Grundstudiumspraktikum

Mehrsprachigkeit im Semantic Web

Cristina Vertan
vertan@informatik.uni-hamburg.de

Inhaltsübersicht

- Was ist und was bringt das Semantic Web ?
- Wie implementiert man das Semantic Web ?
- Was implementieren wir im Praktikum?
- Organisatorische Details

Was ist das Semantic Web ?

- Die Idee wurde erstmal von Tim Berners-Lee 1998 vorgeschlagen
- Es gibt unterschiedliche Definitionen entsprechend den unterschiedlichen Aspekten (e-commerce, Netzwerk, KI, Wissensmanagement, Sprachverarbeitung)
- Semantic Web Agreement Group (SWAG) (2000)

SWAG 2001

Das Semantic Web ist ein Web für Dokumente und Dokumentteile, das explizit die Beziehungen zwischen Objekten beschreibt. Es enthält die semantische Information, die für maschinelle Verarbeitung benötigt wird.

Warum ist das Semantic Web nötig -1-

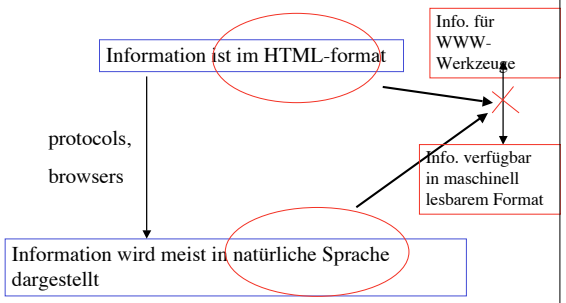
- Ca. 3 Milliarden statische Dokumente in WWW
- ca. 200 Milliarden Benutzer
- diese Ziffern steigen kontinuerlich

WWW heute

- Suchfunktionnen
- Zugänglichkeit
- Darstellungsmöglichkeiten
- Information up-to-date

schwierig

Grenzen des aktuellen WWW -2-



Suche in WWW

- meistens stellen die aktuellen Suchmaschinen viel zu viel irrelevante Information zur Verfügung.
- Die Suchmaschinen geben nur Pointer zu Dokumenten. Um die gewünschte Information zu erhalten, muss man die Dokumente lesen
- Wichtige Information für den gesuchte Begriff wird nicht gefunden weil sie mit anderen als den eingegeben Wörtern (oder in anderer Sprache) repräsentiert ist
- Die Situation ist schlechter, wenn man die Suche auch auf multimodale Information erweitert

Informationsdarstellung im WW

- mit den aktuellen Web-Werkzeugen ist es sehr schwer nicht-redundante und konsistente Information zu pflegen
- Die Webmaster sind sehr oft überfordert
- Viele Webseiten enthalten widersprüchliche Information

Was bringt das Semantic Web ?

Der Inhalt wird für die Maschinen verfügbar

- verbesserte zielorientierte Suche
- widersprüchliche Information wird automatisch entdeckt
- Informationsextraktion erleichtert und erweitert.
- Die Suche wird sich nicht mehr nur auf die Eingabesprache beschränken

Inhaltsübersicht

- Was ist und was bringt das Semantic Web ?
- Wie implementiert man das Semantic Web ?
- Was implementieren wir im Praktikum?
- Organisatorische Details

WWW Veränderung im Semantic Web

Web 2te Generation

Semantic Webseiten:

XML, RDF -Technologien, die Bedeutung mit den Daten verbinden

Web = XML-basierte Datenbank + Softwareagenten

Web 1ste Generation

Dynamische Webseiten:

DHTML, Javascript, Java,

Server-side Technologien

Statische Webseiten:

HTML, Hyperlinks, GIFs, Datenbanken+SQL

Was ist SGML ?
(Standard Generalized Markup Language)

Internationaler Standard, der die Regeln beschreibt, mit denen man Strukturen eines Dokuments in diesem selbst beschreiben kann.

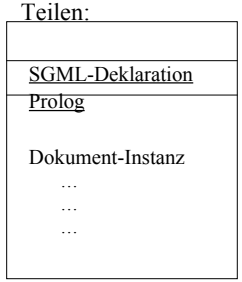
Legt keine spezifische Dokumentstruktur fest, sondern	definiert Regeln und die Syntax zum Aufbau strukturierter Dokumente
SGML-Dokumente sind nicht formatiert oder layoutiert	Die Darstellung der SGML-Dokumente ist Aufgabe eines Browsers
ist kein Format, sondern	Basiert nur auf 7-Bit-ASCII
ist nicht neu, sondern	Wurde 1970 entwickelt für Aufbereitung und Austausch von Rechtstexten

Vorteile von SGML

- Etablierter Standard zur Strukturierung von Dokumenten
- SGML-Dokumente
  - sind plattformneutral
  - besitzen ein einheitliches Erscheinungsbild
  - besitzen eine "Checkliste" für den Redakteur
  - leichter recherchierbar als spezifisch formatierte Texte
  - sind medienneutral aufgebaut
  - benötigen weniger Speicherbedarf als konventionelle Texte
- SGML-Bestandteile sind wiederverwandbar.

Aufbau von SGML-Dokumenten

- Ein vollständiges SGML-Dokument besteht aus 3 Teilen:



Der Prolog einer Markup-Sprache hat nichts mit der Programmiersprache „Prolog“ zu tun!

SGML-Deklaration

- Beschreibt formal die für ein bestimmtes Dokument verwendeten Teile des gesamten SGML-Standards,
- beinhaltet Informationen für die Weiterverarbeitung von SGML-Dokumenten (z.B. durch externe Dateien),
- wird vom SGML-System gelesen und interpretiert,
- definiert:
  - den Dokumentzeichensatz, d.h. die als Markierung zu interpretierenden Zeichen,
  - die zulässige Schachtelungstiefe von Strukturen usw.,
- Ist sehr schwer für SGML-Neulinge zu durchschauen,
- viele SGML-Systeme (z.B. FrameMaker+SGML) greifen deshalb auf eine Standard-SGML-Deklaration zurück.

Der Prolog und die DTD (Dokument Type Definition)

- Der Prolog enthält die Strukturregeln (DTD) für ein SGML-Dokument
- DTD = die Definition der Struktur und der Strukturelemente für eine Klasse von SGML-Dokumenten.

DTD Datei-Beispiel: Datei **anthologie.dtd:**

```
<!ELEMENT anthologie      - - (gedicht+)>
<!ELEMENT gedicht         - - (titel?, strophe+)>
<!ELEMENT titel           - O (#PCDATA)>
<!ELEMENT strophe         - O (reihe+)>
<!ELEMENT reihe           O O (#PCDATA) >
```

Der Prolog zitiert dann die Datei **anthologie.dtd** :

Prolog Beispiel:

```
<DOCTYPE Anthologie SYSTEM “c:\...\anthologie.dtd”>
```


Eine Dokument-Instanz

```
<!ELEMENT anthologie      - - (gedicht+)>
<!ELEMENT gedicht         - - (titel?, strophe+)>
<!ELEMENT titel           - O (#PCDATA)>
<!ELEMENT strophe         - O (reihe+)>
<!ELEMENT reihe           - O (#PCDATA) >
```

```
<anthologie>
<gedicht>
<titel>An die Muse
<strophe>
<reihe> Was ich ohne dich wäre, ich weiß es nicht - aber mir grauet,
<reihe> Seh ich, was ohne dich Hundert’ und Tausende sind.
</strophe>
</gedicht>
<!-- ...andere Gedichte ... -->
</anthologie>
```

Die Syntaxkorrektheit wird von einem Parser überprüft

SGML-Konstrukte

- Erscheinen im DTD-Teil.

Elemente: legen die Bestandteile des Dokumentinhaltes fest
<!ELEMENT gedicht - - (titel?, strophe+)>

Attribute: geben zusätzliche Informationen zu einem Element

<!ATTLIST gedicht sprache (deutsch | englisch) deutsch>

In der Dokumentinstanz: <gedicht sprache=englisch>

Text </gedicht>

Entitäten: abstrakte Bezeichnung für Daten

<!ENTITY uuml "ü">

In der Dokumentinstanz: <reihe>.. ist für... </reihe>

SGML-Tools

- SGML-Parser: überprüfen das SGML-Dokument hinsichtlich:
  - der Gültigkeit der DTD im Sinne von SGML
  - der Konformität der Dokumentinstanz bezüglich DTD
- SGML-Browser: Anzeigesysteme für SGML-Dokumente
- SGML-Editoren:
  - native SGML-Editoren (z.B. look and feel einer Datenbankoberfläche)
  - WYSIWYG - Editoren

SGML-basierte Anwendungen

- MARTIF ist ein SGML-basiertes Austauschformat für terminologische Daten
  - fachsprachliche Kommunikation braucht korrekte Terminologie.
  - Abhilfe für die Probleme
    - traditionelle Medien (Fachwörterbücher, Glossare usw.) wurden durch Entwicklungen im Bereich der elektronischen Datenverarbeitung stark verdrängt
    - Terminologiedatenbanken wurden von jeder Nutzergruppe anders definiert.
  - 1997 wurde das Terminologie-Austauschformat MARTIF (MAchine-Readable Terminology Interchange Format)definiert
  - spezifiziert die DTD des SGML-Dokuments mit den entsprechenden Tags für die Strukturierung der Daten

Martif-Beispiel

```
<martif>
<martifHeader>
.....
</martifHeader>
<text>
<body>
<termEntry>
<descripGrp>
<descrip type='subjectFieldLevel1'>appearence of material</descrip>
<ntig lang=de>
<termGrp>
<term>Opazität</term>
<termNote type='partOfSpeech>n</termNote>
<termnote type='grammaticalGender'>f</termnote>
</termGrp>
<descripGrp>
<descrip type='definition'> Maß für Lichtundurchlässigkeit
</descrip></descripGrp>
</ntig>
.....
```

SGML-basierte Anwendungsklassen

- SGML-basierte bibliographische Datenbank für Nachschlagewerke
- SGML-basierte Publikationprozesse im Verlag
- Semantisches Markup zur Inhaltserschließung von Agenturmeldungen
- Computerunterstützte Textanalyse (Textannotation)

SGML-Dokumente sind plattformneutral

- Da es sich bei SGML-Dokumenten um reine ASCII-Texte mit Strukturauszeichnungen handelt, sind SGML-Dokumente weitestgehend plattform- und softwareunabhängig.
- Neben einer beliebigen Portierbarkeit wird somit die Langlebigkeit von Dokumenten unterstüzt, da SGML eben nicht auf einem herstellerspezifischen Speicherformat, speziellen internen Formatierungen oder gar speziellen Hardwarevoraussetzungen basiert.

