

The Semantic Web

- Motivation
 - Web Datenbanken
 - Das Web heute
- Die Vision des Semantic Web
- Web Mining
- Das 7-Schichten Modell
 - URIs
 - XML und XML Schema
 - RDF und RDF Schema
 - Ontologien und OWL

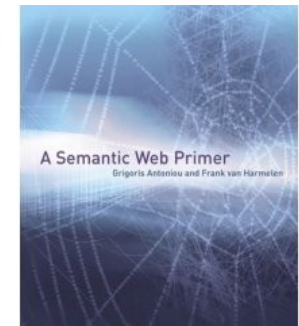
Literatur

■ Artikel

- Tim Berners-Lee, James Hendler and Ora Lassila, The Semantic Web, *Scientific American*, May 2001. (die **Semantic Web Vision**)
- Soumen Chakrabarti, Data Mining for Hypertext: A Tutorial Survey. ACM SIGKDD explorations 1(2):1-11, January 2000.
<http://www.acm.org/sigkdd/explorations/issue1-2/chakrabarti.pdf>
- Johannes Fürnkranz, Web Mining. In O. Maimon and L. Rokach (eds.), *Data Mining and Knowledge Discovery Handbook*, Springer-Verlag, 2005.
<http://www.ke.informatik.tu-darmstadt.de/~juffi/publications/web-mining-chapter.pdf>
- Details zu XML, RDF, OWL finden sich auf den entsprechenden Dokumenten des WWW Consortiums <http://www.w3c.org/>

■ Bücher

- Grigoris Antoniou and Frank van Harmelen:
A Semantic Web Primer. The MIT Press, 2004.
(**sehr lesbare Einführung**)



■ Slides

- Einige der folgenden Slides basieren auf Slides von Nicholas Kushmerick und Ian Horrocks

Web-Datenbanken

- Datenbanken-Anwendungen sind häufig und maßgeschneidert für einen bestimmten Kunden
 - schwer von einer Anwendung auf eine neue übertragbar
 - Datenaustausch zwischen verschiedenen Anwendungen ist nicht trivial
 - sogar das Zusammenführen von verschiedenen Datenbanken innerhalb eines Unternehmens kann eine Herausforderung sein (*Data Warehousing*)
- Informations-Systeme müssen jedoch vermehrt offen gehalten werden
 - nicht nur ein HTML-Abfrage-Interface
 - sondern auch direkter Austausch zwischen Informations-Systemen

Semantische Meta-Daten

- Menschlichen Benutzern wird die Semantik der Daten durch die Präsentation vermittelt
 - natürlich-sprachlicher Text
 - visuelle Hervorhebung wichtiger Komponenten
 - tabellarische Aufbereitung komplexer Daten
- Das ist aber für einen direkten Informationsaustausch zwischen Informations-Systemen ungeeignet
 - Web Mining
- Aufgabe solcher **Meta-Daten** ist es,
 - die Bedeutung (Semantik) einer Datenquelle maschinenlesbar zu beschreiben
 - und dadurch den Austausch zwischen Datenquellen zu erleichtern

Das Web heute

- Das Web ist heute (erst) 20 Jahre alt
 - ca. 1990 entwickelte Tim Berners-Lee (CERN, Schweiz) den ersten grafischen Hypertext Browser
- Seither ist die am Web verfügbare Information exponentiell gewachsen
 - praktisch zu jedem Thema findet sich relevante Information am Web
- Es ist aber immer noch schwer, relevante Information zu finden
 - Das Query-Interface zu Anfragen ans Web hat sich seit den Anfangstagen des Webs kaum verändert
 - Eingabe einiger weniger Stichworte
 - Rückgabe aller Dokumente, die diese Stichworte enthalten
 - in einer Reihenfolge, die die relevanten Dokumente zuerst präsentiert
 - Benutzer haben sich daran angepaßt (statt umgekehrt)

Die häufigsten Probleme

- Hoher Recall, aber niedrige Precision
 - Für viele Queries werden viel zu viele unnötige Dokumente werden retourniert
 - Die Information, die man sucht ist oft auf einer Seite zu finden
- Zu niedriger Recall
 - Manchmal erhält man auch keine befriedigende Antwort
- Sensitivität zum Query-Vokabular
 - Wenn man die Anfragen nicht mit den Worten stellt, die im gesuchten Dokument vorkommen, kann es nicht gefunden werden
 - gewünscht wäre eine Abfrage von semantischen Konzepten
- Resultat sind ganze Web-Seiten
 - die gesuchte Information findet sich irgendwo auf der Seite
 - muß üblicherweise manuell extrahiert werden

Recall: Prozentsatz der gefundenen Dokumenten unter allen relevanten Dokumenten

Precision: Prozentsatz der relevanten Dokumente in den gefundenen Dokumenten

Das grundlegende Problem

- Das Web von heute ist nicht maschinen-lesbar
 - d.h. die Daten können nicht automatisch weiter-verarbeitet werden.
- Maschinen verstehen keine natürliche Sprache
 - können daher nur sehr eingeschränkt auf Information auf natürlich-sprachlichen Web-Seiten zugreifen
 - z.B. Sätze wie
 - Ich bin Informatik-Student
 - Man könnten meinen, ich bin Informatik-Student.
 - sind für Computer sehr schwer zu unterscheiden
- Maschinen haben keine “Augen”
 - daher bleiben wesentliche Teile des Webs für sie unsichtbar (alle Bilder, Töne, ...)
 - viele Web-Seiten bestehen nur aus Bildern, in die der Text als Bitmaps eingearbeitet ist!

Die Vision eines Semantischen Webs

- Das heutige Web so zu erweitern, daß Anfragen mit einer komplexen Semantik, deren Beantwortung z.B. die Integration mehrere Wissensquellen erfordert, gestellt (und beantwortet!) werden können

Zwei grundlegende Ansätze zur Realisierung

- Man beläßt das Web wie es ist
 - aber verbessert die Zugriffsmethoden
 - z.B. durch vermehrten Einsatz von Methoden der künstlichen Intelligenz und des Information Retrievals→ Stichwort: **Web Mining**
- Man macht das Web maschinen-lesbar
 - das bedingt eine neue Repräsentation von Web-Dokumenten
 - und intelligente Methoden, die auf dieser Repräsentation aufsetzen, und die Informationen verarbeiten→ Stichwort: **The Semantic Web**

Beispiel

- Query:
 - *“Welche anderen Vorlesungen halten die Dozenten der Kanonik Data und Knowledge Engineering im nächsten Winter-Semester an der TU Darmstadt?”*
- Alle Informationen, die man benötigt um diese Frage zu beantworten, findet man im Web!
- Man müßte z.B.
 - Die Homepage der Vorlesung finden
 - Darauf die Namen der Dozenten finden
 - Die Homepages dieser Dozenten finden
 - Dort die für die Lehre relevanten Seiten finden
 - und daraus die Liste aller Lehrveranstaltungen extrahieren
 - feststellen, daß die Lehrveranstaltungen für nächstes Semester noch nicht dort sind
 - den Schluß ziehen, daß die im vorigen WS angebotenen Vorlesungen auch im nächsten WS angeboten werden könnten

Anwendungsszenario: Auto-Kauf im Semantic Web

- Sie beauftragen Ihren persönlichen Shopping-Agenten damit, für Sie ein geeignetes Auto ausfindig zu machen
 - Ihr Shopping Agent kennt Ihre Vorlieben, und trifft eine Vorauswahl der in Frage kommenden Modelle
- er kontaktiert die Web-Seiten verschiedener Anbieter
 - wählt Anbieter in der Nachbarschaft aus
 - informiert sich über Preise, Produkt-Details, Lieferbedingungen, führt eventuell Verhandlungen durch
 - informiert sich bei unabhängigen Quellen über die Reputation der betreffenden Anbieter
- beim Auffinden geeigneter Angebote
 - informiert sich Ihr Shopping Agent bei Ihrem Terminkalender über freie Termine
 - vereinbart einen Termin für eine Probefahrt mit dem Agenten des Anbieters
 - informiert Sie auf Ihrem PDA

Web Mining

Web Mining ist Data Mining im Internet

- im Gegensatz zum Data Mining geht Web Mining nicht davon aus, daß das Datenmaterial in der Form von strukturierten Datenbanken vorliegt
 - sondern meistens als unstrukturierte oder (semi-)strukturierte Text-Dokumente

