Seminar
Semantic Grid
Wintersemester 2004/2005

Universität Koblenz

Seminarleiter: Bernhard Tausch

Ontologiesprachen

Daniel Akkaya

Inhaltsverzeichnis

In dieser Arbeit werde ich zunächst darlegen, was unter einer Ontologie zu verstehen ist. Darauf folgend werde ich Standards, die man bei der Erstellung von Ontologien benutzt, vorstellen. Diese wären das Resource Description Framework (RDF), RDF Schema (RDFS), die DARPA Agent Markup Language + Ontology Inference Layer (DAML+OIL), Web Ontology Language (OWL) und schließlich Frame-Logic.

1. Ontologie

Vom griechischen leitet sich der Begriff Ontologie onta (das Seiende) und logos (Lehre) her und bedeutet „die Lehre vom Seienden“. Die Ontologie ist ein Teilgebiet der Metaphysik, welche sich mit der Seinsstruktur der Wirklichkeit und den Gründen des Seins beschäftigt. Die Ontologie beschäftigt sich mit Fragen nach der Natur des Seins (nature of existence) und der Struktur der Realität.
Die Geschichte der Metaphysik lässt sich bis in die Antike zu Aristoteles oder Platon zurückverfolgen. Später haben sich Philosophen wie Descartes, Kant, Leibniz und viele andere mit diesem Thema beschäftigt. Es gibt dementsprechend viele voneinander abweichende Ansichten dazu, auf die ich hier aber nicht näher eingehen möchte, da dies den Rahmen der Arbeit sprengen würde.
Was in der Informatik eine Ontologie ist, was deren Aufgabe und Ziele sind, soll anhand von Zitaten aus einschlägiger Literatur nachvollzogen werden.
Im Buch [Fensel, Dieter, Ontologies:A Silver Bull for knowledge Management and Electronic Commerce, Springer-Verlag,2001] von Dieter Fensel wird auf Seite 11 als treffendste Beschreibung folgendes Zitat angegeben: „An Ontology is a formal, explicit specification of a shared conceptualisation“. Drei Aspekte einer Ontologie werden in diesem Satz zum Ausdruck gebracht:
-Eine Ontologie soll einen formalen Aufbau haben, um von Maschinen gelesen werden zu können.
-Das durch sie ausgedrückte Wissen ist explizit anzugeben.
-„Specification of a shared conceptualisation“ drückt aus, dass eine Ontologie mit Konzepten (Dingen) die Welt beschreiben soll. Das Attribut „shared“ weist auch deutlich aus, dass, zumindest in einem kleinen Kreis, Konsens über diese Konzepte herrschen soll. Mit anderen Worten, eine Ontologie soll in einer abstrakten Weise das akzeptierte Wissen unserer Welt wiedergeben.
Auch folgendes Zitat von Dieter Fensel beschreibt das Wesen einer Ontologie: „[Ontology is] a shared and common understanding of some domain that can be communicatd between people and application systems“ [1, S. 11].
Zusammengefasst kann gesagt werden, dass eine Ontologie ein Konstrukt ist, um Wissen über ein (bestimmtes) Gebiet zu repräsentieren, auszutauschen und (wieder)zu verwenden, sowohl durch den Menschen, als auch durch den Computer.
Ontologien bilden deshalb auch die Grundlage von Wissensbasen für die Künstliche Intelligenz (KI). Sie wurden auch vornehmlich von Wissenschaftlern im Umfeld der Künstlichen Intelligenz entwickelt. Doch Ontologien haben schon längst den Bereich der typischen Künstlichen Intelligenz-Anwendungen, wie z.B. Knowledge Engineering, natürliche Sprachverarbeitung und Wissensrepräsentation verlassen und Gebiete, wie Electronic Commerce, intelligente Informationsintegration, Information Retrieval und Wissensmanagement erobert. Sie haben damit auch in der Wirtschaft große Bedeutung erlangt.
In einer Ontologie wird nicht nur beschrieben, was ein Konzept ist, sondern die Konzepte werden untereinander in Beziehung gesetzt. Es ist damit möglich, aus einer kleinen Menge an Informationen über einen bestimmten Sachverhalt implizit mehr Informationen abzuleiten.
Bevor wir die Bestandteile und den Aufbau einer typischen Ontologie klären werden, möchte ich erst noch ein Beispiel von Ontologien geben, nämlich WordNet. Der Leser soll erst einmal eine real existierende Ontologie kennen lernen.