SGML-Dokumente besitzen ein einheitliches Erscheinungsbild

- Durch "formatierungsunabhängige" Struktur-beschreibungen für Dokumente ist ein einheitlicher und konsistenter Aufbau der Dokumente gewährleistet. Prüfprogramme (Parser) gewährleisten die Vollständigkeit und syntaktische Korrektheit der (Instanz-) Dokumente.

SGML-Dokumente besitzen eine "Checkliste" für den Redakteur

- SGML-fähige Redaktionssysteme, wie z.B. FrameMaker+SGML, unterstützen den Redakteur bei der Erstellung von Dokumenten durch einen kontextsensitiven Elementkatalog

SGML-Dokumente sind leichter recherchierbar

- Gut strukturierte Dokumente ermöglichen eine viel genauere Recherchemöglichkeit von Informationen
- z.B. Eine Volltextrecherche nach dem Begriff *"Werkzeug"* ist unpräziser als die Abfrage *"alle Kapitel die in der Kapitelüberschrift den Begriff "Werkzeug" enthalten"*. Das ist eine Information, die man über das Layout allein nicht erhält.

SGML-Bestandteile sind wiederverwendbar

- Da SGML-Dokumente modular aufgebaut sind, lassen sich die Komponenten eines SGML-Dokumentes einzeln ablegen, z.B. In einer SGML-Datenbank, und in unterschiedlichen Kontexten zur Erreichung von Einheitlichkeit gezielt wiederverwerten.
- Beispiele: Definitionen in einem Lehrbuch, Ergebnisse von Bundesligaspielen, Loseblattsammlung mit Verordnungen

SGML-Dokumente sind trägerneutral aufgebaut

- SGML eignet sich hervorragend für eine träger- (medien-) neutrale Informationsaufbereitung.
- Ein Browser zeigt ein SGML-Dokument an, daraus kann man z.B. drucken
- SGML-Dokumente können z.B. zusammen mit einem geeigneten Browser auf eine CD gebracht und verteilt werden.
- SGML-Dokumente können ohne großen Aufwand nach HTML konvertiert werden und damit Internet-fähig sein.

SGML-Dokumente benötigen weniger Speicherbedarf

- SGML-Dokumente sind nur ASCII-Dokumente deshalb ist der Speicherbedarf um ein Vielfaches geringer als bei herkömmlichen Dokumenten aus einer Textverarbeitung.

SGML und das Web

- SGML ist sehr gut geeignet für “large-scale document management business”.
- Nicht so gut geeignet für Web-Publikationen denn:
  - Komplexe Software ist erforderlich, denn SGML ist sehr kompliziert und erfordert einen entsprechend komplizierten Parser, der für On-line Arbeiten im Netz (z.B. Browsen) zu langsam sein würde,
  - Nach dem SGML-Standard wäre für jedes Dokument unbedingt eine DTD nötig,
  - die DTDs haben eine sehr komplexe Struktur und sind nicht ohne weiteres wiederbenutzbar oder änderbar,
  - zwei Dokumente mit zwei unterschiedlichen DTDs können nicht gemischt werden.

XML versus SGML

- XML ist ein “application profile” von SGML: ein SGML-System kann XML-Dokumente lesen
- XML ist eine beschränkte Teilmenge von SGML
- XML hat keine eigene Deklaration
- XML kann SGML nicht ersetzen. SGML ist eine bessere und abstraktere Lösung für die Erzeugung und Entwicklung von komplexen Dokumenten und Datenbanken.

Was ist XML ?
(Extensible Markup Language)

- XML ist eine eingeschränkte Markup-Sprache für Dokumente, die strukturierte Information enthalten
- strukturierte Information ist z.B.:
  - Inhalt (Wörter, Bilder, usw.)
  - Hinweis über Inhaltbedeutung (Kapiteltitel, Überschriften, usw.)
- Die XML-Spezifikation definiert einen Standard für vereinfachte Markupentwicklung und ohne eine komplizierte Deklaration.
- Dokumente können sein:
  - Konventionelles Standarddokument
  - Vektor-Graphik
  - e-commerce-Transaktionen
  - Mathemetische Formeln
  - meta-Daten, usw.

XML Tools

- Editor: ein normaler ASCII-Editor
- Dokumente können mit Web-Browsern angesehen werden (z.B. Netscape Navigator ab Version 5, Internet Explorer ab Version 4.5)
- HTML-Tags können innerhalb von XML-Dokumenten benutzt werden.
- XML enthält keinen vordefinierten Tag für Hyperlinks. (dafür muss XML Linking Language - XLS - benutzt werden)
- XML-Dokumente können in HTML-Dateien eingelesen und z.B. mit JavaScript ausgewertet werden.

XML Beispiel

- Syntax ist ähnlich wie SGML.

```
<?xml version"1.0"?>                                          } XML-Prolog
<?xml-stylesheet href="style.css" type="text/css"?>
<!DOCTYPE Dokument [
<!ELEMENT Dokument (Titel, Abstrakt?, Kapitel+, Zusammenfassung, Bibliographie)>
        <!ELEMENT Titel (#PCDATA)>
        <!ELEMENT Abstrakt (#PCDATA)>
        <!ELEMENT Kapitel (Titel, #PCDATA)>
        <!ELEMENT Zusammenfassung (#PCDATA)>                  } DTD
        <!ELEMENT Bibliographie (Reihe+)>
        <!ELEMENT  Reihe (#PCDATA)>
]>
<Dokument>                                                    } Dokument-Instanz
<Titel> XML Einf&uuml;hrung</Titel>
<Abstrakt> .....Text .... </Abstrakt>
....
</Dokument>
```

Die Formatierung wird danach durch eine CSS - Datei (Cascading Style Sheets) style.css erzeugt.

XML- Erweiterungen

- **LT-XML**
  - Entwickelt an der Universität Edinburgh (Language Technology Group)
  - XML parser + flexible API + Toolsammlung für XML marked-up Dokumenten-Verarbeitung

```
<p id='p1'>
<s id='s1'>
<w pos='prep'>In</w>
<w pos='art'>the</w>
<w pos='n'>beginning</w>
<w pos='v'>was</w>
<w pos='art'>the</w>
<w pos='n'>word</w><c>.</c>
</s>
</p>
```

Beispiel.xml

%textonly -s ' '<sample.xml
In the begining was the world.

%sgcount <sample.xml
p 1
s 1
w 6
c 1

Ergebnisse der Auswertung

%sggrep -q '.*/w[pos="n"]' sample.xml
<w pos='n'>beginning</w>
<w pos='n'>word</w>

XML vs. HTML

HTML wurde mit SGML definiert

MathML
LT-XML
SMIL

HTML entspricht nicht den XML-Standards

XML-Kode kann nicht in HTML-Dokumente eingebaut werden

XHTML

Was ist XHTML ?