Drei Richtungen

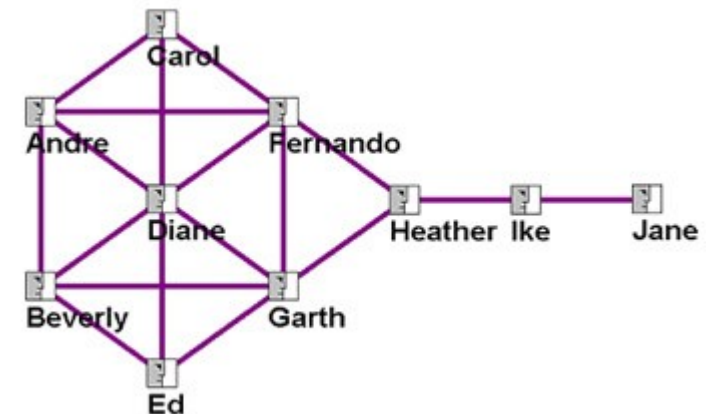
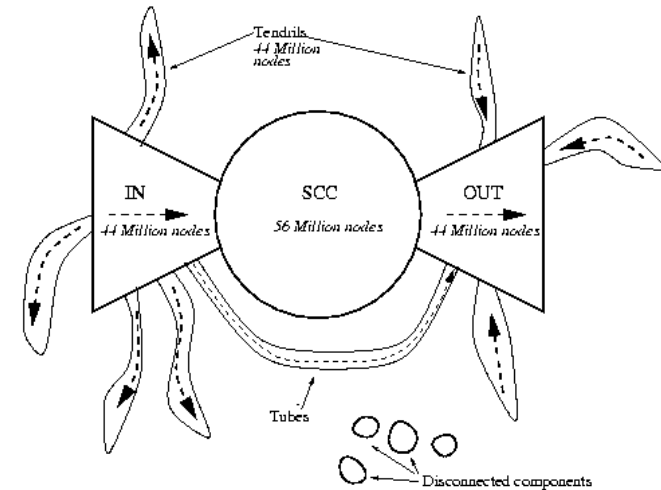
- **Web Content Mining**
 - Analyse des Inhalts von Web Seiten
- **Web Structure Mining**
 - Analyse der Vernetzungsstruktur
- **Web Usage Mining**
 - Analyse von Daten, die bei der Verwendung des Webs anfallen

Web Content Mining

- Analyse des Inhalts von Text-Dokumenten
 - z.B. (semi-)automatische Erstellung von Web-Katalogen durch induktives Lernen von Zuordnungen Dokument → Kategorie
 - basiert auf klassische Methoden zur Text-Analyse
 - **Information Retrieval**
 - Auffinden von Seiten, die für ein bestimmtes Thema relevant sind (Suchmaschinen)
 - z.B. finde Seiten, die Kritiken von Filmen enthalten
 - **Information Extraction**
 - Extraktion relevanter Information auf diesen Seiten
 - z.B. identifiziere den Film-Titel auf diesen Seiten
 - **Information Integration**
 - Zusammenführen von Informationen aus verschiedenen Quellen
 - z.B. führe die Kritiken über die Film-Titel mit Daten über aktuelle Filme in Darmstadt zusammen
- ```
review(Title, Review), show(Title, darmstadt)
```

# Web Structure Mining

- Analyse der Vernetzungsstruktur des Webs
  - das Web ist ein großer Graph
- Die Struktur des Web-Graphen ist wertvolle Information
  - Google's PageRank war ein Durchbruch bei Suchmaschinen
  - Web-Seiten lassen sich oft durch die Information auf benachbarten Seiten besser klassifizieren als durch Information, die sich auf der Seite selbst befindet
  - Objekte (Web-Seiten, Personen, ...), die stark miteinander vernetzt sind, können als Gruppen identifiziert werden, bzw. wichtige Elemente in diesen Gruppen erkannt werden (*Social Network Analysis*)

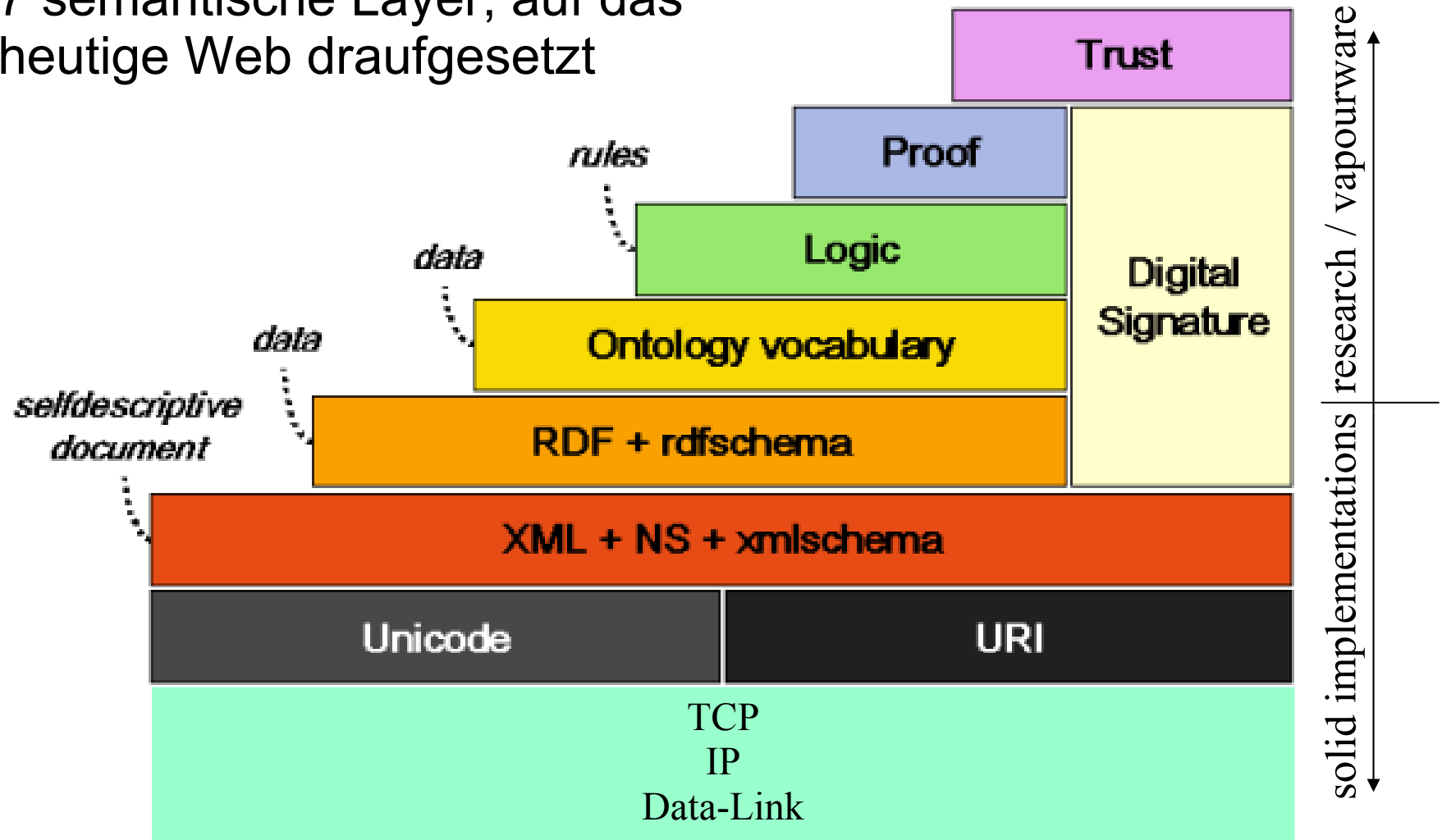


# Web Usage Mining

- Analyse von Daten, die in der Verwendung des Webs anfallen
  - z.B. Web-logs, Benutzerkonten, etc.
- **Clickstream Analysis:**
  - Auffinden auffälliger Muster in den log-Files einer Web-site
  - z.B. um festzustellen, daß Benutzer gewisse Pfade häufig gehen, und möglicherweise eine Abkürzung angebracht wäre
- **Collaborative Filtering:**
  - Vorhersage von Benutzerverhalten aufgrund des Verhaltens “ähnlicher” Benutzer
  - Ähnlichkeit der Benutzer kann dabei wiederum über ihr Verhalten definiert werden
  - z.B. Recommender Systems für Kaufempfehlungen

# Die Sieben Schichten des Semantischen Webs

- Tim Berners-Lee's Vision:
  - 7 semantische Layer, auf das heutige Web draufgesetzt