WordNet ist ein lexikalisches Online-Wörterbuch für die englische Sprache mit über 100.000 Wort-Bedeutungen.
Es unterscheidet fünf Kategorien, nämlich Substantive, Adjektive, Verben, Adverbien, und Funktionswörter, in welche alle gespeicherten Wörter nach Konzepten, wie z.B. ihrer Bedeutung, eingeteilt. Zusätzlich werden Relationen zwischen Wörter beachtet wie z.B. die Synonym-Beziehung. Die Abfrage liefert die verschiedenen Bedeutungen des Wortes, abhängig von der zugehörigen Wortklasse. Zusätzlich kann z.B. von WordNet erfragt werden, welche Synonyme dazu existieren, abhängig vom Sinn des eingegebenen Wortes.
Z.B. Das Wort „dream“ kann also sowohl ein Substantiv, als auch ein Verb sein. Als Substantiv hat das Wort „dream“ sechs verschiedenen Bedeutungen, als Verb zwei. Jeder dieser Bedeutungen wird ausführlich erklärt. Außerdem wird ein Beispielsatz dazu angegeben.
WordNet wurde vom Cognitive Science Laboratory der Universtät Princeton entwickelt. Es kann kostenlos im Netz benutzt oder dort erhalten werden. Obwohl auch eine Schnittstelle für die logische Programmiersprache Prolog existiert, ist WordNet wegen der geringen Formalisierung weniger dazu geeignet, durch Computer verarbeitet zu werden, um auf neues Wissen zu schließen. Durch den geringen Grad an Formalisierung ist es aber sehr benutzerfreundlich gehalten und erfreut sich deshalb großer Beliebtheit.

2. Aufbau einer Ontologie

Um eine Ontologie durch einen Computer benutzen zu können, muss die Ontologie einen regelmäßigen, genau definierten Aufbau besitzen, mit dem sich alles ausdrücke lässt, was man mit ihr ausdrücken möchte. Egal wie die Ontologie intern repräsentiert wird, die Grundbausteine sind immer die gleichen: Konzepte und Relationen.

Konzepte

Es können zwei Arten von Konzepten unterschieden werden. Bei der ersten beschreibt das Konzept eine Klasse oder eine Menge von individuellen Objekten der Welt, die gewisse gemeinsame Eigenschaften aufweisen. Mit dem Konzept „Person“ sind z.B. alle Menschen, egal ob männlich oder weiblich, ob jung oder alt zusammengefasst. Gemeinsam ist, dass sie Menschen sind. Sollen zwischen männlichen oder weiblichen Personen unterschieden werden, kann man die Konzepte „Mann“ und „Frau“ einführen. Diese Art von Konzepten ist vergleichbar mit den Klassen in den objektorientierten Programmiersprachen.
Die zweite Art von Konzepten beschreibt nur ein ganz bestimmtes Objekt der Welt. So könnte z.B. in eine Ontologie das Konzept „Daniel Akkaya“ aufgenommen werden. Diese Art ist eher mit den Objekten in dem OO-Programmierparadigma vergleichbar.

Relationen

Doch alleine mit einer Ansammlung an Konzepten kann unsere Welt noch nicht beschrieben werden. Die Dinge stehen in der richtigen Welt untereinander in Beziehung. Die Beziehungen in einer Ontologie werden durch sogenannte Relationen dargestellt. Zwei Klassen an Beziehungen kann man unterscheiden:

1. Relationen, welche die Konzepte hierarchisch ordnen
2. Relationen, welche die Konzepte untereinander in beliebige Beziehung setzen

Relationen, welche die Konzepte hierarchisch ordnen

Diese Art von Relationen beschreibt, dass ein Konzept Teilmenge eines anderen Konzeptes ist. Dazu gehören:
- die isA-Beziehung
- die Äquivalenz-Beziehung

Die isA-Beziehung zwischen Konzept A und Konzept B drückt aus, dass das Konzept A als Spezialisierung von Konzept B definiert ist („A ist eine Spezialisierung von B“ bedeutet, Konzept A wurde noch detaillierter beschrieben als B).

Dass ein Hund ein (näher spezifiziertes) Tier ist, wird folgendermaßen ausgedrückt:
Hund isA Tier.
Wenn jedoch die beiden Konzepte A und B den gleichen Sachverhalt beschreiben, so kann das durch eine Äquivalenz-Beziehung ausgedrückt werden.

