Knowledge representation

lecture 3: RDFa, RDFs, OWL and rules

Tanel Tammet

TTU

Lecture overview

- Recall RDF
- Html annotations:
    - RDFa
    - Microformats
    - Facebook & Google
    - schema.org
- RDFs
- OWL

Triplets: an obvious idea to Implement schemaless databases

Each row with N cols is represented as N rows of three columns, called sometimes as

Row/Object id	Column name	Value
Object	Property	Value
Subject	Predicate	Object

Similar to key-value

Object Property Value

can be combined to

Object:Property Value

RDF: triples with value types

Object Property **Value** **Valuetype**

With valuetype normally being either:

- One of xml schema datatypes
- Global id: URI
- Local id

How to add metadata to a row?

Like timestamp, changer, row id, status etc etc?

Horrible answer: **reification**

The ugly head of reification

We have

personid:12 salary 20000

Want to add timestamp and entering person?

The reification way

From

personid:12 salary 2000 + timestamp etc

To

datarow:10001	subject	personid:12
datarow:10001	predicate	salary
datarow:10001	object	2000
datarow:10001	timestamp	2009-10-20 13:45
datarow:10001	modifier	personid:345

From the relational db ...

One row, N cols in the relational db

First, get N rows of four cols in RDF

Second, get (N*3)+X rows of four cols after reification

N → 12*N

Problem with containers

RDF provides a *container vocabulary* consisting of three predefined types (together with some associated predefined properties).

A *container* is a resource that contains things. The contained things are called *members*. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers:

rdf:Bag

rdf:Seq

rdf:Alt

Problem with containers

Containers are a fake:

- Containers have no real semantics in RDF
- Container semantics would make calc hard.

Problem with local id-s

Different object id-s:

- Global URI-s.
  - These are fine.
- Local “blank nodes”.
  - Their semantics/use in the RDF spec is broken: creates unnecessary problems.

Storage of RDF in a relational db

Predicate, subject, valuetype URI-s: how to store?

Option 1: keep strings.

Option 2:

- keep a separate table for unique strings
- use numeric string id-s in pred,subject,valuetype

„Semantic“ html: two meanings

First meaning:

- A semantic element clearly describes its meaning to both the browser and the developer.
- Examples of **non-semantic** elements: <div> and <span> - Tells nothing about its content.
- Examples of **semantic** elements: <form>, <table>, and <img> - Clearly defines its content.

Second meaning (the subject of this lecture):

- Encode program-processable data (a la name, age, cost, id etc) into html by a special annotation mechanism.

Semantic html annotations

Big question: how to publish machine-processable data (as opposed to visual-oriented html) on the web

Two main approaches:

- Publish data in csv, xml, json, rdf or some such purely machine-oriented format
- Publish data inside human/visual-oriented html

Semantic html annotations: integrate machine-processable information into visual-design-oriented html

Semantic html „standards“

Important formats:

- HTML5 **data attributes**
- **RDFa**: W3C standard pushed by the semantic web community
  - Facebook Open Graph protocol
  - Google Structured data

- **Microdata**
- **Microformats**

- Important ontologies/dictionaries:

- **schema.org**
- **wordnet**

HTML5 data attributes

Simplest of all!

Just add data-*=“...“ attributes where * can be anything and ... can be anything:

Like:

```
<div  data-name=“jaan“ data-age=“20“ data-fun=“not much“>
    ordinary text
</div>
```

Can use in css selectors, javascript, external software etc exactly as you like: no real „pre-defined“ meaning.

HTML5 data attributes

Dataset API example:

Assuming element:
<div id="myDiv" data-name="myDiv" data-id="myId" data-my-custom-key="This is the value"></div>

Get the element
var element = document.getElementById("myDiv");

Get the id
var id = element.dataset.id;

Retrieves "data-my-custom-key"
var customKey = element.dataset.myCustomKey;

Sets the value to something else
element.dataset.myCustomKey = "Some other value";

Element becomes:
<div id="myDiv" data-name="myDiv" data-id="myId" data-my-custom-key="Some other value"></div>

W3C standard: RDFa

Read:

http://en.wikipedia.org/wiki/RDFa
http://www.w3.org/TR/xhtml-rdfa-primer/

History:

2008: RDFa 1.0 reached W3C Recommendation status
2012: RDFa 1.1 does not require XML-specific namespace mechanism

Example 1:

Initial html:

```
<tr>
    <td>Jaanus Kask</td>
    <td>6024554</td>
    <td><a href=”http://kask.googlepages.com”>click here</a></td>
</tr>
```

annotated using RDFa:

```
<tr about=”#jaanus_kask”>
    <td property=”er:name”>Jaanus Kask</td>
    <td property=”er:phone”>6024554</td>
    <td><a rel=”er:homepage”
                href=”http://kask.googlepages.com”>click here</a></td>
</tr>
```

