Idee des Semantic Web

- Erstmals 1994 von Tim Burness Lee vorgetragen.
- Typisierte Links im Web (Meta-Tags zu Dokumentverknüpfungen)
- Heutiges Anwendungsbeispiel: MetaWiki:
  - Wikipedia (als große Sammlung großer Wikis) beinhaltet viele handgeschriebene Listen.
  - Nur mit exorbitantem Pflegeaufwand können Inkonsistenzen zwischen verschiedenen Wikiseiten und verschiedenen Landeswikis minimiert werden.
  - Daraus wurde die Idee geboren, eine Special-Syntax für Annotationen einzuführen:
  - Diese Annotationen erzeugen RDF-Tripel:
    - domain: Der annotierte Artikel
    - range: Skalar / Anderer Artikel
  - Auf diese Tripel können entweder einfache, direkte Anfragen gestellt werden,
  - Oder die Anfragen können in einen WikiText übernommen werden, um automatisch Listen etc. zu erzeugen.
- Die 3 Kernideen des Semantic Web:
  - **Annotation**: Resourcen auszeichnen
  - **Integration:** Das Web of Data, welches verschiedene Datenquellen verbindet
  - **Inferenz**: Automatische Schlussfolgerungen
    - **Leichtgewichtig:** RDF vs.
    - **Schwergewichtig**: OWL Full

eXtensible Markup Language

- Syntax: Elemente, Attribute, Parseranweisungen, ...
- Wohlgeformt (mit Freitext zwischen den Tags) vs. Gültig
- xmlns="URI" / xmlns:prefix="URI"
  - Gültige Namespaces ab dem deklarierenden Element
  - **Tiefer Namespace:** Substitution des Prefix mit der URI im Gültigkeitsbereich des Namensraums
  - **Flacher Namespace:** Keine URI-Substitution. Verwenden des prefix:tagname-Strings als Tagname.
  - Üblicher Weise verweist der URI eines Namespace auf ein Schemadokument!!
- Physisches Datenmodell: Baumstruktur

Data Type Definitions

Allgemein

- Dient der Validierung von XML-Instanzen.
- Erbe von SGML.

- Wird durch eine einfache, kontextfreie Grammatik beschrieben.

```
<!DOCTYPE wurzelelement [
      <!ELEMENT wurzelelement (ele1*, ele2+, ele3?, ele4)>
      <!ELEMENT ele1 EMPTY>
      <!ELEMENT ele2 (ele5 | ele6*)>
      <!ELEMENT ele3 (#PCDATA)>
      <!ELEMENT ele4 (ele5 | (ele1, ele2)*)

      <!ATTLIST ele1 id ID #REQUIRED>
      <!ATTLIST ele2 name CDATA #IMPLIED>
      <!ATTLISTE ele2 id ID #REQUIRED age CDATA #IMPLIED>
]>
```

Elementdefinition

(x, y, z)	Feste Reihenfolge von Unterelementen
(x | y | z)	Entweder-Oder-Auswahl von Unterattributen

Eine Schachtelung der Klammerausdrücke ist erlaubt!

Kardinalität: *	0 ... n
Kardinalität: +	1 ... n
Kardinalität: ?	0 ... 1
Kardinalität:	1 ... 1

Attributdefinition

- **Allgemeine Syntax:** `Name Typ Zusatz` ...
- **Typen:** `CDATA, ID, IDREF, IDREFS` (Space-getrennte Liste von IDs)
- **Zusatz:** `#REQUIRED` (Pflichtangabe) vs. `#IMPLIED` (Optionale Angabe)

Fallstricke

- Ungewollte Definition der Reihenfolge von Unterelementen.
- Eine Freie Reihenfolge kann nicht ohne Semantikverlust und wachsendem Aufwand realisiert werden.
  - `person ( (name1 | name2), (name1 | name2) )`
  - `person ( (name1 | name2)* )`
- Referenzen können nicht typisiert werden.
- Es gibt kein Namensraumkonzept. (Bzw. nur einen globalen Namensraum).