Relationen, welche die Konzepte untereinander in Beziehung setzen
In diese Kategorie fallen alle anderen Beziehungen, die zwischen zwei Konzepten auftreten können. Dies wären die
- Rollen
- die hasA-Beziehung
Gemeinsam ist diesen, dass sie nicht die Semantik aufweisen, dass ein Konzept eine Teilmenge des anderen ist.
Rollen definieren beliebige, nicht hierarchische Beziehung zwischen Konzepten. Beispiel dafür sind: „istKameradVon“ (Karl istKameradVon Uwe), „mag“ (Karl mag Ute) oder „fressen“ (Tiere fressen Pflanzen).
Die hasA-Beziehung stellt eine besonders wichtige und oft verwendete Rolle dar. A hasA B drückt aus, dass Konzept A Konzept B als Eigenschaft hat oder dass B ein Teil von A ist.
Sollen Beziehungen, wie z.B. ein Elternteil hat Kinder (welche Personen sind), ausgedrückt werden, so wird dies durch eine hasA-Beziehung modelliert. In unserem Beispiel würde die Relation „hatKind“ heißen und würde das Konzept „Elternteil“ mit dem Konzept „Person“ verbinden.

RDF liefert das Datenmodell und bildet die Grundlage von Modellierungssprachen für Ontologien. Ontologiesprachen (Ontology Languages), z. B. DAML+OIL oder OWL, sind also selbst in RDF Schema definiert und erweitern es um weitere Modellierungsmöglichkeiten.
Nun werde ich Modellierungssprachen kurz vorstellen, die man zur Konstruktion von Ontologien benutzen kann bzw. die für das Semantic Web von Bedeutung sind. Beginnen werde ich mit dem *Resource Description Framework* (RDF).

3. RDF

RDF ist eine Empfehlung des World Wide Web Consortium für die Modellierung von Metadaten zur Beschreibung von Ressourcen im Internet. Die Entwickler wollten beim Design von RDF u. a. Folgendes erreichen .
- Es sollte ein einfaches Datenmodell, das für Anwendungen einfach zu manipulieren und zu verarbeiten ist, geschaffen werden.
- Es gibt eine formale Spezifikation der Semantik.
- Das Vokabular ist erweiterbar und eindeutig identifizierbar (URIs).
- Die Austauschbarkeit unter verschiedenen Anwendern/Anwendungen ist durch die Möglichkeit RDF in XML zu repräsentieren gegeben.
Das Ziel von RDF ist es automatische Prozesse im Internet, wie z. B. Suche, durch den Aufbau einer Struktur aus Metadaten, zu verbessern und die Zusammenarbeit von Softwareagenten zu unterstützen.
Das grundlegende Datenmodell von RDF besteht aus den drei Objekttypen Ressource, Property und Statement.
Eine *Ressource* (Resource) kann theoretisch jedes Objekt sein; die einzige Bedingung ist, dass es durch einen *Unified Resource Identifier* (URI) eindeutig identifiziert ist. Da im Konzept der URIs Erweiterbarkeit vorgesehen ist, kann alles eine URI besitzen und somit auch eine RDF Ressource sein.
Eine *Property* ist eine Eigenschaft einer Ressource und wird benutzt, um diese näher zu beschreiben. Eine Property ist selbst auch eine Ressource, woraus folgt, dass sie wiederum durch weitere Properties beschrieben werden kann.

Die Grundstruktur von RDF besteht aus einem Tripel, dessen Bestandteile *Subjekt* (subject), *Prädikat* (predicate) und *Objekt* (object) heißen; dieses Konstrukt wird als *Statement* bezeichnet. Diese Grundstruktur bildet einen gerichteten Graphen, wobei das Subjekt und das Objekt als Knoten fungieren und das Prädikat als Kante mit Richtung vom Subjekt zum Objekt. Das Subjekt eines Statements ist eine Ressource und das Prädikat eine Property. Ein Objekt kann hierbei entweder eine Ressource oder ein *Literal*, also ein Text, oder auch das Subjekt eines anderen Statements sein. Das Objekt eines Statements wird auch als *Wert* (value) der entsprechenden Property bezeichnet. Subjekt, Prädikat und Objekt sind also die Bezeichnungen für die Objekttypen Ressource und Property, die in einem Statement eine bestimmte Rolle einnehmen. Mehrere Statements können zu einer komplexeren Struktur zusammengefasst werden.
Die Gesamtheit aller angelegten Statements heißt *Modell* (Model). Auch ein einzelnes Statement bildet schon ein RDF Modell.
Durch die Aggregation von Statements können komplexe Strukturen aufgebaut werden, die aber, auf Grund der Einfachheit der Grundstruktur Statement, gut verarbeitet werden können, wobei etablierte Techniken aus der Graphentheorie zum Einsatz kommen können. Außerdem wird so eine einfache Erweiterbarkeit eines RDF Modells gewährleistet.
RDF bietet verschiedene syntaktische Möglichkeiten solch ein Statement darzustellen, z. B.: in Tripel-Notation, als Graph oder in XML-Syntax. Die gewählte Darstellungsform hat dabei keinen Einfluss auf die Bedeutung der RDF Statements.
Als weitere Modellierungsmöglichkeiten bietet RDF die Möglichkeit drei verschiedene Arten von *Containern* anzulegen. Zur Auswahl stehen: eine *Bag*, das ist eine ungeordnete Liste von Ressourcen oder Literalen, eine *Sequence*, das ist eine geordnete Liste von Ressourcen oder Literalen, oder eine *Alternative*, das ist eine Liste von Ressourcen oder Literalen, die Alternativen für ein Objekt darstellen. Diese Container sind in der Kapazität nicht beschränkt und erlauben Duplikate.
Außerdem kann ein RDF Statement selbst wieder ein Objekt eines Statements sein, z. B. für eine Aussage der Art: Person A sagt, dass Person B der Autor von Buch C ist. Diese Modellierungsmöglichkeit
heißt *Reification*.

