WIS Data
Semantic Web, RDF(S)

Geert-Jan Houben
Acknowledgements

• Slides are based on a set composed by Jeen Broekstra (with help of his friends :-) )
Contents

- Semantic Web
- RDF
- RDF Schema
- RDF querying
- OWL
Semantic Web
Semantic Web: Goals

• enable *machines* to process information on the Web in a meaningful way
  – allow more specific search than just keyword search
  – allow combination of information from different sources to derive new facts
  – help the user to better leverage the vast amount of information of the Web

• treat the entire Web as a vast *knowledge base*
Semantic Web

• **Metadata**
  – “data about data” to describe salient properties of any piece of information

• **Ontology**
  – a *formal, explicit* specification of a *shared* conceptualization
    • *formal*: machine-processable
    • *explicit*: concepts and how to use them explicitly defined
    • *shared*: consensual knowledge
Semantic Web

• Standards
  – in order to be able to freely share and reuse information, we need to speak the same language
  – a standardized ontology language is the key to having Web Information Systems reuse information from arbitrary sources

• Compromise
  – we are not dealing with closed systems, optimal engineering solutions are not always possible.
  – we need to provide for flexibility
    • allow for missing, incomplete, even inconsistent information.
    • open world assumption
What was XML again?

- **Syntax:**
  - angle brackets, elements and attributes, etc.

- **Data model:**
  - ordered, labeled tree

```xml
<country name="Netherlands">
  <capital name="Amsterdam">
    <areacode>020</areacode>
  </capital>
</country>
```
So why not just use XML?

- No agreement on:
  - structure
    - is country a:
      - object?
      - class?
      - attribute?
      - relation?
      - something else?
    - what does nesting mean?
  - vocabulary
    - is country the same as nation?

- Are the above XML documents the same?
- Do they convey the same information?
- Is that information machine-accessible?
Meta-data

alleviates

August 2 2006

August 2 2006

Meta-data
Meta-data in XML

August 2 2006

Meta-data in XML
Meta-data in RDF

August 2 2006

αλλεσιατεσ

IS-A

σψµπτοµσ

τρεατµεντ

δρυγ

αδµιστρατιον

<ναµε>

<δισεασε>

林克昌 根留台湾 可能增高

在寶監者熱心奔走之下，華裔名指揮家林克昌根留台湾的可行性又提升了幾分。兩廳院主任李炎、國家音樂廳籌備處處長黃前日親赴林克昌、石堅芳富所演譯，並提出多場客席邀約。此外，山泰省立交響樂團團長陳基邦也三度「下聘」，邀請林克昌赴台中綠峰，從八月十日起首演交，為期長達一個月。

在台灣許多公家樂團中，陳基邦是以實際行動表達對林克昌肯定的業界人士之一，曾多次公開表示對林克昌指揮才華的飲佩，而且幾乎每個樂季都邀請林克昌客席演出。

此外，林克昌上個月赴俄羅斯與頂尖的「俄羅斯國家管絃樂團」灌錄了柴可夫斯基晚期三大交響曲以及「羅密歐與茱麗葉」、「謝拉夫進行曲」、「義大利隨想曲」，最後的DAT母帶也在前兩天運回台灣。製作人楊志傑與林克昌試聽之後，都對錄音效果一尤其是實現感感到相當滿意，楊志傑估計呈現了七分林克昌指揮神韻。

俄羅斯國家管絃樂團首席布魯尼日前也讚譽林克昌的指揮藝術有三大特點：一是控制自如的彈性速度；二是強烈的動態對比；三是先知可呼喚的旋律處理。這些對於錄音師而言都是很大的挑戰。俄羅斯錄音師雖然採用多軌錄音，但定位、場面都有可觀之處。
RDF
RDF Schema
What is RDF?