XML-Schema

- Geht in seinem Funktionsumfang weiter über DTD hinaus, da viel mehr Semantik eingefangen wird.
- Ist selbst als XML-Sprache modelliert.
- Rangfolge der Datentypen:
  - **Atomare Typen**
    - Eingebaute Typdefinitionen
    - z.B. für String, Integer, PositiveInteger, Short, Boolean, ...
  - **Einfache Typen**
    - Besitzen keine Unterlemente.
    - Besitzen keine Attribute.
    - Erben von den atomaren Typen und schränken diese ein.
  - **Komplexe Typen**
    - Können Elemente und Attribute besitzen.

```
Wurzelelement: <schema>
```

```
Einfache Definition eines Elements:
<element name="elementname"
               type="datentyp"
               minOccurs="..." maxOccurs="..."
               default="Standardwert" | fixed="Festwert"
/>
```

```
Einfache Definition eines Attributes:
<attribute name="attributname"
           use="optional"             |
           use="required"             |
           use="default" value="..." |
           use="fixed" value="..."   |
/>
```

- Reine Typbeschreibung vs. Dokumentendefinition (Definition von Elementen und Attributen)

Reine Typbeschreibung	Dokumentbeschreibung
`<simpleType name="stype" base="UnsignedShort"> <maxInclusive value="200"/> </simpleType>`	`<attribute name="..." type="stype"/>`
`<simpleType name="myList" derivedBy="list"/>`	`<element name="..." type="myList"/>`

```
Syntax von Elementausprägung durch Typen:
<element type="...">                          (Externe Definition referenzieren)
<element> <complexType>|<simpleType> </element>      (Inline-Definition)
```

```
Syntax komplexer Typdefinitionen:
<complexType>
    <attribute>
    <sequence> (Feste Rhfg.) | <all> (Beliebige Folge) | <choice> (Auswhal)
        <element>
```

- **Typhierarchien:**
  - Baum ohne Mehrfachvererbung
  - Typen sind entweder **atomar** oder eine **extension**/**restriction** eines bestehenden Types.
  - **Erweiterung:**
    - `<complexType> <extension base="..."/>`
    - Rest wie gehabt (s.o.)
  - **Einschränkung:**
    - `<complexType> <restriction base="..."/>`
    - Rest wie gehabt (s.o.)
    - Allerdings können nur Einschränkungen ergänzt oder verschärft werden. (`minOccurs, maxOccurs, use`)
    - Einschränkungen können nicht aufgelockert werden (vgl. `Use`).
    - Die Struktur kann nicht (durch weglassen von Elementen oder Attributen) verändert werden.

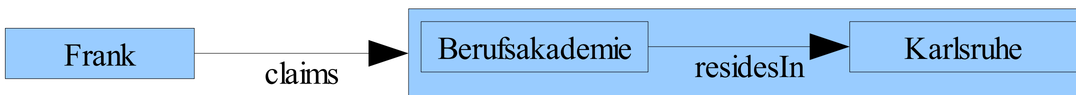
Resource Description Framework

- **Allgemein**
  - RDF ist ein vom W3C definierter Standard zur Modellierung bestimmter Meta-Daten.
  - Einfaches, graph-basiertes Datenmodell.
  - Häufig XML als Träger. Invariant gegen syntaktische Unterschiede innerhalb von XML. (**Syntaktische Invarianz**).

- Physikalisches Datenmodell von XML (Baum) vs. Logisches Datenmodell von RDF (Tripel).

- **Statements:**
  - `(Subjekt, Prädikat, Objekt)-Tripel`
  - z.B. `(Netherlands, hasCapital, Amsterdam)`

  - Beschreiben durch URIs spezifizierte Resourcen. URIs enden gerne mit einer `#fragmentID`
  - Prädikat und Objekt können unterschiedlichen Besitzern gehören (Lokationen im Web)
- **Semantische Netze:**
  - Entstehen dadurch, dass das Prädikat eines Tripels zum Subjekt eines anderen Tripels wird.
- **Reifikation:**
  - Aussage über eine Aussage.
  - `(Frank, claims (Berufsakademie, residesIn, Karlsruhe))`



Allgemeine Syntax von Trippeln

```
<rdf:description rdf:about="identifier">
    Properties:
    <hasCapital rdf:resource="#Amsterdam"/>
    <areaCode>20</areaCode>
</rdf:description>
```

Inlinedefinition einfacher Properties:

```
<rdf:description rdf:about="identifier" areaCode="20">
```

