INF3580 – Semantic Technologies – Spring 2011

Lecture 14: Publishing RDF Data on the Web

Martin Giese

3rd May 2011

Department of Informatics
University of Oslo

Today's Plan

Outline

RDF on the Web

- RDF data exists in many forms:
  - In RDF files, downloadable with HTTP, FTP, etc.
    - FOAF profiles
    - data files from dbpedia.org, geonames, etc.
    - In RSS 1.0 feeds
  - As data model behind SPARQL query endpoints
    - for instance dbpedia.org, dblp, and others
  - Embedded in HTML, as RDFa
  - Embedded in PDF as XMP metadata
  - ...
- How do I *find* data about something?
  - Announcement of a cool new SPARQL endpoint
  - Semantic Web indices and search engines (Google to find some!)
  - Links from HTML pages to RDF data
  - "Linked Open Data" (LOD)

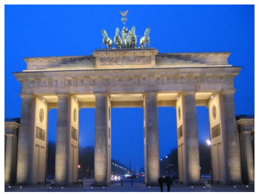
Outline

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://heim.ifi.uio.no/martingi/foaf.rg#me` – a person
  - `tel:+47-22852737` – a telephone number
  - `urn:isbn:0-395-36341-1` – a book
- Two basic types
  - "information resources": downloadable documents
  - "non-information resources": other entities
- Some provide a download protocol, but the resources don't exist
- Others are not dereferencable
- From the RDF standpoint, all are OK
- In practice, software wants to locate information
  - Protocols like http, ftp, etc. are an advantage

The Problem

- Need to differentiate between:
  - A web page or RDF file about Berlin
  - The city of Berlin
- e.g. the city was "created" around 1200...
- A URI for Berlin should not be an existing HTTP resource (why?)
- Need another way to retrieve information about a resource



$\neq$

Two Solutions

- The problem:
  - Need to locate information *about* a resource
  - The URI cannot denote a *downloadable* resource
- 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://heim.ifi.uio.no/martingi/`
  - Connect to port 80 on `heim.ifi.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: heim.ifi.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>
      ...
  ```
- 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
- Various uses with JavaScript (AJAX), PDF viewers, etc.

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://heim.ifi.uio.no/martingi/foaf.rdf#me` - a person
  - `http://heim.ifi.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:me 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`
  - URI says much about data organization. RDF file name baked in!
  - No way to change the organization without changing URIs

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.sun.com/`
- Client sends request to `www.sun.com`

```
GET / HTTP/1.1
Host: www.sun.com
```

- Sun was bought by Oracle. . . Server responds:

```
HTTP/1.1 303 See Other
Location: http://www.oracle.com/
```

- Client sends new request to `www.oracle.com`:

```
GET / HTTP/1.1
Host: www.oracle.com
```

- Server at `www.oracle.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://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/`

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` (registration pending)
  - N3: `text/rdf+n3` (not registered)
- Client sends accepted media types in `Accept:` header:
  - `Accept: text/html, text/plain`
  - Can additionally add “quality factors” to specify preference
- 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
    ```


Outline

Relational Database Management Systems

- “Relational” databases introduced in 1970
  - Replaced navigational and hierarchical systems
- Mostly used with query language SQL
- Most of the world’s business data today is stored in relational databases
- Several freely available systems:
  - PostgreSQL
  - MySQL
  - SQLite
  - ...
- Many commercial systems:
  - Oracle
  - IBM DB2
  - Microsoft Access, SQL Server
  - ...

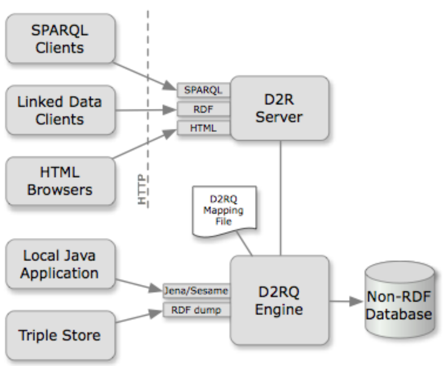
RDBMS to RDF

- Need a way to make data in RDBMS available as RDF
- First idea: RDF export
  - Read all records, export RDF
  - Bad idea: data replication. . .
  - Probably won't switch whole enterprise to RDF store
  - Need to convert to RDF regularly
- Often a better idea: RDF view
  - SPARQL endpoint translates incoming queries to SQL
  - Translates result to SPARQL SELECT result or RDF
  - Data remains where it is, no duplication
  - Drawback: need to keep "old-fashioned" DB backend

Outline

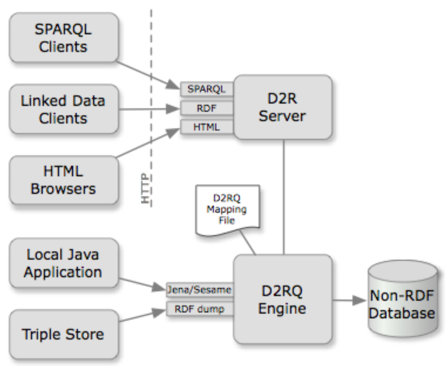
D2R/D2RQ

- Allows to treat relational databases as RDF
- Developed by FU Berlin
- Mapping describes relation between DB and RDF
- Can create SPARQL endpoint without transforming the whole database: *Virtual* RDF graph.
- Also on-demand RDF/HTML pages following LOD protocol



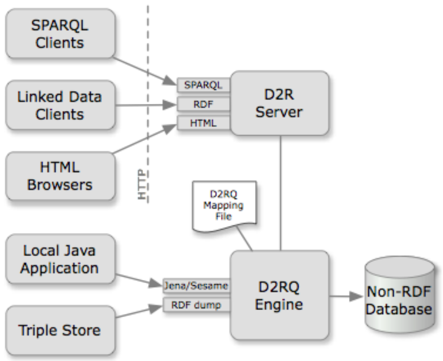
D2RQ Engine

- Reads a "Mapping File"
  - Table → Class
  - Row → Resource
  - Column → Property
  - RDF-encoded
- Translates SPARQL to SQL
- Can also act as Jena Graph
- Or the Sesame equivalent
- Can also export whole DB



D2R Server

- Provides WWW-frontend
- SPARQL Endpoint
- Serves RDF as linked open data
- Pages of data for HTTP browsers
- All requests translated to SPARQL



Example: World Database

- An example database from MySQL distribution
- Table City:
  - ID (key): a unique number
  - Name: the city's name
  - CountryCode: Code for the country the city lies in
  - ...
- Table Country:
  - Code (key): the code for a country
  - Name: the Country's name
  - Continent: the Continent the country lies in
  - Capital: the City ID of the country's capital
  - ...

Example: World Database (cont.)

- Table City:

ID	Name	CountryCode	...
		...	
2806	Kingston	NFK	...
2807	Oslo	NOR	...
2808	Bergen	NOR	...
		...	

- Table Country:

Code	Name	Continent	Capital	...
		...		
NLD	Netherlands	Europe	5	...
NOR	Norway	Europe	2807	...
NPL	Nepal	Asia	2729	...
		...		

Outline

Where Classes Come From

- From a mapping file for the World database:

