INFO216: **Advanced Modelling**

Theme, spring 2017: **Modelling and Programming the Web of Data**

Andreas L. Opdahl
<Andreas.Opdahl@uib.no>

Session 4: Application architecture

- Themes:
  - application architecture for the web of data
  - components of web-of-data applications
    - programming against TDB in Jena
  - basic OWL concepts (“RDFS Plus”)

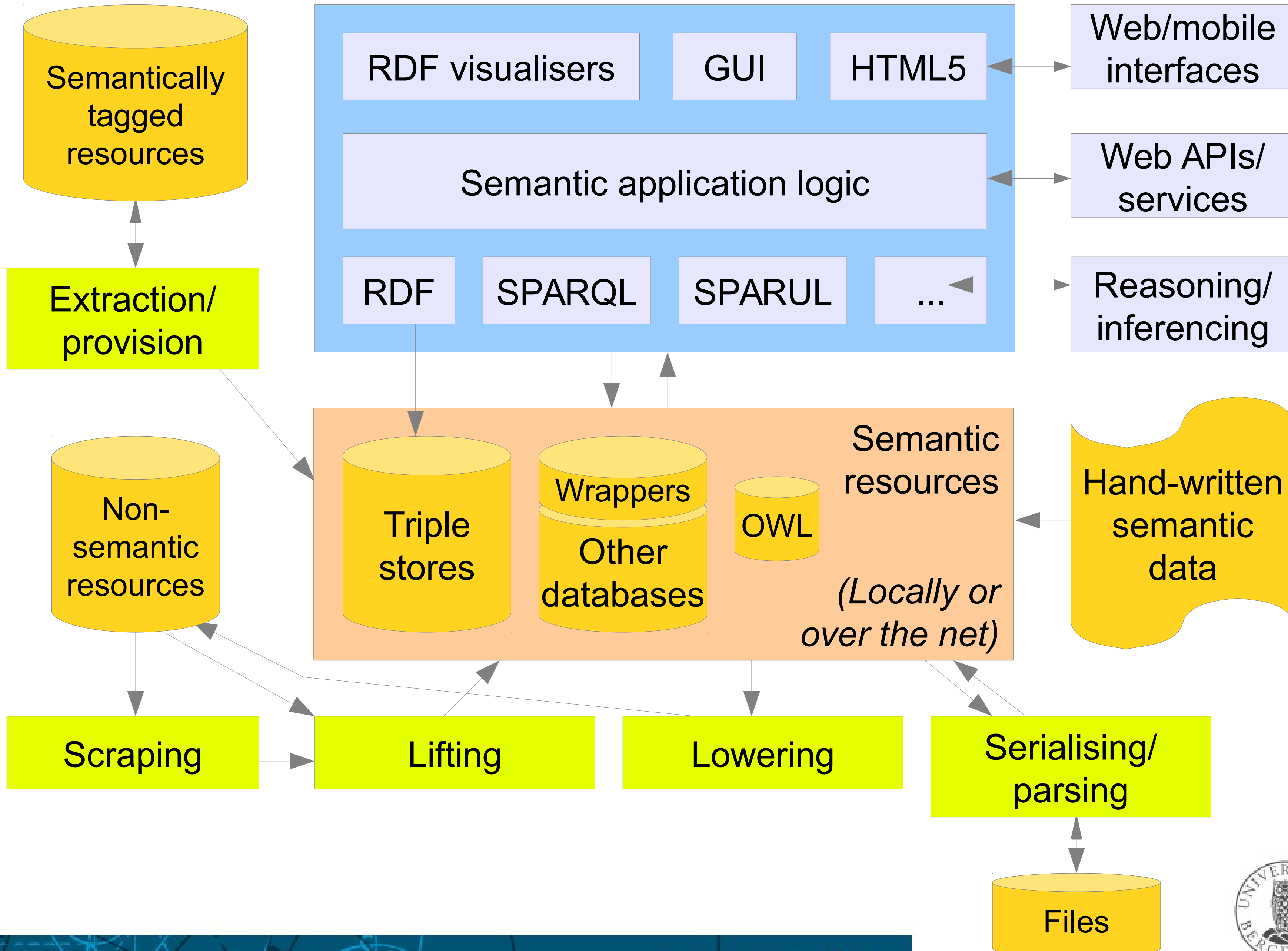
Readings

- Sources:
  - Allemang & Hendler (2011):
    Semantic Web for the Working Ontologist,
    - chapter 4 on application architecture
    - chapter 8 on RDFS Plus
  - materials at wiki.uib.no/info216