Verschachtelte Definition:

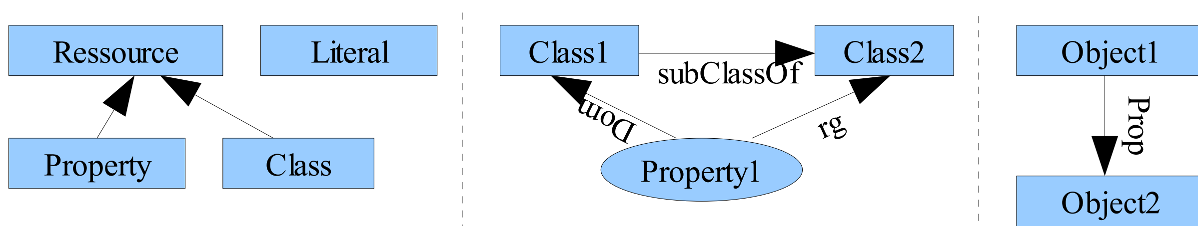
```
<rdf:description rdf:about="identifier">
    <hasCapital rdf:resource="#Amsterdam">
      <areaCode>20</areaCode>
    </hasCapital>
</rdf:description>
```

- Verwendung von **Namensräumen:**
  - **RDF-Elemente und RDF-Attribute:** `rdf:`
  - **Propertey-Elemente:** Keinen oder eigenen
- Es besteht eine formaler Unterschied zwischen **Objekten** und **Relationen zwischen den Objekten**.

- RDF macht Relationen zwischen Objekten explizit, indem diese einfach niedergeschrieben werden.
- Es fehlt aber eine Definition des dafür verwendeten Vokabulars und Hintergrundwissen. (Es folgt RDFS)

Resource Description Framework Scheme

- RDF-Daten sind nur beziehunggebende Metadaten zu anderen Daten. Für den Parser spielen weder die eigentlichen Daten, noch die Metadaten einer Rolle. Für ihn ist alles einfach nur **Zeichenmanipulation**.
- RDFS ist eigentlich RDF, wobei bestimmte Properties eine feste Bedeutung erhalten.
  - ☐ Daraus folgt: RDFS besitzt wieder ein anderes Datenmodell als RDF, da es eben einige Properties „verstehen kann".
  - ☐ Durch diese genormten Properties wird **Hintergrundwissen** modelliert, welches zur **automatischen Inferenz** verwendet werden kann.
- Insgesamt gibt es somit drei logische Abstraktionsebenen: **Sprache, Schema, Instanz**



- Typische Schlüsselwörter:
  - ☐ `rdf:about="...", rdf:resource="...", rdf:type="..."`
  - ☐ `<rdfs:class>, rdfs:subClassOf="..."`
- Durch die **formale Semantik** lassen sich weitere Statements aus dem Modell erzeugen. (**Inferenz**). Es besteht durch die **Ableitungsregeln** also **implizites Wissen**.
  - ☐ `X Property Y`
    - ■ `Property Range S  =>  X Type S`
    - ■ `Property Domain T  =>  Y Type T`
  - ☐ `T1 subClassOf T2, T2 subClassOf T3  =>  T1 subClassOf T3` **(transitiv)**
  - ☐ `X type T1, T1 subClassOf T2  =>  X type T2` **(transitiv)**
  - ☐ `X type Class  =>  X subClassOf X`
  - ☐ *Vorsicht vor den sich daraus ergebenden Randeffekten.*
- RDF und RDFS können über eine gemeinsame Trägerdatei **integriert** werden. Sie können auch getrennt werden, indem in der RDF-Datei ein RDFS-Namespace mit der URI des RDFS-Dokuments eingerichtet wird.

