



Vorlesung Künstliche Intelligenz Wintersemester 2007/08

Teil IV: Wissensrepräsentation im WWW

Kap.11: Semantic Web

Dieses Kapitel basiert weitgehend auf Material von Pascal Hitzler. Weitere Infos gibt es in dem Buch [Grigoris Antoniou, Frank von Harmelen: A Semantic Web Primer, MIT Press, Cambridge 2004]



In diesem Kapitel betrachten wir, wie Wissensrepräsentation aus dem World Wide Web ein Semantic Web machen kann.

Das Semantic Web ist eine Vision von Tim Berners-Lee, dem Erfinder des WWW.

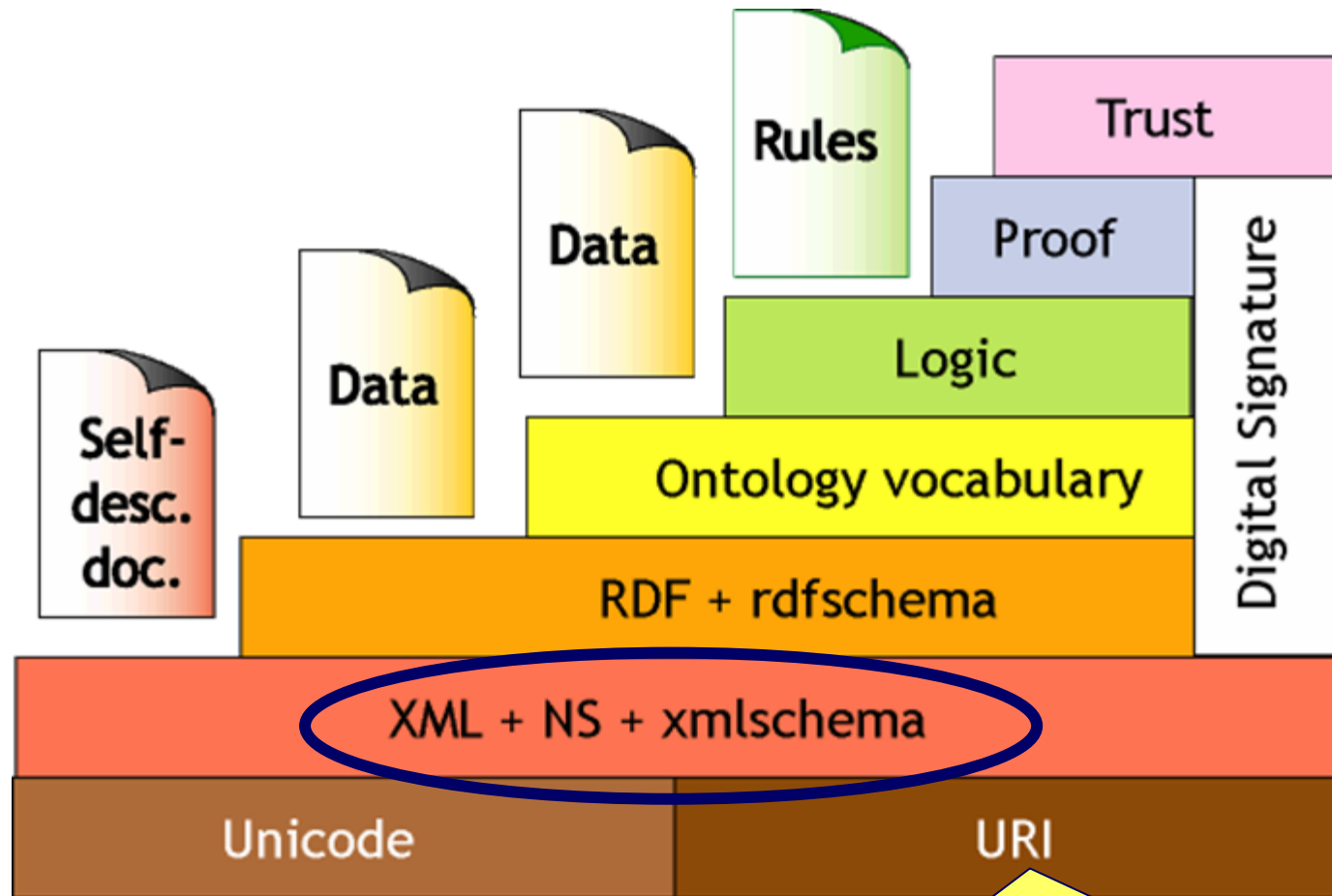
Die Kernidee ist die explizite Repräsentation von Wissen im WWW, so dass es von Such- und anderen Maschinen verwendet werden kann.

Unsere Darstellung basiert auf den Kapiteln 1 bis 4 des Buchs

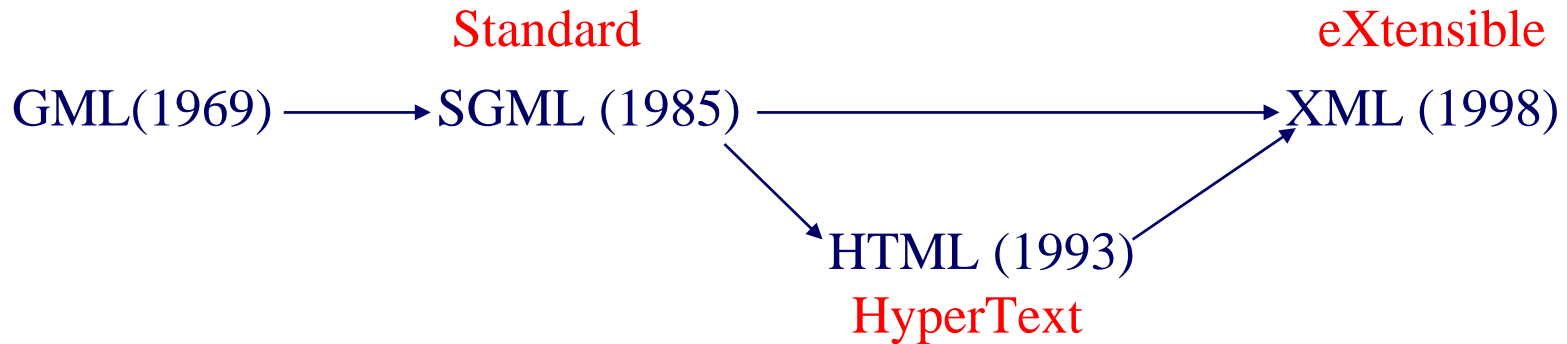
Grigoris Antoniou, Frank von Harmelen: A Semantic Web Primer, MIT Press, Cambridge 2004

Unter <http://www.semanticwebprimer.org> sind Folien, Beispiele etc. online zu finden. Auf diesem Foliensatz basieren auch die nachfolgenden Abschnitte.

