# Semantic Web

Seminar "Konzepte von Informationssystemen SS 2002"
Arbeitsbereich Datenbanken und Informationssysteme
Wilhelm-Schickard-Institut für Informatik
Eberhard-Karls-Universität Tübingen

Betreut von Jochen Hipp

Seminarausarbeitung

Holger Szillat
szillat@informatik.uni-tuebingen.de

3. Juli 2002
Version 1.36

**Zusammenfassung**

Heutige Software kann den Inhalt von Webseiten im allgemeinen nicht verstehen. Die Suche nach bestimmten Informationen beschränkt sich daher meist auf die Suche nach bestimmten Schlüsselwörtern.

*Semantic Web* versucht, basierend auf der bestehenden Internet-Infrastruktur, die Bedeutung von Information maschinen-lesbar zu machen. *RDF* ermöglicht dies mittels eines einfachen, aber sehr leistungsfähigen Modells. *RDF-Schema* erlaubt es dem Benutzer eigene Ontologie-Vokabularien zu definieren. *OIL* und *DAML* erweitern dieses Modell um Logik-Elemente und ermöglichen damit mächtigere Ontologien.

Diese Ausarbeitung meines Seminarvortrags versucht einen Überblick und eine Einführung in das Thema *Semantic Web* und seine Sprachen und Tools zu geben.

# Inhaltsverzeichnis

# 1 *Semantic Web* — Was ist das überhaupt?

Das heutige *World Wide Web* ist in den Jahren seit seiner "Geburt" explosionsartig gewachsen und erlebte in den 90er Jahren ein fast exponentielles Wachstum. Dies war teilweise auch dem Umstand zu verdanken, daß jeder Benutzer relativ einfach eigene HTML-Seiten erstellen konnte. Zu den Microsoft-Betriebssystemen beispielsweise, gab es neben einem Web-Browser auch ein Programm zu Erstellen von eigenen Webseiten: FrontPage Express. Aber auch Netscape's Browser hatte bald ein solches Programm. Auch wer diese nicht nutzen wollte oder konnte, kam mit einem einfachen Texteditor auch schon zu respektablen Ergebnissen.

Obwohl die Infrastruktur in Form des Internets diesem Ansturm zwar gewachsen ist, hat sich das Problem verschärft, in dieser Flut an Informationen die gesuchte zu finden.

## 1.1 Das Problem mit dem heutigen Web

Das Internet hat in den Jahren seit seiner Geburt eine explosionsartige Verbreitung erlebt. Das *World Wide Web* läßt sich mittlerweile nicht nur auf PC's finden, sondern auch auf Handy's, PDA's, in (oder an) Kühlschränken, Mikrowellen, etc. Zwar verfügen diese "kleinen" Internet-Clients nicht über das Leistungspotential eines heutigen PCs, trotzdem ist das kein Hinderungsgrund solche Geräte zu vernetzen.

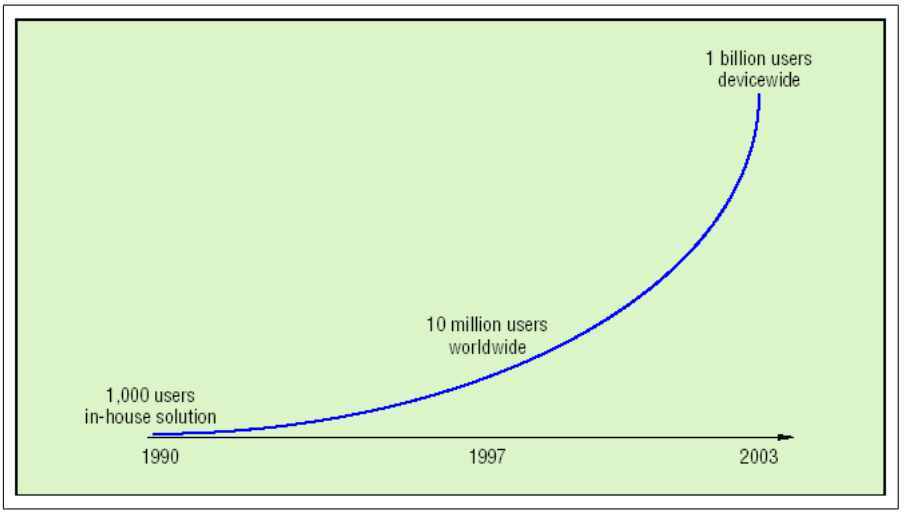


Abbildung 1: Explosionsartige Entwickung des Internets (Stand 1997)

### 1.1.1 Informationen suchen

Ebenso sehr wie sich die Möglichkeiten vervielfacht haben, dem *World Wide Web* zu begegnen, genauso sehr hat sich auch das Angebot an Informationen vermehrt. Umso schwieriger ist es mittlerweile auch geworden einzelne, bestimmte Informationen zu finden. Die Schwierigkeit besteht dabei weniger darin überhaupt etwas zu

finden, als vielmehr, in der Flut an Informationen die unbrauchbaren herauszufiltern. Suchmaschinen versuchen sich diesem Problem mit kleveren Algorithmen zum Text-Retrieval entgegenzustellen, allerdings können auch sie manche Probleme nicht wirklich lösen.

So kann eine Suchmaschine die Bedeutung mancher Worte im Zusammenhang mit anderen Worten nicht "verstehen". Ein Beispiel ist der Beruf des "Müllers", der aber natürlich auch als Name bekannt ist. Eine Suche nach "Müller" bei einer beliebigen Suchmaschine wird also sowohl Ergebnisse für den Beruf als auch für den Namen liefern.

Die Schwierigkeit dem Problem Herr zu werden, entsteht im wesentlichen dadurch, dass Webseiten eigentlich dafür gedacht sind von Menschen gelesen und verstanden zu werden, während Maschinen nur ein Mittel zur Darstellung sind. Dieser Umstand hat auch den Begriff des sog. "Eyeball-Webs" geprägt.

### 1.1.2 Informationen vergleichen

Das heutige Web macht es auch schwierig Informationen zu vergleichen. Seit dem Internet-Boom gibt es immer mehr Anbieter im Web, für alle Arten an Produkten. Preisvergleiche sind im Internet aber eigentlich sehr einfach: Man ruft einfach die entsprechenden Webseiten der Anbieter auf und "legt" die Browser-Fenster nebeneinander.

Das Problem dabei sind dann aber die "versteckten Preise": (Einfuhr-)Steuern, Versandkosten, Mindermengenzuschläge, usw. Diese tauchen meist erst bei der endgültigen Bestellung auf.

Heutzutage verdienen sog. Preisagenturen nicht schlecht an diesem Umstand. Der Versuch dieses Problem mit kleveren Programmen zu umgehen, führen unweigerlich zu Programmen, die für jede Webseite neu geschrieben werden müssen, und einen ziemlich großen Wartungsaufwand verursachen, sobald eine "bekannte Webseite" ein neues Layout erhält.

Doch auch wenn die Informationen in Form einer *XML*-Datei zur Verfügung stehen, gibt es Probleme. Zwei Anbieter eines Produkts führen dasselbe Produkt in ihren Katalogen, doch mit unterschiedlicher Formatierung. Der Versuch diesem Problem dadurch zu begegnen für Produkte standardisierte Tags einzuführen, muß unweigerlich scheitern. Warum sollte sich jemand daran halten? Nur damit sein Konkurrent leichter seine Preise an die eigenen anpassen kann?

## 1.2 Die Probleme lösen

Wie lassen sich also die oben aufgeführten Probleme lösen? Natürlich soll *Semantic Web* der Weg aus dem Dilemma sein, doch wie kann das konkret passieren?

### 1.2.1 Informationen besser suchen

Manche Informationen lassen sich vergleichbar leicht finden: Die Homepage eines gewissen Peter Koch, beispielsweise. Jedoch werden führen alleine die beiden Begriffe "Homepage" und "Koch" bei `www.google.de` zu ca. 148.000 Treffern.

Dieses Ergebnis würde sich zwar bessern je mehr Wörter man angibt, doch eigentlich suche ich einen Kollegen von Herrn Koch. Dieser, so erzählte mir Herr Koch auf einer Konferenz, hätte vor kurzem ein kleines Essay zum Thema *Semantic Web* geschrieben. Natürlich interessiert mich dieses Essay, doch leider erwähnte Herr Koch nicht den Namen seines Kollegen und auf seiner Homepage wird er wahrscheinlich sowieso nicht erwähnt.