# Schicht 1a: Unicode

- Unicode sichert eine weltweit einheitliche Repräsentation von Dokumenten
  - 16 Bits, also  $2^{16} = 65\,536$  möglichen Zeichen.
  - Tatsächlich festgelegt in Unicode ist nur die Bedeutung von knapp 40 000 16–Bit–Werten.
  - ist also noch genügend Platz zum Ausbau
- Inhalte:
  - Schriftzeichen aus allen möglichen Sprachen und Schriftsätzen (zyrillisch, chinesisch, arabisch, ...)
  - mathematische Symbole
  - Sonderzeichen
  - einfache technische Piktogramme
  - einfache geometrische Formen
  - etc.
- Unicode ist Basis-Zeichensatz moderner Programmiersprachen (wie Java)



# Schicht 1b: URI

URI = Uniform Resource Identifier

- Jedes Objekt ist eine “Ressource”
  - Personen, Bücher, Begriffshierarchien, Konzepte, ...
- Jede dieser Ressourcen hat einen einheitlichen Identifier, der es erlaubt, die Ressource im Web zu finden
  - URLs sind die URIs für Web-Seiten
  - URIs sind aber ein allgemeineres Konzept!
- URIs werden von den Inhabern der Objekte vergeben
  - Identifier sind an sich beliebig wählbar
    - es muß nur gewährleistet sein, daß die URIs eindeutig sind.
  - die meisten URIs haben aber eine URL-ähnliche Syntax
    - insbesondere wenn wir annehmen, daß jeder URI für eine Web-Resource steht
  - einige Fragen offen
    - z.B. wie soll man URIs für Personen definieren?

# Schicht 2: XML und XML Schema

- SGML (Standard Generalized Markup Language)
  - ISO Standard zur system-unabhängigen Repräsentation von Information
  - Hauptbestandteile sind Tags der Form  
`<tag>...</tag>`
- HTML (Hyper-Text Markup Language)
  - SGML-Anwendung zur Repräsentation von Hyper-Texten im Internet
  - Hauptzweck ist die Darstellung von Texten
  - Menge der Tags ist daher fixiert
- XML (eXtensible Markup Language)
  - SGML-Anwendung zur Repräsentation von allgemeinen Inhalten
  - XML trennt Inhalt von der Präsentation (z.B. Formatierung)
  - Menge der Tags ist beliebig erweiterbar

# Beispiel

- HTML-Dokument:

```
<h2>A Semantic Web Primer</h2>
<i>by G. Antoniou and F. van Harmelen</i>

The MIT Press, 2004

```

- Mögliche XML-Repräsentation:

```
<book>
 <title>A Semantic Web Primer</title>
 <author>
 <firstname>Grigoris</firstname>
 <lastname>Antoniou</lastname>
 </author>
 <author>
 <firstname>Frank</firstname>
 <lastname>van Hamelen</lastname>
 </author>
 <publisher>The MIT Press</publisher>
 <year>2004</year>
</book>
```

# Überblick über die Syntax von XML

- **Tags (Elements)**
  - treten immer paarweise auf `<tag>...</tag>`
  - können aber leer sein
    - Kurzschreibweise: `<tag/>`
- **Attribute (Properties)**
  - Eigenschaften, die in einem Tag angegeben werden können  
`<tag attribute="value">...</tag>`
- **Processing Instructions (PIs)**
  - beginnen und enden mit einem Fragezeichen
  - z.B. Prolog eines XML Dokuments:  
`<?xml version="1.0" encoding="UTF-16"?>`
  - z.B. Stylesheets  
`<?stylesheet type="text/css" href="x.css"?>`

# Properties und Nested Tags

- Es ist oft nicht klar, was man als Property, und was man verschachtelte Elemente repräsentieren möchte
- Hauptunterschied:
  - Bei Properties ist die Reihenfolge egal
  - bei Elementen ist die Reihenfolge wichtig (Elemente können daher auch mehrfach genannt werden)
- Beispiel:
  - die folgenden beiden Repräsentationen für Autoren-Objekte wären beiden gültig
  - sind aber NICHT äquivalent!

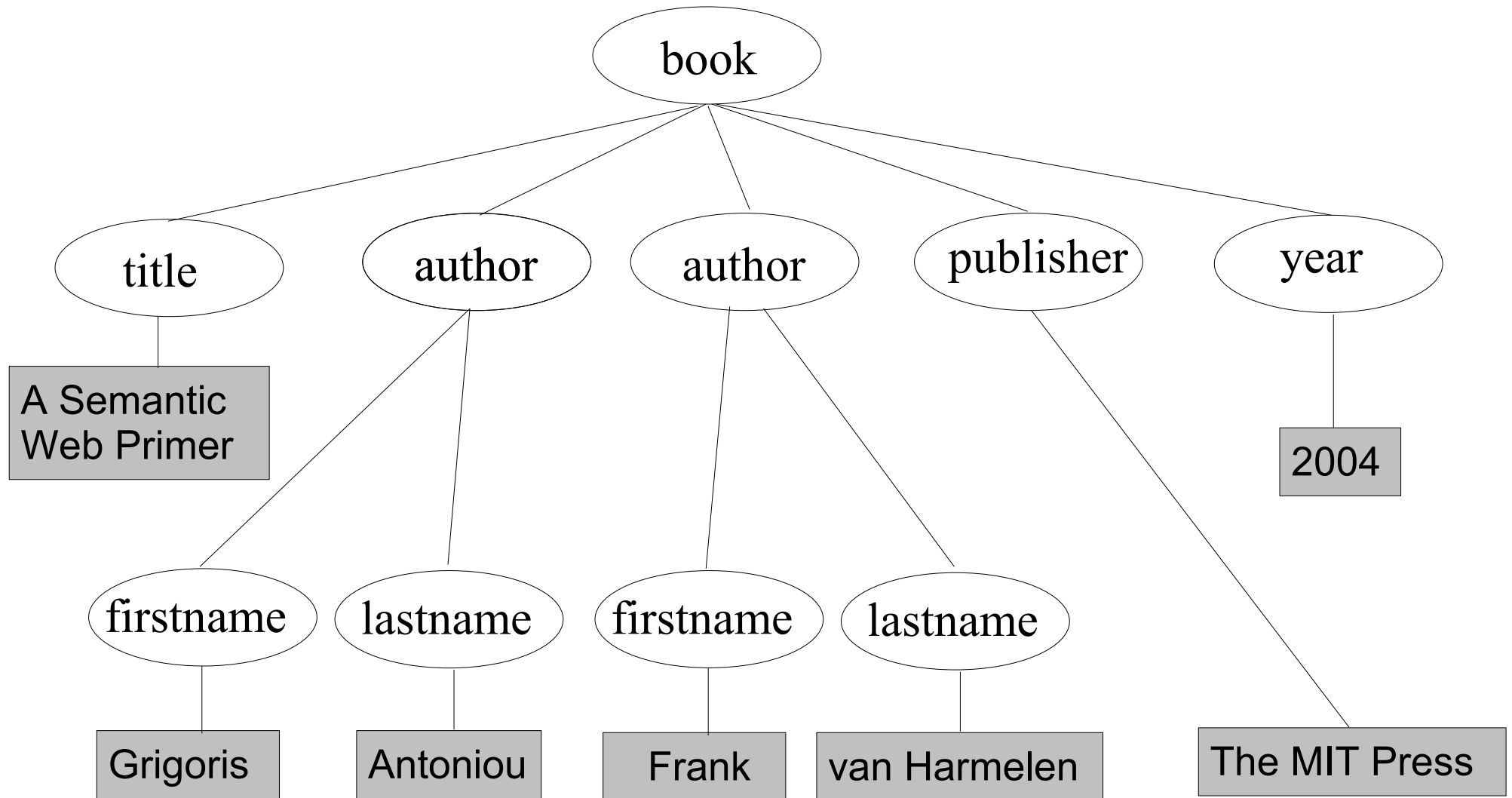
```
<author>
 <firstname>Grigoris</firstname>
 <lastname>Antoniou</lastname>
</author>
```

```
<author firstname="Grigoris" lastname="Antoniou"/>
```

# Baum-Modell von XML

- Tags können ineinander verschachtelt werden
    - öffnende und schließende Tags verschiedener Elemente dürfen sich aber nicht überlappen
  - alle Elemente müssen direkt oder indirekt in einem einzigen Element (dem sogenannten *Root-Element*) eingebettet werden
    - in HTML ist das Root-Element `<HTML> . . . </HTML>`
  - Daher kann ein XML-Dokument als eine Baumstruktur angesehen werden
    - Jedes Element ist ein Knoten
    - die unmittelbar in einem Element X enthaltenen Elemente sind die Nachfolgerknoten von X
    - An den Blättern des Baumes stehen die Informationen in den Tags
- **DOM-Tree** (Document Object Model)