The semantic web layer cake



Universal Resource Identifiers
e.g. <http://www.somedomain.org/thispage#thistag>

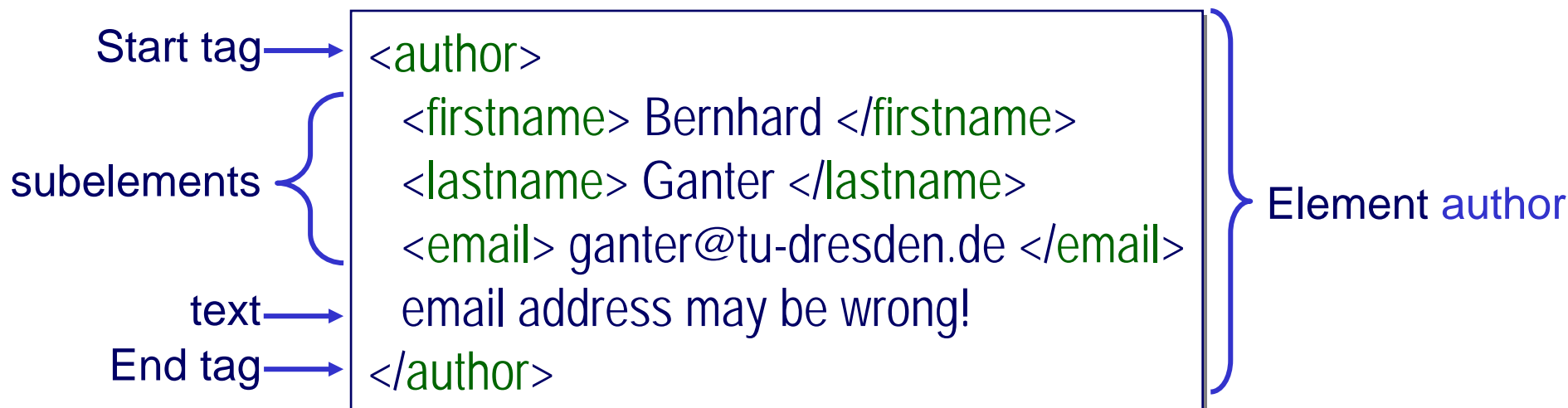


- eXtensible Markup Language
- Web standard (W3C) for data exchange:
 - Description of in- and output data of applications
 - Reduces degrees of freedom for industrial data description standards
- Complementary to HTML:
 - HTML describes presentation
 - XML describes content
- Database perspective: XML as data model for semi-structured data.



XML element:

- Description of an object, which is embraced by matching tags like `<author>` and `</author>`.
- Content of an element: Text and/or other (sub)elements.
- Elements can be nested
- Elements can be empty: `<year></year>` (short: `<year/>`)





XML attribute:

- Name-string pair
- Associated with an element
- Alternative way for describing data

Attribute email

```
<author email="ganter@tu-dresden.de">  
  <firstname> Bernhard </firstname>  
  <lastname> Ganter </lastname>  
</author>
```

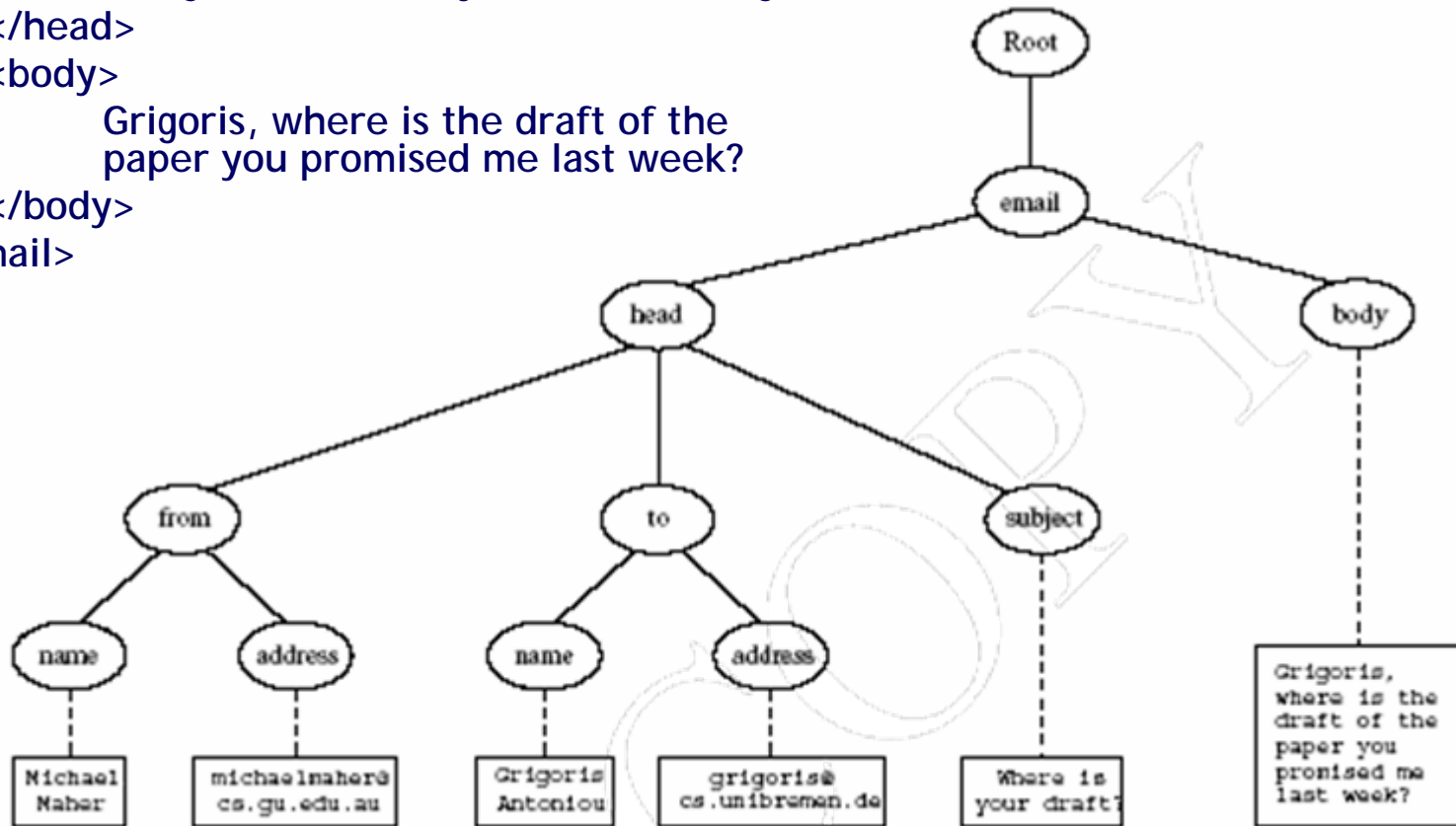
Alternative description of the same(?) data:

```
<author firstname="Bernhard" lastname="Ganter" email="ganter@tu-dresden.de"/>
```

The Tree Model of XML Documents: An Example



```
<email>
  <head>
    <from name="Michael Maher"
      address="michaelmaher@cs.gu.edu.au"/>
    <to name="Grigoris Antoniou"
      address="grigoris@cs.unibremen.de"/>
    <subject>Where is your draft?</subject>
  </head>
  <body>
    Grigoris, where is the draft of the
    paper you promised me last week?
  </body>
</email>
```





The tree representation of an XML document is an ordered labeled tree:

- There is exactly one root
- There are no cycles
- Each non-root node has exactly one parent
- Each node has a label.
- The order of elements is important
- ... but the order of attributes is not important



XPath is core for XML query languages.