Expectations to the meeting Tuesday 7th

- Alone or in groups of 2-3
- Which data sets will you use?
- Which vocabularies will you use?
- What will you use them for?
  - something that cannot be done today
  - something that is harder to do today
  - something that is harder to do flexibly today
- You may bring several alternatives
  - but make sure you have a clear favourite

Application architecture for the Web of Data


Semantically tagged resources
RDF visualisers
GUI
HTML5
Web/mobile interfaces
Semantic application logic
Web APIs/ services
Extraction/ provision
RDF
SPARQL
SPARUL
...
Reasoning/ inferencing
Semantic resources
Triple stores
Wrappers
Other databases
OWL
(Locally or over the net)
Hand-written semantic data
Non-semantic resources
Scraping
Lifting
Lowering
Serialising/ parsing
Files

Parsing/serialising

- Reading from ("parsing") and writing to ("serialising") standard RDF formats
- Why different formats?
  - compactness, XML-dependency
  - can the same data set be stored in many ways?
  - machine versus human readability, abbreviations
  - CURIEs ("Compact URIs") with qname/prefix
  - nested resources
  - scope: only basic RDF or also, e.g., quads, rules , OWL...
- Built into all RDF- (and OWL-) programming frameworks
  - e.g. Jena

Example: TURTLE

- *"Terse RDF Triple Language"*
  - *extends the N-Triple format*
  - *restricts the Notation 3 (N3) -format*
  - *not XML-based (like RDF/XML), but simpler to read*
  - *supports prefixes (and bases)*
  - *writing multiple predicates-objects for the same subject*
  - *writing multiple objects for the same subject-predicate*
  - *flexible notations for blank/anonymous nodes: [], [...]*
  - *TriG extends TURTLE to support named graphs/quads*
  - *SPARQL uses TURTLE-like syntax*
  - *OWL is sometimes written in TURTLE*
    - *but OWL also has its own notations!*

Example: TURTLE

N-TRIPLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Martha_Louise> .

TURTLE:

```
<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> ;
                      <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> ,
                                          <http://r.e.x/Martha_Louise> .
```

- *semicolon (;) means “new predicate, same subject”*
- *comma (,) means “new object, same subject, predicate”*
- *period (.) means “new subject”*

Example: TURTLE

TURTLE:

```
@prefix rex: <http://r.e.x/> .
<rex:Harald> <rex:ektefelle> <rex:Sonja> ;
             <rex:barn> <rex:Haakon_Magnus> ,
                        <rex:Martha_Louise> .
```

- *@prefix allows use of Compact URIs ("Curies")*
- *@base allows use of IRI-fragments*
- *we will look at blank/anonymous nodes later...*

Example: TriG

TriG:

```
@prefix rex: <http://r.e.x/> .
<rex:Royal>   { <rex:Harald> <rex:spouse> <rex:Sonja> ;
                          <rex:kid> <rex:Haakon_Magnus> ,
                                    <rex:Martha_Louise> .  }
<rex:Mine>   { <reg:Andreas> <reg:spouse> <reg:Margareth> ;
                          <reg:kid> <reg:Jens_Christian> .  }
```

- *extends Turtle with named graphs wrapped in { ... }*

Important RDF serialisations

- RDF/XML *(the original XML serialisation)*
- TriX *(XML-based, experimental, named graphs)*
- N-TRIPLE *(maximally simple format, has "canonical form")*
- NQ, NQUAD *(extends N-TRIPLE with quads)*
- TURTLE ("Terse RDF Triple Language")
  (builds on N-TRIPLE, human readable, SPARQL ++)
- TriG *(TURTLE-extension, named graphs)*
- Notation3, N3 *(builds on TURTLE, supports rules, graphs ++)*
- JSON-LD *("JavaScript Object Notation – Linked Data")*
- embedded formats:
  - microformats, (eRDF →) RDFa, microdata
- In addition, OWL has its own serialisations...
  - RDF/XML and TURTLE are sometimes used

Example: JSON

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

Member

Object

Name

Value

JavaScript Object Notation (JSON)
www.json.org

Example: JSON

This is the person's id!