• RDF
  – stands for Resource Description Framework
  – is a W3C Recommendation
    (http://www.w3.org/RDF)

• RDF is a data model
  – for representing metadata (data about data)
  – for describing the semantics of information in a machine-accessible way
RDF in detail: the data model

• **statements** are (subject, predicate, object) *triples*:
  – (Netherlands, hasCapital, Amsterdam)

• statements describe **properties** of **resources**

• **a resource is any object that can be pointed at by a URI:**
  – a document, a picture, a paragraph on the Web
  – a book in the library, 'real-world' objects
    • isbn://006251587X
URIs as means of decoupling

- All identifiers are URIs
  - Allows total decoupling of
    - document
    - thesaurus
    - meta-data

[<x> IsOf-Type <T>]

different owners & locations
RDF: linking statements

• The subject of one statement can be the object of another

• such collections of statements form a \textit{directed, labeled graph}
RDF syntax: XML

- RDF has an XML syntax that has a specific meaning:
  - every Description element describes a resource
  - every attribute or nested element inside a Description is a property of that resource

```xml
<Description about="http://www.countries.org/countries#Netherlands">
  <hasCapital resource="http://www.cities.org/cities#Amsterdam"/>
</Description>

<Description about="http://www.cities.org/cities#Amsterdam">
  <areacode>020</areacode>
</Description>
```

- Does this solve the structure problem?
- Does this solve the vocabulary problem?
RDF/XML syntax: just a syntax

<Description about="http://www.countries.org/countries#Netherlands">
    <hasCapital resource="http://www.cities.org/cities#Amsterdam"/>
</Description>

<Description about="http://www.cities.org/cities#Amsterdam">
    <areacode>020</areacode>
</Description>

<Description about="http://www.countries.org/countries#Netherlands">
    <hasCapital resource="http://www.cities.org/cities#Amsterdam"/>
    <areacode>020</areacode>
</Description>

<Description about="http://www.countries.org/countries#Netherlands">
    <hasCapital resource="http://www.cities.org/cities#Amsterdam"/>
    <areacode>020</areacode>
    <areacode>020</areacode>
</Description>

<Description about="http://www.countries.org/countries#Netherlands">
    <hasCapital resource="http://www.cities.org/cities#Amsterdam">
        <areacode>020</areacode>
    </hasCapital>
</Description>
RDF/XML syntax: namespaces

- like in 'normal' XML, you can define namespaces to disambiguate elements and attributes:

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:geo="http://www.geography.org/schema.rdf#"
  xmlns:words="http://www.dictionary.org/schema.rdf#">
  <rdf:Description rdf:about="#Netherlands">
    <geo:hasCapital rdf:resource="#Amsterdam"/>
    <words:hasCapital> N </words:hasCapital>
  </rdf:Description>
  <rdf:Description rdf:about="#Amsterdam">
    <geo:areacode>020</geo:areacode>
  </rdf:Description>
</rdf:RDF>

- (From now on, abbreviating URIs in the examples)
RDF(S) syntax: Turtle

- Non-XML syntax that is easier to read and write for humans, and more compact.

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix geo: <http://www.geography.org/schema.rdf#>.
@prefix words: <http://www.dictionary.org/schema.rdf#>.
@prefix countries: <http://www.countries.org/countries#>.
@prefix cities: <http://www.cities.org/cities#>.