Language for addressing parts of an XML document.

- It operates on the tree data model of XML.
- It has a non-XML syntax .

Examples

- Address all books with title "Artificial Intelligence"
`/book[@title="Artificial Intelligence"]`
- Address the first author element node in the XML document
`//author[1]`
- Address the last book element within the first author element node in the document
`//author[1]/book[last()]`
- Address all book element nodes without a title attribute
`//book[not @title]`



Complex language for data description:

- Many standardised base types, e.g. float, double, decimal, boolean in particular: string and integer
- Types and typed references
- Class hierarchy and inheritance
- Consistency constraints

Standard („W3C Recommendation“) as extension to XML

Namespaces



An XML document may use more than one DTD or schema
Prefixes are used to avoid name clashes.

Prefixes have URIs as values.
They usually point to a description of the namespace syntax.

Example:

```
<vu:instructors xmlns:vu="http://www.vu.com/empDTD"
                xmlns:gu="http://www.gu.au/empDTD"
                xmlns:unik="http://www.unik.de/empDTD">

  <unik:dozent unik:title="Dr."
              unik:name="Andreas Hotho"
              unik:department="Computer Science"/>

  <gu:academicStaff gu:title="lecturer"
                   gu:name="Mate Jones"
                   gu:school="Information Technology"/>

</vu:instructors>
```



Move data and metadata from one XML representation to another, eg, when applications that use different DTDs or schemas need to communicate.

The *extensible stylesheet language* XSL includes

- a transformation language (XSLT)
- a formatting language

XSLT specifies rules with which an input XML document is transformed to

- another XML document
- an HTML document
- plain text

The output document may use the same DTD or schema, or a completely different vocabulary.

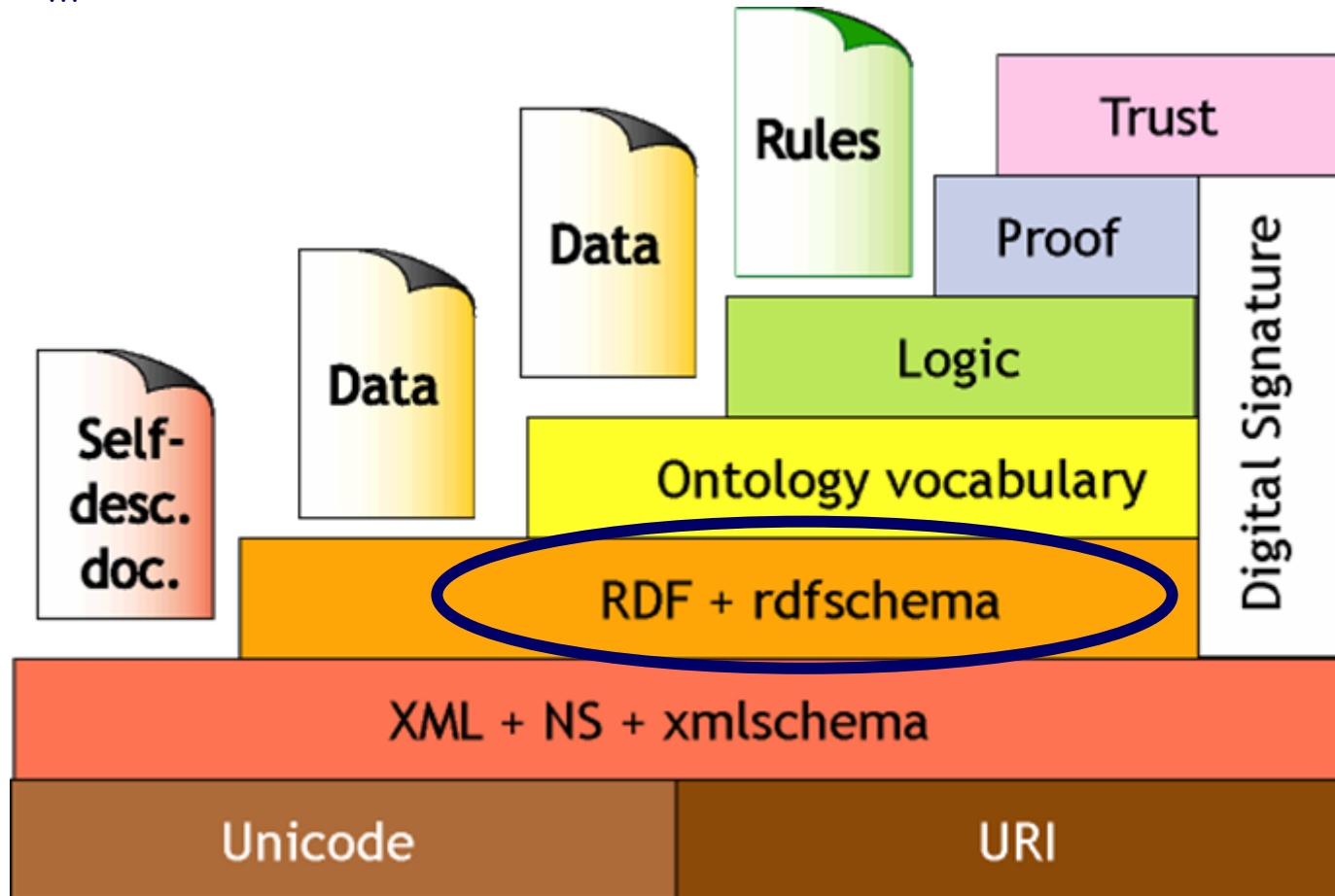
XSLT can be used independently of the formatting language.