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

http://xmlns.com/foaf/0.1/name

http://xmlns.com/foaf/0.1/workplaceHomepage

http://xmlns.com/foaf/0.1/Person

How to represent semantic data in JSON?

Example: JSON-LD

This is the person's id!

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

http://xmlns.com/foaf/0.1/name

http://xmlns.com/foaf/0.1/workplaceHomepage

http://xmlns.com/foaf/0.1/Person

```
{
  "@type" : "http://xmlns.com/foaf/0.1/Person",
  "@id": "http://me.markus-lanthaler.com",
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
       { "@id" : "http://www.tugraz.at/" }
}
```

www.json-ld.org
Topic of lecture S05...

Scraping

- Making less structured data locally available in a well-structured format
- Typically used on internet data:
  - from less to more explicitly structured formats
  - HTML, PDF, DOCX, TXT, tagged file formats
- Storing the result in, e.g., CSV, XML or JSON
- A useful "technical craft"
  - not our focus
  - using scripts, regular expressions
  - *check what others have done before (jsoup)!*
  - *think continuous process – not once-off conversion!*

Semantic lifting

- Making structured data semantic
  - ...important for us
- Often the next step after scraping
  - ...or in parallel with scraping
  - storing the result in, e.g., RDF, RDFS, OWL...
- Tasks:
  1. creating triples *(make everything (s, p, o)-triples)*
  2. creating graphs *(one or several?)*
  3. selecting IRIs *(standard IRIs as identifiers)*
  4. selecting vocabularies *(standard IRIs as predicates)*
  5. selecting types *(standard IRIs as resource types)*
  6. external linking *(owl:sameAs)*

Extraction

- Retrieving RDF triples from (semantically) tagged resources
  - e.g., microformats, (eRDF ->) RDFa, microdata
- Replaces scraping + lifting
  - but is much simpler
    - the tags already do much of the job
  - open-source code is often available

Triple stores

- Basic software for persistent triple stores
  - or: database management systems (DBMSs) for RDF triples
  - general DBMS properties and behaviours
- Examples:
  - *Apache Jena TDB* (simple, file based, RDF-centric)
  - *Eclipse RDF4J (Sesame)* (much used, RDF-centric)
  - *Ontotext GraphDB (OWLIM)* (RDF4J compatible)
  - *Stardog* (RDF4J compatible)
  - *OpenLink Virtuoso* (much used, supports multiple data models, large)

https://www.w3.org/wiki/LargeTripleStores

Why different triple stores?

- A few important properties:
  - capacity (a trillion triples (norsk: “billion”, 10^{12}))
  - performance, security
  - SPARQL version (1.0, 1.1, Update)
  - SQL dependency, supports other data models?
  - FLOSS, license, price
  - in memory / on file
  - local server or cloud-hosted
  - single- / multi-thread and -server
  - reasoning
  - programming language
  - built-in SPARQL or other endpoints?

Visualisation

- APIs:
  - general GUI APIs
  - graph drawing/editing APIs
- Cloud based:
  - graph and general visualisers
  - e.g., embedded in web pages
  - often SPARQL-based
    - a SPARQL query extracts the dataset
    - the SELECTed variables are used to draw
      - graphs, bar charts, pie charts...
    - e.g., *http://mgskjaeveland.github.io/sgvizler/*

Endpoints

- Providing access to semantic resources over the net using standard protocols
  - typically HTTP, SPARQL, RDF, XML, JSON
  - based on
    - pure RDF resources, or
    - “wrapped” resources, e.g., relational databases
- Also simple web interfaces for interactive use
  - e.g., SNORQL (http://dbpedia.org/snorql/), Fuseki

Wrappers

- Wrapping existing structured data resources to present them as semantic resources
  - often relational data
    - but also, e.g., spreadsheets, XML, JSON
  - on-demand (live) semantic lifting
    - attributes/columns are mapped to predicates
  - read-only or read+update?
  - handwritten or wrapper software
    - e.g., D2RQ (http://d2rq.org)
  - wrapped resources can be used locally
    - or made accessible through an endpoint