Weitere Modellierungsmöglichkeiten werden RDF durch RDF Schema hinzugefügt, auf das ich im Folgenden eingehen werde.

3. RDFS

RDF Schema (RDFS) ist eine Erweiterung von RDF, die dazu dient Vokabulare zu erstellen. Ein *Vokabular* ist eine Sammlung von vorgefertigten Ressourcen und Properties; man kann sich ein Vokabular auch als eine Art Wörterbuch vorstellen, das Ausdrücke, die in RDF Statements benutzt werden können, und deren Bedeutung und Beschränkungen für die Benutzung vorgibt. Ein Beispiel für ein solches Vokabular ist Dublin Core, eine Spezifikation für Metadaten, wie sie in einer Bibliothek benutzt
werden könnten. Zum Zweck der Erstellung solcher Vokabulare bietet RDFS noch weitere, über reines RDF hinausgehende, Modellierungsmöglichkeiten.
RDF Schema bietet die Möglichkeit Klassen zu definieren und Instanzen dieser Klassen anzulegen. Der Zusammenhang zwischen Klassen und Properties in RDF Schema hat eine gewisse Ähnlichkeit zum Typsystem in Objektorientierten Programmiersprachen, wie z. B.
Java. Der Unterschied ist, dass in einer Objektorientierten Programmiersprache die Klasse im Vordergrund steht und dieser Attribute (Analogon in RDF: Properties) mit Typen zugeordnet werden. Z. B. wird ein Klasse Buch definiert, die ein Attribut Autor vom Typ Person erhält. Im RDF Schema dagegen steht das Attribut bzw. die Property im Vordergrund. Hier wird eine Property genommen und es wird definiert auf welche Klassen sie angewendet werden darf (*domain*, vergleichbar dem mathematischen Definitionsbereich einer Abbildung) und

aus welchen Klassen die Werte der Property stammen dürfen (*range*, vergleichbar dem mathematischen Wertebereich). Für das Beispiel von oben würde man also für die Property Autor die domain Buch und die range Person definieren. Zudem kann man eine Klasse als Unterklasse (*subClassOf*) einer anderen Klasse definieren; dies ist eine Property. Eine Unterklasse hat alle Eigenschaften der Oberklasse und zusätzlich eigene Eigenschaften; hierdurch wird Vererbung realisiert. Eine Instanz ist eine konkrete Ausprägung von Klassen oder Properties, also z. B. der Name eines Autors und eines von ihm geschriebenen Buches wären Instanzen der Klassen Person bzw. Buch und bilden zusammen mit der Property Autor eine Instanz dieser.
Des Weiteren bietet RDF Schema eigene Klassen für die grundlegenden RDF Objekttypen, wie Resource, Literal, Class, Statement oder Container, und Properties, wie die schon erwähnten domain, range und subClassOf und einige Properties, die in erster Linie zur Dokumentation oder zum Verknüpfen von verschiedenen Vokabularen gedacht sind.
Durch Benutzen der XML-Namespaces ist es möglich Definitionen aus verschiedenen Vokabularen in einem RDF Modell zu benutzen aber gleichzeitig eindeutig festzulegen, welche Definition benutzt wurde. Dies ist nötig, falls es in verschiedenen Vokabularen Definitionen zu denselben Ressourcen gibt.
Der wesentliche Unterschied zwischen RDF und RDF Schema ist, dass in RDF Modellen die Ressourcen und Properties konkret festgelegt werden. Es gibt jedoch in RDF keine Möglichkeit Klassen oder Properties zu deklarieren, ohne eine konkrete Ausprägung anzulegen. Zu diesem Zweck wird RDF Schema benötigt. Das bedeutet, dass wenn man eine bestimmte Property in RDF modellieren möchte, diese auch in jedem Fall im RDF Modell existiert. In einem RDF Modell, das RDF Schema benutzt, ist es jedoch möglich, diese Property zu modellieren, ohne, dass sie sofort auch im Modell existiert. Dadurch ist aber die Möglichkeit gegeben später eine Ausprägung dieser Property im Modell zu erschaffen.