countries:Netherlands a geo:Country;
    geo:hasCapital cities:Amsterdam;
    words:hasCapital "N".
cities:Amsterdam geo:areacode "020".
```
So what can we use this for?

• we can:
  – make explicit statements about web resources
  – have the machine
    • know that these are statements
    • know how the statements relate
    • compare values
• BUT we still miss a way to define a **vocabulary**:
  • should we use 'country' or 'nation'?
  • Is the Netherlands a country? Are there more countries? What properties can countries have?
RDF Schema

- **RDF** gives a data model for meta data annotation, and a way to write it down in XML, but it can not define the vocabulary for a domain.

- **RDF Schema** allows you to define vocabulary terms and the relations between these terms
  - It gives 'extra meaning' to particular RDF predicates and resources
  - this 'extra meaning', or semantics, define how a term should be interpreted
Some observations

• Classes and properties are modeled separately!
  – this is different from 'normal' Object-Oriented modeling where properties (attributes) are part of a class.

• Again: RDF Schema is 'just' RDF, but with some added meaning to particular terms.

• The line between levels is imaginary!
  – for example, it is possible to create some resource X that is both an instance (X type Car) and a class (X subClassOf Vehicle).
formal semantics

- Defines what other statements are *implied* by a given set of RDF(S) statements

- Ensures mutual *agreement on minimal content* between parties without further contact

- In the form of “entailment rules”

- Very *simple to compute* (and not explosive in practice)
RDF(S) semantics: examples

- Netherlands Type EuropeanCountry
  EuropeanCountry subClassOf Country
  ➔ Netherlands Type Country

- aspirin alleviates headache
  alleviates range symptom
  ➔ headache Type symptom
RDF(S) semantics: examples

- Нетерландσ Type ΕυροπεανΧουντρψ ΕυροπεανΧουντρψ subClassOf Χουντρψ
  ➔ Нетерландσ Type Χουντρψ

- ασπιριν αλλεσιατεσ ηεαδαχηε αλλεσιατεσ range σψμπτομ
  ➔ ηεαδαχηε Type σψμπτομ
RDF(S) semantics

- $X \, R \, Y + R \, \text{domain} \, T \rightarrow X \, \text{IsOfOfType} \, T$
- $X \, R \, Y + R \, \text{range} \, T \rightarrow Y \, \text{IsOfOfType} \, T$
- $T_1 \, \text{SubClassOf} \, T_2 +$
  $T_2 \, \text{SubClassOf} \, T_3 \rightarrow T_1 \, \text{SubClassOf} \, T_3$
- $X \, \text{Type} \, T_1 +$
  $T_1 \, \text{SubClassOf} \, T_2 \rightarrow X \, \text{IsOfOfType} \, T_1$
- $X \, \text{Type} \, \text{Class} \rightarrow X \, \text{SubClassOf} \, X$
Domain restrictions

- **GeographicEntity**
  - **subClassOf**
    - **Country**
      - **domain**
        - **population**
          - **Netherlands** → **16 million**
        - **domain**
          - **Continent**
            - **domain**
              - **Europe** → **500 million**
Domain restrictions

- Possible Solutions
  - moving the domain restriction 'up in the hierarchy'
  - creating separate properties, e.g. 'continent_population' and 'country_population'
  - not using a domain restriction at all
RDF Schema syntax

- Class definition
  
  ```
  <rdf:Description rdf:about="#Country">
    <rdf:type rdf:resource="#Class"/>
    <rdfs:subClassOf rdf:resource="#GeographicEntity"/>
  </rdf:Description>
  ```

- or shorter:
  
  ```
  <rdfs:Class rdf:about="#Country">
    <rdfs:subClassOf rdf:resource="#GeographicEntity"/>
  </rdfs:Class>
  ```

- Property definition
  
  ```
  <rdf:Property rdf:about="#hasCapital">
    <rdfs:domain rdfs:resource="#Country"/>
    <rdfs:range rdfs:resource="#Capital"/>
  </rdf:Property>
  ```
So why use RDF / RDFS?

• Because it's there!
  – RDF and RDF Schema provide a common agreement, an \textit{open standard} for annotating web resources and making their semantics explicit.
  – Technically speaking it's not the best possible solution, but a compromise
    • we trade in some convenience for \textit{interoperability}: the ability to communicate with \textit{arbitrary} partners based on the fact that we both use RDF
Ontology language?

• **Ontology**: a formal specification of a shared conceptualization

• RDF Schema allows:
  – specification
    (we have just seen that)
  – sharing
    (because it is an open, Web-based standard)
  – formality
    (RDF Semantics specify the meaning precisely)

• Is RDF Schema expressive enough?
What is still missing?

• Cardinality constraints
  – “a country can have exactly one capital”

• Conjunction, disjunction, negation, equivalence
  – “countries and cities are disjoint: something can not be both a city and a country”

• Localized constraints
  – “when the property 'population' is used on a city, its value must be between 20.000 and 10 million”

• A way to access this information!
  – having it written down is nice and all, but if you want to use it for question answering you need a query language (like SQL for databases)
RDF querying
A Query Language for RDF

• Question:
  – RDF has an XML syntax. Why not just use an existing XML Query Language, like XQuery or XSL?
The Netherlands

The Hague is the seat of government

Path expression:
/country/geography/capital/@name="Amsterdam"
Example in RDF

```xml
<rdf:Description rdf:about="#Netherlands">
  <rdf:type rdf:resource="#Country"/>
  <geo:hasCapital rdf:resource="#Amsterdam"/>
</rdf:Description>
```

XML path expression
`/rdf:Description/rdf:type[@rdf:resource="#Country"]/geo:hasCapital/@rdf:resource`

