Web Data Integration

Web Data Formats
- Part 2 -
Outline

1. Web Data Formats Part I
   1. Character Encoding
   2. CSV
   3. XML

2. Web Data Formats Part II
   1. JSON
      1. Basic Syntax
      2. JSON in JAVA
   2. RDF
      1. RDF Data Model
      2. RDF Syntaxes
      3. RDF Schema
      4. SPARQL
      5. RDF in JAVA
2.1 JavaScript Object Notation (JSON)

• JavaScript
  • a popular programming language on the web
  • understood by Web browsers
  • originally:
    – simple interactions (e.g., image exchange on mouse over)
  • Nowadays:
    – also used for complex applications, Ajax (Asynchronous JavaScript and XML)
    – for instance used to implement Google Docs

– JSON
  – is a lightweight data-interchange format
  – easy to read and write for humans and machines
  – used as a less verbose alternative to XML
JavaScript Object Notation (JSON)

- **Basics:**
  - Objects are noted as in JavaScript
  - Objects are enclosed in curly brackets { … }
  - Data is organized in key value pairs separated by colons { key : value }

- **Example:**

  ```json
  { "firstname" : "John", 
    "lastname" : "Smith", 
    "age" : 46 }
  ```

- **Simple processing in JavaScript:**

  ```javascript
  var obj = JSON.parse(jsonString) ;
  var name = obj.firstname + " " + obj.lastname ;
  ```
The JSON Syntax

Source: json.org
Nested Objects in JSON

JSON

```json
{
  "firstname": "John",
  "lastname": "Smith",
  "age": 46,
  "employer": {
    "name": "Tech Inc.",
    "address": {
      "street": "Main St.",
      "number": 14,
      "city": "Smalltown"
    }
  }
}
```

XML

```xml
<firstname>John</firstname>
<lastname>Smith</lastname>
<age>46</age>
<employer>
  <name>Tech Inc.</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
</employer>
```
JavaScript Object Notation (JSON)

- JSON is a lot like XML
  - Data Model: Tree
  - Opening/closing tags/brackets

- Differences
  - More compact notation than XML
  - No id/ref – JSON data is *strictly* tree shaped
  - Less data types (only strings and numbers)
  - No schema language*
  - No query language*

*although people are working on that and there are various proposals.*
Processing JSON with Java

• **GSON**
  • Library for parsing and serializing JSON in Java
  • https://code.google.com/p/google-gson/

• **Class Definition**

```java
public class Person {
    private String firstname;
    private String lastname;
    private int age;
}
```

• **Object Deserialization**

```java
Person person = gson.fromJson(jsonString, Person.class);
```

• **Object Serialization**

```java
String json = gson.toJson(obj);
```
2.2 Resource Description Framework (RDF)

Graph data model designed for sharing data on the Web.

- Applications:
  - annotation of Web pages (RDFa)
  - publication of data on the Web (Linked Data)

- View 1: Sentences in form Subject-Predicate-Object (called Triples)
  
  „Chris works at University of Mannheim."

- View 2: Labeled directed graph
  - A set of RDF triples forms a graph.
RDF Concepts

• Resources
  – everything (a person, a place, a web site…) is a resource
  – are identified by URI References
  – may have one or more types (e.g. foaf:Person)

• Literals
  – are data values, e.g., strings and integers
  – may only be objects, not subjects of triples
  – may have a data type or a language tag

• Properties (Predicates)
  – Connect resources to other resources
  – Connect resources to literals
RDF as a Labeled Directed Graph

http://dws.uni-mannheim.de/person1

rdfs:label

"Christian Bizer"

http://dws.uni-mannheim.de/papers/paper1

dc:creator

http://dws.uni-mannheim.de/papers/paper1

dc:title

dc:subject

http://dbpedia.org/resource/RDFa

"The WebDataCommons Dataset Series"

Resource

Literal

predicate
The Role of URIs in RDF

- In a typical database or XML document, identifiers are used only internally.
  - They have no meaning outside the database/document.

- RDF uses URI’s as **global identifiers** for resources.
  - Hence, multiple data sets can refer to the same identifier.
  - Key benefit for data integration!

- **Advantage**
  - Global references between data items are possible (Linked Data).

- **Disadvantage**
  - RDF is rather verbose.
  - ➔ most syntaxes use QNames (e.g. dc:subject).
Language Tags and Data Types

- RDF literals may have language tags or data types (but not both).

- Examples:

  ex:Muenchen ex:hasName "München"@de .
  ex:Muenchen ex:hasName "Munich"@en .
  ex:Muenchen ex:hasPopulation "1356594"^^xsd:integer .
  ex:Muenchen ex:hasFoundingYear "1158-01-01"^^xsd:date .

- RDF uses the XML Schema data types.