4. DAML + OIL

Zwar bietet RDF Schema mit seinen Konstrukten bereits einen Ansatz, um semantische Zusammenhänge auszudrücken, muss insgesamt jedoch als eine relativ primitive Sprache angesehen werden. Es erscheint daher notwendig und wünschenswert, eine größere Ausdrucksmächtigkeit bereitzustellen, um einen Wissensbereich mit der erforderlichen Genauigkeit modellieren zu können.
Aufbauend auf diesen Anforderungen wurde 1999 in den USA ein Programm namens „DARPA Agent Markup Language (DAML)“ gegründet, um grundlegende Technologien für ein Web der nächsten Generation zu erforschen. Die gleichnamige, in diesem Rahmen entwickelte Sprache basiert auf RDFS und ergänzt sie um leistungsfähigere Konstrukte zur Beschreibung von Ontologien. Zur gleichen Zeit wurde von einer europäischen Initiative mit vergleichbaren Zielen die Sprache „Ontology Inference Layer (OIL)“ entworfen, die grundsätzliche Gemeinsamkeiten mit DAML aufweist. Da man die gleichen Ziele verfolgte, beschlossen die beiden Gruppen, ihre Arbeiten zu kombinieren. In diesem Zuge wurden die bisher getrennten Sprachen zu einer gemeinsamen Sprache vereint, die als DAML + OIL bezeichnet wurde. Ihre Spezifikation wurde 2001 dem W3C vorgelegt und dient seitdem als Basis für die Entwicklung der Web Ontology Language (OWL), einer einheitlichen, gemeinsamen Ontologiesprache.

DAML+OIL ist ein Vokabular aus Properties und Klassen, das dem RDF Schema folgende Möglichkeiten hinzufügt:
- Properties mit Kardinalitäten zu versehen
- XML Schema Datentypen zu benutzen
- Beschränkte Listen anzulegen
- Mengenoperationen (wie z. B. Vereinigung, Schnitt) zu benutzen

Eine Klasse in DAML+OIL ist eine Subklasse der Klasse aus dem RDF Schema. Im Unterschied zu RDFS bietet DAML+OIL zur Definition von Klassen aber auch die Möglichkeit