```xml
<geo:Country rdf:about="#Netherlands">
  <geo:hasCapital rdf:resource="#Amsterdam"/>
</geo:Country>
```

XML path expression
`/geo:Country/geo:hasCapital/@rdf:resource`
retrieving RDF relations

How do we find the relation between The Netherlands and the areacode '020' in a path expression?
Requirements for an RDF QL

• understand the **data model**
  – directed, labeled, unordered graphs
  – semi-structured

• **path expressions**
  – through the RDF graph, not the XML tree
  – specifying both node and edge labels

• **compositionality**
  – complex queries can be 'built up' by combining simpler queries.

• support for **RDF Schema**
  – understand the semantic of subClassOf, Class, etc.
SeRQL vs. SPARQL

• Both: expressive query and transformation language for RDF
  – SELECT and CONSTRUCT
  – optional path expressions
  – support for context/named graphs

• SeRQL ("circle")
  – nested queries, language tags, …
  – user-friendly syntax
  – very efficient Sesame implementation

• SPARQL ("sparkle")
  – W3C Standard (in progress)
    • tool interoperability: Jena, Redland, 3Store, Sesame, …
SeRQL vs. SPARQL

```
SELECT ?x ?y
WHERE {
  ?x geo:hasCapital ?y .
  FILTER (?z = "020").
}
```

```
SELECT X, Y
FROM {X} geo:hasCapital {Y} geo:areacode {Z}
WHERE Z like "020"
USING NAMESPACE geo = <http://www.geography.org/schema.rdf#> .
```
SeRQL path expressions

- \( \{X\} \) movie:hasPart { :role1 } 
- \( \{X\} \) movie:hasPart \{Y\} 
- \( \{X\} \) P \{Y\} 

Diagram:

```
movie1 ---- movie:hasPart ---- role1 ---- movie:characterName ---- “Edward ScissorHands”
```
Chaining, branching and comparing

- **Chaining:**
  - \{X\} movie:hasPart \{Y\} movie:characterName \{Z\}

- **Branching:**
  - \{Y\} rdf:type {movie:Role};
    
  movie:characterName \{Z\}

- **Comparison operators:**
  - **String comparison:**
    - Z like "*Hands"
  - **Boolean comparison:**
    - X < Y, X <= Y, Z < 20, Z = Y, etc.

```
movie1 movie:hasPart role1 movie:characterName "Edward Scissorhands"
```
SeRQL query composition

• Using the building blocks, we can compose complex queries.
• SeRQL uses a select-from-where syntax (like SQL):
  – select: the variables that you want to return
  – from: the path in the graph that you want to get the information from
  – where: additional constraints on the values using operators

```
SELECT X, Y
FROM {X} movie:hasPart {Y} movie:characterName {Z}
WHERE Z LIKE "edward scissorhands" IGNORE CASE
USING NAMESPACE movie = <http://example.org/movies/>
```
Optional path expressions

• RDF is semi-structured
  – Even when the schema says some object should have a particular property, it may not always be present in the data:
  • Users have names and email addresses, but Geert-Jan is a user without a known email address

```
<table>
<thead>
<tr>
<th>person001</th>
<th>foaf:firstName</th>
<th>type</th>
<th>um:User</th>
</tr>
</thead>
<tbody>
<tr>
<td>person002</td>
<td>foaf:firstName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>foaf:mbox</td>
<td></td>
<td><a href="mailto:j.broekstra@tue.nl">j.broekstra@tue.nl</a></td>
</tr>
<tr>
<td></td>
<td>foaf:firstName</td>
<td></td>
<td>Geert-Jan</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
Optional path expressions

To be able to query for all users, their first names, and *if known* their email address, SeRQL introduces optional path expressions:

```
SELECT
    Person, Name, Email
FROM
    {Person} rdf:type {um:User};
    foaf:firstName {Name};
    [foaf:mbox {Email}]
USING NAMESPACE
    foaf = <http://xmlns.com/foaf/0.1/>,
    um = <http://example.org/usermodel/>
```
CONSTRUCT queries