```
map:City a d2rq:ClassMap ;
    d2rq:dataStorage map:database ;
    d2rq:uriPattern "City/@@City.ID@@" ;
    d2rq:class vocab:City ;
    d2rq:classDefinitionLabel "City" .
```

- identify a "class mapping"
- link to a resource describing the DB connection
- give the pattern for resources of this class
  - contains placeholder with DB table and column
- give the RDFS class for those resources
- give the label for that class.
- Generates:
  - `<http://.../City/2806> a vocab:City.`
  - `<http://.../City/2807> a vocab:City.`
  - `<http://.../City/2808> a vocab:City.`

Mapping Files

Resources for Countries and Continents

- The same for countries:

```
map:Country a d2rq:ClassMap ;
    d2rq:dataStorage map:database ;
    d2rq:uriPattern "Country/@@Country.Code@@" ;
    d2rq:class vocab:Country ;
```

- Can have more classes than tables!
- For continents, add mapping:

```
map:Continent a d2rq:ClassMap ;
    d2rq:dataStorage map:database ;
    d2rq:uriPattern "Continent/@@Country.Continent|urlify@@";
    d2rq:class vocab:Continent ;
    d2rq:classDefinitionLabel "Continent" .
```

- For everything in the Continent column of Country...
- ...generate a resource with URI `.../Continent/...`
- ...removing spaces from "North America", etc.
- E.g. `http://.../resource/Continent/North_America`

Mapping Files

Where Properties Go To

- A mapping for city names:

```
map:City_Name a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap map:City ;
    d2rq:property vocab:name ;
    d2rq:propertyDefinitionLabel "name" ;
    d2rq:column "City.Name" .
```

- Identify a "property bridge"
- that adds properties to the resources described in `map:City`
- give the predicate
- give a label to the predicate
- the object is a *literal* taken from this column
  - `<http://.../City/2806> vocab:name "Kingston".`
  - `<http://.../City/2807> vocab:name "Oslo".`
  - `<http://.../City/2808> vocab:name "Bergen".`
- Also possible to define literals with patterns containing columns

Mapping Files

Linking Cities and Countries

- Want URIs as objects, not literal country codes.
- Use the following property bridge:

```
map:City_CountryCode a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap map:City ;
    d2rq:property vocab:inCountry ;
    d2rq:refersToClassMap map:Country ;
    d2rq:join "City.CountryCode=>Country.Code" .
```

- Foreign key: link to resource from another class map
- Say how columns for `map:City` correspond to those for `map:Country`
- From countries to capitals:

```
map:Country_Capital a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Country;
    d2rq:property vocab:capital;
    d2rq:refersToClassMap map:City;
    d2rq:join "Country.Capital=>City.ID";
```

Resulting Graph

After adding similar mappings for country names and inContinent:

```
<http://.../City/2807> a vocab:City ;
        vocab:name "Oslo" ;
        vocab:inCountry <http://.../Country/NOR> .