The semantic web layer cake



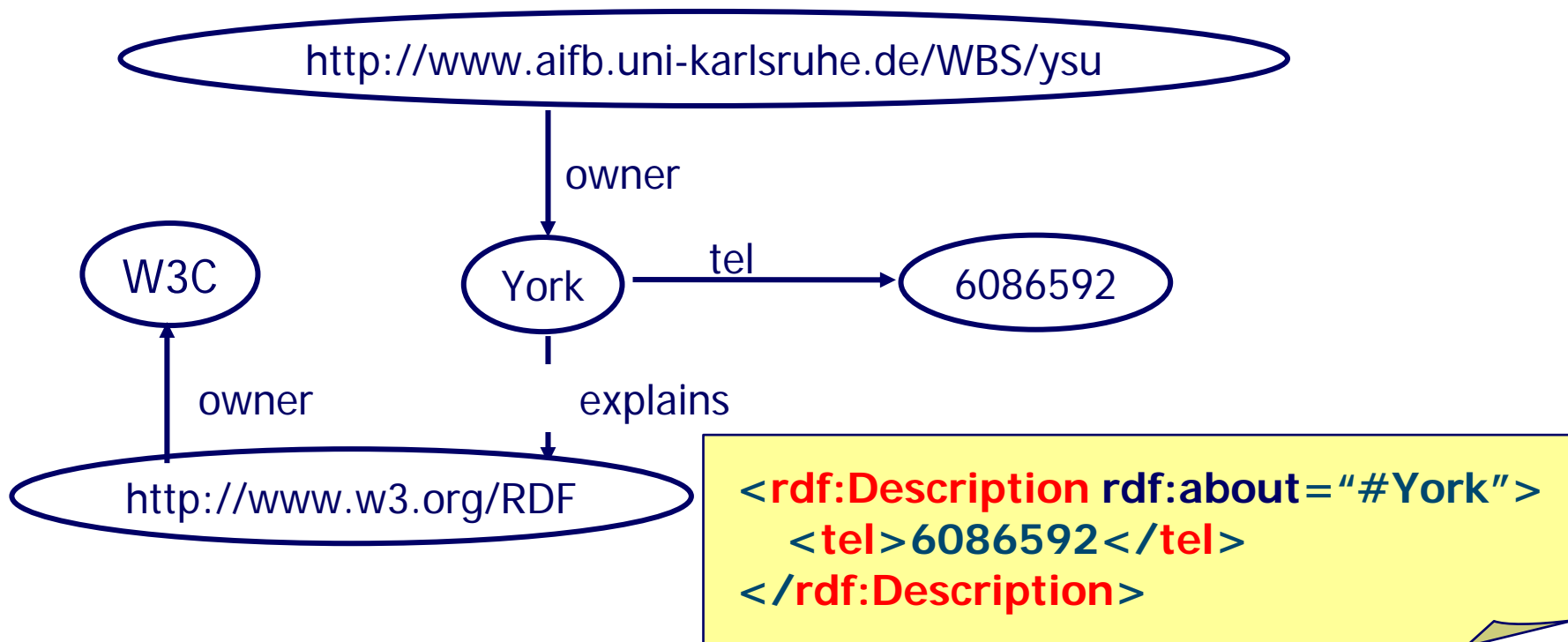
RDF Recommendation consists of several parts

- [RDF Primer](http://www.w3.org/TR/rdf-primer/) (<http://www.w3.org/TR/rdf-primer/>)
- [RDF Schema](http://www.w3.org/TR/rdf-schema/) (<http://www.w3.org/TR/rdf-schema/>)
- ...



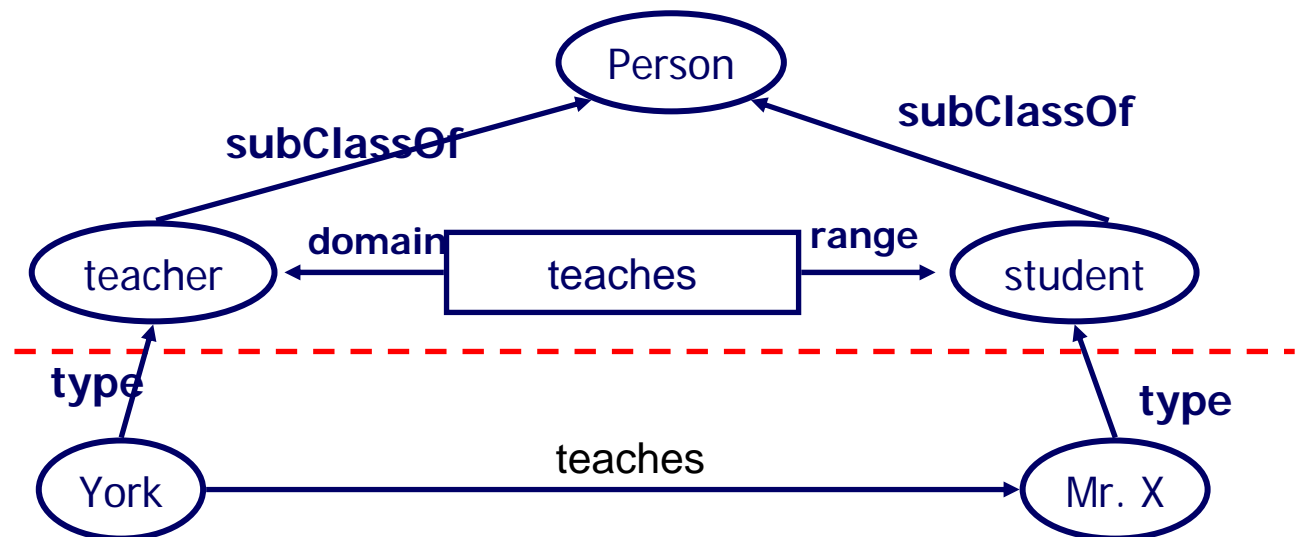


- RDF provides metadata about web resources
- key component: **Object -> Attribute -> Value** triple
- Interconnected triples constitute a labelled graph
- RDF uses XML syntax





- RDFS defines **vocabulary** for RDF
- Vocabulary is organised as **type hierarchy**
 - Class, subClassOf
 - type
 - Property, subPropertyOf
 - domain, range





```
<rdf:Description ID="Person">  
  <rdf:type resource="http://www.w3.org/...#Class"/>  
  <rdfs:subClassOf rdf:resource="http://www.w3.org/...#Resource"/>  
</rdf:Description>
```

```
<rdf:Description ID="Teacher">  
  <rdf:type resource="http://www.w3.org/...#Class"/>  
  <rdfs:subClassOf rdf:resource="#Person"/>  
</rdf:Description>
```

```
<rdf:Description ID="teaches">  
  <rdf:type resource="http://www.w3.org/...#Property"/>  
  <rdfs:domain rdf:resource="#Teacher"/>  
  <rdfs:range rdf:resource="#Student"/>  
</rdf:Description>
```

```
<rdf:Description ID="teaches well">  
  <rdf:type resource="http://www.w3.org/...#Property"/>  
  <rdfs:subPropertyOf rdf:resource="#teaches"/>  
</rdf:Description>
```




Classes: unary predicates
subClassOf relation: implication

tutor \sqsubseteq student

$(\forall x) (\text{tutor}(x) \rightarrow \text{student}(x))$

Properties: binary predicates
subPropertyOf relation: implication

supervises \sqsubseteq responsibleFor

$(\forall x)(\forall y) (\text{supervises}(x,y) \rightarrow \text{responsibleFor}(x,y))$