Three-level architecture

- Raw data sets:
  - available in a standard format
    - perhaps virtually
  - SPARQL end points, RDF files
- Abstract data representation (RDF):
  - graph of nodes and arrows
- Queries:
  - standard query languages
  - based on the abstract data representation
- *Enabled by the semantic technologies*

Basic OWL ("RDFS-Plus")

Inverse properties

- Properties can be each other's reverses (with subject and object swapped), e.g.,
  - rex:HaakonMagnus fam:hasParent rex:Harald .
  - rex:Harald fam:hasChild rex:HaakonMagnus .
- P1 owl:inverseOf P2:
  - fam:hasParent owl:inverseOf fam:hasChild .
  - owl:inverseOf owl:inverseOf owl:inverseOf .
  - owl:inverseOf a owl:ObjectProperty .
- Entailment rules:
  - if *P1 owl:inverseOf P2* then
    - *P2 owl:inverseOf P1 .*
  - if *S P1 O . P1 owl:inverseOf P2* then
    - O P2 S .

Symmetric properties

- Some properties are their own inverse, e.g.,
    - rex:Harald fam:marriedTo rex:Sonja .
    - rex:Sonja fam:marriedTo rex:Harald .
- P rdf:type owl:SymmetricProperty:
    - fam:marriedTo a owl:SymmetricProperty .
    - owl:inverseOf a owl:SymmetricProperty .
    - owl:SymmetricProperty rdfs:subClassOf owl:ObjectProperty .
- Entailment rules:
    - if *P a owl:SymmetricProperty* then
        - P owl:inverseOf P .
    - if *S P O . P a owl:SymmetricProperty* then
        - O P S .

Asymmetric, reflexive, irreflexive properties

- New in OWL2:
  - both *symmetric* and *asymmetric* properties:
    - fam:marriedTo rdf:type owl:SymmetricProperty .
    - fam:hasChild rdf:type owl:AsymmetricProperty .
    - *many properties are neither!*
  - both *reflexive* and *irreflexive* properties:
    - owl:sameAs rdf:type owl:ReflexiveProperty .
    - fam:hasChild rdf:type owl:IrreflexiveProperty .
    - *many properties are neither!*

Transitive properties

- Some properties can form chains so that the result is the property itself, e.g.:
  - rex:HaakonMagnus fam:hasAncestor rex:Harald .
  - rex:Harald fam:hasAncestor rex:Olav .
  - rex:HaakonMagnus fam:hasAncestor rex:Olav .
- P a owl:TransitiveProperty:
  - fam:hasAncestor a owl:TransitiveProperty .
  - rdfs:hasSubClass a owl:TransitiveProperty .
  - rdfs:hasSubProperty a owl:TransitiveProperty .
- Entailment rules:
  - “if *S P X . X P O . P a owl:TransitiveProperty* then
    - S P O .”

Functional properties

- Each subject *can only have one* object value for the functional property, e,g.,
  - fam:mother a owl:FunctionalProperty .
  - fam:birthdate a owl:FunctionalProperty .
  - owl:FunctionalProperty rdfs:subClassOf owl:Property .
- "Entailment rule":
  - if *S P O1 . S P O2 . P a owl:FunctionalProperty* then
    - *O1 owl:sameAs O2 .*
  - ...for owl:ObjectProperties
    - similar rule for owl:DatatypeProperties

Inverse functional properties

- Two different subjects cannot have the same object for an inverse functional property, i.e.,
  - fam:persNum a owl:InverseFunctionalProperty .
  - owl:FunctionalProperty
    owl:inverseOf owl:InverseFunctionalProperty .
- Inverse functional properties are *unique* for each individual
  - used for *identifiers* in OWL 1
  - OWL 2 has a built-in *owl:hasKey* property for identifiers:
    - similar to inverse functional properties
    - can only be used with OWL 2's *owl:NamedIndividuals*
    - ...not for anonymous *owl:Individuals*