# Beispiel



# XPath

- Sprache zur eindeutigen Identifikation von Objekten in einem DOM Tree
- Bezeichnet einen Knoten durch die Abfolge der Tags
- Beispiele:
  - `/book/title`
    - extrahiert den Titel unter dem Knoten `book`
  - `//title`
    - finde alle Titel
  - `/book/author[1]`
    - finde den ersten Autor
  - `/book/author/@lastname`
    - Zugriff auf die `lastname` Property eines Autors
  - `//author/@lastname="Antoniou"`
    - findet alle `lastname` Knoten mit dem Wert `Antoniou`
  - `//author[@lastname="Antoniou"]`
    - findet alle Autoren, deren `lastname` `Antoniou` ist



# XML zum Datenaustausch

- XML hat sich vom ursprünglichen Zweck, eine verbesserte Markup-Sprache für Dokumente zu bieten, zu einem generellen Datenaustausch-Format entwickelt
  - viele Applikationen können Ihre Daten in XML exportieren und aus XML importieren
- viele Applikationen speichern ihre Daten in XML ab
  - z.B. OpenOffice Suite
- alle Daten und Werkzeuge des semantischen Webs sind in XML kodiert
  - z.B. XML Schema, RDF, RDF Schema, OWL, etc.
- Grundproblem beim Datenaustausch:
  - die beiden Partner müssen sich auf eine gemeinsame Struktur von XML-Dokumenten festlegen
  - d.h. dieses muß ebenfalls maschinen-lesbar definiert werden

# XML Schema

- Eine Sprache zur Beschreibung der Struktur von XML-Dokumenten
  - ein anderer Formalismus ist DTD
  - XML Schema ist aber vielfältiger
- Eigenschaften
  - erhöht die Lesbarkeit von Dokumenten
  - unterstützt die Wiederverwendbarkeit von Strukturen
  - stellt eine große Bibliothek von wiederverwendbaren Datentypen zur Verfügung

# Überblick über XML Schema Syntax

- Syntax basiert auf XML
- erlaubt die Definition von
  - Elementen
  - Attributen
  - Datentypen
- Header:

```
<xsd:schema
 xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
 version="1.0">
```

- `xsd` ist der Name-Space
  - kann bei einfachen Strukturen auch weggelassen werden
- Hier wird das XML-Schema importiert, das auf der W3C Web-Seite angegeben wird.

# Überblick über XML Schema Syntax

- `<element name="..." />`
  - spezifiziert den Namen eines Elementes, das im XML-Dokument auftreten darf
  - Weitere mögliche Attribute:
    - `type`, `minOccurs`, `maxOccurs`
- `<attribute name="..." />`
  - spezifiziert den Namen eines Attributs, das im XML-Dokument auftreten darf
  - Weitere mögliche Attribute:
    - `type`: Datentyp
    - `use`: optional oder required
    - `value`: Angabe eines Default-Werts
- **Beispiel:**

```
<element name="email" />
<element name="head" minOccurs="1" maxOccurs="1" />
<element name="to" minOccurs="1" />
```

# Datentypen in XML Schema

- Vordefinierte Datentypen:
  - numerische Typen
    - `integer`, `Short`, `Byte`, `Long`, `Float`, ...
  - String Daten Typen
    - `string`, `ID`, `IDREF`, `CDATA`, `Language`, ...
  - Zeit und Datum
    - `time`, `Date`, `Month`, `Year`, ...
  - etc.
- Benutzer-definierte Datentypen:
  - Einfache Datentypen:
    - können keine Elemente und Attribute verwenden
  - Komplexe Datentypen:
    - werden aus verschiedenen Elementen (und Attributen) zusammengebaut
    - ähnlich zusammengesetzten Typen in höheren Programmiersprachen

# Komplexe Datentypen

- werden zusammengesetzt aus existierenden Datentypen durch Angabe von
  - `all`: Alle müssen auftreten, aber die Ordnung ist nicht wichtig
  - `sequence`: Ordnung ist wichtig
  - `choice`: eine der folgenden Angaben muß auftreten
- **Beispiel:**

```
<complexType name="lectureType">
 <sequence>
 <element name="firstname" type="string"
 minOccurs="0" maxOccurs="unbounded"/>
 <element name="lastname" type="string"/>
 </sequence>
 <attribute name="title" type="string" use="optional"/>
</complexType>
```

# Extension existierender Datentypen

- Man kann auch existierende Datentypen ausbauen
- Beispiel:

```
<complexType name="lectureTypeWithEmail">
 <extension base="lectureType">
 <sequence>
 <element name="email" type="string"
 minOccurs="0" maxOccurs="1"/>
 </sequence>
 </extension>
</complexType>
```

- Es ist auch möglich, existierende Datentypen zu erweitern, indem man den Wertebereich einschränkt

```
<complexType name="lectureTypeWithProf">
 <restriction base="lectureType">
 <attribute name="title" type="string" use="required"/>>
 </restriction>
</complexType>
```

# Namespaces

- Ein XML-Dokument kann mehrere XML-Schemas implementieren
- Damit es beim Zusammenführen mehrerer Schemas zu keinen Problem kommt, gibt es Namespaces
  - erlauben eine Disambiguation der Quelle
- Syntax:
  - Innerhalb des XML-Elements können mehrere XML-Schemas importiert werden:

```
<book xmlns:ns1="http://www.ns1.com/mySchema"
 xmlns:ns2="http://www.xyz.org/ourXMLSchema">
```

- Die Elemente jedes Schemas können dann in der Folge mit dem entsprechenden Präfix verwendet werden

```
<ns1:title>Ein Titel nach Schema n1</ns1:title>
<ns2:title>Ein Titel nach Schema n2</ns2:title>
```



# Beispiel

## XML-Dokument

```
<purchaseOrder orderDate="1999-10-20">
 <shipTo country="US">
 <name>Alice Smith</name>
 <street>123 Maple Street</street>
 <city>Mill Valley</city>
 <state>CA</state>
 <zip>90952</zip>
 </shipTo>
 <billTo country="US">
 <name>Robert Smith</name>
 <street>8 Oak Avenue</street>
 <city>Old Town</city>
 <state>PA</state>
 <zip>95819</zip>
 </billTo>
 <comment>Hurry, my lawn is going wild!</comment>
 <items>
 <item partNum="872-AA">
 <productName>Lawnmower</productName>
 <quantity>1</quantity>
 <USPrice>148.95</USPrice>
 <comment>Confirm this is electric</comment>
 </item>
 <item partNum="926-AA">
 <productName>Baby Monitor</productName>
 <quantity>1</quantity>
 <USPrice>39.98</USPrice>
 <shipDate>1999-05-21</shipDate>
 </item>
 </items>
</purchaseOrder>
```

## zugehöriges XML Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
 <xsd:documentation xml:lang="en">
 Purchase order schema for Example.com.
 Copyright 2000 Example.com. All rights reserved.
 </xsd:documentation>
 </xsd:annotation>
 <xsd:element name="purchaseOrder" type="PurchaseOrder Type"/>
 <xsd:element name="comment" type="xsd:string"/>
 <xsd:complexType name="PurchaseOrder Type">
 <xsd:sequence>
 <xsd:element name="shipTo" type="USAddress"/>
 <xsd:element name="billTo" type="USAddress"/>
 <xsd:element ref="comment" minOccurs="0"/>
 <xsd:element name="items" type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>
 <xsd:complexType name="USAddress">
 <xsd:sequence>
 <xsd:element name="name" type="xsd:string"/>
 <xsd:element name="street" type="xsd:string"/>
 <xsd:element name="city" type="xsd:string"/>
 <xsd:element name="state" type="xsd:string"/>
 <xsd:element name="zip" type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN"
 fixed="US"/>
 </xsd:complexType>
 <xsd:complexType name="Items">
 <xsd:sequence>
```

# Präsentation vom XML-Dokumenten

- CSS2 (Cascading Style Sheets 2)
- XSL (eXtensible Stylesheet Language)
  - Spezifikation für die Formatierung eines XML-Dokuments
- XSLT (XSL Transformations)
  - Spezifikation für Transformation von XML-Dokumenten
  - z.B. kann ein Style-sheet realisiert werden, indem man eine Transformation von XML in HTML spezifiziert

# XML Zusammenfassung

- XML ist eine Sprache zur Kennzeichnung von Inhalten von Dokumenten
- Die Struktur der Sprache kann durch XML Schemas festgelegt werden
- Strenge Trennung von Struktur und Präsentation
- Ist sowohl maschinen-lesbar als auch (mit Mühe) von Menschen interpretierbar
  - viele Tools unterstützen Navigation in XML-Dokumenten
- Unterstützung für Datenaustausch zwischen verschiedenen Anwendungen
- Einfache Queries durch XPath

# Schicht 3: RDF und RDF Schema

- RDF (Resource Description Framework)
  - einfache Sprache, um Fakten zu spezifizieren
  - jedes Faktum ist ein Tripel `Obj | Attr | Wert`
  - Semantik:
    - Objekt `Obj` hat für Attribut `Attr` den Wert `Wert`
- Jegliche Information wird im Semantic Web als RDF Statements wiedergegeben
  - J. Fürnkranz | has-email | `juffi@ke.informatik.tu-darmstadt.de`
  - BuchSW | has-title | A Semantic Web Primer
  - etc.
- Terminologie in RDF:
  - Objekte sind Ressourcen
    - werden auch durch URIs identifiziert
  - Attribute sind Properties
    - können ebenfalls durch URIs identifiziert werden

# Notationen von RDF

RDF kann auf verschiedene Arten und Weisen repräsentiert werden

- graphisch



- logisch