RDF statements are triples (Object, Property, Object)

■ Objects can be

- URIs constants
- classes unary predicates
- properties binary predicates
- triples(!) -- (\rightarrow reification, i.e. second-order)



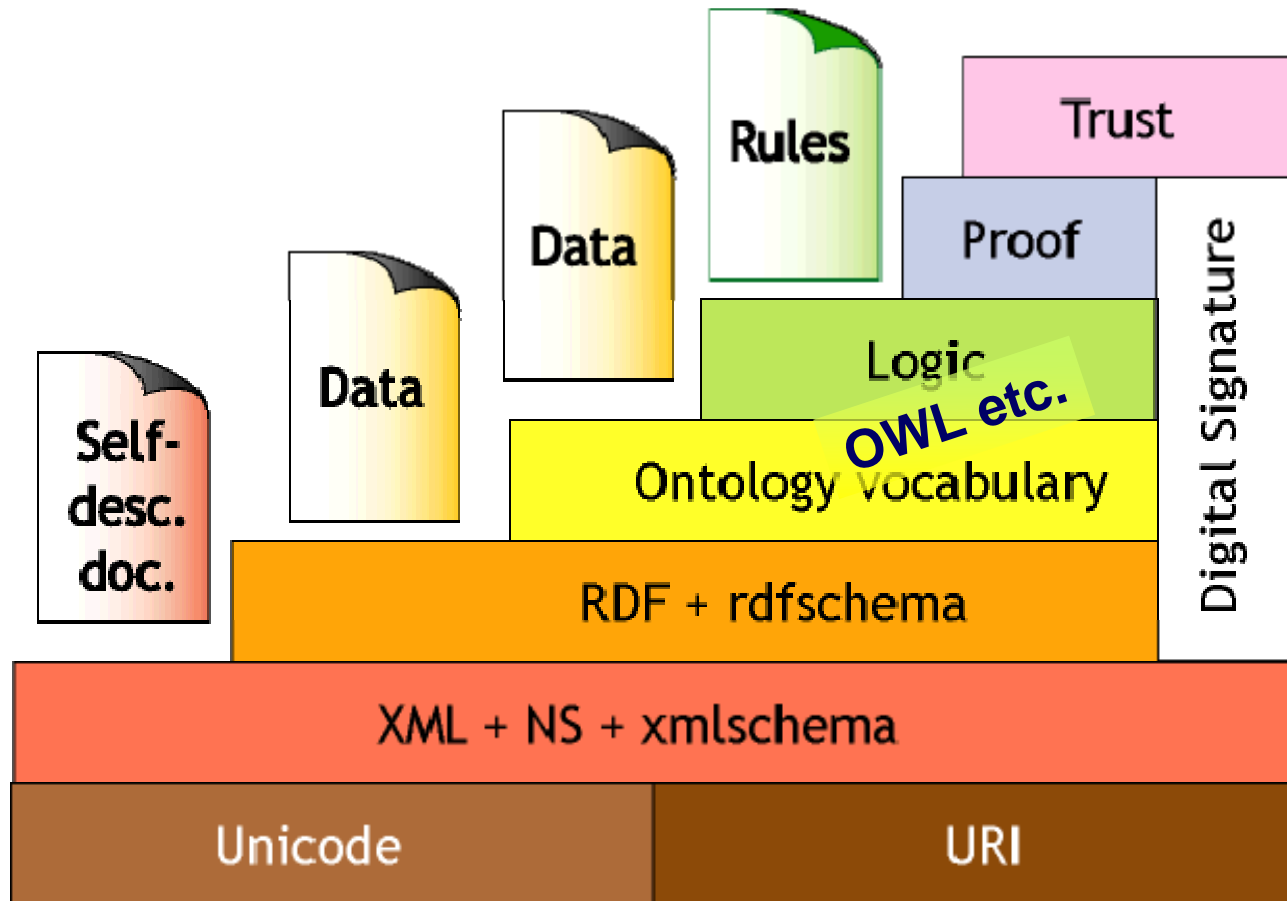
RDF(S) is useful for simple ontologies, but not for complex modelling

→ „Need for more expressivity!“

More expressive languages:

- OWL (based on description logics)
- F-Logic (based on logic programming)
- Hybrids and rules extensions for OWL

The Semantic Web layer cake





- W3C Recommendation since 2004
- Semantic fragment of FOL (First-order predicate logic)
- Three variants: OWL Lite \subseteq OWL DL \subseteq OWL Full
- RDFS is fragment of OWL Full.
- No reification in OWL DL.
- OWL DL is decidable
- OWL DL = SHOIN(D)



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties



OWL documents are RDF Documents.

They consist of

- Head with general information
- Rest with the ontology

Head of an OWL document



Definition of namespaces in the root

```
<rdf:RDF
```

```
  xmlns = "http://www.semanticweb-grundlagen.de/beispielontologie#"
```

```
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"
```

```
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
```

```
  xmlns:owl = "http://www.w3.org/2002/07/owl#">
```

```
...
```

```
</rdf:RDF>
```

General information

```
<owl:Ontology rdf:about="">
```

```
  <rdfs:comment    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

```
    SWRC Ontology December 2005
```

```
  </rdfs:comment>
```

```
<owl:versionInfo>v0.5</owl:versionInfo>
```

```
<owl:imports rdf:resource="http://www.semanticweb-grundlagen.de/foo"/>
```

```
<owl:priorVersion    rdf:resource="http://ontoware.org/projects/swrc"/>
```

```
</owl:Ontology>
```



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties



Basic components of OWL ontologies:

Classes

- like resources in RDFS
- like classes in DL

Individuals

- like resources in RDFS
- like individuals in DL

Roles

- like attributes in RDFS
- like roles in DL



Definition

```
<owl:Class rdf:ID="Professor" />
```

■ predefined:

■ **owl:Thing**



■ **owl:Nothing**





Definition by class membership

```
<rdf:Description rdf:ID="RudiStuder">  
  <rdf:type rdf:resource="#Professor" />  
</rdf:Description>
```

- equivalent:

Professor(RudiStuder)

```
<Professor rdf:ID="RudiStuder" />
```



Abstract roles defined like classes

```
<owl:ObjectProperty
  rdf:ID="Affiliation" />
```

Domain and Range of abstract Roles

```
<owl:ObjectProperty rdf:ID="Affiliation">
  <rdfs:domain rdf:resource="#Person" />
  <rdfs:range rdf:resource="#Organisation" />
</owl:ObjectProperty>
```

Domain:	$\exists \text{Affiliation} . \top \sqsubseteq \text{Person}$
Range:	$\top \sqsubseteq \forall \text{Affiliation} . \text{Organisation}$



Concrete roles have datatypes in range

```
<owl:DatatypeProperty rdf:ID="firstname" />
```

Domain and range of concrete roles

```
<owl:DatatypeProperty rdf:ID="firstname">  
  <rdfs:domain rdf:resource="#Person" />  
  <rdfs:range rdf:resource="&xsd:string" />  
</owl:DatatypeProperty>
```

One can use many XML Schema Datatypes.

The standard requires at least integer and string.