- CONSTRUCT-queries return RDF statements
  - each RDF statement matching the query pattern is returned
- The query result is
  - a subgraph of the original graph, or;
  - a transformed graph
- This mechanism is quite powerful and even allows formulation of simple rules
CONSTRUCTing subgraphs

• Retrieve for each movie the title and year as RDF Statements

```
CONSTRUCT * 
FROM {M} movie:year {Y};
    movie:title {T}
USING NAMESPACE 
movie = <http://example.org/ontology/movie/>`

Graph Transformations

• Retrieve for each movie the title and year as RDF Statements

CONSTRUCT {Y} my:inCountry {X}
FROM {X} geo:hasCapital {Y}
USING NAMESPACE
  movie = <http://example.org/ontology/movie/>,
  my = <http://example.org/mynamespace/>
The query result

• SeRQL select-queries return variable bindings
  – For each variable in the query, it gives a value.
  – The result is a table, where each column represents a variable and each row a set of values
• Result is returned as an RDF/XML document
Query result: example

- Query:
  return all capital cities

  \[
  \text{select } Y \\
  \text{from } \{X\} \text{ geo:hasCapital } \{Y\}
  \]

- Result:

  \[
  <\text{rs:ResultSet rdf:about=''} > \\
  <\text{rs:resultVariable}>Y</\text{rs:resultVariable}> \\
  <\text{rs:solution}> \\
  <\text{rs:ResultSolution}> \\
  <\text{rs:binding rdf:parseType='Resource'> \\
  <\text{rs:variable}>Y</\text{rs:variable}> \\
  <\text{rs:value rdf:resource='http://www.geo.com/cities#London'/> \\
  </\text{rs:binding}> \\
  </\text{rs:ResultSolution}> \\
  <\text{rs:ResultSolution}> ... \\
  </\text{rs:solution}> \\
  </\text{rs:ResultSet}>
  \]
Summary

• **RDF** is a simple graph model
  – has different syntaxes: RDF/XML, Turtle, ...

• **RDF Schema** is a Vocabulary Description Language
  – adds additional primitives (Class, subClassOf, domain, range) to RDF
  – allows description of domain terms and relations: a simple ontology language

• **SeRQL and SPARQL** are query languages for RDF/RDF Schema

• **Missing**: more expressive ontology modeling
  – For many modeling tasks, RDF Schema does not have sufficient power
Further material

- RDF Primer – an introduction into RDF
  [http://www.w3.org/TR/rdf-primer/](http://www.w3.org/TR/rdf-primer/)
- RDF Semantics specification
  [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)
- SeRQL query language manual
- SPARQL query language (CR)
  [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
- Sesame Framework – try some queries online!
OWL
Web Ontology Language
Extensions of RDFS

• Do we need more?

• Limitations of RDFS
  – only \textbf{simple} modeling primitives:
    Class, Property, subClassOf, subPropertyOf, type, domain, range
  – \textbf{not} very \textbf{expressive}

• Well known extensions:
  – Domain independent: OIL, DAML+OIL, OWL
  – Domain specific: FOAF, SKOS
Evolution of OIL into OWL

- Ontology Inference Layer **OIL** (2000)
  - developed by group of European researchers
- DAML Ontology Language **DAML-ONT** (2001)
  - developed by group of US researchers
- Efforts merged in **DAML+OIL** (2002)
  - further development by EU/US joint committee
  - tasked to develop W3C standard based on DAML+OIL
  - result is called **OWL**: Web Ontology Language (standard by February 2004)
Ideas behind O*L

• Three roots:
  – looks like frame based modeling languages (like UML, Java)
  – reasoning from Description Logic
  – founded in web languages (XML, RDF)

• Goals:
  – expressive enough
  – efficient reasoning support
  – usable on the Web
Additions of OWL wrt RDFS

- Local scope of properties
  - different characteristics in different classes
- Two meanings for properties
  - allValuesFrom (∀): if it has_daughter, it must be of type female (but it is still possible to have no children)
  - someValuesFrom (∃): every bike consist_of at least a wheel (it is not possible to be a bike without having at least a wheel)
- Cardinality of properties
  - an insect has has_leg min-cardinality (same for maximum cardinality and exact cardinality)
Additions of OWL wrt RDFS

• Characteristics of properties
  – inverse property: define the inverse relation
    (e.g. has_owner is inverse of owned_by)
  – symmetric: if A has-prop B, then also B has-prop A
    (e.g. is_married_with, borders_with)
  – transitive: if A has-prop B and B has-prop C, then also
    A has-prop C. (e.g. bigger_than, located_in)
  – functional: maximal one value per instance
    (e.g. has_mother, capital)
  – inverse-functional: unambiguous, the only possible
    value (e.g. is_mother_of)
Additions of OWL wrt RDFS

• Boolean expression of classes
  – disjunction: car or bike
  – conjunction: vehicle and status_symbol
  – negation: not animal

• Defined classes
  – not only necessary conditions: every lion eats meat
    • lion \( \Rightarrow \) eats meat
  – but also sufficient conditions: every person that eats fish nor meat is a vegetarian!
    • vegetarian \( \Leftrightarrow \) person eats (not (meat or fish))
Additions of OWL wrt RDFS

• Equivalence and difference
  – same **class** or **property**: car = automobile, or has_leader = has_head
  – same **individual**:  
    TUe = Technische Universiteit Eindhoven
  – different **individuals**: TUD ≠ TUe

• Data types
  – use **XML Schema** for data types (builtin + new)