```
site_owner("http://www.cit.gu.edu.au/~db",
 "David Billington")
```

- und natürlich in XML
  - siehe Beispiel auf nächster Folie
  - werden wir aber nicht weiter im Detail behandeln

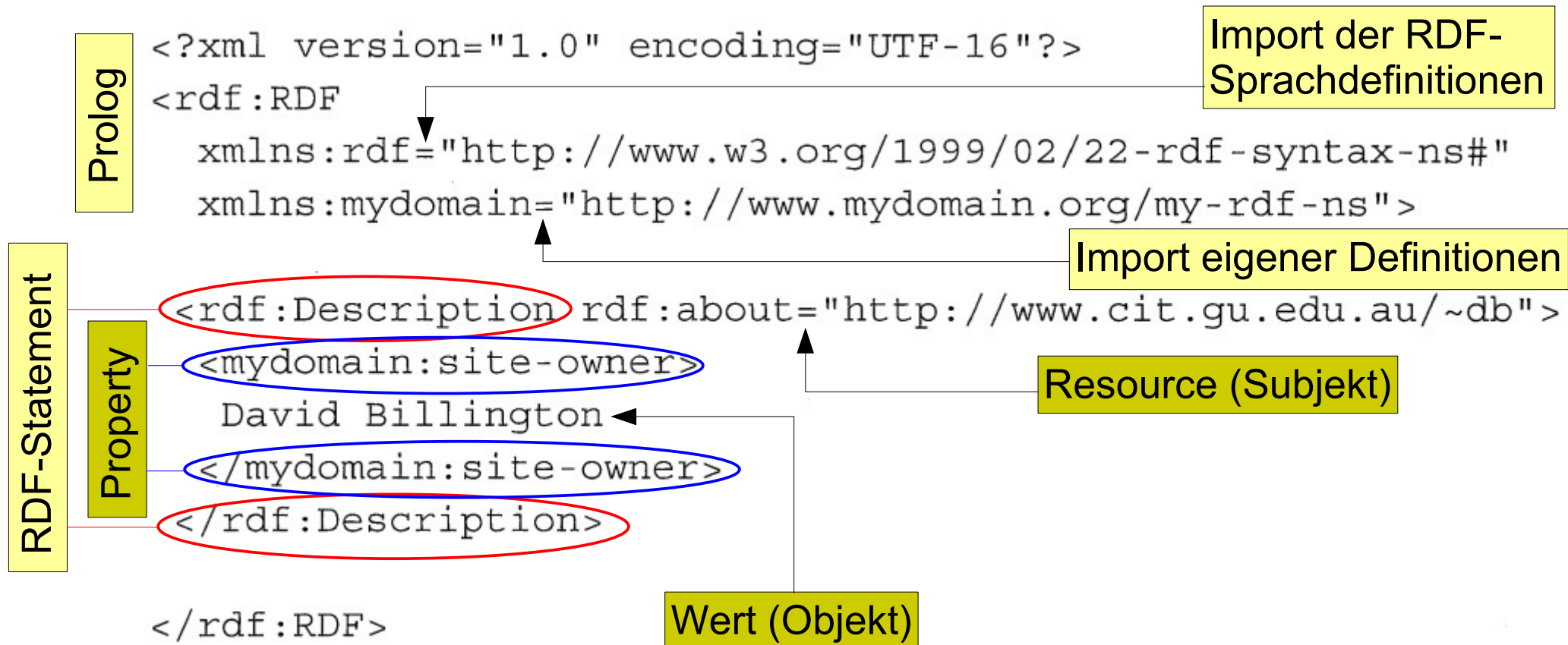
# Beispiel RDF XML-Syntax

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">

 <rdf:Description rdf:about="http://www.cit.gu.edu.au/~db">
 <mydomain:site-owner>
 David Billington
 </mydomain:site-owner>
 </rdf:Description>

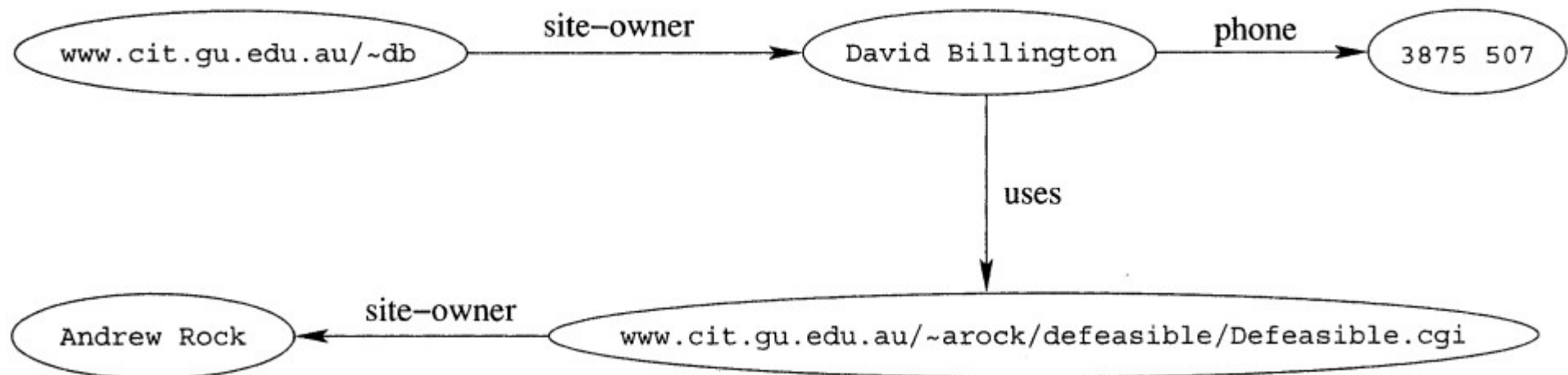
</rdf:RDF>
```

# Beispiel RDF XML-Syntax



# Semantische Netzwerke

- Werden mehrere Statements über dieselben Objekte getroffen, so spricht man auch von einem **Semantischen Netzwerk**
- Beispiel:





# Mehrstellige Prädikate

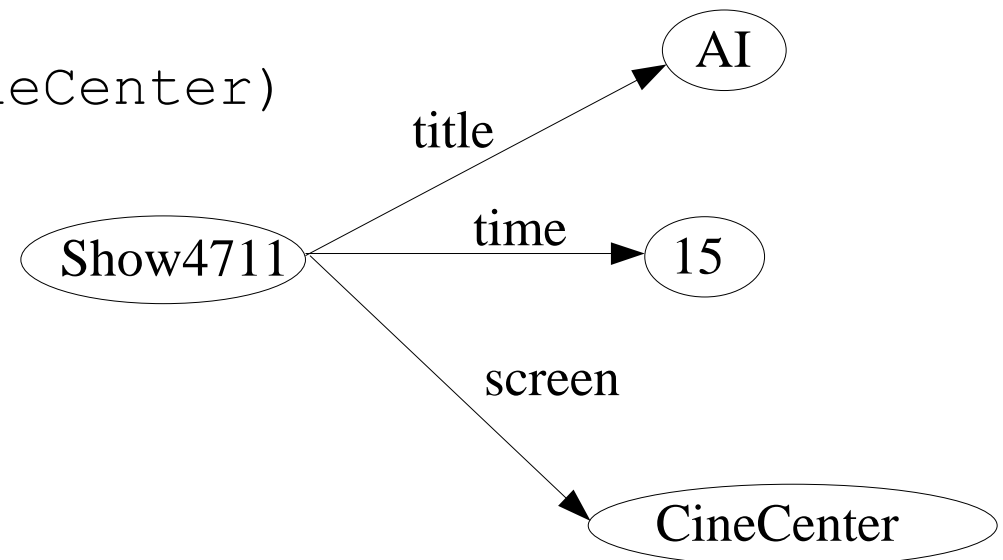
- An sich kann RDF nur zweistellige Relationen repräsentieren
  - Allgemeine Form: `property(resource, value)`



- Man kann aber n-stellige Relationen auf zweistellige Relationen zurückführen
  - indem man eine Konstante R einführt, die die mehrstellige Relation repräsentiert
  - und für jedes Attribut A der mehrstelligen Relation eine binäre Aussage trifft, die in etwa "R hat Attribut A" bedeutet
- Das ist zwar durchführbar, aber nicht sehr elegant und lesbar (Nachteil von RDF).