```
<Person rdf:ID="RudiStuder">  
  <Affiliation rdf:resource="#AIFB" />  
  <Affiliation rdf:resources="#ontoprise" />  
  <firstname rdf:datatype="&xsd:string">Rudi</firstname>  
</Person>
```

Affiliation(RudiStuder,AIFB)
Affiliation(RudiStuder,ontoprise)
Firstname(RudiStuder,"Rudi")

Roles are in general not functional.



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties

Simple class relationships



```
<owl:Class rdf:ID="Professor">  
  <rdfs:subClassOf  
    rdf:resource="#Faculty"/>  
</owl:Class>
```

Professor \sqsubseteq Faculty

```
<owl:Class rdf:ID="Faculty">  
  <rdfs:subClassOf rdf:resource="#Person"/>  
</owl:Class>
```

Faculty \sqsubseteq Person

It can be inferred that Professor is a subclass of Person.

Simple class relationships



```
<owl:Class rdf:ID="Professor">
```

Professor \sqsubseteq Faculty

```
  <rdfs:subClassOf rdf:resource="#Faculty"/>
```

```
</owl:Class>
```

```
<owl:Class rdf:ID="Book">
```

Book \sqsubseteq Publication

```
  <rdfs:subClassOf rdf:resource="#Publication"/>
```

```
</owl:Class>
```

```
<owl:Class rdf:about="#Faculty">
```

```
  <owl:disjointWith rdf:resource="#Publication"/>
```

```
</owl:Class>
```

Faculty \sqcap Publication $\equiv \perp$

We infer that Professor and Book are also disjoint classes.



```
<owl:Class rdf:ID="Book" >  
  <rdfs:subClassOf  
    rdf:resource="#Publication" />  
</owl:Class>
```

Book \sqsubseteq Publication

```
<owl:Class rdf:about="#Publication" >  
  <owl:equivalentClass  
    rdf:resource="#Publikation" />  
</owl:Class>
```

Publication \equiv Publikation

We infer that Book is a subclass of Publikation.



```
Author(SemanticWebGrundlagen, YorkSure)
Author(SemanticWebGrundlagen, PascalHitzler)
```

```
<Book rdf:ID="SemanticWebGrundlagen">
  <Author rdf:resource="#YorkSure"/>
  <Author rdf:resource="#PascalHitzler"/>
</Book>
```

Book \sqsubseteq Publication

```
<owl:Class rdf:about="#Book">
  <rdfs:subClassOf rdf:resource="#Publication"/>
</owl:Class>
```

We infer that SemanticWebGrundlagen is a Publication.



```
<Professor rdf:ID="RudiStuder" />
<rdf:Description rdf:about="#RudiStuder">
  <owl:sameAs
    rdf:resource="#ProfessorStuder" />
</rdf:Description>
```

**Professor(RudiStuder)
RudiStuder = ProfessorStuder**

We infer that ProfessorStuder is a Professor.

Inequality of individuals expressed using
owl:differentFrom.



```
<owl:AllDifferent>
  <owl:distinctMembers
    rdf:parseType="Collection">
    <Person rdf:about="#RudiStuder" />
    <Person rdf:about="#YorkSure" />
    <Person rdf:about="#PascalHitzler" />
  </owl:distinctMembers>
</owl:AllDifferent>
```

Shortcut for multiple usage of `owl:differentFrom`.

Closed classes (nominals)



```
<owl:Class rdf:about=#SecretaryOfStuder>
  <owl:oneOf rdf:parseType="Collection">
    <Person rdf:about="#GiselaSchillinger"/>
    <Person rdf:about="#AnneEberhardt"/>
  </owl:oneOf>
</owl:Class>
```

There are **exactly those two** Individuals in the class SecretaryOfStuder.

SecretaryOfStuder \equiv {GiselaSchillinger,AnneEberhardt}



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties



Conjunction:

`owl:intersectionOf`

Disjunction:

`owl:unionOf`

Negation:

`owl:complementOf`

Can be used to construct complex classes from class names.



SecretaryOfStuder \equiv Secretary \sqcap MemberAGStuder

```
<owl:Class rdf:about="#SecretaryOfStuder">
  <owl:equivalentClass>
    <owl:intersectionOf
      rdf:parseType="Collection">
      <owl:Class rdf:about="#Secretary"/>
      <owl:Class
        rdf:about="#MemberAGStuder"/>
    </owl:intersectionOf>
  </owl:equivalentClass>
</owl:Class>
```

We infer that all individuals in SecretaryOfStuder are also in Secretary.

Disjunction



```
<owl:Class rdf:about="#Professor">
  <owl:subClassOf>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#activeTeacher"/>
      <owl:Class rdf:about="#retired"/>
    </owl:unionOf>
  </owl:subClassOf>
</owl:Class>
```

Professor \sqsubseteq activeTeacher \sqcup retired



```
<owl:Class rdf:about="#Faculty">
  <owl:subClassOf>
    <owl:complementOf rdf:resource="#Publication"/>
  </owl:subClassOf>
</owl:Class>
```

Faculty \sqsubseteq \neg Publication

This is a complicated way of saying the following:

```
<owl:Class rdf:about="#Faculty">
  <owl:disjointWith rdf:resource="#Publication"/>
</owl:Class>
```

Faculty \sqcap Publication $\equiv \perp$



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties



Using roles for defining complex classes

```
<owl:Class rdf:ID="Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:allValuesFrom rdf:resource="#Professor"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e. *all* examiners of an exam must be professors.

Exam $\sqsubseteq \forall \text{hasExaminer. Professor}$

Role restrictions (someValuesFrom)



```
<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:someValuesFrom rdf:resource="#Person"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

I.e. each exam must have *at least one* examiner.

Exam $\sqsubseteq \exists \text{hasExaminer. Person}$

Number restrictions (cardinalities)



```
<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        2
      </owl:maxcardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Exam $\sqsubseteq \leq 2$ hasExaminer

An exam must have *at most two* examiners.

Number restrictions (cardinalities)



```
<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopic"/>
      <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        3
      </owl:mincardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Exam $\sqsubseteq \geq 3$ hasTopic

An exam must cover *at least three* topics.

Number restrictions (cardinalities)



```
<owl:Class rdf:about="#Exam">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopic"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">
        3
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Exam \sqsubseteq =3 hasTopic

An exam must cover *exactly three* topics.