Syntax der Klassendefinition:

```
<rdf:description rdf:about="#Klassenname">
    <rdf:type rdf:resource="#class"/>
    <rdfs:subClassOf rdf:resource="#Klassenname"/>
    ....
</rdf:description>
```

```
<rdfs:class rdf:about="#Klassenname">
    <rdfs:subClassOf rdf:resource="#Klassenname"/>
</rdfs:class>
```

```
Syntax der (bewusst externen) Propertydefinition:
<rdfs:property rdf:about="#Propname">
    <rdfs:domain rdf:resource="#Klassenname"/>
    <rdfs:range  rdf:resource="#Klassenname"/>
</rdfs:property>
```

Web Ontology Language (OWL)

- **Ontologie**: Wissensrepräsentation in der KI anhand von **Begriffsbildung (**Klassen), **Begriffsverwendung (**Instanzen) und **Relationen** zwischen den Instanzen (auch Eigenschaften genannt).
- RDFS ist nur als „kleine Ontologiesparache" geeignet. Daraus folge der „Need for Expresivity".
  - OWL Full (OWL-Dokument = RDF-DOkument !!!) (Unentscheidbar) (wenig Tools)
    - RDF/S
    - OWL DL (z.B. keine Reification) (Entscheidbar) (gute Toolunterstützung)
      - OWL Lite (wenigst viele Features aber gute Performance)

OWL	RDF
Klasse	Klasse
Instanz	Objekt
Rolle	Property

```
owl:Nothing --subClassOf--> myClass --subClassOf--> owl:Thing
```

```
Kopfdaten eines OWL-Dokuments:
<rdf:RDF>
    <owl:ontology about="">
        <owl:imports rdf:reccourse="..."/>
        rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
        owl:versionInfo, owl:priorVersion, owl:backwardsCompatibleWith, ...
    </owl:ontology>
    Bodybereich hier
<rdf:RDF>
```

Definition einer Klasse:

```
<owl:class rdf:ID="Klassenname">
    <rdfs:subClassOf rdf:resource="#Klassenname"/>      (Erbschaft)
    <owl:disjointWith rdf:resource="#Klassenname"/>     (Abgrenzung)
    <owl:equivalentClass rdf:resource="#Klassenname"/> E(Synonym zu best. K.)
</owl:class/>
```

Komplexe Klassenbeziehungen:

```
<owl:class rdf:about="Bestehende Klasse bei about">
    <owl:equivalentClass>
        Schnittmenge
        <owl:intersectionOf rdf:parseType="Collection"> Classes </>

        Logisches Oder
        <owl:unionOf rdf:parseType="Collection">        Classes </>

         Negation (bei nur einer Class: Besser owl:disjointWith !!)
        <owl:complementOf rdf:parseType="Collection">   Classes </>
    </owl:equivalentClass>
</owl:class>
```

Instantiierung von Individuen:

```
<Klassenname rdf:ID="Instanzname"/> oder
<rdf:Description rdf:ID="Instanzname"> <rdf:type rdf:ID="Klassenname"/> </>

<MeineKlasse rdf:ID="MeinIndividuum">
    <RolleNo1 rdf:ressource="URIno1"/>
    <RolleNo2 rdf:Datatype="&xsd;String"> Klaus </RolleNo2>

    <owl:sameAs rdf:resource="#IndName"/>
    <owl:differentFrom rdf:resource="IndName"/>
</MeineKlasse>
```

Abkürzung statt vieler owl:DifferentFrom

```
<owl:allDifferent>
    <owl:distinctMembers>
       Instantiierungen hier
    <owl:distinctMembers>
</owl:allDifferent>
```

Abgeschlossene Klassen, (Es gibt nur DIESE ...)

```
<owl:class rdf:about="...">
    <owl:oneOf rdf:parseType="Collection">
        Instantiierungen hier
    </owl:oneOf>
</owl:class>
```

Abstrakte Rollen: (zwischen zwei Klassen)

```
<owl:ObjectProperty rdf:ID="Propname">
    <owl:subPropertyOf rdf:resource="#Propname"/>
    <owl:equivalentProperty rdf:resource="#Propname"/> ENDE (Synonym zu)
    <owl:inverseOf rdf:resource="#Propname"/> ENDE (Autom. Umkehrbeziehung)

    <rdfs:domain rdf:resource="Klassenname"/>
    <rdfs:range  rdf:resource="Klassenname"/>
</owl:ObjectProberty>
```

Konkrete Rollen: (Eine Klasse, ein atomarer Typ)

```
<owl:DatatypeProperty rdf:ID="Propname">
    <owl:subPropertyOf rdf:resource="#Propname"/>
    <owl:equivalentProperty rdf:resource="#Propname"/> ENDE (Synonym zu)

    <rdfs:domain rdf:resource="Klassenname"/>
    <rdfs:range  rdf:resource="&xsd;Typnme"/>
</owl:DatatypeProperty>
```