```xml
<xsd:SimpleType name="lessThan200">
  <xsd:restriction base="xsd:PositiveInteger">
    <xsd:maxExclusive value="200">
    </xsd:maxExclusive>
  </xsd:restriction>
</xsd:SimpleType>
```
Role of richer meta-data

- `<name>` is unique
- every `<Xς>` has one `<work>`,
- one `<work>` can belong to multiple `<Xς>`’s
- `<ενισερσιτψ>` is a part-of `<work>`
OWL has formal semantics

• Meaning beyond words!
• Defined by mapping to very expressive Description Logic (DL)
  
  \[ \text{eats value (meat or fish)} = \exists \text{eats:meat} \cup \exists \text{eats:fish} \]

• Mapping is used to provide reasoning support from a DL system (e.g. FaCT, Pellet)
Benefits of formal semantics

• **Reason** about class membership, equivalence and inconsistency
  – herbivore ⇔ animal eats (plant or (part_of plant))
  – tree ⇒ plant
  – branch ⇒ part_of tree
  – leaf ⇒ part_of branch
  – giraffe ⇒ animal eats leaf
  – part_of = transitive

• now we can **derive** that:
  – giraffe ⇒ herbivore
Why reasoning support?

• Important
  – as design support tool
  – for large ontologies
  – with multiple authors
  – for integrating and sharing ontologies
• because it allows to
  – Establish inter-ontology relationships
  – Check for consistency
  – Check for (unexpected) implied relationships
• Shown useful for DB schema integration
• Can facilitate query answering
OWL Syntax

• OWL is extension of RDFS
  – same trick as before:
    • give meaning to some predicates
  – but: semantics should be compatible

• OWL builds on top of RDF-S

  1 OWL is defined as **RDFS extension**
    • extension = addition to meta-model
    • new constructs besides rdfs:Class etc.

  2 OWL primitives are related to RDFS
    • owl:Class $\subseteq$ rdfs:Class
OWL Syntax

• How are these extensions expressed?

• Global characteristics of properties:
  – two types of properties, subclasses of `rdf:Property`:
    • `owl:DatatypeProperty`: for datatype values
    • `owl:ObjectProperty`: for instance values
  – other characteristics are subclasses again
    • `owl:TransitiveProperty` ⊆ `owl:ObjectProperty`
    • `owl:SymmetricProperty` ⊆ `owl:ObjectProperty`
    • etc.