Role restrictions (hasValue)



```
<owl:Class rdf:ID="examProfStuder">
  <rdfs:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer" />
      <owl:hasValue rdf:resource="#RudiStuder" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

owl:hasValue always points to an individual. This is equivalent to the example on the next slide.

Role restrictions (hasValue)



```
<owl:Class rdf:ID="examProfStuder">
  <rdfs:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasExaminer" />
      <owl:someValuesFrom>
        <owl:oneOf rdf:parseType="Collection">
          <owl:Thing rdf:about=#RudiStuder />
        </owl:oneOf>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

examProfStuder $\equiv \exists$ hasExaminer.{RudiStuder}



Head of a document

Classes, roles and Individuals

Class relationships

Complex class definitions

- Boolean class constructors
- Role restrictions

Role properties



hasExaminer \sqsubseteq hasParticipant

```
<owl:ObjectProperty rdf:ID="hasExaminer">
  <rdfs:subPropertyOf
    rdf:resource="#hasParticipant" />
</owl:ObjectProperty>
```

Similar: owl:equivalentProperty

Roles can be inverse to each other:

```
<owl:ObjectProperty rdf:ID="hasExaminer">
  <owl:inverseOf rdf:resource="#examinerOf" />
</owl:ObjectProperty>
```

hasExaminer \equiv examinerOf⁻



- Domain
- Range
- Transitivity i.e. (a,b) and $r(b,c)$ implies $r(a,c)$
- Symmetry i.e. $r(a,b)$ implies $r(b,a)$
- Functionality i.e. $r(a,b)$ and $r(a,c)$ implies $b=c$
- Inverse functionality

Domain and Range



```
<owl:ObjectProperty rdf:ID="Affiliation">  
  <rdfs:range rdf:resource="#Organisation"/>  
</owl:ObjectProperty>
```

Is equivalent to the following:

```
<owl:Class rdf:about="\&owl;Thing">  
  <rdfs:subClassOf>  
    <owl:Restriction>  
      <owl:onProperty rdf:resource="#Affiliation"/>  
      <owl:allValuesFrom rdf:resource="#Organisation"/>  
    </owl:Restriction>  
  </rdfs:subClassOf>  
</owl:Class>
```

Range: $\top \sqsubseteq \forall \text{Affiliation. Organisation}$



```
<owl:ObjectProperty rdf:ID=„Affiliation“>
  <rdfs:range rdf:resource=„#Organisation“ />
</owl:ObjectProperty>
<Number rdf:ID=„Five“>
  <Affiliation rdf:resource=„#PrimeNumber“ />
</Number>
```

It follows that `PrimeNumber` is an `Organisation`!

$\top \sqsubseteq \forall \text{Affiliation. Organisation}$
 $\top \sqsubseteq \forall \text{Number(Five)}$
 $\text{Affiliation(Five, PrimeNumber)}$

Role properties



```
<owl:ObjectProperty rdf:ID="hasColleague">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasProjectLeader">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isProjectLeaderFor">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty" />
</owl:ObjectProperty>
<Person rdf:ID="YorkSure">
  <hasColleague rdf:resource="#PascalHitzler" />
  <hasColleague rdf:resource="#AnupriyaAnkolekar" />
  <isProjectLeaderFor rdf:resource="#SEKT" />
</Person>
<Projekt rdf:ID="SmartWeb">
  <hasProjectLeader rdf:resource="#PascalHitzler" />
  <hasProjectLeader rdf:resource="#HitzlerPascal" />
</Projekt>
```

Logical consequences from example

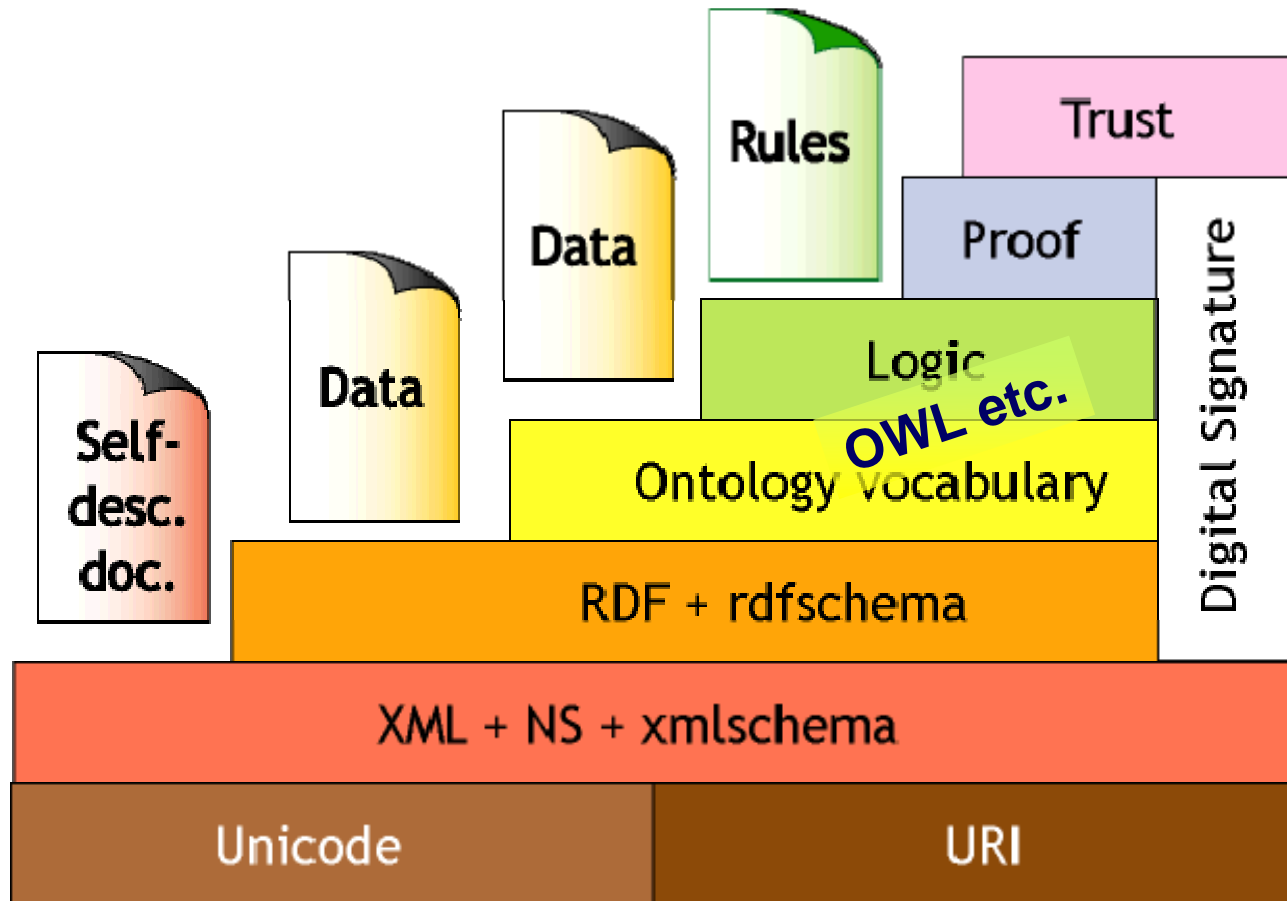


AnupriyaAnkolekar hasColleague YorkSure

AnupriyaAnkolekar hasColleague PascalHitzler

PascalHitzler owl:sameAs HitzlerPascal

The Semantic Web layer cake





OWL Full

- Contains OWL DL and OWL Lite
- Contains all of RDFS
- Undecidable
- Limited support by existing software

OWL DL (= SHOIN(D))

- Contains OWL Lite and is contained in OWL Full
- decidable
- Tools available
- complexity NExpTime (worst-case)

OWL Lite (= SHIF(D))

- Is contained in OWL DL and OWL Full
- decidable.
- Less expressive.
- Complexity ExpTime (worst-case)



Unlimited usage of all OWL and RDFS constructs (must be valid RDFS).