Rollen einer Klasse einschränken: (Innerhalb subClassOf / equivalentClas)

```
<owl:restriction>
```

```
    <owl:onProperty rdf:resource="#Propname"/>

    Einschränkung ALLER Range-Klassen:
    <owl:allValuesFrom rdf:ressource="#Klassenname"/>
    (Langform für owl:range aus der Property-Definition)

    Einschränkung MANCHER Range-Klassen: (mindestens eine)
    <owl:someValuesFrom rdf:ressource="#Klassenname"/>

    Wert-/Instanzeindschränkung:
    <owl:hasValue rdf:resource="#InstName"/>
    <owl:someValuesFrom> <owl:oneOf> <owl:Thing rdf:about="#InstName"/></></>

    Kardinalitäten:
    <owl:minCardinality rdf:datatype="&xds;nonNegativeInteger"> ... </>
    <owl:maxCardinality rdf:datatype="&xds;nonNegativeInteger"> ... </>
    <owl:Cardinality rdf:datatype="&xds;nonNegativeInteger">    ... </>
</owl:restriction>
```

(In Property deklarierbare) Rolleneigenschaften

- Domain und Range
- **Transitiv**: r(a, b) und r(b,c) => r(a,c) **(Nur in OWL Full deklarierbar)**
  `<rdf:type rdf:resource="&owl;transitiveProperty"/>`
- **Symmetrisch:** r(a,b) => r(b,a) **(Nur in OWL Full deklarierbar)**
  `<rdf:type rdf:resource="&owl;symmetricProperty"/>`
- **Funktional:** r(a,b) und r(a, c) => b = c **(Nur in OWL Full deklarierbar)**
  `<rdf:type rdf:resource="&owl;functionalProperty"/>`
- **Invers Funktional:** ??? **(Nur in OWL Full deklarierbar)**
  `<rdf:type rdf:resource="&owl;inverseFunctionalProperty"/>`
- **Weiteres:** r(a, b) und X r Y => X typeOf a und Y typeOf b
- ...

Typische OWL-Anfragen

- **Klassenäquivalenz**
- **Subklassenbeziehung**

- **Disjunktheit** von Klassenbeziehungen
- **Globale Konsistenz** (Erfüllbarkeit, Keine Widersprüche)
- **Klassenkonsistenz** (inkonsistente Klassen sind äquivalent zu owl:Nothing)
  ```
  <owl:subClassOf rdf:ressource="#Klasse">
  <owl:disjointWith rdf:ressource="#SelbeKlasse">
  ```
- Suche nach **Individuen einer Klasse**
- Frage **ob** ein **gegebenes Individuum zu** einer **Klasse gehört**

Alle OWL-Sprachelemente im Überblick

Kopf
- ***rdfs:comment***
- ***rdfs:label***
- ***rdfs:seeAlso***
- ***rdfs:isDefinedBy***
- ***owl:versionInfo***
- ***owl:priorVersion***
- ***owl:backwardCompatibleWith***
- ***owl:incompatibleWith***
- ***owl:DeprecatedClass***
- ***owl:DeprecatedProperty***
- ***owl:imports***

Beziehungen zwischen Individuen
- ***owl:sameAs***
- ***owl:differentFrom***
- ***owl:AllDifferent*** (zusammen mit ***owl:distinctMembers***)

Vorgeschriebene Datentypen
- ***xsd:strong***
- ***xsd:integer***

Klassenkonstruktoren und -beziehungen
- ***owl:Class***
- ***owl:Thing***
- ***owl:Nothing***
- ***rdfs:subClassOf***
- ***owl:disjointWith***
- ***owl:equivalentClass***
- ***owl:intersectionOf***
- ***owl:unionOf***
- ***owl:complementOf***

Rollenrestriktionen
- ***owl:allValuesFrom***
- ***owl:someValuesFrom***
- ***owl:hasValue***
- ***owl:cardinality***
- ***owl:minCardinality***
- ***owl:maxCardinality***
- ***owl:oneOf***

Rollenkonstruktoren, -beziehungen und -eigenschaften
- ***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"***