- Redefinition von HTML tags entspechend der XML-Syntax
- die Dokumentstruktur bleibt dieselbe

```
<html>
   <head>
    <!--   ... Head-Inhalt ....-->
   <title> Dokumenttitel </title>
   <!-- ...  Head-Inhalt ... -->
 </head>
 <body>
    <!-- ... Body-Inhalt ... -->
</body>
</html>
```

- Einige Tags haben in XHTML **NICHT** dieselben Namen wie in HTML

XHTML vs. HTML

- Alle Tag- und Attribut-Namen müssen in Kleinbuchstaben geschrieben werden (weil XML kleine und grosse Buchstaben unterscheidet)
- Es gibt keine optionalen Ende-tags. Alle Tags müssen ein Paar sein (also auch z.B. ...** **)
- Alle leeren Tags enthalten wie in XML ein Leeres-Element-Tag)
  <hr />
- es gibt nur ein einziges `head`- und ein einziges `body`- Element. Stattdessen kann man nur ein einziges `frameset` -Element einfügen
- jedes `head`-Element darf nur ein einziges `title` -Element (Tag) enthalten

XHTML vs. HTML (Beispiel)

```
<html>
<head>
<title> Vorlesung CP
   Content</title>
</head>
<body>
<h1> Vorlesung CP </h1>
<hr / >
<h2>Inhalt</h2>
<ul>
  <li>01 Intro </li>
  <li> 02 Theorie </li>
</ul>
</body>
</html>
```

```
<HTML>

<body>
<h1> Vorlesung CP </h1>
<hr>
<h2>Inhalt</h2>
<ul>
  <li>01 Intro
  <li> 02 Theorie
</ul>
</body>
</html>
```

Viele HTML-Editoren ergänzen HTML zu XHTML

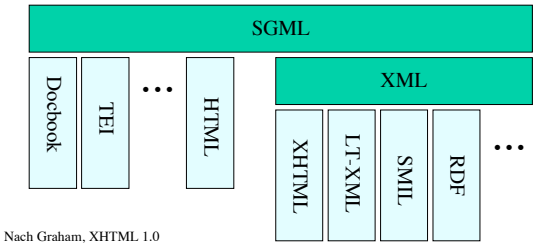
Strictly Conforming XHTML

- Entspricht einem strikten XML-Formalismus:
  - Spezifiziert, dass das Dokument vollständig XML-formatiert ist:
    <?xml version="1.0" charset="iso-8859-1" ?>
  - benennt eine DTD
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict// EN"
    "http://www.w3.org/TR/xhtml1/DTD/strict.dtd" >
  - das <html> Element muß ein "xmnls" Attribut enthalten, um zu spezifizieren wo die Elementnamen definiert sind:
    <html xmnls="http://www.w3.org/TR/xhtml1">
    ...
    </html>

Strictly Conforming XHTML - Beispiel

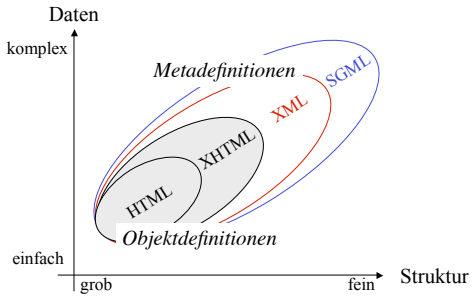
```
<?xml version="1.0" charset="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Strict// EN"
 "http://www.w3.org/TR/xhtml1/DTD/strict.dtd" >
<html xmnls="http://www.w3.org/TR/xhtml1">
<head>
<title> Vorlesung CP Content</title>
</head>
<body>
.....
</body>
</html>
```

Derivate von SGML und XML



Nach Graham, XHTML 1.0

SGML, XML, XHTML und HTML



Bestandteile von XML

- Vorspann:
  - Version: z.B. <?xml version="1.0">
  - DTD: Verweis auf Grammatik: <!DOCTYPE…>
- Elemente (Tags): z.B. <offer>,<store>,<book> usw.
  - Änlich wie in HTML aber selbstdefiniert
- Attribute: z.B. für >price>: type="retail".
  - Änlich wie in HTML aber selbstdefiniert
- Entitäten
  - Zur Wiederverwendung und modularisierung von Texten

Elemente

- Elemente sind die grundlegenden Komponenten eines XML-Dokuments
- Sie werden durch beliebige Namen eingeschlossen in < und > repräsentiert.
- D.H. zur Definition eines neues Elements definiert man einen neuen Tag sowie den zugehörigen Abschlusstag
- Elemente können innerhalb von anderen Elementen definiert werden (Hierarchien)
- Namenskonventionen wie bei der Programmierung

Beispiel

```
<?xml version="1.0">
<kontakt>
      <name>
            <vorname>Peter </vorname>
           <nachname>Mustermann</nachname>
     </name>
    <email>peter@firma.de</email>
   <telefon> 0236589 </telefon>
</kontakt>
```

- Wurzelelement: `kontakt`
- Elemente: `name, vorname, email, telefon`
- Drei Hierarchieebenen

Attribute

- Ein Element kann nicht nur Unter-elemente besitzen, sondern auch durch Attribute erweitert werden.
- Ein Attribut wird samt seinem Wert (immer in anführungszeichen) dem Start-Tag eines Elements hinzugefügt.
- Beispiel: ein Kapitel eines Buches hat einen Title und eine Nummer:

```
<chapter title="Introduction" number="4">
….
</chapter>
```


Wann Element, wann Attribut?

- XML gestattet die Datenformatierung auf viele Arten; u.a. ist es praktisch immer möglcih anstelle von Attributen auch Elemente zu verwenden und umgekehrt.
- Einiege Tipps:
  - Wenn die information selbst wieder komplex ist verwendet man Elemente
  - Modelllierung geordneter Information: Elemente
  - Leserlicher ist die Elementvariante, sie nimmt deshalb auch mehr Platz ein.
  - Ist die modellierte Information ein Bestandteil des übergeordneten Elements, verwendet man ebenfalls ein element. Ist sie eine Eigenschaft (wie etwa eine Farbe), dann verwendet man Attribute.

Validierung

- Die strengen Syntaxkonventionen erleichtern die Programmierung von XML Applikationen bereits erheblich (wohlgeformte Dokumente)
- Validierung erlaubt es, die möglichen Baumstrukturen einzuschränken (valide Dokumenten)
- Validierung findet immer gegenüber einer Grammatik statt, d.h., ein valides Dokument ist:
  - Wohlgeformt und
  - Konform zu einer vorgegebenen Grammatik
- Ein grosser Teil der Fehlerabfragen kann so von der applikation an einen Parse übertragen werden

Definition von Grammatiken

- Wenn zwei Anwendungen dieselbe Grammatik verwenden, haben sie dasselbe Vertändnis von Daten (wenigstens die Hierarchie)
- Es werden heute zwei verschiedene ansätze verwendet, um eine Grammatik zu beschreiben:
  - Document Type Definition (DTD)
  - XML Schema

XML Schema als Alternative zur DTD

- Die DTD-Syntax ist nicht XML-konform (sondern SGML)
- Keine Datentypen, nur Strings
- Begrentzte Erweiterbarkeit (vererbung)

- XML Schema behebt diese Probleme
  - Das Konzept bleibt dasselbe
  - Verschiedene Datentypen
  - Geschrieben in XML
- Vorgehen: schreibe ein Schema in XML, referenziere es von der XML-Datei aus.

Vorteile von XML-Schema

- Jedes XML Schema ist selbst ein XML-Dokument (im Gegensatz zu DTD keine spezielle Syntax mit speziellen Verarbeitungswerkzeugen erforderlich)
- Auch komplexe Integritätsbedingungen formulierbar
- XML Schema enthält vordefinierte und eigendefinierbare Datentypen, wodurch Typprüfung möglich wird
- Bei Datentypen werden Vererbung und substitution unterstützt.
- Benennungskonflikte können durch Verwendung von XML-Namensräumen vermieden werden.

Beispiel für XML-Schema -1-

```
<xsd:schema xmlns:xsd=http://www.w3.org/2002/XMLSchema”>
<element name=“paper” type=“papertype” />
<xsd:complextype name=“papertype”>
  <xsd:sequence>
     <xsd:element name=“autor” type=“xsd:string”/>
     <xsd:element name=“titel” type=“xsd:string”/>
     <xsd:element name=“datum” type=“xsd:gYearMonth”/>
     <xsd:element name=“link” type=“xsd:anyURI min Occurs=“0”/>
  </xsd:sequence>
  <xsd:attribute name=“typ” type=“art”/>
</xsd:complextype>
```

Beispiel für XML-Schema -2-

```
<xsd: simpleType name="art">
    <xsd: restriction base="xsd:string">
        <xsd:enumeration value="proc"/>
        <xsd:enumeration value="report"/>
        <xsd:enumeration value="journal"/>
        <!-- usw. -->
    </xsd:restriction>
</xsd:simpleType>
```


Parser

- Um ein XML-Dokument in einer Anwendung zu verwenden, muss man es einlesen und in interne Datenstrukturen der verwendetenProgrammiersprache umwandeln
- Es werden heute wesentlichen zwei Modelle für das Parsen von XML verwendet:
  - Simple API für XML (SAX)
  - Document Object Model (DOM)
- JavaXML-library bietet Lösungen für beides an.

Wie verarbeiten wir die Daten?

- Möglichkeit 1: schreibe eine anwendungfür den verwendeten Parser.
  - Z.B: lese Dokument als DOM-Baum ein, bearbeite dann den Baum
- Möglichkeit 2: bearbeite das Dokument mittel vorgegebener Regeln
- Hierzu verwendet man sog. Stylesheet-Transformationen (XSL)

Zusammenfassung

- XML an sich ist eine sehr grundlegende Technologie
- Die Stärken liegen in:
  - Der grossen Akzeptanz
  - Der enormen Menge an frei erhältlicher Software
  - Der Vielzahl der Anwendungsbereiche

Warum reicht XML-Annotierung nicht aus ?

- Beispiel : wir suchen etwas über *„Java programming language“*
- Google wirft alle Dokumente aus, die eines der eingegebenen Wörter enthalten. Deswegen erhalten wir auch Seiten über die Insel Java...
- mit XML-Annotierung kann man unterscheiden zwischen z.B.:

  <programming> Java </programming> und

  <geographisch>Java </geographisch>
- aber ....