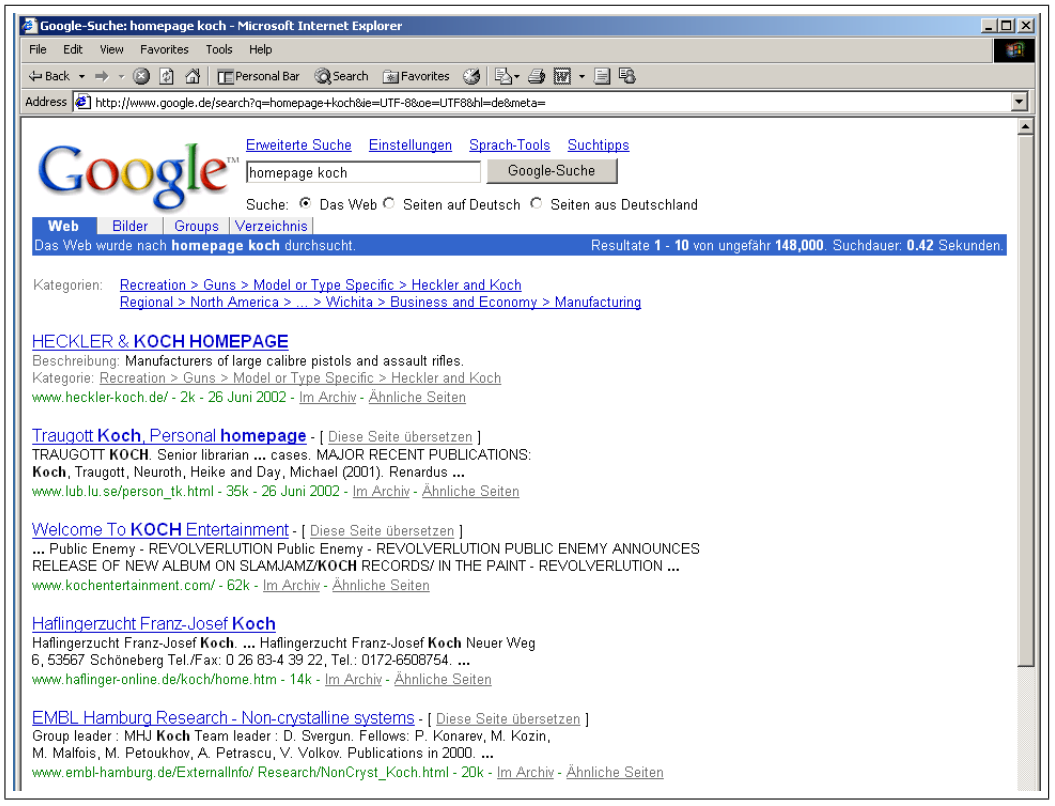


Abbildung 2: Suche bei `www.google.de`

Eine solche Suche wäre mit dem heutigen Web wahrscheinlich erfolglos. Eine allgemeine Suche nach dem Wort "*Semantic Web*" wird auch nicht zum Erfolg führen. Mein Problem: Ich suche eine Information, die nicht "direkt" gesucht werden kann. Die Graphik in Abbildung 3 verdeutlicht das Problem.

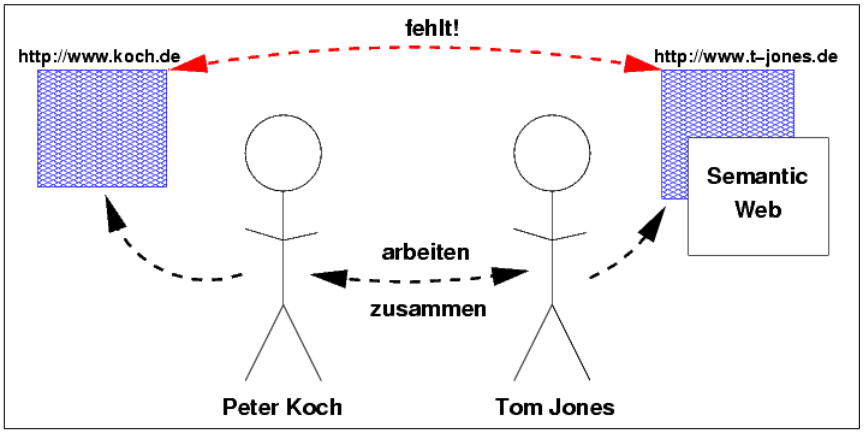


Abbildung 3: Nicht verlinkte Informationen kann man nicht finden.

*Semantic Web* ermöglicht es dieses Problem zu lösen. Die Information, daß die Homepage "`http://www.koch.de`" (natürlich) von Herrn Koch kreiiert wurde, würde er natürlich zur Verfügung stellen, wie es in Abbildung 4 graphisch dargestellt ist.

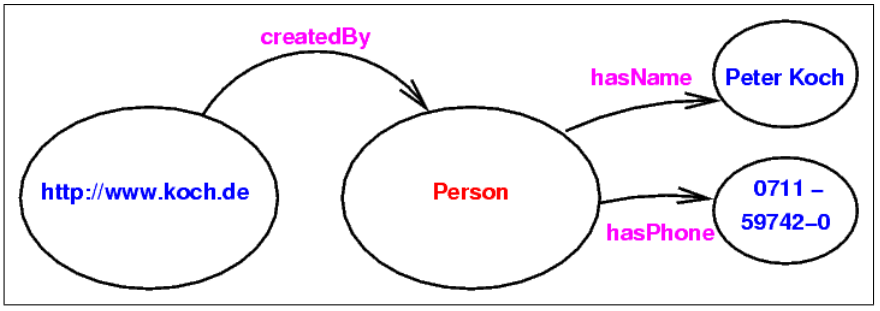


Abbildung 4: Herr Koch hat seine eigene Homepage kreiiert

Eine solche Meta-Information nennt man auch "Ontologie". Eine genauere Definition findet sich später.

Die Ontologie in Abbildung 5, daß Herrn Kochs Kollege "Tom Jones" eine Homepage hat, auf der sich ein Essay zum Thema *Semantic Web* befindet, würde Herr Jones erzeugen. Schließlich beschreibt diese Ontologie Informationen auf seiner Homepage.

Was jetzt noch fehlt, ist die Verbindung zwischen Herrn Koch und Herrn Jones. Diese Information könnte von der Firma kommen, für die beide arbeiten, denn die Firma möchte vielleicht aus Supportgründen bekannt machen, daß diese beiden an einem bestimmten Projekt zusammengearbeitet haben. Graphisch repräsentiert in Abbildung 6.

Nun ist der Pfad vollständig: Eine Suche würde auf der Homepage von Herrn Koch beginnen, eine weitere Ontologie würde zur Firma von Herrn Koch führen, die Zusammenarbeit zwischen Herrn Koch und Herrn Jones zur Homepage von Herrn

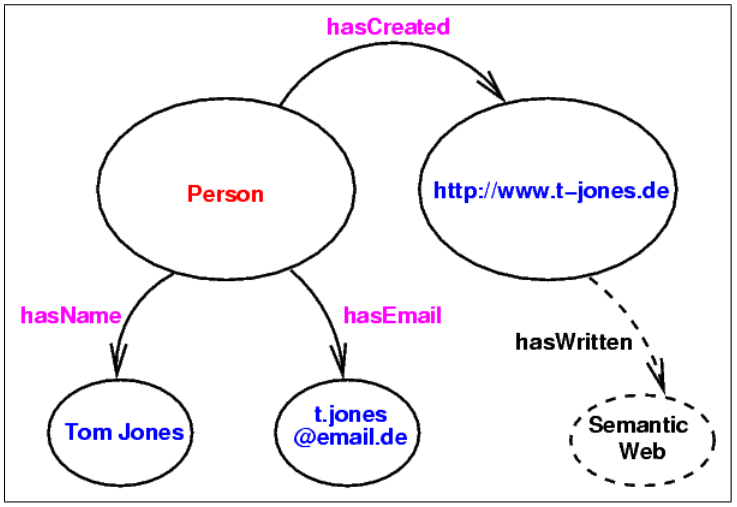


Abbildung 5: Herr Jones hat auch eine Homepage kreiiert

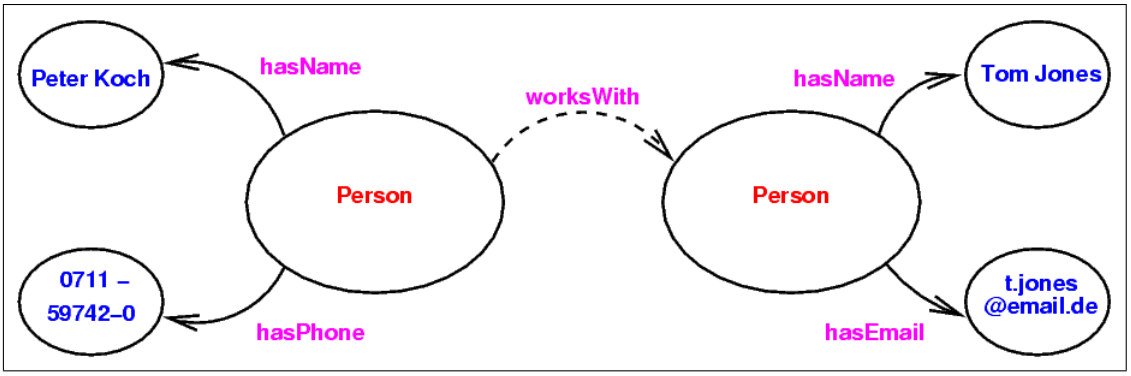


Abbildung 6: Herr Koch und Herr Jones arbeiten zusammen

Jones und dort schließlich zum gesuchten Essay über *Semantic Web*. Siehe Abbildung 7.