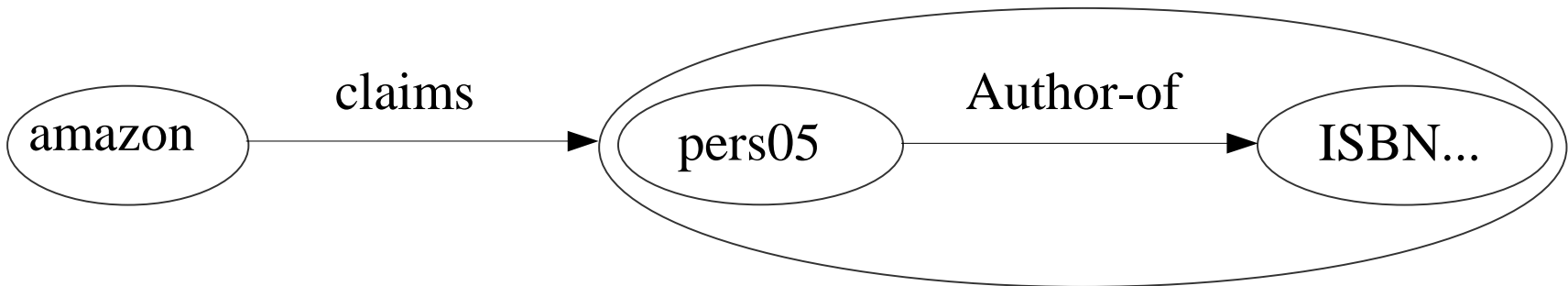
# Beispiel

- Der Film “AI” wird um 15h im CineCenter gezeigt
  - Natürliche Repräsentation `shows(ai, 15, cineCenter)`
  - Repräsentation mit binären Relationen
    - Konstante `show4711` repräsentiert die Relation
      - eine Konstante für jedes Tupel
    - `title(show4711, ai)`
    - `time(show4711, 15)`
    - `screen(show4711, cineCenter)`



# Reifikation

- Statements können ebenfalls Ressourcen sein
- Das heißt, man kann Statements über Statements machen
  - das heißt dann **Reifikation**



```
<rdf:Description rdf:about="#amazon">
 <claims>
 <rdf:Description rdf:about="#pers05">
 <authorOf>ISBN...</authorOf>
 </rdf:Description>
 </claims>
</rdf:Description>
```

# RDF Schema

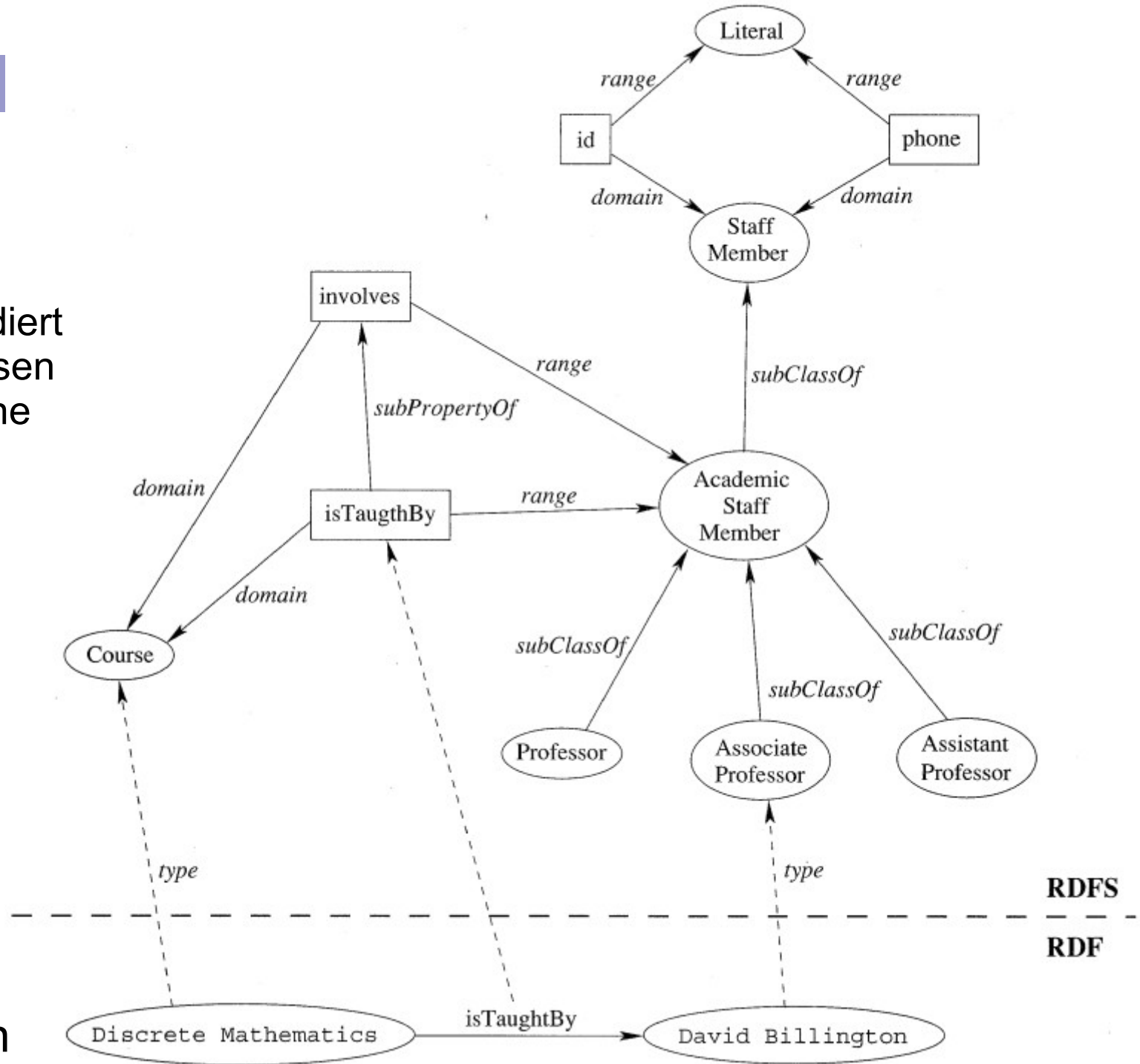
- Definiert ein einfaches Meta-Vokabular, das man braucht, um die Semantik von Domänen zu beschreiben
  - insbesondere wird eine Klassendefinitionssprache und ein Vererbungsmechanismus ähnlich wie in objekt-orientierten Programmiersprachen definiert
  - Dieses Vokabular kann dann seinerseits verwendet werden, um eine bestimmte Domäne zu beschreiben

# Die wichtigsten Elemente

- **Class: Eine Klasse beschreibt ein abstraktes Konzept**
  - `type`: gibt eine Instanz einer Klasse an
  - `subClassOf`: definiert eine Unterklasse (mit Vererbung)
  - `Resource`: the mother of all classes
- **Property: Definiert eine Property (binäre Relation)**
  - `domain`: Einschränkungen über die möglichen Ressourcen für eine Property (das erste Element der Relation)
  - `range`: Einschränkungen über den Wertebereich von Properties (das zweite Element der Relation)
    - z.B. nur Dozenten dürfen eine Vorlesung unterrichten
  - `subPropertyOf`: Definiert eine Unter-Property
    - es kann also genauso wie es Klassen-Hierarchien gibt, auch Hierarchien von Properties geben
  - `Statement`: the mother of all properties

# Beispiel

RDF Schema kodiert  
allgemeines Wissen  
über die Domäne



RDF kodiert  
spezifische Fakten

RDFS  
RDF

# RDF-Schema Syntax in RDF/XML

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

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

```
<rdf:Description ID="involves">
 <rdf:type resource="http://www.w3.org/...#Property" />
 <rdfs:domain rdf:resource="#Course" />
 <rdfs:range rdf:resource="#AcademicStaffMember" />
</rdf:Description>
```

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

...

