IN3060/4060 – Semantic Technologies – Spring 2021

Lecture 15: Publishing RDF Data on the Web

Jieying Chen

14th May 2021

Department of Informatics

University of Oslo

Today's Plan

1. Relevant highlights from RDF lecture
2. Linked (Open) Data
3. Conclusion

Outline

1. Relevant highlights from RDF lecture
2. Linked (Open) Data
3. Conclusion

RDF

- Why URIs?
  - URIs naturally have a “global” scope, unique throughout the web.
  - URLs are also addresses.
  - *“A web of data.”*
- Why triples?
  - Any information format can be transformed to triples.
  - Relationships are made explicit and are elements in their own right
  - Again, *“A web of data”*.

RDF on the web: Where is it?

- In files:
  - In some serialisation format: XML/RDF, Turtle, . . .
  - Typically small RDF graphs, i.e., max. a few 100 triples, e.g.,
    - Vocabularies: `http://xmlns.com/foaf/spec/index.rdf`.
    - Tiny datasets: `http://folk.uio.no/martingi/foaf.rdf`.
- "Behind" *SPARQL endpoints*:
  - Data kept in a *triple store*, i.e., a database.
  - RDF is served from endpoint as results of *SPARQL queries*.
  - Exposes data (in different formats)
    - with endpoint frontends, e.g., `http://dbpedia.org/resource/Norway`, or
    - by direct SPARQL query: `http://dbpedia.org/sparql`.

Publishing RDF on the web

- If URIs of resources are dereferencable. . .
- . . . clients can use URIs to request a description of the resource.
- Make data available in different formats. Typically:
  - HTML for humans,
  - RDF for computers.
- This is called *content negotiation*.
- Endpoint frontends will do all of this for you.
- In this lecture, we look at some of the technicalities.

Outline

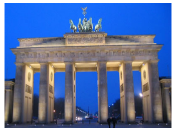
1. Relevant highlights from RDF lecture
2. Linked (Open) Data
3. Conclusion

URIs

- URIs in RDF can have many different forms:
  - `http://www.google.com/` – a web page
  - `mailto:jsmith@example.com` – a mailbox
  - `http://dbpedia.org/resource/Oslo` – a town
  - `http://folk.uio.no/martingi/foaf#me` – a person
  - `tel:+47-22855050` – a telephone line
  - `urn:isbn:0-395-36341-1` – a book
- Two basic types
  - "information resources": downloadable documents.
  - "non-information resources": other entities.

The Problem and Two Solutions

- The problem:
  - Need to locate information *about* a resource.
  - The same URI cannot denote a *downloadable* resource.
- Example: Need to differentiate between:
  - A web page or RDF file about Berlin.
  - The city of Berlin.

 ≠

- Two W3C-recommended solutions:
  - The hash-namespace solution.
  - The slash-namespace solution (aka HTTP 303 redirects).
- To fully understand them, we need to have a look at HTTP.

HTTP

- HTTP Server listens to "requests" (usually on TCP/IP port 80).
- An HTTP client sends requests to the server and obtains responses.
- A typical request: `http://folk.uio.no/martingi/`.
  - Connect to port 80 on `folk.uio.no`.
  - Send:

    ```
    GET /martingi/ HTTP/1.1
    User-Agent: Mozilla/5.0 (X11; U; Linux i686; ...
    Accept: text/html,application/xhtml+xml,...
    Accept-Language: no, en
    Host: folk.uio.no
    ...
    ```

    followed by a blank line.
- Other "methods": `HEAD`, `POST`, `PUT`,...

HTTP (cont.)

- A typical response to the GET request:

  ```
  HTTP/1.1 200 OK
  Date: Wed, 05 May 2010 14:15:24 GMT
  Server: Apache/2.2.14 (Unix) ...
  Content-Length: 14348
  Content-Type: text/html

  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
  <html> ...  </html>
  ```

- Result may vary depending on the `Accept:` choices in request.
- `200 OK` is not the only possible response ("status code"):
  - `404 Not Found`
  - `401 Unauthorized`
  - `303 See Other`