dieses durch direktes Aufzählen der Mitglieder oder durch Angeben von Vereinigungen oder Schnitten oder Verknüpfungen von diesen zu tun. In einer DAML+OIL Klassendefinition kann man außerdem angeben, ob die Klasse zu einer anderen Klasse disjunkt oder äquivalent sein soll. Jedes dieser Elemente kann mehrfach in einer Klassendefinition vorkommen.
Die Properties werden in DAML+OIL nach *ObjectProperty* und *DatatypeProperty* unterschieden. DatatypeProperties setzen Klassen mit einfachen Datentypen in Beziehung.
DAML+OIL benutzt für diese einfachen Datentypen Definitionen aus XML Schema. ObjectProperties setzen Klassen miteinander in Beziehung.
Neben den schon aus RDFS bekannten Angaben für domain und range einer Property, bietet DAML+OIL die Möglichkeit Properties mit Kardinalitätsbeschränkungen zu versehen. Zu diesem Zweck wird eine anonyme Klasse, eine *Restriction*, definiert, in der angegeben wird auf welche Property sich die Beschränkung bezieht und wie genau die Beschränkung aussieht, also eine untere oder eine obere Beschränkung oder der genaue Wert der Anzahl der Werte, die eine Property erhalten darf, und ob die Werte der Property einer bestimmten Klasse angehören müssen. Die Klasse, die in der domain der zu Beschränkenden Property vorkommt und für die diese Beschränkungen gelten sollen (also das Subjekt eines Statements), wird als Unterklasse der Restriction definiert. Wenn z. B. einer Person eine eindeutige Identifikationsnummer mit dem Typ ID-Nummer zugeordnet werden soll, würde man eine Restriction erzeugen, die sich auf die Property Identifikationsnummer bezieht, eine Kardinalitätsbeschränkung auf eins enthält und angibt, dass die Werte dieser Property aus der Klasse ID-Nummer stammen müssen. Von dieser Restriction würde man die Klasse Person mittels subClassOf ableiten. Eine auf diese Weise erzeugte Beschränkung ist eine lokale Beschränkung und gilt nur für die Klasse, die von ihr abgeleitet ist.
Eine andere Möglichkeit Properties zu beschränken ist, die Einschränkungen direkt in der Definition der Property anzugeben. Hier ist es möglich anzugeben, ob die Property von einer anderen abgeleitet wurde (*subPropertyOf*), welche domain bzw. range gilt (wie bei RDFS; siehe oben), ob die Property einzigartig (*UniqueProperty*), unzweideutig (*UnambiguousProperty*) oder transitiv (*TransitiveProperty*) ist. Bei einer einzigartigen Property bestimmt das Subjekt eindeutig das Objekt, d. h., dass es zu jedem Subjekt höchstens ein Objekt gibt; dies entspricht einer oben genannten Kardinalitätsbeschränkung auf einen Wert von eins. Ein Beispiel hierfür ist, dass zu einem Kind (Subjekt) eindeutig eine Mutter (Objekt) gehört (Subjet: Kind; Prädikat: hatMutter; Objekt: Mutter). Bei einer unzweideutigen Property bestimmt das Objekt eindeutig das Subjekt, d. h., dass es zu jedem Objekt höchstens ein Subjekt gibt; Unzweideutigkeit ist das Inverse zur Einzigartigkeit.
Für das Beispiel bedeutet dies, dass eine Mutter (Subjekt) mehrer Kinder (Objekte) haben kann, aber es zu jedem Kind (Objekt) nur eine Mutter (Subjekt) gibt (Subjekt: Mutter; Prädikat: hatKind; Objekt: Kind). Diese Art von Beschränkungen ist global und bewirkt, dass jede Klasse, auf die die Property angewendet werden soll, die Beschränkungen erfüllen muss. Für das Beispiel, das ich für die Restriction angeführt habe, hieße dies, dass man die Eindeutigkeit auch dadurch erreichen könnte, dass man die Property Identifikationsnummer als unzweideutig definierte (UnambiguousProperty). Dies hätte zur Folge, dass jede Klasse, auf die die Property Identifikationsnummer angewendet wird, eine eindeutige IDNummer haben muss, wohingegen sich dies in der ersten Möglichkeit nur auf die Klasse Person bezöge und es bei anderen Klassen noch möglich wäre verschiedenen Subjekten dieselbe ID-Nummer zuzuordnen (auch wenn dies zugegebenermaßen nicht besonders sinnvoll wäre).
Dass eine Property transitiv ist bedeutet, dass, wenn die Property A mit B und B mit C verbindet, dann verbindet sie auch A mit C, z. B. wenn Hugo der Vorfahre von Karl ist und Karl der Vorfahre von Wilhelm, dann ist Hugo auch Vorfahre von Wilhelm.
Anders als in RDF bzw. RDFS ist es in DAML+OIL möglich eine beschränkte Liste anzulegen; diese ist vom Typ *Collection*. Die Struktur einer solchen Liste lässt sich rekursiv aus den Elementen *first*, *rest* und *nil* zusammensetzen, wobei nil das Ende der Liste kennzeichnet. Dadurch ist gewährleistet, dass die Liste beschränkt ist und keine Elemente

hinzugefügt werden können, ohne die Struktur der Liste zu verändern bzw. eine vorhandenes Element zu überschreiben. Diese Liste wird bei der Definition einer Klasse durch Aufzählen der Instanzen (*Enumeration*) benutzt.

5. OWL

Die *Web Ontology Language* (OWL) ist eine Sprache, die, wie RDFS und DAML+OIL, dazu dient, die Bedeutung von Ausdrücken und deren Beziehungen explizit in einem Vokabular festzulegen und damit eine Ontologie zu erzeugen. Für die Entwicklung von OWL dienten die Erfahrungen, die während der Entwicklung und des Einsatzes von DAML+OIL gemacht wurden, als Ausgangspunkt. Daher haben beide Modellierungssprachen ähnliche Funktionalitäten und OWL kann als Revision von DAML+OIL betrachtet werden.
OWL besteht aus drei Untersprachen: OWL Lite, OWL DL und OWL Full.

OWL Lite ist die einfachste und eingeschränkteste dieser drei Untersprachen. Es wird benutzt, wenn man in erster Linie eine Hierarchie und nur einfache Definitions- und Wertebereichs-Einschränkungen benötigt. Es ist z. B. für Kardinalitätsbeschränkungen nur möglich diesen die Werte Null oder Eins zu geben. OWL Lite bietet weniger Ausdrucksmöglichkeiten als OWL DL oder OWL Full, ist dafür aber auch weniger Komplex und es ist daher auch einfacher Software, die diese Sprache unterstützt, zu entwerfen.