# Schicht 4: Ontologien

- Begriffklärung:
  - in der Philosophie:
    - Ontologie = “die Lehre vom Sein”
  - in der Informatik:
    - eine Ontologie ist die Beschreibung einer Domäne
- Üblicherweise bezeichnet man mit Ontologie
  - eine hierarchische Struktur von Konzepten
  - die aber auch untereinander komplexe Querverbindungen haben können
  - im Prinzip sind die Beispiele, die wir gesehen haben, alles einfache Ontologien



# RDF Schema und Ontologien

- RDF Schema ist geeignet, um einfache Ontologien zu modellieren
- einige Dinge fehlen jedoch:
  - lokale Einschränkungen der Domäne oder Ranges für einzelne Klassen
    - z.B. range von `child` ist `Person` für Personen und `Animal` sonst
  - Quantoren und Einschränkungen der Kardinalität von Mengen
    - z.B. jede Person hat 2 Eltern, jeder Vater hat mind. 1 Kind
  - Angabe von speziellen Eigenschaften von Properties
    - z.B. Transitivität, Gegenteil-von, etc.
  - komplexe Mengen-Operationen auf Klassen
    - Klassen können disjunkt sein
    - mit Bool'schen Operatoren aus anderen Klassen zusammengesetzt werden
- Trade-off zwischen Ausdruckskraft und Unterstützung von effizientem Reasoning

# OWL

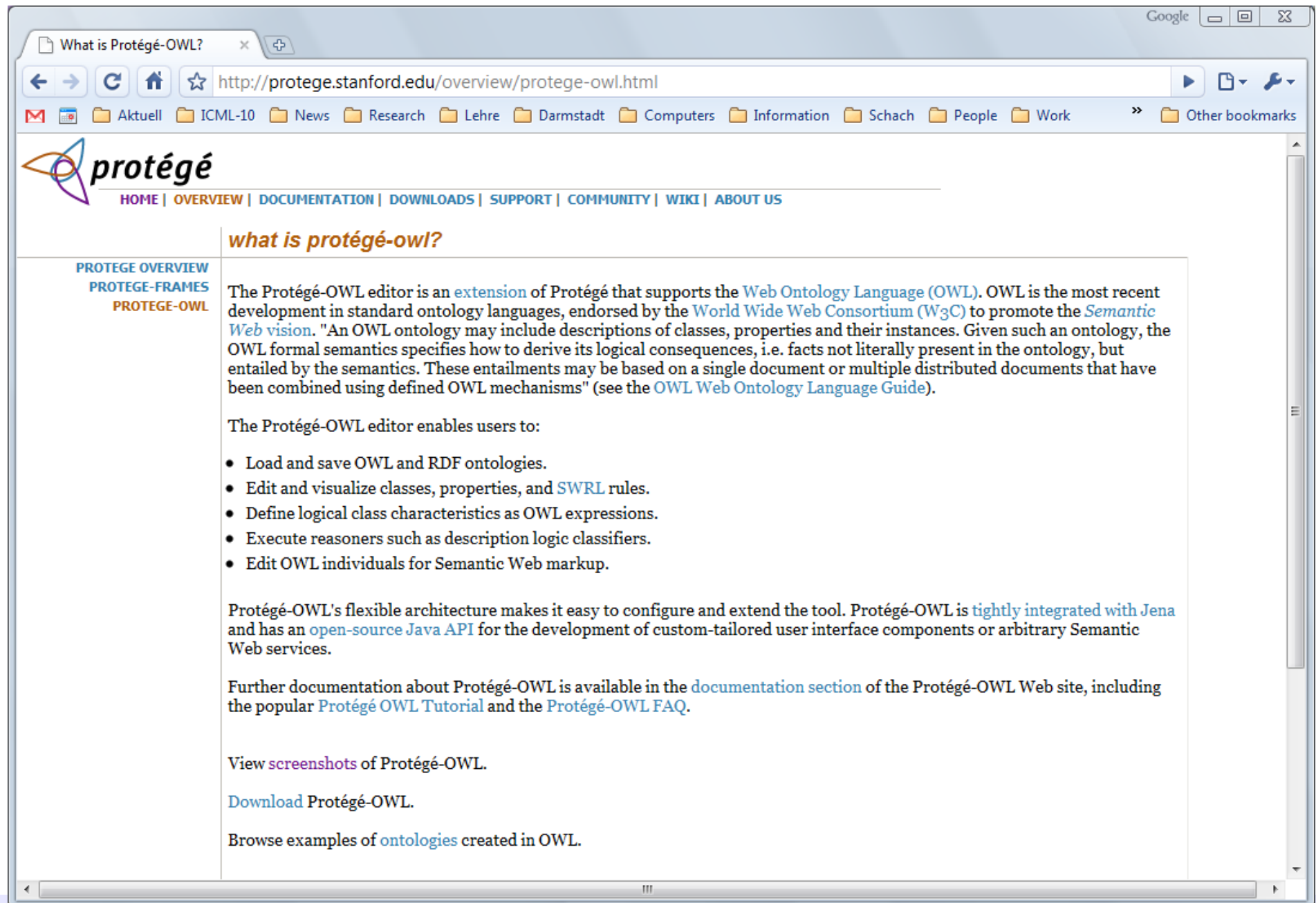
- OWL: Web Ontology Language
  - syntaktische Erweiterung von RDFS, das viele Probleme behebt
- Geschichte von OWL
  - DAML: Europäische Entwicklung
  - OIL: US-Amerikanische Entwicklungwurden von W3C zu einem gemeinsamen Rahmen (OWL) fusioniert
- Eigentlich gibt es drei Sprachen:
  - **OWL full**:
    - OWL Syntax + RDF
  - **OWL DL** (Description Logic)
    - Einschränkungen auf Teilmenge, die Description Logic entspricht
    - Nicht mehr alle RDF-Dokumente sind OWL DL Dokumente
    - dafür effizienteres Reasoning
  - **OWL Lite**
    - ein noch einfacher zu verstehendes Subset

# Beispiel OWL/RDF Syntax

- Die Klasse `facultyInCS` ist dadurch charakterisiert, daß sie die Schnittmenge zweier Klassen ist
  - der Menge aller Faculty Mitglieder (Klasse `faculty`)
  - der Menge aller Objekte, die zum CS Department gehören

```
<owl:Class rdf:ID="facultyInCS">
 <owl:intersectionOf rdf:parseType="Collection">
 <owl:Class rdf:about="#faculty">
 <owl:Restriction>
 <owl:onProperty rdf:resource="#belongsTo"/>
 <owl:hasValue rdf:resource="CSDepartment"/>
 </owl:Restriction>
 </owl:intersectionOf>
 </owl:Class>
```

# Protégé Ontologie-Editor



The screenshot shows a web browser window with the address bar displaying `http://protege.stanford.edu/overview/protege-owl.html`. The browser's bookmark bar contains folders for 'Aktuell', 'ICML-10', 'News', 'Research', 'Lehre', 'Darmstadt', 'Computers', 'Information', 'Schach', 'People', and 'Work'. The page content features the Protégé logo and a navigation menu with links for HOME, OVERVIEW, DOCUMENTATION, DOWNLOADS, SUPPORT, COMMUNITY, WIKI, and ABOUT US. The main heading is 'what is protégé-owl?'. A sidebar on the left lists 'PROTEGE OVERVIEW', 'PROTEGE-FRAMES', and 'PROTEGE-OWL'. The main text describes the Protégé-OWL editor as an extension of Protégé that supports the Web Ontology Language (OWL). It mentions that OWL is endorsed by the World Wide Web Consortium (W3C) and is used to promote the Semantic Web vision. The text explains that an OWL ontology includes descriptions of classes, properties, and their instances, and that the formal semantics specify how to derive logical consequences. A list of capabilities of the Protégé-OWL editor is provided, including loading and saving OWL and RDF ontologies, editing and visualizing classes and properties, defining logical class characteristics, executing reasoners, and editing OWL individuals. The text also notes that Protégé-OWL has a flexible architecture and is tightly integrated with Jena. Finally, it provides links to further documentation, screenshots, and examples.