Fragment identifiers

- A *fragment identifier* is the part after # in a URI:

  ```
  http://en.wikipedia.org/wiki/Fragment_identifier#Examples
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  ```

- HTTP specifies that fragment identifiers are processed client-side:
  - GET request is sent without the fragment identifiers:

    ```
    GET /wiki/Fragment_identifier HTTP/1.1
    ```

  - fragment identifier is processed by client.
- For HTML or XHTML:
  - Elements (sections titles, paragraphs, etc.) can have *id* attributes:

    ```
    <h2 id="Examples">Examples</h2>
    ```

  - Browser will jump to element identified by fragment identifier.

Hash namespaces

- For RDF served over HTTP: fragment identifiers identify resources:
  - `http://bla.bla/bla#resource` is a resource
  - `http://bla.bla/bla` is a document describing the resource
- e.g., FOAF files:
  - `http://folk.uio.no/martingi/foaf.rdf#me` - a person
  - `http://folk.uio.no/martingi/foaf.rdf` - an RDF/XML file
- *By convention* the RDF file contains some triples involving resources identified by its fragments.
- Can use the part of the URI until # as namespace

```
@prefix myfoaf:  <http://.../martingi/foaf.rdf#>
myfoaf:martin foaf:givenname "Martin" .
```

- This is known as a “hash namespace”.

Hash namespaces – pros and cons

- Hash namespaces solve our problem:
  - Resources are separate from documents about them.
  - It is possible to find a document given a resource URI.
- Moreover:
  - Fetching the right document is done automatically by HTTP.
  - It is enough to publish the RDF file on an HTTP server.
  - Very low tech and fool proof, in other words.
- However:
  - All data published this way about all entities in a hash namespace needs to be stored in the same RDF file

    ```
    http://brreg.no/bedrifter.rdf#974760673
    ```
  - Too tight coupling of URI schema (name design) and physical storage (file name).

HTTP Redirection

- Reminder: HTTP responses start with a “status code”:
  - Usually “200 OK”, if the document was found and can be served.
  - “404 Not Found”, if the document does not exist.
- One of the possible status codes is “303 See Other”.
- Always comes with a `Location:` field in the response.
- Tells the client to submit a “GET” request to that location.
- Also known as “303 redirection”.
- Followed by all modern HTTP clients.
- Often used when URIs have changed.

Example of 303 Redirection

- User requests `http://www.dnvgl.com/`.
- Client sends request to `www.dnvgl.com`:

  ```
  GET / HTTP/1.1
  Host:  www.dnvgl.com
  ```
- DNV GL changed name to DNV . . . Server responds:

  ```
  HTTP/1.1 303 See Other
  Location:  http://www.dnv.com/
  ```
- Client sends new request to `www.dnv.com`:

  ```
  GET / HTTP/1.1
  Host:  www.dnv.com
  ```
- Server at `www.dnv.com` responds:

  ```
  HTTP/1.1 200 OK
  Content-Type:  text/html
  ...
  ```

303 Redirection for RDF

- Find information about `http://dbpedia.org/resource/Oslo`.
- Send “GET” request to server `dbpedia.org`:
  ```
  GET /resource/Oslo HTTP/1.1
  Accept:  application/rdf+xml
  ```
- Server `dbpedia.org` recognizes this as a non-information resource.
- Redirects to a file with data about the city of Oslo:
  ```
  HTTP/1.1 303 See Other
  Location:  http://dbpedia.org/data/Oslo.xml
  ```
- Browser can now send a new request for that location:
  ```
  GET /data/Oslo.xml HTTP/1.1
  Accept:  application/rdf+xml
  ```
- This time the server responds with the requested document:
  ```
  HTTP/1.1 200 OK
  Content-Type:  application/rdf+xml
  ...
  ```

Slash Namespaces

- Common to use URIs with a slash (/) as last non-identifier character:

  `http://dbpedia.org/resource/Oslo`
- Can use URI up to last slash as namespace:
  ```
  @prefix dbpedia:  <http://dbpedia.org/resource/>
  dbpedia:Oslo dbprop:maySnowCm "0" .
  ```
- Known as a “slash namespace”.
- Advantages over hash namespaces:
  - Whole URI is sent to server, so. . .
  - Possible to redirect different resources to different documents.
  - Possible to change redirection without changing URIs.
- Requires some more server configuration.
- See recipes at `http://www.w3.org/TR/swbp-vocab-pub/`.
- See also `http://linkeddatabook.com/`.

Serving Vocabularies

- What about classes and properties?
- Identified by URIs:
  ```
  http://xmlns.com/foaf/0.1/Person
  http://xmlns.com/foaf/0.1/knows
  http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
  http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  ```
- What should be served in response to these?
  - A description of the “vocabulary” defining the term.
  - Often an RDF file with RDFS or OWL/RDF content.
  - Sometimes (FOAF) just an HTML page with documentation.
- Mechanisms are the same as for “ordinary” RDF data.
- A single RDF file (hash namespace) is usually OK.
- Should also serve the vocabulary description for the “vocabulary URI”:
  ```
  http://xmlns.com/foaf/0.1/
  http://www.w3.org/1999/02/22-rdf-syntax-ns#
  ```