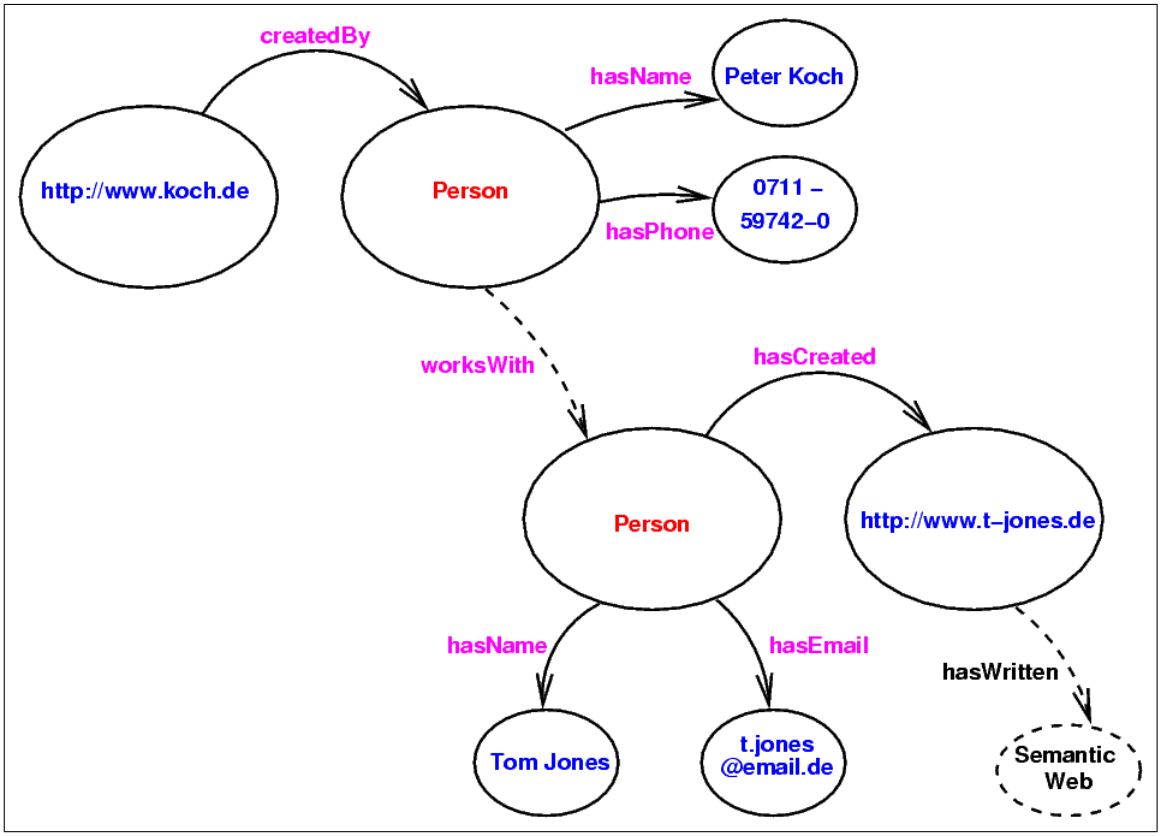


Abbildung 7: Eine verlinkte Information läßt sich finden

### 1.2.2 Informationen besser vergleichen

Informationen lassen sich mit *Semantic Web* auch besser vergleichen. Wenn zwei Anbieter in ihrem Katalog dasselbe Produkt anbieten, läßt sich ein Vergleich ziehen, wenn beide eine Ontologie an die Einträge in ihrem Katalog knüpfen. Damit wird beschrieben, welche Einträge welchem Konzept entsprechen. Ein sog. "Translation-Server" kann dann die Einträge vergleichen und nötigenfalls in ein gemeinsames Datenformat übertragen.

Ein Beispiel soll dies verdeutlichen: In Abbildung 8 bieten zwei Anbieter dasselbe Produkt an, einen "`Daimler 230 SE`". Doch beide benutzen nicht nur verschiedene Währungen, auf der einen Seite Dollar, auf der anderen DM, sondern beide Preise sind mit verschiedenen Tags markiert, obowhl beide dasselbe Konzept bezeichnen: Den Preis des Produkts.

Der Translation-Server könnte mit Ontologien nicht nur die verschiedene Syntax der beiden Katalogeinträge "verstehen", er könnte auch eine Währungskonversion durchführen und es somit dem Benutzer ermöglichen die Produkte zu vergleichen.

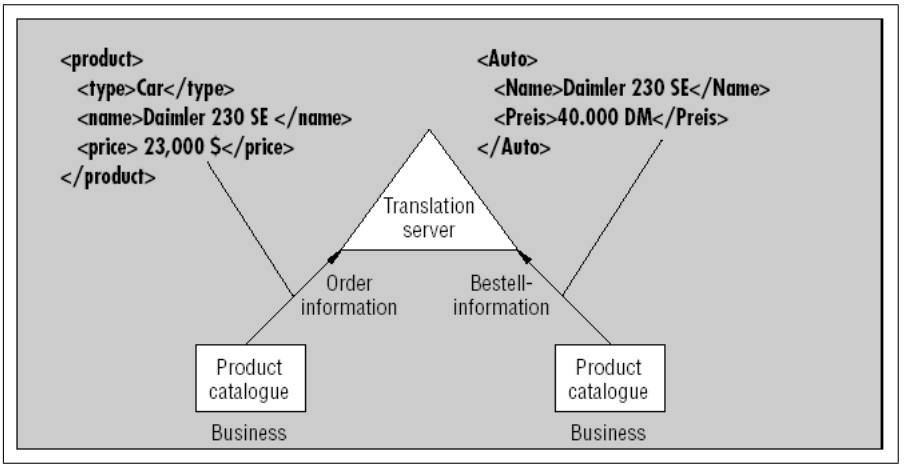


Abbildung 8: Informationen vergleichen

### 1.2.3 Vision: Proof-Checking

Mit *Semantic Web* ist auch "Proof-Checking" möglich, d.h. ein Agent kann eine Behauptung gegenüber einem anderen Agenten beweisen. Ein Beispiel:

Der Server-Agent behauptet dem Client-Agenten gegenüber, daß dieser ihm einen Betrag von $30 schuldet, wie in Abbildung 9 dargestellt. Natürlich zweifelt der Client diese Behauptung an und verlangt einen Beweis. Der Server beweist seine Behauptung anhand von zwei Fakten: Der Benutzer `user1` hat bei AOL ein Buch `book1` gekauft, bewiesen durch eine "Confirmation-URL", bei der als dritte Instanz beim damaligen Kauf der Nachweis einer Transaktion hinterlegt wurde (Eine Art Kopie des Kassenbelegs bei einem Notar). Der Server beweist ferner durch einen Link auf die Preishistorie des Artikels bei AOL, daß der Preis des Buches damals $30 betrug. Als letztes zitiert der Server eine Ontologie, die besagt, daß wenn "Ein Kunde $a$ einen Artikel $b$ bei $c$ gekauft hat und dieser Artikel $b$ den Preis $d$ hatte, dann folgt daraus, daß $a$ $c$ den Betrag $d$ schuldet". Der Client muß sich dieser Logik beugen und steckt den Scheck in die eMail...

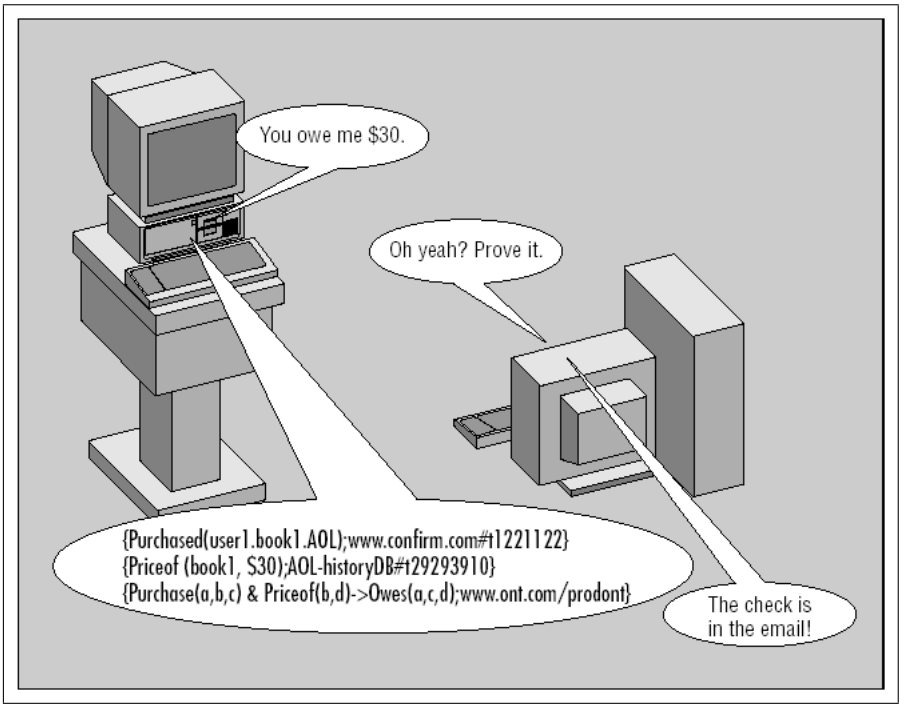


Abbildung 9: Proof of Trust

### 1.2.4 Vision: Endliche Automaten mit *Semantic Web*

Eine weitere Anwendungsmöglichkeit von *Semantic Web* liegt in der Möglichkeit CGI-Programme als abstrakte Maschinen darzustellen. Warum sollte man dies tun? Jeder der mal bei einem großen Internet-Buchhändler eingekauft hat, kennt den Vorgang, daß man durch mehrere Schritte gehen muß, bei denen Liefer- und Rechnungsaddresse, Kreditkarteninformationen u.ä. abgefragt werden. Manche Anbieter benutzen dafür zwar Cookies und Datenbanken auf dem Server um den Vorgang zu beschleunigen und zu vereinfachen, doch wenn man zum ersten Mal bei einem Anbieter kauft, hilft das natürlich wenig.

Der Buchhändler kann aber nun den Kaufvorgang als abstrakte Maschine darstellen und als Ontologie kodieren. Dadurch kann ein Client-Agent “verstehen”, was in welchem Schritt verlangt wird und die entsprechenden Informationen zur Verfügung stellen, ohne daß eine weitere Benutzerinteraktion nötig ist. Auch kann der Client damit auf Fehlerzustände “intelligent” reagieren, indem z.B. die Zahlungsart gewechselt wird.

## 1.3 Die Vision

Tim Berners-Lee, einer der Erfinder des Web wie wir es heute kennen, hat 1999 in seinem Buch "Weaving the Web" [BL99] das Problem erkannt: Die nächste Generation des Webs wird sich mehr mit Meta-Informationen beschäftigen müssen um die gewaltige Flut an Informationen in den Griff zu kriegen.