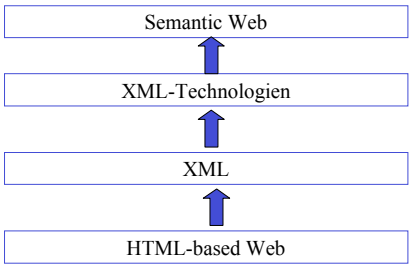
Was kann XML nicht lösen

- z.B. ein tag:

<programmierung>Java</programmierung>

- wird auch ignoriert weil die Maschine nicht versteht, dass die Bedeutung von programming und programmierung dieselbe ist

XML und WWW



XML-Sprachen für das Semantic Web

- Erweitern Web-Daten und Web-Ressourcen durch Bedeutung
- Diese Bedeutung ist hierarchisch spezifiziert
- RDF (Resource Description Framework) definiert ein einfaches Datenmodel als ein Tripel (Subjekt, Prädikat, Objekt)

  URI URI URI/Literal

- Ein RDF Schema (RDFS) beschreibt RDF-Merkmale, und ermöglicht die Darstellung einfacherer Ontologien

- DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer), OWL erweitern RDFS für die Darstellung von komplexeren Ontologien

Was ist RDF ?

- RDF ist eine Modellierungssprache, mit der man Informationnen über Ressourcen in WWW darstellen kann
- Man beschreibt Web-Ressourcen mit RDF um sie maschinell interpretierbar zu machen
- Ressourcen werden :
  - eindeutig im Web durch URI (Uniforme Resource Identifier) identifiziert
  - durch Merkmalen und deren Werte beschreibt

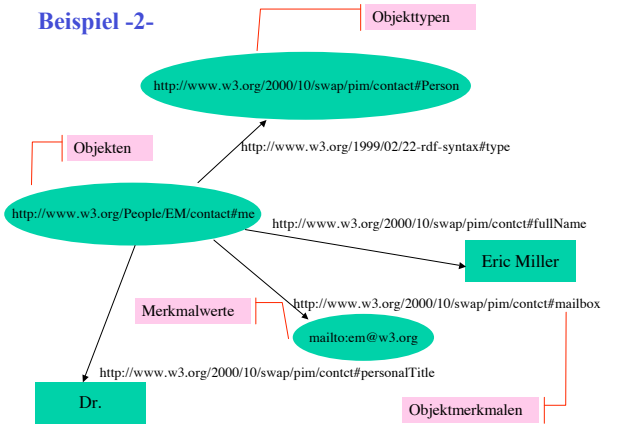
Beispiel -1-

- Wie kann man folgenden Ausdrücke darstellen:

„ es gibt eine Person die :

- von http://www.w3.ord/People/EM/contact#me identifiziert ist
- hat eine Name: Eric Miller
- hat eine e-mail Adresse: em@w3.org
- hat ein Titel: Dr.

Beispiel -2-



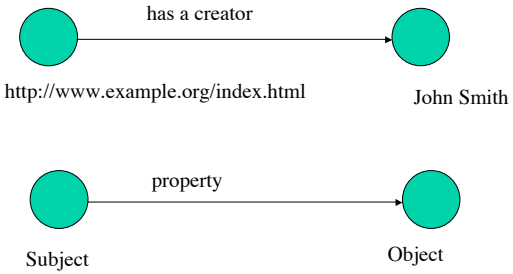
RDF/XML Syntax - Beispiel

```
<?xml version=„1.0“?>
<rdf:RDF xmnls:rdf=„http://www.w3.org/1999/02/22-rdf-syntax-
   ns#
   xmnls:contact=„http://www.w3.org/2000/10/swap/pim/contact#“>
  <contact:Person
   rdf:about=„http://www..w3.org/People/EM/contact#me>
      <contact:fullName>Eric Miller </contact:fullName>
      <contact:mailbox rdf:resource=„mailto:em@w3.org“/>
      <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

RDF: Feststellungen über Ressourcen treffen

- http://www.example.org/index.html has a creator whose value is John Smith
- Immer ein RDF-Triple
  - Etwas,über das ein Feststellung (statement) gemacht wird (http://www.example.org/index.html)- Subject
  - Eine Eigenschaft(properety), das beschrieben wird (has a creator)- Property / Prediacte
  - Wert der Eigenschaft (John Smith) - Object

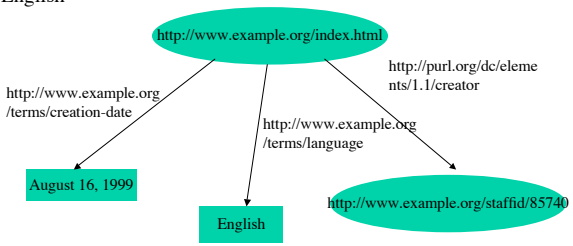
RDF Darstellung als Graph



Komplexere Beispiele

http://www.example.org/index.html has a creation-date whose value is August 16, 1999

http://www.example.org/index.html has a language whose value is English



URI /URL

- Berners-Lee (1998): “Uniform Resource Identifiers (URI) provide a simple and extensible means for identifying a resource”
- Zur Adressierung von Subject, Property oder Object eines Triples.
- URLs sind nur ein URI-Beispiel

Qualified Names und XML Namespaces

xmnls: ex="http://www.example.org/"
xmnls:dc="http://purl.org/dc/elements/1.1/"
xmnls:exterms= "http://www.example.org/terms/"
xmnls:exstaff=http://www.example.org/staff/

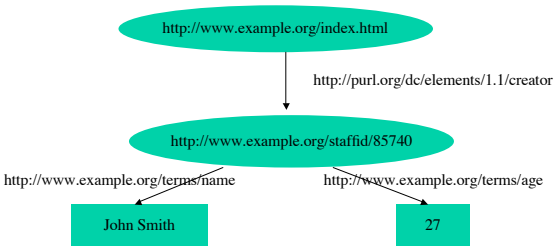
- RDF Tripel

ex:index.html	dc:creator	exstaff:85740
ex:index.html	exterms:creation-date	"August 16, 1999"
ex:index.html	exterms:language	"English"

Wer ist John Smith?

Identifizier eindeutig über ID seines Unternehmens

Name und Alter sind Objekte von John smith



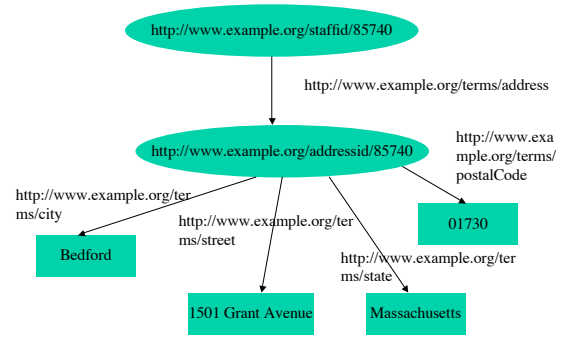
Identität

- Wie identität allgemeingültig beschreiben?
- Name (John Smith) ungünstig
- ID besser, aber noch nicht perfekt
- Andere Autoren können andere Ids benutzen.
- Mit RDF allein kann man nur syntaktische Gleichheit feststellen nicht inhaltliche

Komplexere Beispiel -1-

exstaff:85740	exterms:address	exaddressid:85740
exaddressid:85740	exterms:street	"1501 Grant Avenue"
exaddressid:85740	exterms:city	"Bedford"
exaddressid:85740	exterms:state	"Massachusetts"
exaddressid:85740	exterms:postalCode	"01730"

Komplexere Beispiel -2-



Blank Node- Beispiel-1

exstaff:85740 externs:address _:johnaddress

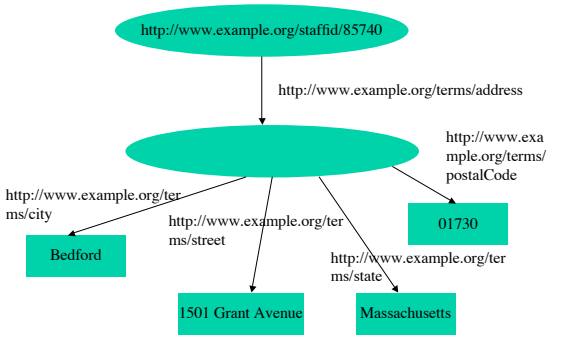
_:johnaddress externs:street “1501 Grant Avenue”

_:johnaddress externs:city “Bedford”

_:johnaddress externs:state “Massachusetts”

_:johnaddress externs:postalCode “01730”

Blank Node- Beispiel-2



Blank Node

- Nicht als Teil eines RDF Graphs angesehen
- Ein Mittle, um RDF-Graph in Triple-From darzustellen
- Keine Identität
- Nicht referenzierbar von außen
- Weg um n-stellige Relationen Triples zu wandeln
- Interessanterweise Hilfsmittel um nicht direkt über das beziehbare Dinge zu beschreiben

Jane Smith als Autorin

_:jane exterms:mailbox
_:jane rdf:type exterms:Person
_:jane exterms: name “Jane Smith”
_:jane exterms empID “23748”
_:jane exterms:age “26”

- Version 1:

Ex2terms:book78354 rdf:type ex2terms:Book
Ex2terms:book78354 ex2terms:author “Jane Smith”

- Version 2:

Ex2terms:book78354 rdf:type ex2terms:Book
Ex2terms:book78354 ex2terms:author _:authr78354
_:author78354 rdf:type ex2terms:Person
_:author78354 ex2terms:name “Jane Smith”

- Version 2 robuster.

Typed Literals

http://www.example.org/staffid/85740
http://www.example.org/terms/age>
“27”^^http://www.w3.org/2002/XMLSchema#integer

- Oder mit Qnames:

Exstaff:85740 exterms:age “27”^^xsd:integer

RDF Syntax in XML

- Ex:index.html exterms:creation-date “August 16, 1999”