Example 2:

Initial html:

```
    <div>
    <a class="tootaja_nimi"
href="http://www.mkm.ee/index.php?id=7187&amp;tootaja=10000450">
    Taivo Kivistik</a></div>
    <div class="tootaja_inf">
    asekantsler (õigusala)
    <br>Tel:  6256346 | E-post:
    <a class="kontakt_mail" href="javascript:void(0)"
    onclick="sendmail('taivo.kivistik', 'mkm.ee')"
    >taivo.kivistik
    <img src="majandus_juhtkond_files/e_post.gif"
    class="e_post_gif" border="0">mkm.ee</a> <br></div><br>
```

Example 2:

annotated using RDFa:

```
<span about="#Taivo_Kivistik">
      <div>
      <a class="tootaja_nimi" rel="er:homepage"
href="http://www.mkm.ee/index.php?id=7187&amp;tootaja=10000450">
      <span property="er:name">Taivo Kivistik</span></a></div>
      <div class="tootaja_inf">
      <span property="er:job">asekantsler (õigusala)</span>
      <br>Tel:  <span property="er:home">6256346</span> | E-post:
      <a class="kontakt_mail" href="javascript:void(0)"
      onclick="sendmail('taivo.kivistik', 'mkm.ee')" property="er:email"
      content="taivo.kivistik@mkm.ee">taivo.kivistik
      <img src="majandus_juhtkond_files/e_post.gif"
      class="e_post_gif" border="0">mkm.ee</a> <br></div><br>
</span>
```

RDFa more concretely:

This should mostly suffice:

object id:	**about**=”#local_id”
link property:	**rel=**”er:propertyname”
property:	**property**=”er:propertyname”
type tüüp:	**type**=”xsd:typename”
Property value:	normally present in html
Alternatively property value:	**content=**”value”

Facebook & RDFa: Open Graph protocol

https://developers.facebook.com/docs/opengraphprotocol/

```
<meta content="Sightsmap" property="og:title">
<meta content="http://www.sightsmap.com" property="og:url">
<meta content="Sightsmap" property="og:site_name">
<meta content="website" property="og:type">
<meta content="Sightseeing popularity heatmaps for the whole world, based
    on Panoramio photos, Wikipedia and FourSquare."
    property="og:description">
<meta content="http://www.sightsmap.com/wpng/siteimage_small_150.jpg"
    property="og:image">
<meta content="tanel.tammet" property="fb:admins">
<meta content="330325837036787" property="fb:app_id">
```

Google & RDFa:
Rich snippets

http://support.google.com/webmasters/bin/topic.py?hl=en&topic=1088472&parent=21997&ctx=topic

Use either:

- RDFa
- Microdata
- Microformats

Microdata:

```
<body itemtype="http://schema.org/WebPage" itemscope="">
```

Or a longer example:

```
<div itemscope itemtype="http://data-vocabulary.org/Person">
  My name is <span itemprop="name">Bob Smith</span>
  but people call me <span itemprop="nickname">Smithy</span>.
  Here is my home page:
  <a href="http://www.example.com" itemprop="url">www.example.com</a>
  I live in Albuquerque, NM and work as an <span itemprop="title">engineer</span>
  at <span itemprop="affiliation">ACME Corp</span>.
</div>
```

Compare RDFa & microdata

RDFa:

```
<div xmlns:v="http://rdf.data-vocabulary.org/#" typeof="v:Person">
  My name is <span property="v:name">Bob Smith</span>,
  but people call me <span property="v:nickname">Smithy</span>.
  Here is my homepage:
  <a href="http://www.example.com" rel="v:url">www.example.com</a>.
  I live in Albuquerque, NM and work as an <span property="v:title">engineer</span>
  at <span property="v:affiliation">ACME Corp</span>.
</div>
```

Microdata:

```
<div itemscope itemtype="http://data-vocabulary.org/Person">
  My name is <span itemprop="name">Bob Smith</span>
  but people call me <span itemprop="nickname">Smithy</span>.
  Here is my home page:
  <a href="http://www.example.com" itemprop="url">www.example.com</a>
  I live in Albuquerque, NM and work as an <span itemprop="title">engineer</span>
  at <span itemprop="affiliation">ACME Corp</span>.
</div>
```

Microformats

Google example: data encoded in **class=„...“** attributes, here concretely **vcard format** data

```
<div class="vcard">
  <img class="photo" src="www.example.com/bobsmith.jpg" />
  <strong class="fn">Bob Smith</strong>
  <span class="title">Senior editor</span> at <span class="org">ACME Reviews</span>
  <span class="adr">
    <span class="street-address">200 Main St</span>
    <span class="locality">Desertville</span>, <span class="region">AZ</span>
    <span class="postal-code">12345</span>
  </span>
</div>
```