Summary: more specific properties

- owl:inverseOf
- owl:SymmetricProperty, owl:AsymmetricProperty
- owl:ReflexiveProperty, owl:IrreflexiveProperty
- owl:TransitiveProperty
- owl:FunctionalProperty, owl:InverseFunctionalProperty
- owl:hasKey
- Also:
    - negated properties (later)
    - chained properties, e.g.:

```
fam:hasGrandparent
        owl:propertyChainAxiom
                ( :hasParent  :hasParent ) .
```

Individual equivalence

- Two individuals (with different IRI-s) may represent the same thing:
  - http://dbpedia.org/resource/Amanda_Plummer
  - http://yago-knowledge.org/resource/Amanda_Plummer
  - http://data.linkedmdb.org/resource/actor/34880
- I1 owl:sameAs I2:
  - owl:sameAs a owl:ReflexiveProperty .
  - owl:sameAs a owl:SymmetricProperty .
  - owl:sameAs a owl:TransitiveProperty .
- owl:sameAs is an *equivalence relation*:
  - because it is *reflexive*, *symmetric* and *transitive*

Unique Name Assumption (UNA)

- If two resources have different names, do they necessarily represent different things?
- RDF and OWL does *not* assume this!
    - *in RDF and OWL, we do not know whether resources with different names represent different things or not*
- We can use
    - owl:sameAs – two resources represent the same thing!
    - owl:differentFrom – they represent different things!
- Some ICT-languages and technologies use UNA, others do not!

Individual difference

- A *pair* of individuals with different names (IRI-s) may represent different things, e.g.,
  - cal:Spring owl:differentFrom cal:Summer .

Individual difference

- A *pair* of individuals with different names (IRI-s) may represent different things, e.g.,
  - cal:Spring owl:differentFrom cal:Summer .
- A *group* of individuals with different names (IRI-s) may represent different things, e.g.,
  - [a owl:AllDifferent] owl:distinctMembers (
    cal:Spring cal:Summer cal:Autumn cal:Winter
) .
  - *owl:AllDifferent* and *owl:distinctMembers* are special constructs in OWL
    - they must always be used together
  - ...corresponds to pairwise *owl:differentFrom* between *all* individuals in the *owl:distinctMembers*-list

Equivalent classes

- Two classes (with different IRI-s) represent the same class:
- C1 owl:equivalentClass C2:
    - owl:equivalentClass a owl:ReflexiveProperty .
    - owl:equivalentClass a owl:SymmetricProperty .
    - owl:equivalentClass a owl:TransitiveProperty .
- owl:equivalentClass is another *equivalence relation*:
    - it is *reflexive*, *symmetric* and *transitive*

Disjoint classes

- Some classes may not have the same individual as a member,
  - fam:Male owl:disjointWith fam:Female .
  - owl:disjointWith a owl:SymmetricProperty .
    - ...but it is *not* transitive
- I.e., no individual can have both classes as its rdf:type
  - ...corresponds to owl:differentFrom between *all* pairs of individuals in fam:Male and fam:Female
- Preferred in *formal* versions of OWL (no “punning”):
  - owl:Class owl:disjointWith owl:Property .
  - owl:Class owl:disjointWith owl:Individual .
  - owl:Property owl:disjointWith owl:Individual .

Equivalent properties

- Two properties (with different IRI-s) represent the same property:
- C1 owl:equivalentProperty C2:
  - owl:equivalentProperty a owl:ReflexiveProperty .
  - owl:equivalentProperty a owl:SymmetricProperty .
  - owl:equivalentProperty a owl:TransitiveProperty .
- owl:equivalentProperty is another *equivalence relation*:
  - it is *reflexive*, *symmetric* and *transitive*
- *Also disjoint properties:*
  - :hasParent owl:propertyDisjointWith :hasSpouse .

Summary: sameness and difference

- Individuals:
  - pairwise: owl:sameAs, owl:differentFrom
  - groupwise difference: owl:AllDifferent
- Classes:
  - pairwise: owl:equivalentClass, owl:disjointWith
  - groupwise difference: owl:AllDisjointClasses
- Properties:
  - pairwise: equivalentProperty, propertyDisjointWith
  - groupwise difference: owl:AllDisjointProperties
- Membership in the groups:
  - owl:distinctMembers *(preferred)* or owl:members