- Be careful, the following three literals are different:
  - "München"
  - "München"@de
  - "München"^^xsd:string .
There are various syntaxes for serializing RDF graphs.

1. N-Triples and Turtle: Plain text syntaxes
2. RDF/XML: RDF serialization in XML
3. RDFa: Syntax for embedding RDF into HTML pages
4. RDF/JSON: RDF serialization in JSON
N-Triples and Turtle

- **N-Triples** is a line-based, plain text serialization format for RDF graphs.

  `<http://www.dws.uni-mannheim.de/teaching/wdi>`
  `<http://purl.org/dc/elements/1.1/subject>`

- **Turtle** extends N-Triples with QNames

```
@BASE <http://www.dws.uni-mannheim.de/teaching/>
@PREFIX dc: <http://purl.org/dc/elements/1.1/>
@PREFIX dbpedia: <http://dbpedia.org/resource/>
:wdi dc:title "Web Data Integration"@en .
```
RDF/XML

- XML-based serialization format for RDF.

- Defining resources:
  
  ```xml
  <rdf:Description rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi">
    <dc:creator>Heiko Paulheim</dc:creator>
  </rdf:Description>
  ``

- Resources with a type:
  
  ```xml
  <rdf:Description rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi">
    <rdf:type rdf:resource="http://www.dws.uni-mannheim.de/teaching/Lecture"/>
    <dc:creator>Heiko Paulheim</dc:creator>
  </rdf:Description>
  ``

- Alternative Notation:
  
  ```xml
  <dws:Lecture rdf:about="http://www.dws.uni-mannheim.de/teaching/wdi"
  xmlns:dws="http://www.dws.uni-mannheim.de/dws/teaching" />
  ```
2.3 RDF Schema

Language for defining RDF vocabularies.

- RDF schema provides for defining:
  - classes (that are used as types) and
  - properties (that are used as predicates)

- Example of an RDF schema vocabulary definition:
  
  ```
  dws:Teacher rdf:type rdfs:Class .
  dws:Course rdf:type rdfs:Class .
  dws:teaches rdf:type rdf:Property .
  ```

- Example of using the vocabulary:
  
  ```
  dws:ChrisBizer rdf:type dws:Teacher .
  dws:WebDataIntegration rdf:type dws:Course .
  dws:ChrisBizer dws:teaches dws:WebDataIntegration .
  ```
Classes and Properties may form Hierarchies

- **Sub-class Definition**
  
  \[ \text{dws:UniversityTeacher} \text{ rdfs:subClassOf dws:Teacher .} \]

- **Sub-property Definition**
  
  \[ \text{dws:CourseName} \text{ rdfs:subPropertyOf dc:title .} \]

- **Implication:** All dws:UniversityTeachers are also dws:Teachers

- **Multiple inheritance is allowed**
Domain and Range Definitions

• RDF Schema provides for defining domains and ranges of properties.
  
  dws:teaches rdf:type rdf:Property .
  dws:teaches rdfs:domain dws:Teacher .
  dws:teaches rdfs:range dws:Lecture .

• Implications:
  • All resources that have a dws:teaches property are of rdf:type dws:Teacher.
  • All objects of dws:teaches triples are of rdf:type dws:Lecture.

• Domains and ranges are inherited to sub-properties.
Purpose of RDF Schema

• Recap: XML Schema defines allowed structures
• RDF Schema does \textit{not} constrain anything

• Purpose of XML Schema
  – validation of XML documents

• Purpose of RDF Schema
  – machine interpretation of RDF data
  – Inference of additional triples
  – \textit{NOT} validation
Purpose of RDF Schema

- Given the RDF schema
  
  dws:Teacher rdfs:subClassOf dws:Person .
  dws:teaches rdfs:domain dws:Teacher .
  dws:teaches rdfs:range dws:Lecture .

- and the single statement
  
  dws:ChrisBizer dws:teaches
  dws:WebDataIntegration .

- A machine could infer (interpret) that
  
  dws:ChrisBizer rdf:type dws:Teacher .
  dws:ChrisBizer rdf:type dws:Person .
  dws:WebDataIntegration rdf:type dws:Lecture .

- OWL (Web Ontology Language)
  
  - provides for more expressive definitions and inferences
  - see lecture: Semantic Web Technologies
2.4 SPARQL

Language for querying RDF graphs.