```
<?xml version=“1.0”?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmnls:exterms=http://www.example.org/terms/” >
  <rdf:Description rdf:about=http://www.example.org/index.html “>
     <exterms:creation-date>August 16, 1999 </exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```


Mehr Statements über ein Subject

- Ex:index.html exterms:creation-date “August 16, 1999”
- Ex:index.html exterms:language “English”

```
<?xml version=“1.0”?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmnls:exterms=http://www.example.org/terms/” >
  <rdf:Description rdf:about=http://www.example.org/index.html “>
     <exterms:creation-date>August 16, 1999 </exterms:creation-date>
  </rdf:Description>
<rdf:Description rdf:about=http://www.example.org/index.html “>
     <exterms:language>English </exterms:language>
  </rdf:Description>
</rdf:RDF>
```

Abkürzung für mehr Statements für ein Subject

```
<?xml version="1.0"?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmnls:exterms=http://www.example.org/terms/"
         xmnls:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description rdf:about=http://www.example.org/index.html ">
     <exterms:creation-date>August 16, 1999 </exterms:creation-date>
     <exterms:language>English </exterms:language>
     <dc:creator rdf:resource=http://www.example.org/staffid/85740 "/>
  </rdf:Description>
</rdf:RDF>
```


Typed Literal

```
<?xml version="1.0"?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmnls:exterms=http://www.example.org/terms/" >
  <rdf:Description rdf:about=http://www.example.org/index.html ">
     <exterms:creation-date
          rdf:datatype=http://www.w3.org/2001/XMLSchema#date>
1999-08-16 </exterms:creation-date>
  </rdf:Description>
</rdf:RDF>
```


RDF bags -1-

```
<?xml version="1.0"?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
          xmnls s=http://www.example.org/students/vocab#/" >
  <rdf:Description rdf:about=http://www.example.org/courses/6.001 ">
    <s:students>
      <rdf:Bag>
       <rdf:li rdf:resource="http://example.org/students/Amy"  />
       <rdf:li rdf:resource="http://example.org/studentsJohann"  />
       <rdf:li rdf:resource="http://example.org/students/Eduard"  />
     </rdf:Bag>
    </s:students>
</rdf:Description>
</rdf:RDF>
```


RDF-Bags-2

```
<?xml version="1.0"?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
          xmnls:s=http://www.example.org/packages/vocab#/" >
  <rdf:Description rdf:about=http://www.example.org/packages/X11 ">
    <s:DistributionSite>
      <rdf:Alt>
       <rdf:li rdf:resource="ftp://ftp.example.org"  />
       <rdf:li rdf:resource= ="ftp://ftp1.example.org" />
      <rdf:li rdf:resource= ="ftp://ftp2.example.org" />
     </rdf:Alt>
    </s:DistributionSite>
</rdf:Description>
</rdf:RDF>
```

RDF- Collection

```
<?xml version="1.0"?>
<rdf:RDF xmnls:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
          xmnls s=http://www.example.org/students/vocab#/" >
  <rdf:Description rdf:about=http://www.example.org/courses/6.001 ">
   <s:students  rdf:parseType="Collection">
     <rdf:Bag>
      <rdf:li rdf:resource="http://example.org/students/Amy"  />
      <rdf:li rdf:resource="http://example.org/studentsJohann"  />
     <rdf:li rdf:resource="http://example.org/students/Eduard"  />
    </rdf:Bag>
  </s:students>
</rdf:Description>
</rdf:RDF>
```


RDF-Reification

```
<?xml version"1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
    "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmnls:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmnls:dc="http://purl.org/.dc/elements/1.1/"
          xmlns:exterms="http://www.example.com/terms/"
         xml:base="http://www.example.com/2002/04/products">
<rdf:Description rdf:ID="item10245">
    <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
</rdf:Description>
<rdf:Statement rdf:about="#triple12345">
    <rdf:subject
     rdf:resource="http://www.example.com/2002/04/products#item10245"/>
   <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/>
  <rdf:object rdf:datatype="&xsd;decimal">2.4>/rdf:object>
  <dc:creator rdf:resource="http://www.example.com/staffif/85740"/>
</rdf:Statement>
</rdf:RDF>
```


Anmerkungen zu RDF -1-

- URI's sind URL's sehr änlich aber nicht identisch. Ein URI bezeichenet eine Ressource die nicht unbedingt im WWW ist
- Die Semantik von "Vocabulary" und "Schema" ist dieselbe. Für RDF wird nur Vocabulary benutzt.
- Qname. Präfix :local name
  - Das Präfix ist einem "Parent folder"änlich. In RDF ist das Präfix ein URI.
  - Wenn mehrere URI's ein gemeinsames Präfix haben, bedeutet das nicht dass es zwischen die URI's irgendwelche Beziehungen gibt.

Anmerkungen zu RDF -2-

- Man muß sich gut überlegen ob ein Objekt eines Statements nicht Subjekt für ein weiteres Statement sein kann. Literals können nicht Subjekt sein !
- URI's identifizieren eine Ressource einmalig ABER: unterschiedliche URI's können dieselbe Ressource identifizieren. Es ergibt sich dann die Nötwendigkeit eine Beziehung zwischen diese URI's zu bilden (Ontologien).
- Was die Menschen von der URI bedeutung verstehen ≠ was die Maschinen verstehen

Anmerkungen zu RDF -3-

- Leere Knoten (Blank-nodes) werden meistens um eine N-are Relation in eine binäre Relation zu tranformieren.
- Mehrere leere Knoten können dieseleben Name haben
- Typed Literals: www.w3.org/2001/XMLSchema#type. Im Vergleich zu Programmierungssprachen, kann man nicht prüfen ob die Assoziazion: Literal ↔ Data-Typ korrekt ist.

Ontologien

- Sind hierarchische Konzeptstrukturen mit semantischen Relationen (Ober/Unterbegriff, Teil-von, etc.).
- Sie definieren Beziehungen zwischen Tags und erlauben damit semantische Annotation und ihre Abbildung auf natürlichsprachliche Wörter.

Grenze von RDF

- Alle RDF-tags geben Informationen über Bedeutungen von Wörtern, Ausdrücken, Texte, Webseiten aber:
- Woher soll ein automatischer Prozess wissen, welche Semantik diese Tags haben ?
- Z.B. woher soll ein Prozess wissen, dass "Altstadt" ein Unterbegriff von "Stadt" ist. Oder dass er deutsche Namen nur für siebenbürgische Städte suchen soll und nicht für alle Städte in Rumänien ?
- Für solche inhaltliche Beziehungen muss man die Tags in einer Struktur organisieren

Was ist eine Ontologie ? -1-

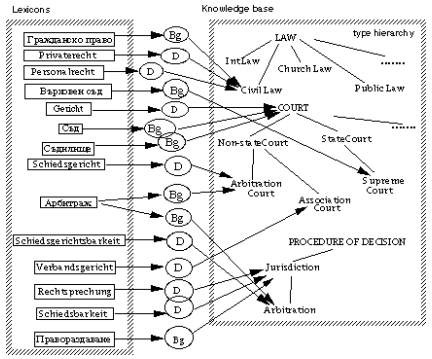
- Ursprünglich war es ein Begriff aus der Philosophie (seit Aristoteles) für:
  - Ein Forschungsgebiet der Metaphysik, das sich mit der Natur und Existenz beschäftigt oder
  - Eine spezielle Theorie über Existenz und Existenztypen
- In der Mathematik (seit dem XIX. Jahrhundert): Eine formale Ontologie ist eine Logik-Theorie
- In der Informatik - eine grobe Definition: Eine Konzepthierarchie

Was ist eine Ontologie? -2-

- Eine Ontologie definiert (Bezeichner für) eine begriffliche Beschreibung einer Domäne:
  - maschinell interpretierbare Definitionen von grundlegenden Begriffen des Domänes und
  - Beziehungen zwischen diesen Konzepten
- Ontologien sind meistens domänenspezifisch
- Da sie auf Konzepten und nicht auf natürlichsprachlichen Namen basieren, sind Ontologien grundsätzlich sprachunabhängig.

Ontologie - Beispiel -

Ontologie in DBR-MAT MAT System



Ontologien in Computeranwendungen

- Ontologien werden seit Jahrzehnten intensiv benutzt bei:
  - der Wissensrepräsentation
  - dem Wissensengineering
  - Sprachverarbeitung (z.B. MT, MAT)
  - Information Retrieval und Information Extraction
  - Anderen Web-Anwendungen
- Im Semantic Web sind die Ontologien der Hauptmechanismus für die Darstellung von semantischen Beziehungen zwischen Ressourcen.

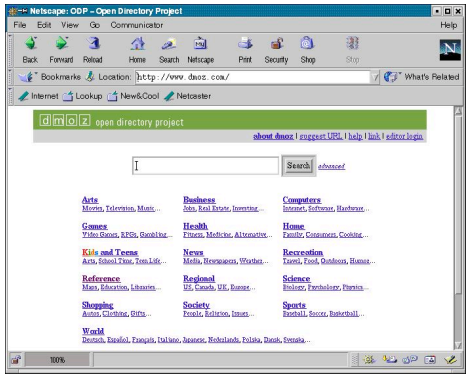
Ontologietypen

- Die primitivste Form einer Ontologie ist ein Thesaurus, (Glossar) d,h, man spezifiziert nur, welche Begriffe speziell für eine Domäne sind.
- Eine typische Ontologie enthält eine hierarchische Organisation von Begriffen. Wenn man nur die Beziehung "Subclass-Of" ("Superclass-Of") repräsentiert, heisst die Ontologie Taxonomie
- Eine spezielle Form sind die strukturierten Ontologien, die auch auch Klassenmerkmale und Werteeigenschaften und -bereiche (häufig leider auch "Domänen" genannt) enthalten.