Eine kleine Vision soll die Zukunft mit *Semantic Web* klar machen: Betrachten wir den kleinen Diplom-Informatiker Bob Alice. Bob arbeitet seit kurzer Zeit an der Universität Tübingen. An diesem Morgen fühlt er sich nicht besonders gut, geht aber trotzdem, weil er sehr pflichbewußt ist, zur Arbeit. Doch dort merkt er nach ein paar Stunden, daß ein Arztbesuch vielleicht doch keine so schlechte Idee ist.

Leider kennt sich Bob in der Gegend nicht so gut aus, sein Hausarzt ist sowieso seit ein paar Jahren in Ruhestand und weit fahren möchte er auch nicht (mit den Kopfschmerzen...).

Er könnte in dieser Situation natürlich zum "Örtlichen Telefonbuch" greifen und in der Rubrik "Ärzte" schnell einige Allgemeinmediziner finden, doch was nützt ihm diese Information? Im "Örtlichen" stehen weder die Sprechstunden noch eine Anfahrtsroute, geschweige denn eine individuelle Anfahrtsroute von seinem Arbeitsplatz zum Arzt seiner Wahl.

Zum Glück hat Bob Internetzugang, einen PalmTop-Computer (mit Bluetooth) und *Semantic Web* hat sich bereits durchgesetzt. Schnell lässt sich Bob eine Liste aller medizinschen Einrichtungen und Ärzte in Tübingen auf seinen PalmTop übertragen. Dieser startet einen "Agenten", der aus dieser Liste die Allgemeinmediziner heraussucht, die semantischen Informationen ihrer Homepages abruft und dadurch schnell herausfinden kann, welcher Arzt denn in der nächsten halben Stunde Sprechstunde hat. Danach nimmt der Agent Kontakt zum Navigationssystem in Bob's Auto auf und errechnet die schnellste Route zum jeweiligen Arzt. Die Ärzte, für die Bob (mit seinem Fahrstil) länger als 20 Minuten bräuchte, streicht der Agent vorläufig von der Liste, alle anderen zeigt er Bob an.

Mittlerweile hat ein Kollege Bob eine eMail geschickt und ihm zu einem bestimmten Arzt geraten. Bob teilt dem Agenten seines PalmTops mit, daß er diesen Arzt auch gerne auf der Liste hätte und bitte mit höchster Priorität.

Bob kann nun auf dieser Liste sehen, welche Ärzte er wie schnell erreichen kann und wie lang er bräuchte, um zum Arzt des Kollegen zu fahren. Kurzerhand entscheidet er sich für einen Arzt, der Agent schickt die Telefonnummer der Praxis auf das Handy und Bob kann einen Termin für heute 15:00 Uhr ausmachen.

Lassen wir Bob nun zum Arzt fahren und betrachten wir, was denn eigentlich passiert ist: Natürlich ist es etwas optimistisch, anzunehmen, daß jede Arztpraxis in und um Tübingen tatsächlich über eine eigene Webseite verfügt und auf dieser Sprechstunden und Anschrift veröffentlicht. Schliesslich ist der praktische Nutzen im *World Wide Web* zu werben, für Arztpraxen gering, da diese meist durch "Mundpropaganda" zum selben Ziel kommen.

Interessanter ist jedoch, daß das *Semantic Web* die für Bob wichtigen Daten in einem Format zu Verfügung stellen konnte, die es seinem PDA und einem Agenten (d.h. ein "Stück Software") ermöglichten schnell an diese Informationen zu kommen. Der Agent war dann in der Lage diese Informationen zu benutzen um eine Strecken-

und Terminplanung durchzuführen.

# 2 *Semantic Web* — Wie geht das?

Nachdem nun die Motivation für *Semantic Web* vorgestellt wurde, stellt sich natürlich als nächstes die Frage, wie man eine beliebige Webseite mit Meta-Informationen ausrüstet. Wie "macht" man also *Semantic Web*?

Eine eigene, proprietäre Lösung wäre möglich, würde das Problem aber bestimmt nicht lösen und würde für großen Programmier- und Wartungsaufwand sorgen.

Die Idee, daß HTML-Seiten über Meta-Informationen verfügen, führt fast unweigerlich zum Ansatz, HTML-Seiten direkt mit Meta-Informationen anzureichern. Diesen Ansatz verfolgt auch *SHOE* seit 1997. Das Problem dabei ist, daß dies die Wartung und Pflege solcher Seiten erschwert, insbesondere mit Tools, welche die Meta-Informationen nicht verstehen und auch nicht unterstützen und daher entweder ignorieren oder schlimmstenfalls zerstören. Eine Gefahr, die durch fast jeden HTML-Editor gegeben ist.

Das *W3C* verfolgt daher eine andere Lösung, die auf *XML* aufbaut und eine stärkere Trennung zwischen der Information und ihrer Meta-Information betreibt: *RDF* und *RDFSchema*. Mit *RDF* lassen sich Informationen darstellen, die (fast) beliebige Ressourcen beschreiben können. Mit *RDFSchema* läßt sich diese Darstellung in (begrenztem) Maße beliebig um eigene Vokabularien erweitern.

*OIL* und *DAML* schließlich, nutzen *RDF* und *RDFSchema* und erweitern sie um mächtigere Konzepte und Logik.

In den folgenden Kapiteln wird jeder dieser Ansätze kurz untersucht.

## 2.1 Was ist eine Ontologie?

In den nachfolgenden Kapiteln wird ein Begriff sehr häufig benutzt werden, der Begriff der "Ontologie". Doch was eine Ontologie überhaupt? Das *Oxford English Dictionary* definiert eine Ontology als: *"[The] Study or Science of Being."*

In der Philosophie wird eine Ontologie definiert als *"A particuliar theory about being or reality"*.

In der Forschung der künstlichen Intelligenz ist eine Ontologie *"... a representation of a shared conceptualization of a particular domain"*.

Dies alles ist mehr oder weniger philosophisch und man kann sich nur schwer etwas Konkretes darunter vorstellen, daher ist es am einfachsten sich eine Ontologie als "Gesamtheit von Fakten, Regeln und Logik" vorzustellen, salopp ausgedrückt als "Wissen" (über etwas).

## 2.2 *SHOE* — Ein erster Prototyp

*SHOE* gilt fast schon als Veteran der *Semantic Web*-Sprachen. 1997 zum ersten Mal spezifiziert, ist der Ansatz bei *SHOE*, daß semantische Informationen in die eigentliche HTML-Seite eingebettet wird. Die Information und die Meta-Information bilden also mehr oder weniger eine Einheit. Dies macht auch der expandierte Name von *SHOE* deutlich: **S**imple **H**TML **O**ntology **E**xtensions.

Der Umstand, daß *SHOE* bereits so lange existiert, hat den Vorteil, daß es bereits einige Tools gibt, die *SHOE* unterstützen, z.B.:

- Der "Knowledge Annotator" ermöglicht es HTML-Seiten mit Ontologien anzureichern und dabei primär auf bestehende Ontologien (im Web) zurückzugreifen.
- Exposé ist quasi das Gegenstück und erlaubt es über Seiten mit SHOE-Ontologien zu suchen.

Diese beiden Tools sind wohl die wichtigsten, die man braucht um eine beliebige Webseite *SHOE*-fähig zu machen.

In den folgenden Beispielen möchte ich *SHOE* und einige seiner Möglichkeiten etwas näher erläutern.

### 2.2.1 Ein *SHOE*-Beispiel

Peter Koch hat, wie im vorigen Kapitel beschrieben, eine Homepage. Auf dieser hat er auch ein paar "seiner" Ontologien gespeichert:

```
<HTML>
<HEAD>
 <TITLE>Ontologien von Peter Koch</TITLE>
 <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
 <H1>Ontologien von Peter Koch</H1>
 Diese Seite hat die URL: http://www.koch.de/onto.html
 und speichert Ontologien.
 ...
 <ONTOLOGY ID="koch-ontology" VERSION="1.0">
  <USE-ONTOLOGY ID="base-ontology" VERSION="1.0" PREFIX="b"
                URL="http://www.cs.umd.edu/projects/plus/SHOE/base.html">
  <DEF-CATEGORY NAME="Person" ISA="b.SHOEentity">
  <DEF-CATEGORY NAME="Webpage" ISA="b.SHOEentity">

  <DEF-RELATION NAME="createdBy">
   <DEF-ARG POS="1" TYPE="Person">
   <DEF-ARG POS="2" TYPE="Webpage">
  </DEF-RELATION>

  <DEF-RELATION NAME="hasCreated">
```

```
   <DEF-ARG POS="1" TYPE="Webpage">
   <DEF-ARG POS="2" TYPE="Person">
  </DEF-RELATION>

  <DEF-INFERENCE>
   <INF-IF>
    <RELATION NAME="createdBy">
     <ARG POS="1" VALUE="x" VAR>
     <ARG POS="2" VALUE="y" VAR>
    </RELATION>
   </INF-IF>
   <INF-THEN>
    <RELATION NAME="hasCreated">
     <ARG POS="1" VALUE="y" VAR>
     <ARG POS="2" VALUE="x" VAR>
    </RELATION>
   </INF-THEN>
  </DEF-INFERENCE>
 </ONTOLOGY>
</BODY>
</HTML>
```

Herr Koch definiert hier also erstmal eine Ontologie namens "`koch-ontology`" samt einer Version-nummer und gibt an, daß er Ontologien von der Seite http://www.cs.umd.edu/projects/plus/SHOE/base.html verwendet. Ferner definiert er zwei Kategorien nämlich `Person` und `Webpage`, die sog. `SHOEentity`s sind.