Schema.org

Created by: Google, Microsoft, and Yahoo

Goal: create a shared **markup vocabulary** supported by major search engines

See also: http://www.sitemaps.org/

Compare to: wordnet (a very different animal)

See additionally:

http://googlewebmastercentral.blogspot.com/2012/12/introducing-data-highlighter-for-event.html
http://googlewebmastercentral.blogspot.com/2012/07/introducing-structured-data-dashboard.html

RDFS

RDFS: RDF Schema

Three kinds of simple taxonomy rules added to RDF

„if X is a car, X is a vehicle“

„if X has a property profession, X is a person“

„if X has a property brother, value of the property is a person“

First (main) rule

example:

ex:MotorVehicle   rdf:type   rdfs:Class

exthings:companyCar   rdf:type   ex:Van

ex:Van   rdfs:subClassOf   ex:MotorVehicle

Main rule

same facts in the predicate calculus notation:

rdf:type(ex:MotorVehicle, rdfs:Class)

rdf:type(exthings:companyCar,ex:Van)

rdfs:subClassOf(ex:Van, ex:MotorVehicle)

built-in rule assumed in rdfs:

rdf:type(X,Y) & rdfs:subClassOf(Y,Z) => rdf:type(X,Z)

Second rule

```
ex:Person   rdf:type     rdfs:Class .
ex:author   rdf:type     rdf:Property .
ex:author   rdfs:range   ex:Person
```

and the fact

```
ex:person1  ex:author  ex:hamlet
```

should derive:

```
ex:author rdf:type  ex:Person
```

Logically …

built-in rule assumed in rdfs:

Y(X,Z) & rdfs:range(Y,U) => rdf:type(X,U)

Third rule

```
ex:Book     rdf:type      rdfs:Class .
ex:author   rdf:type      rdf:Property .
ex:author   rdfs:domain   ex:Book .
```

and the fact

```
ex:person1  ex:author  ex:hamlet
```

should derive:

```
ex:hamlet  rdf:type  ex:Book
```

Logically …

built-in rule assumed in rdfs:

Y(X,Z) & rdfs:domain(Y,U) => rdf:type(Z,U)

Additional encoding layer!

built-in rule assumed in rdfs:

rdf:type(X,Y) & rdfs:subClassOf(Y,Z) => rdf:type(X,Z)
Y(X,Z) & rdfs:range(Y,U) => rdf:type(X,U)
Y(X,Z) & rdfs:domain(Y,U) => rdf:type(Z,U)

have to be encoded in classical 1st order logic as

rdf(X,rdf:type,Y) & rdf(Y,rdfs:subClassOf,Z) => rdf(X,rdf:type,Z)
rdf(X,Y,Z) & rdf(Y,rdfs:range,U) => rdf(X,rdf:type,U)
rdf(X,Y,Z) & rdf(Y,rdfs:domain,U) => rdf(Z,rdf:type,U)

Reification

Example:

- Fact: http://xx#121 ex:firstname "Jaan"
- Metainfo about the fact:
  timestamp: 10jaan2008,
  enteredby:peeter,
  trustlevel:0.9

Example encoded as triples (invent object "13"):

13	rdf:object	http://xx#121
13	rdf:predicate	ex:firstname
13	rdf:subject	"Jaan"
13	ex:timestamp	10jaan2008
13	enteredby	peeter

Reification rule

built-in rule assumed in rdf:

```
  rdf(X,rdf:subject,Y) &
  rdf(X,rdf:predicate,Z) &
  rdf(X,rdf:object,U)

  =>

  rdf(Y,Z,U).
```

NOT provided in rdfs:

- cardinality constraints on properties, e.g., that a Person has exactly one biological father.
- specifying that a given property (such as ex:hasAncestor) is transitive, e.g., that if A ex:hasAncestor B, and B ex:hasAncestor C, then
  A ex:hasAncestor C.
- specifying that a given property is a unique identifier (or key) for instances of a particular class.
- specifying that two different classes (having different URIrefs) actually represent the same class.
- specifying that two different instances (having different URIrefs) actually represent the same individual.

NOT provided in rdfs:

- specifying constraints on the range or cardinality of a property that depend on the class of resource to which a property is applied, e.g., being able to say that

- for a soccer team the ex:hasPlayers property has 11 values,
- for a basketball team the ex:hasPlayers property has 5 values.

- the ability to describe new classes in terms of combinations (e.g., unions and intersections) of other classes, or to say that two classes are disjoint (i.e., that no resource is an instance of both classes).

Ways to extend rdfs

Two main approaches:

- Using traditional rules.
  - Current mainstream: RIF (Rule Interchange Format).
  - Older favorite: RuleML (Rule Meta-Language)
  - A complex evolving standard: CL (Common Logic)
- Using a specialised description-logic based language

  OWL (Web Ontology Language)