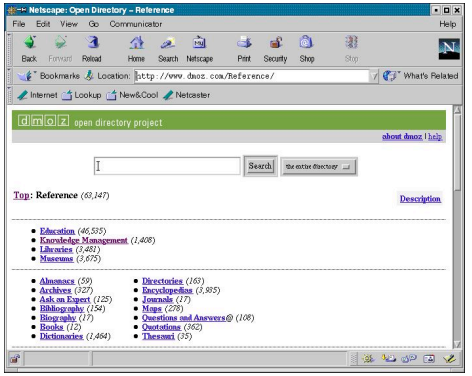
Einfache Ontologien - Taxonomien

- Sind relativ einfach zu entwerfen
- Merkmale:
  - kontrollierter Wortschatz für die Begriffe eines dargestellten Gebiets
  - meistens benutzt zur Datenorganisation
  - liefern die allgemeine Struktur: die Konzepte können dannach instantiiert werden (z.B. “The Universal Standard Product and Services Classification)
  - Die in Web vorhandenen Taxonomien enthalten auch browsing support
- Viele Browers bieten als Navigationsmöglichkeit das Browsing eine Taxonomie (z.B. DMOZ: directory Mozille www.dmoz.com hat über 460,000 Klassen und 45,000 Editoren)

DMOZ -Beispiel- 1

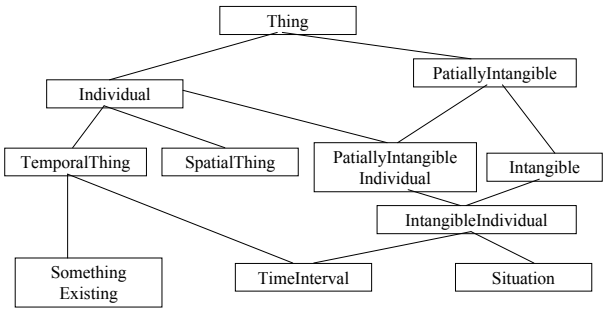


DMOZ -Beispiel -2



Cyc Ontology

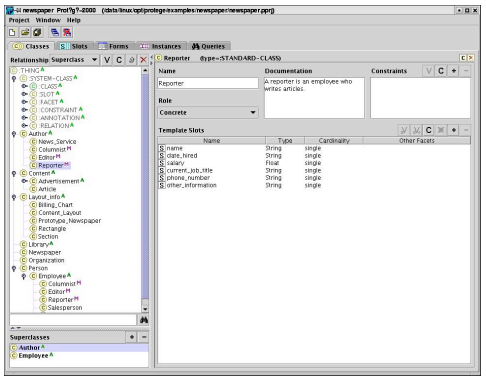
Komplexere Ontologien



Strukturierte Ontologien

- Die Klassen enthalten auch
  - Merkmale
  - Werte und gültige Domänen für diese Merkmale
- Vorteile:
  - Man kann sehr schnell einen Konsistenztest machen,
  - Man kann Inferenzregeln beschreiben,
  - Die Beziehungen zwischen Konzepten können detailliert beschrieben werden
  - Die Suche kann speziell oder allgemein sein (durch Beschränkung der Werte)

Strukturierte Ontologien -Beispiel



Hauptschritte beim Ontologieentwurf

- Definition von Ontologieklassen
- Entwurf einer Hierarchie (Taxonomie) zwischen Klassen (Super-/Subclass)
- Definition von Konzeptattributen /Merkmalen (slots) und deren Werten
- Eine Ontologie zusammen mit Klasseninstanzen = Wissensbasis.

Hauptprinzipien des Ontologieentwurfs

- Es gibt mehrere Alternativen, eine Domäne zu modellieren (d.h es gibt nicht nur **eine** Ontologie). Sehr oft ist die Ontologie anwendungsorientiert und es gibt mehrere Ontologien über ein Gebiet je nach Anwendungssicht)
- Der Entwurf einer Ontologie ist ein iterativer Prozess.
- Die Bezeichner sollen plausibel für die logischen oder physische Objekte und Beziehungen der modellierten Domäne sein. Sehr oft benutzt man für eine Ontologie:
  - Nomen für die Objekte ("Drehbank")
  - Verbausdruck für die Beziehungen ("hat_Eigenschaft")

Schritt 1- Domäne und Ziel der Ontologie

- Man muss das Gebiet und die Anwendung, in der die Ontologie benutzt wird, klar identifizieren d.h:
  - Welche Domäne soll die Ontologie modellieren?
  - In welcher Anwendung wird die Ontologie benutzt,
  - Für welche Typen von Anfragen will man die Ontologie durchsuchen?
  - Wer wird die Ontologie weiter pflegen?

Schritt 2: Wiederbenutzbarkeit?

- Es gibt schon viele Ontologien für die unterschiedlichsten Gebiete. Sehr oft sind sie anwendungsunabhängig kodiert (besonders solche, die für Webanwendungen entworfen wurden)
- Ontologie-Bibliotheken:
  - http://www.ksl.stanford.edu/software/ontolingua
  - http://www.daml.org/ontologies

Schritt 3 - Wichtige Konzepte

- Man muss wichtige Konzepte im modellierten Gebiet identifizieren
- Diese Identifizierung steht immer in Zusammenhang mit der Anwendung, d.h. man muss sich überlegen, welche Konzepte sollten danach erklärt werden
- In unserem Fall z.B. muss man das Fragekorpus ansehen und festlegen, welche mögliche angefragte Konzepte sind.

Schritt 4 - Klassenhierarchie

- Es gibt 2 Möglichkeiten, eine Klassenhierarchie zu definieren:
  - Bottom-up man beschreibt die sehr spezifischen Klassen (Blätter) und dann gruppiert man sie schrittweise in allgemeineren Konzepten
  - Top-down : man definiert erst die sehr allgemeinen Konzepte und dann verfeinert man sie.

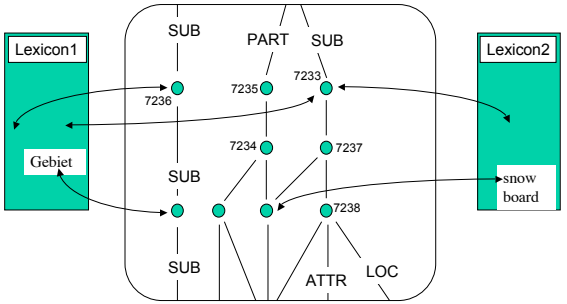
Schritt 5 - Entwurf einer strukturierter Ontologie

- Man muss für jede Klasse Merkmale identifizieren und für jedes Merkmal
  - Die Kardinalität: wieviele mögliche Werte
  - Werttypen (string, number, bolean, enumeration)
  - Wertedomäne = die Klassen die dieses Merkmal haben
- Für Wertedomänen muss man Redundanz vermeiden:
  - Wenn in eine Wertedomäne eine Klasse und zugelich eine ihrer Unterklassen enthält, muss man die Unterklasse löschen
  - Wenn in einer Wertedomäne nur Unterklassen von Klasse X aber nicht die Klasse X selbst sind, muss man die Unterklassen durch die Klasse X ersetzen

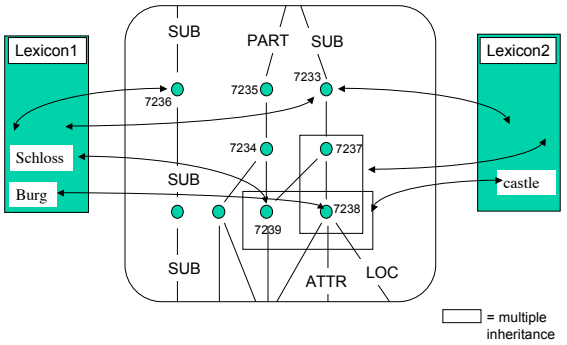
Schritt 6 Instantiierung

- Für eine Klasseninstantiierung muss man:
  - Eine Klasse auswählen
  - Ein Individuum dieser Klasse spezifizieren
  - Die Attributwerte spezifizieren
- Man muss sich immer überlegen, welche Begriffe eines Gebietes Klassen sind und welche Begriffe Instanzen von Klassen

Zweiseitige lexikalische Anbindung



Mehrfache Vererbung für die unterschiedliche Konzeptualisierung



Weitere Hinweise

- Ontology Development: A guide to Creating Your First Ontology

http://protege.stanford.edu/publications/ ontology_development/ ontology 101.html

Was ist RDFS ?

- RDFS = Resource Description Framework Schema
- Beschreibt genau, was wir brauchen: Die Konzepte und deren Bezeichner (Wortschatz, Vocabulary) für eine RDF Annotierung d.h. man erklärt mit RDFS-Ausdrücken, welche semantischen Beziehungen es zwischen den RDF-Tags gibt.
- Man kann damit Klassen und Mermale beschreiben
- RDFS entspricht dem RDF-Modell, d.h jeder RDFS-Ausdruck ist ein Tripel (Subjekt, Prädikat, Objekt)

Wie beschreibt man Klassen?