<http://.../Country/NOR> a vocab:Country ;
        vocab:name "Norway" ;
        vocab:capital <http://.../City/2807> ;
        vocab:inContinent <http://.../Continent/Europe> .
```


Linking to DBpedia

- Add property bridge:

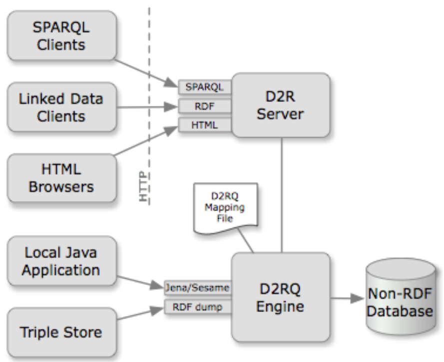
```
map:Country_DBpedia a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Country;
    d2rq:property owl:sameAs;
    d2rq:uriPattern
"http://dbpedia.org/resource/@@Country.Name|urlify@@" .
```

- No problem to use “external” properties or classes
- No problem to link to “external” URIs.
- Careful: Generating links like this often fails for some cases:
  - World DB country name: `Sao Tome and Principe`
  - DBpedia URI: `http://.../São_Tomé_and_Príncipe`
- Better in general to have a DB table with corresponding URIs

Outline

The Jena Adapter

- No direct way of adding reasoning to D2R
- An RDF view of a database can be made available as a Jena Model
- Requires mapping file and d2rq.jar
- Add reasoning to that model



The Jena Adapter: Example

`Model m = new ModelD2RQ("file:mapping.n3");`

- Create a model backed by a DB through D2R
- No data is read into memory

`OntModel om = ModelFactory.createOntologyModel();`
`om.read("file:world.owl");`

- Create model with ontology, e.g.
- `vocab:City rdfs:subClassOf vocab:Place`
- `vocab:Country rdfs:subClassOf vocab:Place`

`Model infm = ModelFactory.createRDFSModel(om, m);`

- Asking `infm` for all objects of type `vocab:Place`...
- ...gives all cities...
- ...and all countries!
- Can use Jena query engine for SPARQL queries with reasoning
- But does it still not read data into memory?

Forward Chaining vs. Backward Chaining

- Given: reasoning rules, like e.g.:

$$\frac{x \texttt{ rdf:type } C \quad C \texttt{ rdfs:subClassOf } D}{x \texttt{ rdf:type } D}$$

- Forward Chaining:
  - Add all consequences of rules to the model
  - Queries can be answered using the expanded model
- Backward Chaining:
  - Leave model as it is
  - Answer queries by applying rules "backwards"
  - A bit like Prolog!

Example of Forward Chaining

- Given triples:

```
:City rdfs:subClassOf :Place
:Oslo rdf:type :City
```

- Inferred triples:

```
:Oslo rdf:type :Place
:Place rdf:type rdfs:Class
:Place rdfs:subClassOf rdfs:Resource
...
```

- To answer x `rdf:type :Place`:
  - Simply look in model:
  - $x \to$ `:Oslo`

Example of Backward Chaining

- Given triples:

```
:City rdfs:subClassOf :Place
:Oslo rdf:type :City
```

- To answer x `rdf:type :Place`:
  - Look for direct occurrences: none
  - Look for instances of:
    - C `rdf:subClassOf :Place`
    - x `rdf:type` C
  - E.g. $C \to$ `:City`, $x \to$ `:Oslo`
- In general, need to backward-chain over many rules!
  - E.g. C `rdf:subClassOf :Place` could come from other rules

Forward Chaining vs. Backward Chaining

Forward	**Backward**
reason once	repeated computation
diffuse	goal-oriented
adds to data	data unchanged
much space	little space
expensive up-front	cheap up-front
fast queries	slow queries
possibly non-terminating expansion	possibly non-terminating backward chaining

- "Hybrid" approaches possible, e.g. Jena RDFS reasoner
  - Forward chaining for sub-class/prop. hierarchy, ranges, domains
  - Backward chaining for `rdf:type`
- Forward chaining difficult for data in databases
  - RDFS reasoner OK for databases
  - Pellet etc. in general not

OWL 2 Profiles

OWL QL Based on "DL-Lite$_\mathcal{A}$". Allows query answering by "query rewriting", i.e. backward chaining. Same data-efficiency as SQL.

OWL RL Based on "pD*" semantics for OWL. Allows terminating exhaustive forward chaining.

OWL EL Based on "$\mathcal{EL}^{++}$". Shown to allow query answering by query rewriting after some amount of preprocessing.

- QL and RL "maximal" with these properties. EL originally defined for efficient classification.
- Query processors for these profiles still academic.
  - Google for "ontology-based data access" for work on OWL QL/DL-Lite.

The Future of D2R/D2RQ

- Last version of D2R, v0.7, is from August 2009
- No full-fledged alternatives yet
- W3C working draft on R2ML:

  `http://www.w3.org/TR/r2rml/`

- Very similar in most respects
- Only partial, experimental implementations so far
- Full implementations will come, and replace D2R