Außerdem definiert er sich hier eine Relation `createdBy`, die beschreibt, daß eine Webseite von einer bestimmten Person geschrieben wurde. Die andere Relation `hasCreated` ist quasi die Symmetrie dazu, und beschreibt, daß eine bestimmte Person eine Webseite geschrieben hat. Interessant ist hierbei die mögliche Herleitung dieser Relation, die besagt, daß wenn eine Webseite $x$ von einer Person $y$ geschrieben wurde, dann hat diese Person $y$ eben auch die Webseite $x$ geschrieben. So trivial dieses Konzept für uns Menschen sein mag, ein Computer kann damit schon seine Schwierigkeiten haben.

Man beachte, daß die Relation `hasCreated` im nachfolgenden Beispiel aber auch direkt angegeben werden könnte. Die Inferenz ist nur eine "Möglichkeit".

Ferner hat Herr Koch natürlich auch eine Einstiegsseite auf seiner Homepage, die die obige Ontologie nutzt:

```
<HTML>
<HEAD>
 <TITLE>Homepage von Peter Koch</TITLE>
 <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
 <H1>Homepage von Peter Koch</H1>
 Diese Seite hat die URL: http://www.koch.de/index.html
 Ich bin Peter Koch...
 ...
```

```
 <INSTANCE KEY="http://www.koch.de/index.html">
  <USE-ONTOLOGY ID="koch-ontology" VERSION="1.0" PREFIX="my"
                URL="http://www.koch.de/onto.html">

  <RELATION NAME="my.createdBy">
   <ARG POS="1" VALUE="Peter Koch">
   <ARG POS="2" VALUE="http://www.koch.de">
  </RELATION>
 </INSTANCE>
</BODY>
</HTML>
```

Hier benutzt Herr Koch seine Relation `createdBy` um zu modellieren, daß er (Peter Koch) seine Webseite (`http://www.koch.de`) kreiiert hat.

### 2.2.2 Vor- und Nachteile von *SHOE*

Betrachtet man sich das obige Beispiel näher, so stellt man fest, daß es durchaus ein Vorteil sein mag, daß Information und Meta-Information in ein und derselben Datei gespeichert sind. Jedoch wird man auf den zweiten Blick feststellen, daß die allgemeine Ontologie auf einer anderen Seite steht, als die tatsächliche Nutzung derselben. Zwar ist das nicht unbedingt ein Nachteil, aber es trennt die beiden Konzepte "allgemeines Konzept" und "konkrete Nutzung" doch voneinander, was für gewisse Ontologien durchaus ein Nachteil sein kann.

Außerdem trennt *SHOE* Information und Meta-Information dann an anderer Stelle doch nicht voneinander: Die Meta-Information über die Homepage von Herrn Koch befindet sich in derselben Datei. Eine klare Trennung fehlt! Ein Umstand, der in der heutigen Zeit durchaus für Unbehagen sorgen kann.

Auf der Haben-Seite kann *SHOE* seine Tools verbuchen, die einen Benutzer sowohl in der Erzeugung von Ontologien als auch bei der Suche mit Ontologien unterstützen.

## 2.3 *RDF* — Resource Description Framework

Einen völlig anderen Ansatz verfolgt der "**R**esource **D**escription **F**ramework", welcher vom *W3C* favorisiert und auch standardisiert (werden) wird. *RDF* baut auf *XML* auf und kann mit *RDFSchema* ähnlich *XMLSchema* um eigene Vokabularien erweitert werden.

### 2.3.1 Ein *RDF*-Beispiel

*RDF* ist eigentlich aber nur ein Datenmodell, dessen bevorzugte Repräsentation eben mit Hilfe von *XML* geschieht.

Ein *RDF*-Statement besteht dabei immer aus einem Subjekt, einem Prädikat und einem Objekt, in Abbildung 10 graphisch dargestellt.

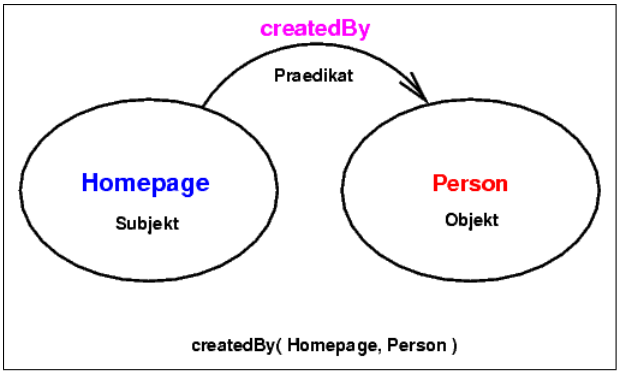


Abbildung 10: *RDF* Statement als Graph

Im Graphen ist dargestellt, daß eine Resource "Homepage" von einer Ressource "Person" erzeugt wurde. Diese Relation läßt sich auch als Prädikat schreiben.

*RDF* verlangt außerdem, daß jeder Teil eines Statements "Webenabled" ist. D.h. eigentlich müßte die Person eine URI haben. Der "Creator" einer Homepage müßte also Zeit seines Lebens mit einem Netzwerkkabel an seinem Körper herumlaufen um das obige Statement nicht als falsch abzutun. Die Lösung ist jedoch einfach: Man erzeugt eine sog. "anonyme Resource" und weißt ihr eine "`URI`"[1] zu. (So kann der "Creator" doch noch dem Fluch einer permanenten Verkabelung entkommen...)

Man erkennt auch leicht, daß *RDF*-Statements in dieser Darstellung einen benannten[2], gerichteten Graphen darstellen. Zyklen sind nicht verboten, wahrscheinlich aber selten erwünscht.

Eine (bevorzugte) Darstellung mittels *XML* sieht dann wie folgt aus:

```
<?xml version='1.0' encoding='UTF-8'?>
<rdf:RDF xmlns:rdf='http://www.w3c.org/rdf-syntax-ns#'
         xmlns:dc='http://purl.org/elements/1.0/'
         xmlns:NS0='http://www.w3c.org/...' >
```

[1] *U*niform *R*essource *I*dentifier.

[2] d.h. daß sowohl Knoten als auch Kanten mit Namen versehen sein müssen

```
 <rdf:Description rdf:about='http://www.koch.de'>
  <dc:createdBy rdf:resource='#A0'/>
 </rdf:Description>
 <rdf:Description rdf:about='#A0'>
  <NS0:hasName>Peter Koch</NS0:hasName>
  <NS0:hasPhone>0711 - 59742-0</NS0:hasPhone>
 </rdf:Description>
</rdf:RDF>
```

In diesem Beispiel beschreiben die Tags `rdf:Description` sowohl das Subjekt als auch das Objekt, beide werden aus dem *XML*-Namensraum "`http://www.w3c.org/rdf-syntax-ns#`" importiert. Das jeweilige Attribut `rdf:about` bezeichnet dabei die Ressource, die beschrieben wird. Der Wert dieses Attributes kann dabei entweder eine *URL* sein oder eine sog. anonyme Ressource, hier als "`#A0`" bezeichnet. Der Unterschied liegt wie oben erläutert dabei in der "Erreichbarkeit" der Ressource[3].

In diesem Beispiel ist die Person "Peter Koch" also eher ein abstraktes Konzept und bekommt daher eine "vorübergehende" Web-Erreichbarkeit in Form einer anonymen Ressource.

Die Prädikate "`createdBy`", "`hasName`" und "`hasPhone`" werden aus den jeweiligen anderen Namensräumen importiert. Der Prefix erlaubt dabei Eindeutigkeit auch wenn derselbe Name jeweils mehrfach in den Namensräumen definiert werden würde.

Eigene "*RDF*-Vokabularien" können durch *RDFSchema* erzeugt werden, analog zu *XMLSchema*. In den meisten Fällen wird man jedoch bereits existierende *RDF*-Dateien lediglich mit den benötigten Informationen anreichern.

### 2.3.2 Vor- und Nachteile von *RDF*

*RDF* ist in seiner Konzeption fast kanonisch. Ein eigener Ansatz würde einen wohl sehr nahe an *RDF* zurückbringen.

Ein Vorteil von *RDF* gegenüber *SHOE* ist, daß *RDF* beliebige Ressourcen beschreiben kann und nicht an HTML gebunden ist. Die Trennung zwischen der Ressource und seiner Beschreibung ist viel stärker und die etwas künstliche Trennung von Ontologien und Instanzen wie bei *SHOE* ist hier nicht aufgezwungen, kann aber trotzdem erfolgen.

Eigene Ontologie-Vokabularien lassen sich mit *RDFSchema* erzeugen und benutzen ähnlich *XMLSchema* und werden im nächsten Kapitel gezeigt.

Ein großer Nachteil ist jedoch, daß *RDF* nicht dieselben Ausdrucksmöglichkeiten wie *SHOE* besitzt. So sind in *RDF* nur binäre Relationen möglich und Logik-Elemente wie z.B. die Inferenzen bei *SHOE* fehlen völlig.

Beides muß aber nicht notwendigerweise als Nachteil angesehen werden, da *RDF* sich zuerst nur die Beschreibung von Ressourcen als Ziel setzt. Die daraufaufbauende Logik kann dann mit anderen Mitteln beschrieben werden. Auch mehr-stellige Relationen könnte man mittels mehreren binären Relationen "Schön-Finkeln", obwohl dies natürlich erstmal ein kosmetisches Defizit darstellt.

[3] Der "Erzeuger" einer Webpage ist hier auch vielmehr ein abstraktes Konzept denn als eine physische Entität zu verstehen.