- Jede Klasse ist eine Ressource, die das Merkmal rdf:type und als Wert (Objekt) die Ressource rdfs:Class hat
- Z.B.

  Praktikum : Burg rdf:type rdfs:Class
- Entsprechend festgelegten Abkürzungsregeln kann man das "rdf:type" weglassen:

  <rdfs:Class rdf:ID=„Burg“/>

Wie beschreibt man Unterklassen ?

- Eine Unterklasse ist eine Ressource, die als Merkmal: rdf:subClassOf und als Objekt die Oberklasse hat
- Z.B.

  Praktikum: Kirchburg rdfs:subClassOf praktikum:Burg

  <rdfs:class rdf:ID=„Kirchburg“>
  <rdfs:subClassOf rdf:resource="#Burg“/>
  </rdfs:Kirchburg>

Wie instantiiert man Klassen

- Ein Individuum ist eine Ressource, die als Merkmal rdf:type hat und als Objekt eine definierte Klasse

z.B.

praktikum : Schässburg rdf:type rdfs:Burg

<praktikum:Burg rdf:ID=„ Schässburg“/>

Wie beschreibt man Klassenmerkmale

- Merkmale in RDF sind als Instanzen der Klasse rdf:Property beschrieben.
- zB.

Praktikum:AltStadt rdf:type rdfs:Class

Praktikum: deutscherName rdf:type rdf:Property

Praktikum:deutscherName rdfs:domain praktikum:AltStadt

<rdf:Property rdf:ID=„deutscherName“>

<rdfs:domain rdf:ressource =„#AltStadt“>

</rdf:Property>

Ontologiebasierte Anwendungen

- Information Retrieval and Extraction
- Enterprise Integration: Beim Zusammenschluss von weltweiten aber auch regionalen Firmen treten viele Probleme auf, die sich durch unterschiedliche Unternehmenstruktur, berschiedene Arbeitsbereiche oder einfach durch unterschiedliche Sprachen ergeben. Eine Ontologie könnte hier helfen, die Probleme zu durchschauen und schneller zu lösen.
- Wissensmanagement
- Multilinguale (Web-)Anwendungen

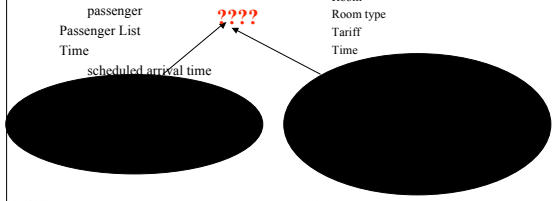
Merging 2 Ontologien

- Konzepte in einer Fluggeselchaft-Ontologie

Flight
Flight number
Price
Person
  passenger
Passenger List
Time
  scheduled arrival time

- Konzepte in einer Hotel-Ontologie

Amenieties
Occupancy
Person
  guest
Reservation
Room
Room type
Tariff
Time



Ontologien für Semantic Web

- To be useful for the Semantic Web, an ontology language must do more than just define a vocabulary and place constraints on the use of terms.It must be
  - Able to reference concepts defined elswhere on the Web
  - Sharable over the Web
  - Able to work with one or more languages in use
  - Able to emrge several ontologies
  - Widely accepted as a standard
  - Expressive enough for serious use
  - Able to support kinds of logical reasoning that are found to be needed to conduct the business of the Semantic Web

OWL ©W3C

OWL-1-

- Veröffentlichung & Austausch von Ontologien
- Beschreibungs-Sprache
- Austausch-Syntax RDF/XML
- Elements
  - Taxonomische Beziehungen zwischen Klassen
  - Eigenschaften und Datentypen
  - Objekteigenschaften (Eigenschaften von Individuen)
  - Instanzen von Klassen und Eigenschaften

OWL -2-

- Menge von OWL-Aussagen & Schlussfolgerungen = Wissensbasis (Knowledge base (KB))
- Typen von OWL:
  - Drei zunehmend komplexe Sprachen
    - OWL Lite
    - OWL DL
    - OWL Full

Header

- Header
  - Namensräume (rdf, rdfs, owl, dc, …)
  - Kommentare
  - Versionskontrolle
  - Einschließen anderer Ontologien

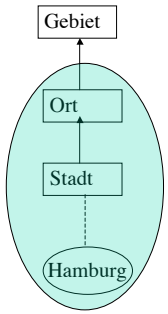
Basis-Definitionen -1-

- Bezeichnung mit rdf:ID
- Hierarchische Klassen
  - Class, subClassOf (rdfs)

  `<owl:Class rdf:ID="Gebiet"> …`
- Individuen (elemente)
  - Implizit Elemente von `owl:Thing`

    `<Gebiet rdf:ID=…`
  - Thing (Zuordnung zu einer Klasse mit `<owl:Thing rdf:about=….`)

Beispiel



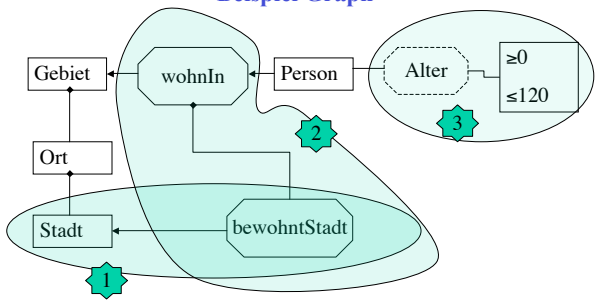
<owl:Class rdf:Id="Ort"/>

<owl:Class rdf:ID="Stadt">
  <rdfs:subclassOf rdf:resource="#Ort"/>
</owl:class>
<Stadt rdf:ID="Hamburg"/>

Basis - Definitionen

- Definieren von eigenschaften
  - Objekteigenschaften
    - ObjectProperty (hierarchisch)
      - Domain (Definitionsbereich)
      - Range (Wertebereich, Objekte)
    - subPropertyOf (rdfs)
      - Rdf:resource (Einzuschränknde Objekteigenschaft)
      - Range
  - Datentypeigenschaften
    - DataTypeProperty
      - Domain
      - Range(Wertebereich, Strings oder einfache XML-Datentypen)

Beispiel Graph



OWL-Beispiel-Code

```
<owl:ObjectProperty rdf="wohntIn">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Gebiet"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="bewohntStadt">
  <rdfs:subPropertyOf  rdf:resource="#wohntIn"/>
  <rdfs:range rdf:resource="Stadt"/>
</owl:ObjectProperty>

<owl:DataTypeProperty rdf:ID="Alter">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="&dt;Alter>
</owl:DataTypeProperty>
```

1 2 3

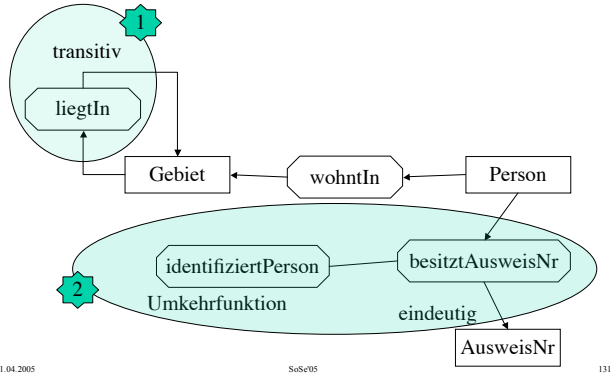
Basis Definitionen

- Eigenschafts-Merkmale
  <owl:TransitiveProperty rdf:ID=…
  - TransitiveProperty z.B. LiegtInn
  - SymmetricProperty z.B. NachbarVon
  - FunctionalProperty (eindeutige Abbildung) z.B. hatGeburtsjahr
  - inverseOf (Umkehrfunktion)

  <owl:inverseOf rdf:resource="Objekteigenschaft"/>….
  z.B. produzentVon und hatProduzent
  - InverseFunctionalProperty

  <owl:inverseFunctionalProperty rdf:ID=…">
  <owl:inverseOf rdf:resource="Objekteigenschaft">…

Beispiel-Graph



OWL-Beispiel-Code

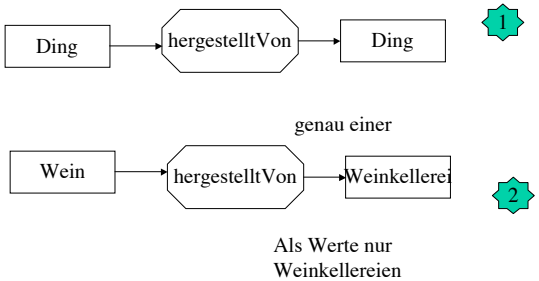
```
<owl:TransitiveProperty rdf:ID="liegtIn">
    <rdfs:domain  rdf:resource="#Gebiet"/>
    <rdfs:range  rdf:resource="#Gebiet"/>
</owl:TransitiveProperty>

<owl:FunctionalProperty  rdf:ID="besitztAusweisNr">
    <rdfs:domain  rdf:resource="#Person"/>
   <rdfs:range  rdf:resource="#AusweisNr"/>
<owl:FunctionalProperty>

<owl:inverseOf  rdf:ID="identifiziertPerson">
  <owl:inverseOf  rdf:resource="besitztAusweisNr"/>
</owl:inverseOf>
```