```xml
<owl:FunctionalProperty rdf:ID="husband">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:FunctionalProperty>
```
OWL Syntax

• Equivalence, disjointness, boolean expressions of classes:
  – via specific properties on rdfs:Class or rdf:Property (like rdfs:subClassOf)
    • boolean expressions: owl:unionOf, owl:complementOf, owl:intersectionOf
    • element
      - class
      - property
      - individual
      | equivalence         | disjointness       |
      |---------------------|--------------------|
      | owl:equivalentClass | owl:disjointWith   |
      | owl:equivalentProperty | -                |
      | owl:sameAs          | owl:differentFrom  |

<owl:Class rdf:ID="Man">
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
OWL Syntax

- Local characteristics of properties
  - via anonymous RDFS class `owl:Restriction`
  - the restriction class has properties:
    - `owl:onProperty`, and
    - `owl:cardinality` or `owl:allValuesFrom` or `owl:someValuesFrom` or `owl:hasValue`
  - class is made subclass of restriction

```xml
<owl:Class rdf:ID="Human">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has-parent"/>
      <owl:allValuesFrom rdf:resource="#Human"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
OWL Syntax

• Classes can be defined by enumeration
  – via owl:oneOf
  – also lists can be defined

• Each ontology starts with header
  – version information
  – imports
  – comments
  – compatibility

<owl:Ontology rdf:about=""">
  <rdfs:comment> ... </rdfs:comment>
  <owl:VersionInfo> ... </owl:VersionInfo>
  <owl:imports rdf:resource="http://.."/>
  <owl:priorVersion rdf:resource="http://.."/>
</owl:Ontology>
OWL Layered approach

• One size doesn’t fit all; complaints:
  – “some constructs are difficult to understand”
    • inverse functional, negation, disjunction
  – “language should be decidable”
  – “more expressiveness”

• Solution:
  – layered approach with extended capabilities,
    e.g.
    • simple, intuitive modeling, or
    • efficient reasoning, or
    • high expressiveness, or ...
OWL Layering

- Three language “layers” called:
  - OWL Full
  - OWL DL
  - OWL Lite

- “Layering” in two interpretations:
  - syntactic:
    - tags in OWL Lite $\subseteq$ tags in DL $\subseteq$ tags in Full
  - semantic:
    - OWL DL semantics = OWL Full semantics (within DL fragment)
    - OWL Lite semantics = OWL DL semantics (within Lite fragment)
• **No restriction** on use of OWL vocabulary
  (as long as legal RDF)
  – classes as instances (and much more)
    • e.g. JeensClasses: the class of classes that are defined by Jeen
  – not decidable ⇒ no automated reasoning support
• Use of OWL vocabulary **restricted**
  – can’t be used to do “nasty things”
    • i.e., modify OWL
  – no classes as instances

• Standard DL/FOL model theory
  – direct correspondence with (first order) logic
  – **automated reasoning** via DL engines
    • e.g., FaCT, RACER, Pellet
OWL Lite

• Like DL, but **fewer constructs**
  – no explicit negation or union
  – restricted cardinality (zero or one)
  – no enumerations (oneOf)
  – **goal**: language that is easier to use

• Semantics as per DL
  – reasoning via standard DL engines (+datatypes)
Putting it all together

• What can we do with these things?
• General picture:
  – Semantic web requires:
    • structured data
    • knowledge of where the data is about (ontology)
• What about
  – XML on itself:
    • provides structured data in documents
    • no knowledge of where the data is about
      ⇒ not appropriate for reasoning on the web
Putting it all together - 2

– RDF(S) techniques:
  
  • **data** is captured in RDF descriptions

```
<rdf:Description ID="Z31">
  <rdf:type resource="http://my.business.com/prod#Inkjet"/>
  <onto:price>199,=</onto:price>
</rdf:Description>
```

• some **knowledge** is described in RDF Schema

```
<rdfs:Class ID="Inkjet">
  <rdfs:subClassOf resource="#Printer"/>
</rdfs:Class>
```

⇒ possible to find out that “Z31” is a printer of type inkjet with price 199,=
Putting it all together - 3

– extended techniques (OWL):
  
  • **data** is still captured in RDF descriptions
  
  • **knowledge** is captured in OWL ontology

```
<owl:Class rdf:ID="CheapPrinter">
  <owl:intersectionOf>
    <rdfs:Class rdf:about="#Printer"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#price"/>
      <owl:hasClass rdf:resource="...#lessThan200"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

⇒ possible to find “Z31” when looking for a CheapPrinter
⇒ check whether Inkjet and CheapPrinter are equivalent, etc.
Read further

• OWL specification
  http://www.w3.org/TR/owl-features/

• OWL + RDFS Reference Card
  http://ebiquity.umbc.edu/v2.1/resource/html/id/97/