## 2.4 *RDFSchema* — *RDF* erweitern

Mit *RDF* kann man also beliebige Ressourcen beschreiben. In den meisten Fällen wird man in sog. "Ontologie-Repositorien" Klassen finden, die man dann im gegebenen Fall benutzen kann. Manchmal aber, braucht man doch eigene Vokabularien. Diese werden in *RDFSchema* geschrieben.

*RDFSchema* ist *XMLSchema* sehr ähnlich, beschäftigt sich aber weniger mit der Syntax der Tags; das ist schließlich Aufgabe von *XMLSchema*. *RDFSchema* erlaubt es vielmehr neue Klassen und Eigenschaften zu definieren und damit eine Beziehung zwischen Klassen und ihren Eigenschaften zu modellieren.

Die Idee von *RDFSchema* ist objektorientiert: Es gibt Klassen und Unterklassen, sowie Eigenschaften und Untereigenschaften. Dies ermöglicht den Einsatz von Objektorientierten Analyse- und Design-Methoden um Ontologien zu entwerfen und zu pflegen. Dies senkt u.U. die Kosten für beide Phasen.

### 2.4.1 Ein Beispiel in *RDFSchema*

Im folgenden Beispiel soll eine Klasse `Herbivore` modelliert werden, die Pflanzenfresser darstellt. Ferner wird eine Eigenschaft definiert, die den Namen eines Tieres mit diesem assoziiert.

```
<rdfs:Class rdf:ID='Herbivore'>
 <rdfs:type rdf:resource='http://www.w3c.org/rdf-schema/#DefinedClass'/>
 <rdfs:subClassOf rdf:resource='#Animal'/>
</rdfs:Class>
...
<rdfs:Property rdf:ID='hasName'>
 <rdfs:label>The Name of an animal</rdfs:label>
 <rdfs:domain rdf:resource='#Animal'/>
 <rdfs:range rdf:resource='http://www.w3c.org/rdf-schema#Literal'/>
</rdfs:Property>
```

In der Klassendefinition mag auf den ersten Blick der Einsatz sowohl eines "`type`s" als auch einer Unterklassenrelation (`subClassOf`) verwirren. Jedoch ist es wichtig hier zwischen dem Typ der Klasse und dem Typ einer Instanz der Klasse zu unterscheiden. Der Typ der Klasse ist "`...#DefinedClass`", eine Instanz der Klasse `Herbivore` ist allerdings eine Unterklasse der Klasse `Animal`. Dieser Unterschied ist wichtig!

Die Eigenschaft (`Property`) definiert sowohl eine Definitions- als auch eine Wertemenge (im Beispiel `rdfs:domain` bzw. `rdfs:range` genannt) und macht damit deutlich, daß eine Eigenschaft eigentlich als eine binäre Abbildung aufgefaßt werden kann. Hier ist die Definitionsmenge eingeschränkt auf eine Instanz der Klasse `Animal`, während die Wertemenge nur ein Literal sein darf. Damit wird die Möglichkeit verhindert, daß das Objekt des `hasName`-Prädikats wieder reiifiziert werden kann. (Es sei dahingestellt, ob dies hier Sinn macht oder nicht!)

Benutzt wird die neue Klasse dann ganz einfach:

...

```
<Herbivore rdf:about='http://www.farm.de/Cow/Sieglinde'>
 <hasName>Sieglinde</hasName>
</Herbivore>
...
```

Die Ressource `http://www.farm.de/Cow/Sieglinde` ist also ein Pflanzenfresser und hat den Namen "Sieglinde".

Es sei noch erwähnt, daß analog zu Klassen- und Unterklassen-Beziehungen, *RDFSchema* auch die Möglichkeiten von Eigenschaften- und Untereigenschaften-Beziehungen existieren und analog funktionieren. Bei Untereigenschaften kann dann natürlich die Angabe der `domain` bzw. `range` entfallen, wenn die jeweiligen Werte der "Elterneigenschaft" verwendet werden sollen.

### 2.4.2 Vor- und Nachteile von *RDFSchema*

*RDFSchema* erlaubt einem Benutzer eigenen Ontologie-Vokabularien zu entwerfen und damit seine Umwelt präziser zu modellieren, als dies mit bei Ontologie-Repositorien verfügbaren Vokabularien vielleicht möglich ist. Auch kann ein Benutzer bestehende Vokabularien um eigene Eigenschaften oder Beziehungen erweitern. Beides vereinfacht möglicherweise die Erzeugung von Ontologien in einem konkreten Fall.

Andererseits ist die Ausdruckskraft von *RDFSchema* sehr beschränkt. So können nur binäre Relationen modelliert werden und diese Relationen können auch keine "höheren" Eigenschaften haben, wie z.B. Transitivität oder Symmetrie.

Der Objektorientierte Ansatz ermöglicht allerdings, wie schon erwähnt, den Einsatz von OOA und OOD Methoden wie sie beispielsweise in der Software-Technik Anwendung finden. Dies ist ein gewichtiges Argument, denn es darf nicht außer Acht gelassen werden, daß der Entwurf und, vor allen Dingen, die spätere Pflege von Ontologien mitunter sehr aufwendig und damit teuer ist. Jede Möglichkeit hier Kosten zu sparen, ist damit ein nicht unerheblicher Faktor!

Unterm Strich bleibt aber die dann doch schwache Ausdruckskraft von *RDFSchema*, ein Defizit, daß nur durch Sprachen wie *OIL* oder *DAML* ausgeglichen werden kann.

## 2.5 *OIL* — Logik für *RDF*

Mit *RDF* kann man im Wesentlichen nur Ressourcen beschreiben, für die Logik braucht man andere Hilfsmittel. Eines dieser Hilfsmittel ist *OIL*.

*OIL* ist primzipiell in Schichten gegliedert wie in Abbildung 11 dargestellt, deren Unterschiede in den Beschreibungsmöglichkeiten liegen. Während *Core OIL* praktisch nur aus *RDF*, *RDFSchema* und einigen wenigen Logik-Elementen besteht, enthält *Instance OIL* auch Datenbank-Elemente. *Heavy OIL* ist noch Zukunftsmusik und soll natürlich noch mehr können als die beiden unteren Schichten.

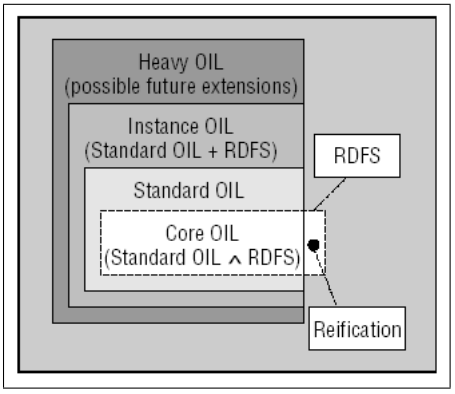


Abbildung 11: *OIL* Schichten

### 2.5.1 Ein *OIL*-Beispiel

Wie sieht eine Ontologie nun konkret in *OIL* aus? Ein Beispiel aus der Biologie soll hier helfen; in der Biologie klassifiziert, unter anderem jedes Lebewesen, als entweder einen Pflanzenfresser oder einen Fleischfresser.

Im folgenden Beispiel ist ein Pflanzenfresser modelliert:

```
<rdfs:Class rdf:ID='Herbivore'>
 <rdfs:type
rdf:resource="http://www.ontoknowledge.org/oil/RDFS-schema/#DefinedClass"/>

 <rdfs:subClassOf rdf:resource="#Animal"/>
 <rdfs:subClassOf>
   <oil:NOT>
    <oil:hasOperand rdf:resouce="#Carnivore"/>
   </oil:NOT>
 </rdfs:subClassOf>
</rdfs:Class>
```

Im Beispiel wird eine Klasse "`herbivore`" als Unterklasse von "`animal`" definiert, ganz genauso wie auch beim Beispiel zu *RDFSchema*. Hier kommt aber noch ein *OIL*-Element hinzu, das diese Klasse auch noch als disjunkt zur Klasse "`carnivore`" kennzeichnet.

Man sieht auch, daß die Klasse mit *RDFSchema* modelliert wird. Der Grund dafür ist, daß *OIL* es damit Applikationen, die nur *RDFSchema* "kennen", ermöglicht, die Modellierung der Klassen "`herbivore`" und "`carnivore`" zu "verstehen", auch wenn die Disjunktheit der beiden dabei ignoriert wird. Damit geht zwar Information verloren, aber, da die Applikation *OIL* sowieso nicht versteht, muß dies kein (großes) Problem darstellen.

Während diese Darstellung für Menschen gerade noch lesbar ist, wird ein Programm keine Probleme damit haben; ein *XML*-Parser aus dem Regal, der *XML-Schema*-fähig ist, genügt schon.

Für Menschen ist die folgende Darstellung aber wohl leichter lesbar:

```
...
class-def defined carnivore
  subclass-of animal
  slot-constraint eats value-type animal

class-def defined herbivore
  subclass-of animal
  slot-constraint eats
    value-type plant
    OR
    (slot-constraint is-part-of has-value plant)

disjoint carnivore herbivore
...
```

Auch hier werden sowohl zwei Klassen "`herbivore`" und "`carnivore`", als auch deren Disjunktheit definiert. Beide Klassen sind Unterklassen der Klasse "`animal`".

Das Attribut "`defined`" besagt zusätzlich, daß die Eigenschaften der Klasse nicht nur notwendig, sondern auch hinreichend sind, um ein Objekt als zur Klasse zugehörig zu befinden. Ohne dieses Attribut ist eine *OIL*-Klasse praktisch identisch zu einer *RDFSchema*-Klasse.