Difficult is e.g. the non-existent type separation (Classes, Roles, Individuals), hence:

- `owl:Thing` is the same as `rdfs:resource`
- `owl:Class` is the same as `rdfs:Class`
- `owl:DatatypeProperty` subclass of `owl:ObjectProperty`
- `owl:ObjectProperty` is the same as `rdf:Property`

Why types are not separated in OWL Full



```
<owl:Class rdf:about="#Book">
  <germanName rdf:datatype="&xsd:string">
    Buch
  </germanName>
  <frenchName rdf:datatype="&xsd:string">
    livre
  </frenchName>
</owl:Class>
```

One often does not really need inferencing over such information.



- is SHOIN(D).
- Allowed are only certain RDFS constructs (like those in the examples).
Not allowed: `rdfs:Class`, `rdfs:Property`
- Type separation. Classes and Roles must be declared explicitly.
- Concrete Roles must not be transitive, symmetric, inverse or inverse functional.
- Number restrictions must not be used with transitive roles, their subroles, or their inverses.



- is SHIF(D).
- All restrictions for OWL DL apply.
- Not allowed: `oneOf`, `unionOf`, `complementOf`, `hasValue`, `disjointWith`
- Number restrictions only allowed with 0 and 1.
- Some restrictions on the occurrence of anonymous (complex) classes apply, e.g. they must not occur in the subject of `rdfs:subClassOf`.



Editors

- Protegé, <http://protege.stanford.edu>
- SWOOP, <http://www.mindswap.org/2004/SWOOP/>
- OWL Tools, <http://owltools.ontoware.org/>

Inference engines

- Pellet, <http://www.mindswap.org/2003/pellet/index.shtml>
- KAON2, <http://kaon2.semanticweb.org>
- FACT++, <http://owl.man.ac.uk/factplusplus/>
- Racer, <http://www.racer-systems.com/>
- Cerebra, <http://www.cerebra.com/index.html>



Head

`rdfs:comment`

`rdfs:label`

`rdfs:seeAlso`

`rdfs:isDefinedBy`

`owl:versionInfo`

`owl:priorVersion`

`owl:backwardCompatibleWith`

`owl:incompatibleWith`

`owl:DeprecatedClass`

`owl:DeprecatedProperty`

`owl:imports`

Relations between individuals

`owl:sameAs`

`owl:differentFrom`

`owl:AllDifferent`

(together with

`owl:distinctMembers`)

Required datatypes

`xsd:string`

`xsd:integer`



Class constructors and relationships

`owl:Class`
`owl:Thing`
`owl:Nothing`
`rdfs:subClassOf`
`owl:disjointWith`
`owl:equivalentClass`
`owl:intersectionOf`
`owl:unionOf`
`owl:complementOf`

Role restrictions

`owl:allValuesFrom`
`owl:someValuesFrom`
`owl:hasValue`
`owl:cardinality`
`owl:minCardinality`
`owl:maxCardinality`
`owl:oneOf`



Role constructors, relations and properties

`owl:ObjectProperty`

`owl:DatatypeProperty`

`rdfs:subPropertyOf`

`owl:equivalentProperty`

`owl:inverseOf`

`rdfs:domain`

`rdfs:range`

`rdf:resource="&owl;TransitiveProperty"`

`rdf:resource="&owl;SymmetricProperty"`

`rdf:resource="&owl;FunctionalProperty"`

`rdf:resource="&owl;InverseFunctionalProperty"`



Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human \sqcap Male	$C_1(x) \wedge \dots \wedge C_n(x)$
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor \sqcup Lawyer	$C_1(x) \vee \dots \vee C_n(x)$
complementOf	$\neg C$	\neg Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \dots \sqcup \{x_n\}$	{john} \sqcup {mary}	$x = x_1 \vee \dots \vee x = x_n$
allValuesFrom	$\forall P.C$	\forall hasChild.Doctor	$\forall y.P(x, y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	\exists hasChild.Lawyer	$\exists y.P(x, y) \wedge C(y)$
maxCardinality	$\leq_n P$	≤ 1 hasChild	$\exists \leq_n y.P(x, y)$
minCardinality	$\geq_n P$	≥ 2 hasChild	$\exists \geq_n y.P(x, y)$

XMLS datatypes as well as classes in $\forall P.C$ and $\exists P.C$

- E.g., \exists hasAge.nonNegativeInteger

Arbitrarily complex nesting of constructors

- E.g., Person $\sqcap \forall$ hasChild.Doctor $\sqcup \exists$ hasChild.Doctor



E.g., $\text{Person} \sqcap \forall \text{hasChild}.\text{Doctor} \sqcup \exists \text{hasChild}.\text{Doctor}$:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType=" collection">
    <owl:Class rdf:about="#Person" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild" />
      <owl:toClass>
        <owl:unionOf rdf:parseType=" collection">
          <owl:Class rdf:about="#Doctor" />
          <owl:Restriction>
            <owl:onProperty rdf:resource="#hasChild" />
            <owl:hasClass rdf:resource="#Doctor" />
          </owl:Restriction>
        </owl:unionOf>
      </owl:toClass>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```



Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human \sqsubseteq Animal \sqcap Biped
equivalentClass	$C_1 \equiv C_2$	Man \equiv Human \sqcap Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$ Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} \equiv {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg$ {peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter \sqsubseteq hasChild
equivalentProperty	$P_1 \equiv P_2$	cost \equiv price
inverseOf	$P_1 \equiv P_2^-$	hasChild \equiv hasParent ⁻
transitiveProperty	$P^+ \sqsubseteq P$	ancestor ⁺ \sqsubseteq ancestor
functionalProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1$ hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1$ hasSSN ⁻

Axioms (mostly) reducible to inclusion (\sqsubseteq)

- $C \equiv D$ iff both $C \sqsubseteq D$ and $D \sqsubseteq C$

Obvious FOL equivalences

- E.g., $C \equiv D$ iff $\forall x. C(x) \Leftrightarrow D(x)$,
- $C \sqsubseteq D$ iff $\forall x. C(x) \Rightarrow D(x)$