HTTP Content Type Negotiation

- In HTTP, data formats are identified by “internet media types”.
  - Previously known as MIME types.
  - `text/html, image/jpeg, application/pdf,...`
- RDF media types:
  - RDF/XML: `application/rdf+xml`.
  - Turtle: `text/turtle`.
  - N3: `text/n3`.
- Client sends accepted media types in `Accept:` header:
  - `Accept:  text/html, text/plain`
- Server chooses sent media type:
  - Picks the preferred one among available types.
  - Sends the media type of the response in the header.
  - `Content-Type:  text/html`

Content Type Negotiation for RDF

- Given the URI of a non-information resource. . .
  - A semantic web applications wants RDF data, as discussed.
  - A regular WWW browser wants HTML, human readable.
- This can be achieved using HTTP content type negotiation.
- Semantic web client:
  - Requests RDF, e.g., `Accept: application/rdf+xml, text/turtle`.
  - Server uses e.g., 303 redirection to an RDF file.
- HTML web client:
  - Requests text, e.g., `Accept: text/html, text/plain`.
  - Server uses e.g., 303 redirection to an HTML file.
- Also possible with hash namespaces, see `http://www.w3.org/TR/swbp-vocab-pub/`.

Example: dbpedia.org

- Requesting the URI `http://dbpedia.org/resource/Oslo`
- From an HTML web browser:
  - Sends `Accept: text/html` in request
  - Server returns:

```
HTTP/1.1 303 See Other
Location: http://dbpedia.org/page/Oslo
```

  - Client requests `http://dbpedia.org/page/Oslo`
  - Server sends HTML document:

```
HTTP/1.1 200 OK
Content-Type: text/html
```

Example: dbpedia.org (cont.)

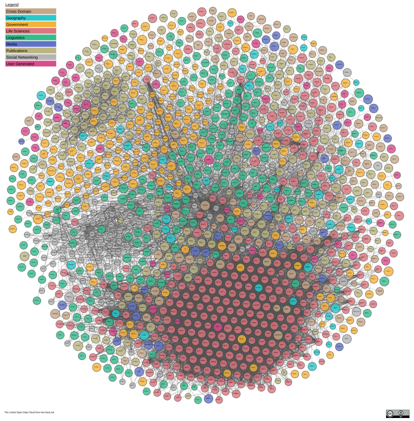
- Requesting the URI `http://dbpedia.org/resource/Oslo`
- From a semantic web browser:
  - Sends `Accept: application/rdf+xml` in request
  - Server returns:

```
HTTP/1.1 303 See Other
Location: http://dbpedia.org/data/Oslo.xml
```

  - Client requests `http://dbpedia.org/data/Oslo.xml`
  - Server sends RDF/XML document:

```
HTTP/1.1 200 OK
Content-Type: application/rdf+xml
```

The Linked Open Data Cloud



`http://lod-cloud.net/`

Outline

1. Relevant highlights from RDF lecture
2. Linked (Open) Data
3. Conclusion

Topics Covered

- RDF, principles, Turtle syntax
- The Jena API for RDF
- The SPARQL Query Language
- Basics of the RDFS and OWL ontology languages
- Basics of model semantics and reasoning
- Linked Open Data
- Constraints, SHACL
- OTTR Templates

Topics *Not* Covered

- Rule Languages (SWRL, RIF, Jena rules, etc.)
- SW application structures
- Semantic Web Services
- Details of RDF/RDFS model semantics
- Some details of OWL
- Details of OWL 2 profiles
- Logical theory: Soundness, Completeness,. . .
  - (You ain't seen nothing yet :-)
- And many more!

Help! I Can't Get Enough!

- For more information on theory:
  - Book on Foundations of SW Technologies
  - Take a course in logic or automated reasoning
- For more information on practical questions:
  - Book on Semantic Web Programming
  - Standards texts on W3C Web pages
  - Google
- Still not enough?
  - Contact us for possible MSc topics!

That's all Folks!