature language specification ready to be implemented.
- Techniques can be applied to extensions of popular scripting languages.