1 2

Basis -Definitionen

- Eigenschaftsbeschränkungen
  <rdfs:subClassOf>
  <owl:Restriction>
  <owl:onProperty rdf:resource=“ “/>…
  – allValuesFrom, someValuesFrom <…rdf:resource=“Eigenschaft”>…
    - Bisherige Mechanismen global
    - Hier Gültigkeit lokal für umschließende Klasse
    - Entsprechen All- und Existenzquantor
  – Kardinalitäten (Wert wird von Tags umschlossen)
    - minCardinality
    - maxCardinality
    - Cardinality
  – hasValue
    - Fixiert den Wert dieser Eigenschaft (Konstante)

Beispiel-Graph



OWL-Beispiel-code

1

```
<owl:ObjectProperty rdf:ID=“hergestelltVon”>
  <rdfs:domain rdf:resource=“&owl:Thing”/>
  <rdfs:range rdf:resource=“&owl;Thind”/>
</owl:ObjectProperty>
```

2

```
<owl:Class rdf:ID=2Wein”>
  <rdfs:subclassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource=“hergestelltVon”/>
      <owl:allValuesFrom rdf:resource=“Weinkellerei”/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

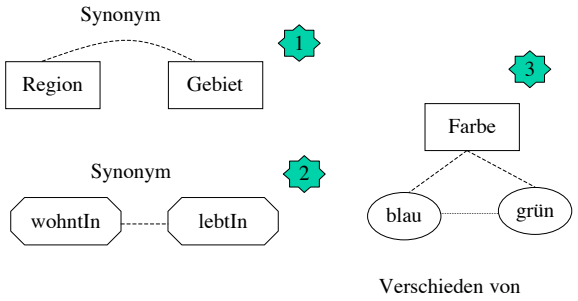

Abbildung von Ontologien

- Ontologien weit verbreitet übers internet
- Zusammenführung anspruchsvoll, aber notwendig
- Auszeichen gleicher Elemente
  – sameClassAs
    - Auszeichnung gleicher Klassen unterschiedlicher Ontologien
    - Klassifizierung von Individuen bestimmter Eigenschaften

```
<owl:Class rdf:ID=“Deutsche”>
  <owl:sameClassAs>
    <owl:restriction>
      <owl:OnProperty rdf:resource=“geborenIn”/>
      <allValuesFrom rdf:resource=“Deutschland”/>
```

    - samePropertyAs
    - sameIndividualAs (Synonyme)
    - differentIndividualFrom (Unterscheidung von Individuen (Werten), z.B. süß und sauer)

Beispiel - Grpah



OWL-Beispiel-Code

```
<owl:Class rdf:ID="Region">
    <owl:sameClassAs  rdf:resource="#Gebiet"/>
</owl:Class>

<owl:ObjectProperty  rdf:ID="lebtIn">
   <owl:samePropertyAs   rdf:resource="#wohntin"/>
</owl:ObjectProperty>

<Farbe  rdf:ID="gruen"/>
<Farbe  rdf:ID="blau"
    <owl:differentIndividualFrom   rdf:about=""#gruen"/>
</Farbe>
```

1 2 3

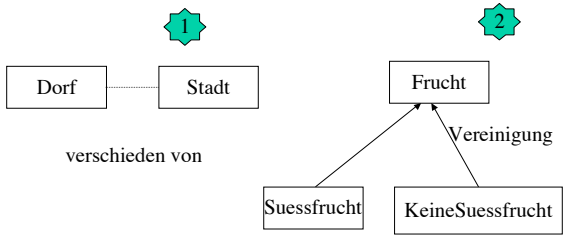
Komplexe Klassen

- Mengeoperatoren
  - Durchschnitt (intersectionOf)

```
<owl:Class rdf:ID="Weisswein">
    <owl:intersectionOf   rdf:parseType="Collection">
        <owl:Class  rdf:about="Wein"/>
        <owl:Restriction>
            <owl:onProperty   rdf: resource="#faerbung"/>
            <owl:hasValue  rdf:resource="#Weiss"/>…
```

  - Vereinigung (unionOf)
  - Komplement (complementOf)
  - Aufzählungsklasse (one of)
    alle Individuen der Klasse werden aufgezählt
  - Dijunkte Klassen (disjointWith)
  - Separierung einer Menge von klassen
  - Leere Schnittmenge entstehet

Beispiel-Graph



OWL-Beispiel-Code

```
<owl:Class rdf:ID="Dorf">
    <owl:disjointWith  rdf:resource="#Stadt"/>
</owl:Class>

<owl:Class  rdf:ID="Frucht">
   <owl:unionOf  rdf:parseType="Collection">
          <owl:Class rdf:about="#Suessfrucht"/>

  </owl:unionOf>
          <owl:Class rdf:about="#KeineSuessfrucht"/>
</owl:Class>
```

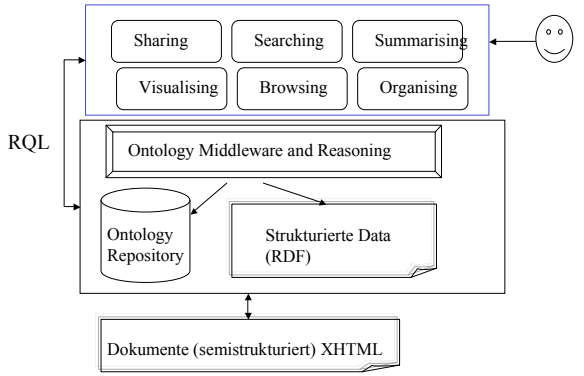
1

2

„Layer-cake" Architektur (nach Tim Berners-Lee)

<table>
<tr><td colspan="2">Trust</td></tr>
<tr><td colspan="2">Proof</td></tr>
<tr><td colspan="2">Logic</td></tr>
<tr><td rowspan="5">XML Familie</td><td>Ontologies & Ontology Interchange lang.</td></tr>
<tr><td>RDF Schemas</td></tr>
<tr><td>RDF, Topic Maps, ännliche Technologien</td></tr>
<tr><td>XML-Schemas, Namespaces</td></tr>
<tr><td>XML-Dokumente</td></tr>
<tr><td>URI</td><td>Unicode</td></tr>
</table>

Anwendungsarchitektur im Semantic Web



Inhaltsübersicht

- Was ist und was bringt Semantic Web ?
- Wie implementiert man das Semantic Web ?
- Was implementieren wir im Praktikum?
- Organisatorische Details

Funktionalität des Systems - Grobe Beschreibung

- Ein Demosystem zur Durchsuchung und zum Information-Retrieval von zweisprachigen Webseiten
- Die Webseiten werden durch eine Sammlung von deutschen und englischen Texten (Korpora) simuliert,.
- Die Eingaben des Systems sind in natürliche Sprache. Schlüsselwörter werden daraus extrahiert.
- Mit Hilfe von RDF-Annotationen im Text und eine Konzeptontologie werden relevante Paragraphen durchgesucht und als Antwort präsentiert.

Aufgaben mit den Texten

- Man muss die Texte erstmal analysieren und relevante Konzepte extrahieren.
- Dieses konzepte werden danach als tags für Paragraphe/Ausdrucke/Wörter benutzt werden.
- Beispiel:

<beschreibung>Das historische Zentrum von <stadt_burg>Sighisoara (Schässburg)<stadt_burg> wurde von der UNESCO auf die Liste des Weltkulturerbes gesetzt. Und das zurecht. In der Mitte steht wie einst die mächtige Burg , die besterhaltene Siebenbürgens, die .....

</beschreibung>

- Die Tags sollen auf deutsch für die deutsche Texte und auf englisch für die englische Texte

Aufgaben mit den Tags

- Für jedes Tag muss beschrieben werden:
  - Welches andere Tag kann es enthalten (= Properties in RDFS)
  - In welcher Beziehung steht es mit anderen Tags (= Class/Subclass in RDFS)
  - Welche ist (wenn vorhanden) die Korrespondenz in anderen Sprachen (= sameClassAs in RDF)

Inhaltsübersicht

- Was ist und was bringt Semantic Web ?
- Wie implementiert man das Semantic Web ?
- Was implementieren wir im Praktikum?
- Organisatorische Details 

Gruppen - Erste Phase

- XML und XSLT
- RDF
- RDFS
- Ontologieerstellung (inklusiv Protege)
- OWL
- RQL (inklusiv JENA)

Termine

- 11.04 Einführung
- 18.04, 25.04, 02.05 Erstellung des Korpus (Einführungsmodulen für die Gegebene Themen)
- 02.05 System Architektur
- 09.05 und 23.05 Extraktion von Konzepten, Ontologie Erstellung
- 30.05,06.06, 13.06 Ontologie implementierung, Data annotationen
- 20.06,27.06,04.07, RQL Anfragen
- 27.06 Evaluationskriterien
- 11.07 End-Präsentation

Scheinkriterien

- Individueller Nachweis von
  - Erstellung einer Teil des Korpus im HTML.
  - Programmieranteil (obligatorisch)
  - Präsentation
  - Teil im Endbericht
  - Teil der Evaluation (nach Kriterien)

Web-Seite des Praktikums

- http://nats-www.informatik.uni-hamburg.de/view/MSW05/
- Wenn sie selbst Materialien und Zwischenergebnisse einfügen, melden Sie sich bitte an unter
- http://nats-www.informatik.uni-hamburg.de/view/TWiki/TWikiRegistration
  als "Group" bitte auswählen "MSW05Group"