What is Protégé-OWL?

HOME | OVERVIEW | DOCUMENTATION | DOWNLOADS | SUPPORT | COMMUNITY | WIKI | ABOUT US

## what is protégé-owl?

**PROTEGE OVERVIEW**  
**PROTEGE-FRAMES**  
**PROTEGE-OWL**

The Protégé-OWL editor is an [extension](#) of Protégé that supports the [Web Ontology Language \(OWL\)](#). OWL is the most recent development in standard ontology languages, endorsed by the [World Wide Web Consortium \(W3C\)](#) to promote the *Semantic Web vision*. "An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms" (see the [OWL Web Ontology Language Guide](#)).

The Protégé-OWL editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and [SWRL](#) rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

Protégé-OWL's flexible architecture makes it easy to configure and extend the tool. Protégé-OWL is [tightly integrated with Jena](#) and has an [open-source Java API](#) for the development of custom-tailored user interface components or arbitrary Semantic Web services.

Further documentation about Protégé-OWL is available in the [documentation section](#) of the Protégé-OWL Web site, including the popular [Protégé OWL Tutorial](#) and the [Protégé-OWL FAQ](#).

View [screenshots](#) of Protégé-OWL.

[Download](#) Protégé-OWL.

Browse examples of [ontologies](#) created in OWL.

# Protégé Ontologie-Editor

The screenshot displays the Protégé 3.1 ontology editor interface. The main window title is "travel Protégé 3.1 (file:VC:\protege-owl\owl\travel.pprj, OWL Files (.owl or .rdf))". The interface is divided into several panes:

- Left Pane (Asserted Hierarchy):** Shows a tree view of classes. The "Destination" class is expanded, and "FamilyDestination" is selected.
- Top Pane (CLASS EDITOR):** Shows the class "FamilyDestination" (instance of owl:Class). It includes tabs for "Name", "SameAs", and "DifferentFrom". The "Name" tab shows "FamilyDestination". The "rdfs:comment" tab shows the text: "A destination with at least one accommodation and at least 2 activities."
- Right Pane (Annotations):** Shows a table of annotations for the class.
 

Property	Value	Lang
rdfs:comment	A destination with at le...	
- Bottom Pane (Asserted Conditions):** Shows the class "FamilyDestination" with the following conditions:
  - Destination (NECESSARY & SUFFICIENT)
  - hasAccommodation ≥ 1
  - hasActivity ≥ 2
- Right Pane (Properties):** Shows the class "FamilyDestination" with the following properties:
  - hasAccommodation (multiple Accommodation) with cardinality 1
  - hasActivity (multiple Activity) with cardinality 2
  - hasPart (multiple Destination)
- Bottom Pane (Disjoints):** Shows the class "RetireeDestination" as a disjoint class.

The interface also includes a menu bar (File, Edit, Project, OWL, Code, Window, Tools, Help), a toolbar with various icons, and a status bar at the bottom with "Logic View" and "Properties View" options.

# Protégé Ontologie-Editor

The screenshot displays the Protégé 3.1 ontology editor interface. The title bar shows the file path: `(file:\C:\protege-owl\owl\travel.pprj, OWL Files (.owl or .rdf))`. The menu bar includes `File`, `Edit`, `Project`, `OWL`, `Code`, `Window`, `Tools`, and `Help`. Below the menu is a toolbar with various icons for file operations and editing. The main interface is divided into several sections:

- Navigation and View Controls:** Includes tabs for `OWLClasses`, `Properties`, `Forms`, `Individuals`, `Metadata`, and `OWL Viz`. Below these are icons for different visualization styles and search functions.
- Model Selection:** Buttons for `Asserted Model` and `Inferred Model`.
- CLASS BROWSER:** A sidebar on the left showing the class hierarchy for the project `travel`. It lists `Asserted Hierarchy` and displays a tree view of classes.
- Main Ontology Diagram:** A large central area showing a network of classes and their relationships. The `Destination` class is the central node, with several subclasses: `RuralArea`, `UrbanArea`, `BudgetHotelDestination`, `RetireDestination`, `Beach`, `FamilyDestination`, `QuietDestination`, and `BackpackersDestination`. Other classes include `Farmland`, `NationalPark`, `Town`, `City`, `Capital`, `AccommodationRating`, `Contact`, `Surfing`, and `Sight`. Relationships are indicated by arrows labeled `is-a`.
- Legend:** A legend in the bottom right corner explains the symbols used in the diagram:
  - A yellow circle represents `Destination`.
  - A circle with a horizontal line represents `NECESSARY & SUFFICIENT`.
  - A circle with a vertical line represents `NECESSARY`.
  - A circle with a diagonal line represents `Sufficient`.

# OWL Klassen-Konstruktoren

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	$\leq_n P$	$\leq 1$ hasChild	$\exists \leq_n y.P(x, y)$
minCardinality	$\geq_n P$	$\geq 2$ hasChild	$\exists \geq_n y.P(x, y)$



# OWL Constraints

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 <sup>-</sup>
transitiveProperty	$P^+ \sqsubseteq P$	ancestor <sup>+</sup> $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1$ hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1$ hasSSN <sup>-</sup>



# Probleme mit Ontologien

- **Ontologie-Erstellung**
  - Unterstützung zur Entwicklung von Ontologien aus Ressourcen wie HTML-Seiten
  - Automatisierung z.B. durch Clustering-Algorithmen
- **Ontologie-Einordnung**
  - Mapping bestehender Einheiten (z.B. Web-Seiten) auf bestehende Ontologien
  - Automatisierung z.B. durch Klassifikations-Algorithmen
- **Ontologie-Mapping bzw. Ontologie-Merging**
  - verschiedene Ontologien können denselbem Sachverhalt mit unterschiedlichem Vokabular und unterschiedlicher Struktur beschreiben
  - Wie erkenne ich, daß zwei Knoten das gleiche Konzept repräsentieren?

# Komplexe Ontologien

- WordNet
  - <http://wordnet.princeton.edu/>
  - Eigendefinition: a lexical database for the English language
  - für Suchworte der Englischen Sprache finden sich
    - englische Umschreibungen
    - Synonyme, Antonyme, Oberbegriffe, Unterbegriffe,...
- Cyc und OpenCyc
  - <http://www.opencyc.org/>, <http://www.cyc.com/>
  - Eigendefinition: the world's largest and most complete general knowledge base and commonsense reasoning engine
  - Resultat eines Ende der 80'er sehr stark geförderten Projektes mit dem Ziel, Alltagswissen und alltägliche Schlüsse zu formalisieren
    - Die Resultate blieben ein wenig hinter den Zielen zurück
- verschiedenste Domain-spezifische Ontologien
  - z.B. zur Kategorisierung medizinischer Fachartikel, etc.

# Schichten 5-7: Logic, Proof, Trust

- sind zwar an sich gut erforschte Konzepte
  - die Umsetzung im Szenario des Semantic Webs ist aber noch in den Kinderschuhen
- aktive Forschungsgebiete

# Schichten 5-7: Logic, Proof, Trust

I would like to buy this book;  
please send my company an invoice

I am an employee of XYZ Corp  
(because it says so on this Web  
page, which is an XYZ Corp  
official document)



ugly XML encoding

```
<definitions name="HelloService"
targetNamespace="http://www.ecerami.com/wsdl/HelloService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <message name="SayHelloRequest">
 <part name="firstName" type="xsd:string"/>
 </message>
 <message name="SayHelloResponse">
 <part name="greeting" type="xsd:string"/>
 </message>

 <portType name="Hello_PortType">
 <operation name="sayHello">
 <input message="tns:SayHelloRequest"/>
 <output message="tns:SayHelloResponse"/>
 </operation>
 </portType>

 <binding name="Hello_Binding" type="tns:Hello_PortType">
 <soap:binding style="rpc"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="sayHello">
 <soap:operation soapAction="sayHello"/>
 <input>
 <soap:body
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 namespace="urn:examples:helloservice"
 use="encoded"/>
 </input>
 <output>
 <soap:body
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 namespace="urn:examples:helloservice"
 use="encoded"/>
 </output>
 </operation>
 </binding>
</definitions>
```

Proof Verifier

OK, book successfully ordered

Yes this proof is correct

No this proof is flawed

Sorry, we need a credit card!

(Easy to build once  
the Logic layer is fixed)