Ferner definiert "*eats*" eine Relation zwischen den Instanzen dieser Klasse und den Instanzen einer anderen Klasse, wobei der Wert ("value", die rechte Seite) der Relation darauf eingeschränkt wird, daß es im Falle eines `carnivore` nur ein `animal` sein darf. D.h. ein Fleischfresser frißt nur andere Tiere, keine Pflanzen (oder Menschen!).

Im Falle der Klasse "`herbivore`" ist der Wert der Eigenschaft eingeschränkt auf Pflanzen oder Teile einer Pflanze. Zum Beispiel ist der Stamm eines Baumes meistens nicht essbar, die Blätter an den Zweigen, die vom Stamm aus wachsen aber schon.

*OIL* ermöglicht es also, Relationen und Beziehungen zwischen Objekten zu beschreiben. *OIL* ermöglicht es ferner Relationen inverse, symmetrische, funktionale und/oder transitive Eigenschaften zu geben. Ob es zudem Reflexivität, Irreflexivität, Anti-Symmetrie, Asymmetrie u.ä. Eigenschaften geben soll, ist aber noch umstritten.

### 2.5.2 Vor- und Nachteile von *OIL*

*OIL* erweitert *RDFSchema* offensichtlich nicht nur um Logik-Elemente, sondern auch um eine menschenlesbare Darstellung von Ontologien. Dies ist von großem Wert, da es sehr aufwendig und teuer ist, größere Ontologien zu entwerfen und zu pflegen, insbesondere wenn dort komplexere Konzepte verwendet oder modelliert werden.

Auch die Möglichkeit, daß bestehende Applikationen die *OIL*-Erweiterungen erstmal ignorieren und trotzdem die *RDF*- und *RDFSchema*-Strukturen lesen und bearbeiten können ist von großem Nutzen.

Ein Problem von *OIL* mag aber sein, daß es in erster Linie aus dem Universitätsbereich kommt und dort auch weiter gepflegt wird, doch wie lange noch? Daher wird *OIL* wahrscheinlich in *DAML* aufgehen, dessen Entwicklung momentan von einer großen staatlichen Organisation (**DARPA**) finanziert wird und das viele große Namen vom *W3C* involviert. Daher wird *DAML* wohl früher oder später als *W3C*-Standard gelten, etwas was *OIL* wohl verwehrt bleiben wird.

## 2.6 *DAML* — Das Beste für's *Semantic Web*?

*DAML* ist vielleicht die mächtigste Sprache im Semantic Web (oder soll es zumindest einmal werden). Dies ist nicht verwunderlich, denn das wohlbekannte **DARPA**, welches uns u.a. das Internet bescherte, finanziert die Forschung und Entwicklung von *DAML* und einer der Beteiligten ist Tim Berners-Lee, der Erfinder des Webs schlechthin, neben vielen anderen bekannten Namen vom *W3C*. Daher wird sich das *W3C* letztenendes wohl auch *DAML* annehmen und standardisieren.

*DAML* ist noch sehr jung: Erst 2001 erblickte es das Licht der Welt. Trotzdem ist es schon weitgediehen, denn *DAML* baut auf den Erfahrungen mit *SHOE* und *OIL* auf und benutzt (natürlich) *RDF* und *RDFSchema*.

### 2.6.1 Ein *DAML*-Beispiel

Ein kleines Beispiel soll die Möglichkeiten von *DAML* näher bringen:

```
...
<daml:Class rdf:ID='Animal'>
 <rdfs:label>Animal</rdfs:label>
 ...
</daml:Class>

<daml:Class rdf:ID='Herbivore'>
 <rdfs:subClassOf rdf:resource='#Animal'/>
 <daml:disjointWith rdf:resource='#Carnivore'/>
</daml:Class>

<daml:Class rdf:ID='Carnivore'>
 <rdfs:subClassOf rdf:resource='#Animal'/>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource='#eats'/>
   <daml:toClass rdf:resource='#Herbivore'/>
  </daml:Restriction>
 </rdfs:subClassOf>
</daml:Class>
...
```

Auch in diesem Beispiel werden zwei Klassen definiert, die (wieder) Fleisch- und Pflanzenfresser darstellen sollen. Am Anfang des Beispiels wird eine Klasse `Animal` definiert; die Tags haben zwar den Prefix `daml`, sind im wesentlichen aber identisch zu *RDFSchema*. *DAML* erlaubt an dieser Stelle noch einen Kommentar (`comment`) zu schreiben, der die Klasse in menschenlesbarer Form beschreibt. (Dies könnte notfalls aber auch noch in *RDFSchema* gemacht werden...)

Interessanter ist dann aber die Klasse der `Herbivore`n: Hier kann man tatsächlich ein Tag `disjointWith` erkennen, daß diese Klasse als disjunkt zur Klasse der `Carnivore`n definiert. Ein Feature, das so nicht in *RDFSchema* möglich ist.

In der zweiten Klasse wird dann eine Restriktion für eine Eigenschaft (d.h. Relation) eingeführt: Sie besagt, daß die Relation `eats` als Wert (d.h. `toClass` im

Gegensatz zu `fromClass`) nur eine Instanz der Klasse `Herbivore` haben darf. Dies entspricht der (zugegeben etwas künstlichen) Einschränkung, daß ein Fleischfresser nur Pflanzenfresser frißt.

### 2.6.2 Vor- und Nachteile von *DAML*

Ein großer Vorteil von *DAML* ist seine Ausdruckskraft: *DAML* ermöglicht es nicht nur Klassen- und Unterklassen-Beziehungen sehr präzise zu modellieren, sondern auch Eigenschaften und Syntax der Klassen genau zu spezifizieren.

Auch wenn es im Beispiel nur zu erahnen ist, so baut *DAML* massiv auf *RDF* und *RDFSchema* auf. Damit erlaubt *DAML* es, Applikationen verhältnismäßig einfach um *DAML*-Fähigkeiten zu erweitern.

Ein Nachteil von *DAML* gegenüber *OIL* ist, daß *DAML RDFSchema* um eigene Tags erweitert. Eine Applikation, die nicht *DAML* "kennt", kann mit den Klassenbeschreibungen auch nichts anfangen, im Gegensatz zu *OIL*.

# 3 *Semantic Web* — Warum überhaupt?

Zum Schluß bleibt aber immer die Frage: Braucht man das überhaupt? Es ist klar, daß das *World Wide Web* in seinem jetzigen Zustand große Probleme hat, die es nicht lösen kann. Und die Probleme werden von Jahr zu Jahr allein durch die schiere Zahl an Information, immer größer!

Das *World Wide Web* kann aber nicht einfach weggeworfen und durch etwas komplett Neues ersetzt werden, die "Trägheitsmasse" ist dafür zu enorm. Vielmehr muß ein *World Wide Web* der nächsten Generation auf dem bestehenden Web aufbauen und versuchen auf dieser Meta-Ebene die Probleme in den Griff zu bekommen. *Semantic Web* kann dies schaffen, aber nur, wenn es ähnlich wie das *World Wide Web* in den 90er Jahren genügend Masse aufbauen kann[4], d.h. es müssen ausreichend viele Leute *Semantic Web* "machen" damit es ein Erfolg werden kann.

Doch wer wird es "machen"? Das *Semantic Web* läßt sich in grob in zwei Seiten und zwei Benutzergruppen gliedern: Die Clients und die Server, sowie Endbenutzer und Firmen. Sie unterscheiden sich in der Art wie sie Meta-Informationen verarbeiten wollen, sowie in den finanziellen und personellen Möglichkeiten dies zu tun.

## 3.1 *Semantic Web* für Endbenutzer

Der klassische Endbenutzer, der auch dem Web in seiner jetzigen Form zum Durchbruch geholfen hat, wird in erster Linie Client von *Semantic Web* sein. Als "Server" bietet sich ihm meistens nur die Möglichkeit seine eigene Homepage mit semantischen Informationen anzureichern und wird über Standardbeispiele wie "Diese Homepage wurde von Peter Koch erstellt" und "Diese Homepage wurde am 3.Juli 2002 erstellt" wohl kaum hinausgehen.

Der typische Endbenutzer wird also wohl eher nur daran interessiert sein, Informationen zu finden. Diese Informationen wird er aber nur dann finden, wenn es Suchmaschinen gibt, die das Web nach semantischen Informationen durchsuchen. Seine Suche wird sich dann allerdings nicht mehr auf einfache Stichworte beschränken können, sondern der Benutzer muß semantische und logische Zusammenhänge angeben oder zumindest verstehen können. Und nur dann wird ihm eine Suche mit der "neuen" Suchmaschine effektiv mehr bringen. Zwar kann eine Suchmaschine diesen Prozess in gewissen Umfang unterstützen, dennoch wird viel "Denkarbeit" beim Benutzer verbleiben. Ob der durchschnittliche Endbenutzer dafür überhaupt die Motivation aufbringen wird, ist allerdings fraglich.

Es ist denkbar, daß sich dies bessert, wenn sich die Agenten-Technologie mehr durchsetzt und es damit ermöglicht von PDAs und Handy aus von Unterwegs aus das *Semantic Web* zu nutzen. Dies steht und fällt allerdings mit der Verfügbarkeit eines bezahlbaren und zuverlässigen mobilen Zugangs zum Internet. Die bisherigen Technologien sind dafür unbrauchbar oder unbezahlbar (im Verhältnis zum Ergebnis) und ob hier zukünftige Technologien eine Trendwende einleuten werden, sei dahingestellt.

[4] *Semantic Web* wird die "Explosion" des WWWs aber kaum wiederholen können, dafür ist es nicht "visuell" genug.