OWL DL bietet viele Ausdrucksmöglichkeiten und gewährleistet dabei die Vollständigkeit und die Entscheidbarkeit, d. h., dass alle aus gegebenen Prämissen ableitbaren Folgerungen auch tatsächlich abgeleitet werden und dass dieser Prozess terminiert, d. h. in endlicher Zeit berechnet wird. In OWL DL können alle Möglichkeiten von OWL genutzt werden, allerdings unterliegen einige Einschränkungen, um die Vollständigkeit und die Entscheidbarkeit gewährleisten zu können. Das DL im Namen ergibt sich aus dem Zusammenhang mit description logics, die die formale Grundlage für OWL bilden. Description logics sind eine Familie von Sprachen zur Wissensrepräsentation, die besonders auf dem Gebiet der künstlichen Intelligenz untersucht werden.

OWL Full bietet die kompletten Modellierungsmöglichkeiten von OWL ohne Einschränkungen, aber auch ohne Garantien für die Berechenbarkeit.
Jede dieser Untersprachen ist eine Erweiterung des einfacheren Vorgängers. Dies wiederum bedeutet, dass OWL Lite Ontologien auch OWL DL Ontologien und OWL DL Ontologien auch OWL Full Ontologien sind; die Umkehrung gilt allerdings nicht.
OWL Full kann als eine Erweiterung von RDF angesehen werden, wohingegen dies für OWL Lite und OWL DL nur eingeschränkt gilt. Dies hängt damit zusammen, dass aufgrund der Einschränkungen in OWL Lite und OWL DL nicht alle RDF Dokumente OWL Lite oder OWL DL Dokumente sind. Es sind jedoch alle RDF Dokumente auch OWL Full Dokumente.
Bei dem Entwurf von OWL wurden im Vergleich zu DAML+OIL einige Modellierungsmöglichkeiten hinzugefügt und wenige entfernt, so dass die Ausdrucksmöglichkeiten von OWL Full in etwa vergleichbar mit denen von DAML+OIL sind.

Im Folgenden werden einige Sprachkonstrukte von OWL vorgestelt. In OWL werden Konzepte oft als Klassen, Individuen als Instanzen und Beziehungen als Properties (Eigenschaften) bezeichnet. Die entsprechenden Begriffe werden im Folgenden synonym verwendet.

Klassen
Klassen werden in OWL mit dem Schlüsselwort owl:Class definiert. Ein Beispiel für einfache Klassendefinitionen sind:

```
<owl:Class rdf:ID="Wein"/>
<owl:Class rdf:ID="trinkbareFlüssigkeit"/>
```

Damit wird eine Klasse mit der Bezeichnung Wein und eine Klasse mit der Bezeichnung trinkbareFlüssigkeit eingeführt. Innerhalb der Ontologie können diese Klassen mit rdf:resource="#Wein" bzw. rdf:resource="#trinkbareFlüssigkeit" referenziert werden. Um nun Wein als Unterklasse bzw. Unterkonzept von trinkbareFlüssigkeit zu definieren verwendet man rdfs:subClassof:

```
<owl:Class rdf:about="#Wein"/>
<rdfs:subClassOf rdf:resource="#trinkbareFlüssigkeit"/>
...
</owl:Class>
```

Durch rdf:about="#Wein" kann die Definition einer Klasse erweitert werden. Damit können auch Klassen aus anderen Ontologien erweitert werden, ohne ihre ursprüngliche Definition wiederholen zu müssen.

Instanzen

Ein Individuum wird angegeben, indem man es als Element einer bestimmten Klasse definiert:

```
<owl:Class rdf:ID="Riesling"/>
<Riesling rdf:ID="CriesbacherKocherbergRiesling"/>
```

In diesem Beispiel passieren gleich zwei Dinge. Zum einen wird die Klasse Riesling definiert, und zum anderen wird angegeben, dass CriesbacherKocherbergRiesling ein Individuum vom Typ Riesling ist. In ausführlicher Form würde das Beispiel so aussehen (beide sind semantisch äquivalent!):

```
<owl:Class rdf:ID="Riesling"/>
<owl:Thing rdf:ID="CriesbacherKocherbergRiesling"/>
<owl:Thing rdf:about="#CriesbacherKocherbergRiesling">
<rdf:type rdf:resource="#Riesling"/>
</owl:Thing>
```

In OWL gibt es zwei vordefinierte Klassen owl:Thing und owl:Nothing. Die Klasse owl:Thing besteht aus allen Individuen eines betrachtetenWeltausschnitts, owl:Nothing enthält keinerlei Individuen, steht also für die leere Menge, und ist Unterklasse aller Klassen. Die Zugehörigkeit von CriesbacherKocherbergRiesling zur Klasse owl:Thing wird in dieser ausführlichen Variante explizit angegeben. Mit rdf:type wird die Beziehung zwischen einer Instanz und ihrer Klasse definiert, sprich CriesbacherKocherbergRiesling ist ein Riesling.

Properties

In OWL werden zwei Arten von Properties unterschieden.