- Queries are expressed in the form of triple patterns.
- Query results are tabular and given as XML, JSON, or CSV.
- The SPARQL Protocol is used to query remote endpoints.
- Example query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person .
  ?person foaf:name ?name .
}
```
Triple Pattern Matching

RDF Graph

![RDF Graph Diagram]

Triple Pattern

```
foaf:made
```

```
dbpedia: The_Beatles
```

```
dc:title
```

```
"Help!"
```

```
dc:title
```

```
"Abbey Road"
```

```
dc:title
```

```
"Let It Be"
```

Query Result

<table>
<thead>
<tr>
<th>?album</th>
<th>?title</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://...">http://...</a></td>
<td>&quot;Help!&quot;</td>
</tr>
<tr>
<td><a href="http://...">http://...</a></td>
<td>&quot;Abbey Road&quot;</td>
</tr>
<tr>
<td><a href="http://...">http://...</a></td>
<td>&quot;Let It Be&quot;</td>
</tr>
</tbody>
</table>

Source: EUCLID - Querying Linked Data
Optional Triple Patterns

- Declaring triple patterns as OPTIONAL allows you to get query results even if only a subset of the patterns matches.

\[
\text{WHERE } \{ \text{A OPTIONAL } \{ \text{B} \} \} \]

- Keep all solutions from A whether or not there is a matching solution for B.

- Important for querying endpoints with a lot of missing values.

- Example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth ?death
WHERE {
  ?person foaf:name ?name .
  OPTIONAL { ?person dbo:deathDate ?death . }
}
```
FILTER Clauses

- FILTER clauses eliminate solutions that do not cause an expression to evaluate to true.

- Example

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth
WHERE {
    ?person foaf:name ?name .
    FILTER (?birth < "1900-01-01"^^xsd:date) .
}
```

- Comparators:  =  !=  <  >  <=  >=

- Logical Operators:  &&  ||  !

- Functions: SUBSTR(), regex(), month(now()), isURI(), …
  - More functions: http://www.w3.org/TR/sparql11-query/#SparqlOps
Solution Modifier

- Sort results

  ORDER BY ?name

- Restrict number of results

  LIMIT 100

- Page over result list

  LIMIT 100
  OFFSET 0

  LIMIT 100
  OFFSET 100

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name ?birth
WHERE {
  ?person foaf:name ?name .
}
ORDER BY ?name
LIMIT 10
OFFSET 100
Exercise: Querying DBpedia

- Query tool
  - http://dbpedia.org/snorql/

- Example Queries
  - http://wiki.dbpedia.org/OnlineAccess

- Query 1: What is the population and area code of Mannheim?
  - http://dbpedia.org/resource/Mannheim

- Query 2: Find all German cities that have a population of more than 100,000 people?
2.5 Processing RDF in Java: Jena

• Jena is a popular framework for processing RDF in Java
• Download: https://jena.apache.org/
• Capabilities
  – Support for various RDF syntaxes
  – SPARQL support
  – Support for RDF Schema and OWL inference
  – Various storage back ends
• Central concepts
  – Model (i.e., RDF graphs): `class Model`
  – Resource: `class Resource`
Processing RDF in Java: Jena

- Read a graph from a URL (or local file):
  ```java
  model.read("http://dbpedia.org/resource/Mannheim");
  ```

- Navigating through a model
  ```java
  Resource mannheim = 
  model.getResource("http://dbpedia.org/resource/Mannheim");

  Literal areaCode = mannheim.getProperty("http://dbpedia.org/ontology/areaCode")
  .getLiteral();
  ```
String queryString = "SELECT ?x ...";
Query query = QueryFactory.create(queryString);
QueryExecution qe =
    QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
while(results.hasNext()) {
    QuerySolution sol = results.next();
    String s = sol.get("x").toString();
    ...
}
Querying a Public SPARQL Endpoint

- Many RDF data sources provide SPARQL endpoints
  - e.g. DBpedia, DBLP, BBC, MusicBrainz, Linked Movies Database, ...
  - List of public endpoints: https://www.w3.org/wiki/SparqlEndpoints

- Access with Jena

  ```java
  String query = "SELECT ...";
  String endpoint = "http://dbpedia.org/sparql";
  Query q = QueryFactory.create(strQuery);
  QueryExecution qexec =
      QueryExecutionFactory.sparqlService(endpoint, q);
  ResultSet RS = qexec.executeSelect();
  ```
Wrap-up: Web Data Formats

• Data is published on the Web in various formats
  – CSV
  – XML
  – JSON
  – RDF
  – …

• The data formats provide us with syntaxes for exchanging data.

• They do not solve the actual data integration challenges:
  1. Do two records describe the same real-world entity?
  2. Which elements in different schemata have the same meaning?
  3. Which data values from different sources should I trust?

• These challenges will be the topics of the upcoming lectures.
3. References

- Standards and specifications
  - JSON: http://www.json.org/
  - RDF: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/
  - RDF Schema: http://www.w3.org/TR/rdf-schema/
  - SPARQL: http://www.w3.org/TR/sparql11-overview/

- Tutorials
  - GSON: http://code.google.com/p/google-gson/
  - RDF: http://www.w3schools.com/webservices/ws_rdf_intro.asp
  - JENA: http://jena.apache.org/documentation/
  - Euclid Curriculum: http://www.euclid-project.eu/