In P2P-Netzwerken könnte *Semantic Web* die Suche erheblich vereinfachen, doch P2P-Netzwerke haben nicht selten ohnehin ein eigenes Datenformat, bei dem die Suche durch zusätzliche Stichworte z.B. in einem Kommentarfeld verbessert werden kann. Diese einfache (und unschöne) Lösung bietet den geringsten Widerstand gegenüber einer Lösung mit *RDF*, *RDFSchema* und *DAML*, die dann doch viel Mehraufwand mit sich bringt.

*Semantic Web* wird für den durchschnittlichen Endbenutzer also wenig oder gar nichts bringen. Der Aufwand ist größer als der Nutzen.

(Es ist vielleicht lustig anzumerken, daß der durchschnittliche Endbenutzer eigentlich schon über ein *Semantic Web* verfügt: Freunde, Bekannte und Kollegen tauschen rege Links und Informationen aus, sowohl verbal als auch per eMail oder SMS. Dieses "*Semantic Web*" ist in seiner Effizienz leider kaum zu schlagen.)

## 3.2 *Semantic Web* für Firmen

Auf der anderen Seite befinden sich Firmen, die es als einen (Werbe-)Vorteil ansehen könnten, wenn sie ihre Produkt- oder Katalog-Seiten mit semantischen Informationen anreichern und damit leichter zu finden wären. Damit hätte eine Firma also auch einen wirtschaftlichen Vorsprung vor seinen Konkurrenten. Andererseits wird sich *Semantic Web* natürlich nur dann durchsetzen können, wenn es eben auch die Konkurrenz macht. Damit geht der Vorteil aber wieder verloren und wird eigentlich zum Nachteil, da ein Benutzer nun leichter Preise und Produkte vergleichen kann.

Ferner ist es so, daß die meisten Strukturen im Internet in diesem Bereich bereits gefestigt sind, die Kunden kennen ihre Anbieter und Lieferanten und sind (gute Erfahrungen vorausgesetzt) selten bereit diese zu wechseln.

Zudem können die Kunden einer Firma ihre Produkte über die Portalseite der Firma genauso leicht finden, wie mit *Semantic Web*, notfalls mit einer "darunterliegenden" proprietären Lösung. Hier wäre also *Semantic Web* nur als Alternative zu sehen und der Benutzer im Web würde so oder so nichts davon zu sehen kriegen.

Für größere Firmen, die vielleicht weltweit agieren, ist es vielleicht interessant, daß das Firmenwissen im Intranet der Firma leichter zu finden und zu nutzen wäre, als mit den bisherigen Strukturen. Aber auch hier, wäre *Semantic Web* nur eine Alternative und wäre vor allen Dingen nicht von "außen", d.h. Nicht-Firmen-Angehörige, nutzbar. Damit würde diese Lösung nicht zur "kritischen Masse" beitragen, die nötig wäre, damit sich *Semantic Web* in der derzeit anvisierten Form durchsetzt.

Bei großen Firmen wird *Semantic Web* daher in erster Linie wohl "unter der Oberfläche" seinen Dienst verrichten, denn hier sind auch genügend Ressourcen vorhanden, sowohl finanziell als auch personell. Die großen Firmen können es sich leisten ihre Angestellten in Richtung *Semantic Web* auszubilden. Dies muß auch passieren, denn selbst die schönsten Ontologien müssen erstmal einmal geschrieben und später auch gewartet werden. Diese Kosten können sich eigentlich nur größere Firmen leisten, weil sie auch am schnellsten diese Investitionen wieder zurückbekommen würden.

## 3.3 Fazit: Ja! oder doch nicht?

Das Fazit ist also, daß diejenigen, die dem Web ursprünglich zur Revolution verholfen haben (eben der kleine Endbenutzer von nebenan), eigentlich am wenigsten vom *Semantic Web* haben. Würde jeder seine Webseite mit semantischen Informationen anreichern, dann wären Informationen zu bestimmten Themen tatsächlich viel leichter zu finden. Ein gewaltiger Vorteil gegenüber dem undurchdringlichen Durcheinander an Informationen, das sich einem Benutzer heutzutage den wirklich gesuchten Informationen in den Weg stellt.

Andererseits, was hält einen "bösen" Benutzer davon ab, auf seiner Homepage die gerade "hippen" semantischen Informationen anzubieten, obwohl er letztendlich dann doch nur seinen neuesten Seifenersatz verkaufen will. Eine Suche nach "*Semantic Web*" würde dann vielleicht auf diese Homepage zutreffen, doch die tatsächlichen Informationen haben damit gar nichts zu tun.

Das *eBusiness* könnte wahrscheinlich am meisten von *Semantic Web* profitieren: Der Datenaustausch zwischen Businesspartnern erfolgt heutzutage sowieso mittels *XML* und die Partner müßten sich vorher auf die Bedeutung der jeweiligen Tags einigen oder sich diese zumindest mitteilen. Dieser Vorgang würde sich mit *Semantic Web* vereinfachen oder sogar komplett wegfallen, denn hier könnte dann auf Ontologien zurückgegriffen werden.

Unter dem Strich bleibt der Eindruck, daß das *Semantic Web* wohl seinen größten Erfolg unter der "Oberfläche" erleben wird und vielleicht noch nicht einmal bis zu einem Endbenutzer durchsickern wird. Die Visionen mancher Autoren werden wohl auf jeden Fall noch etwas auf sich warten lassen und manche werden bestimmt nie Realität werden.

Setzen sich die Techniken des *Semantic Web* durch, dann wird dies auf jeden Fall für ein angenehmeres Web sorgen und vielleicht die nächste Generation des *World Wide Web* einleuten. Bis dahin...

*Surf long and prosper...*

# 4 Glossar

**Ontologie** Das *Oxford English Dictionary* definiert eine "Ontology" als "the science or study of being". Im Bereich der künstlichen Intelligenz wird der Begriff als "the specification of a conceptualization" definiert. Manche Autoren verstehen darunter aber die Menge an "Wissenstermen", das dazugehörige Vokabular, die semantischen Verbindungen und einige einfache Inferenz- und Logik-Regeln.

***SHOE*** steht für **S**imple **H**TML **O**ntology **E**xtensions und ist eine Ontologie-basierte Sprache zur Repräsentation von Wissen, eingebettet in HTML-Dokumente.

***XML*** ist die Abkürzung für e**X**tensible **M**arkup **L**anguage. *XML* kann nur zur Beschreibung von Syntax benutzt werden, für semantische Informationen ist *XML* nicht mächtig genug.

***RDF*** ist die Abkürzung für **R**esource **D**escription **F**ramework und ist eine "*General Purpose Language for representing information in the World Wide Web*" [MM].

***RDFSchema*** ist die Abkürzung für *RDF***S**chema. *RDFSchema* erlaubt es *RDF* mit eigenen Ontologie-Vokabularien auszurüsten oder bestehende Vokabularien zu erweitern.

***OIL*** steht entweder für **O**ntology **I**nference **L**anguage oder für **O**ntology **I**nference **L**ayer, ganz nach Geschmack. *OIL* ist eine Sprache um Ontologien und Beziehungen zwischen Ontologien zu beschreiben und erweitert *RDF* und *RDF-Schema* um Logik.

***DAML*** ist die **D**ARPA **A**gent **M**arkup **L**anguage. *DAML* ist eine Initiative der DARPA-Behörde um *RDF* und *RDFSchema* um Logik zu erweitern und gilt als Nachfolger von *OIL*.

# Literatur


[BL99] Tim Berners-Lee. *Weaving the Web*. Orion Business Books, London, 1999.

[Cha] Pierre-Antoine Champin. An RDF Tutorial, 5.4.2001. http://www710.univ-lyon1.fr/~champin/rdf-tutorial/.

[DF01] Mark A. Musen Dieter Fensel. The Semantic Web: A Brain for Humankind. *IEEE Intelligent Systems*, 16(2):24—25, 2001.

[D.H01] D.Hensel, I.Horrocks, F.Van Harmelen, S.Decker, M.Erdmann, and M.Klein. OIL in a nutshell, 2001.

[Die01] Dieter Fensel, Frank von Harmelen, Ian Horrocks, Deborah L. McGuiness, Peter F.Patel-Schneider. OIL: An ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38—45, 2001.

[Hef01] Jeffrey Douglas Heflin. Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment. *Dissertation submitted to the Faculty of the Graduate School of the University of Maryland, College Park*, 2001.

[Hen01] James Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30—37, 2001. http://www.cs.umd.edu/~hendler/AgentWeb.html.

[Jef01] Jeff Heflin and James Hendler. A portrait of the Semantic Web in action. *IEEE Intelligent Systems*, 16(2):54—59, 2001.

[JH00] Sean Luke Jeff Heflin, James Hendler. SHOE: A prototype language for the Semantic Web. Technical report, Department of Computer Science, University of Maryland, 2000.

[JH01] James Hendler Jeff Heflin. Semantic Interoperability on the Web. Technical report, University of Maryland, 2001.

[Kle01] Michel Klein. XML, RDF, and Relatives. *IEEE Intelligent Systems*, 16(2):26—28, 2001.

[MM] Frank Manola and Eric Miller. An RDF primer. http://www.w3.org/TR/2002/WD-rdf-primer-20020319/.

[ROa] Uche Ogbuji Roxane Ouellet. Introduction to DAML: Part 1. http://www.xml.com/lpt/a/2002/01/30/daml.html.

[ROb] Uche Ogbuji Roxane Ouellet. Introduction to DAML: Part 2. http://www.xml.com/lpt/a/2002/03/13/daml.html.

[She01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46—53, 2001.

[TBLL] James Hendler Tim Berners-Lee and Ora Lassila. The Semantic Web. http://www.sciam.com/2001/0501issue/0501berners-lee.html.

[WC00] Reinhold Klapsing Wolfram Conen. A Logical Interpretation of RDF. Technical report, Linköping Electronic Articles in Computer and Information Science, 2000.