Datatype properties setzen Instanzen mit primitiven Datentypen in Beziehung.
Object properties setzen Instanzen zweier Klassen miteinander in Beziehung.
Zur Definition von Beziehungen verwendet man owl:ObjectProperty bzw. owl:DatatypeProperty:

```
<owl:ObjectProperty rdf:ID="hergestelltAus">
<rdfs:domain rdf:resource="#Wein"/>
<rdfs:range rdf:resource="#Rebsorte"/>
</owl:ObjectProperty>
```

Hier wird die Beziehung hergestelltAus definiert. Ihr Definitionsbereich (rdfs:domain) ist auf Instanzen der Klasse Wein und ihr Wertebereich (rdfs:range) auf Instanzen der Klasse Rebsorte eingeschränkt. Ein Beispiel für die Verwendung von hergestelltAus:

```
<owl:Thing rdf:ID="CriesbacherKocherbergRiesling">
<hergestelltAus rdf:resource="#RieslingTraube" />
</owl:Thing>
```

Dies bedeutet, dass der Wein CriesbacherKocherbergRiesling aus der Rebsorte RieslingTraube hergestellt wird. Durch die Angabe von rdfs:range bei hergestelltAus kann abgeleitet werden, dass RieslingTraube eine Rebsorte sein muss.
In einer Klassendefinition kann auch angegeben werden, wie viele verschiedene Werte eine bestimmte Property haben darf bzw. muss. Dies wird auch Kardinalitätseinschränkung genannt. Es folgt ein Beispiel für die Beziehung hergestelltAus:

```
<owl:Class rdf:ID="Wein">
<rdfs:subClassOf rdf:resource="#trinkbareFlüssigkeit"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hergestelltAus"/>
<owl:minCardinality ...
... rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>

</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Der hervorgehobene Block owl:Restriction...definiert eine anonyme Klasse. Diese Klasse repräsentiert die Menge aller Individuen, die mindestens einen Wert (owl:minCardinality=1) für die hergestelltAus-Beziehung haben. Da Wein als Unterklasse dieser anonymen Klasse definiert ist, muss jede Instanz von Wein mit einer Instanz von Rebsorte in der hergestelltAus-Beziehung stehen. Zur Kardinalitätseinschränkung stehen noch owl:maxCardinality und owl:Cardinality zur Verfügung.
Mit den hier vorgestellten Sprachkonstrukten lassen sich bereits einfache Ontologien definieren. Eine umfassende Einführung in alle Sprachkonstrukte von OWL gibt das Dokument OWL Web Ontology Language Guide des W3C.

6. Frame-Logic

Als geistige Väter von Frame-Logic (F-Logik) gelten Michael Kifer und Georg Lausen. Ein erstes Framework wurde schon 1986 von Michael Kifer entwickelt. In dem technischen Report „Logical Foundations of Object-Oriented and Frame-Based Languages“ von 1989 stellten Kifer und Lausen F-Logik zum ersten Mal als formale Sprache vor.
F-Logik ist eine mächtige Sprache, die deduktive und Objektorientierte Konzepte (Konzept, Relation, Klassenhierarchien, Vererbung, Signaturen) verbindet. Der deduktive Ansatz versucht, die Mächtigkeit der Datenbankabfragesprache zu erweitern. Hierbei können Daten durch Regeln definiert sein, wodurch die Möglichkeit zu rekursiven Anfragen gegeben ist. Insbesondere kann der deduktive Ansatz mit einer deklarativen, modelltheoretischen Semantik verbunden werden. Bei dem Objektorientierten Ansatz steht die Erweiterung des Datenmodells im Vordergrund. Hierbei werden komplex strukturierte Konzepte mit einer Objektidentität eingeführt, die hierarchisch angeordneten Klassen zugeordnet werden können. Die Modellierung von Beziehungen zwischen Konzepten erfolgt mit Relationen, die vererbt werden können.
In F-Logik steckt zum einen der Begriff Logik und zum zweiten der Begriff Frame. Logik kann als universeller Formalismus zur präzisen Beschreibung von Sachverhalten definiert werden. Die Logik kann man in so genannte Aussagenlogik(AL), Prädikatenlogik(PL) und Horn-Logik(HL) aufteilen. Und Frame bedeutet Anordnung oder Bild. Mit einem Frame lassen sich Attribute und Beziehungen zwischen Konzepten darstellen. Man kann mit ihnen zum Beispiel Klassenhierarchien grafisch veranschaulichen.
Es gibt noch ein paar grundlegende Begriffe von F-Logik, die im Folgenden ständige Verwendung finden und zunächst kurz erläutert werden. F-Logik ist eine Objektorientierte Datenbanksprache und ermöglicht somit den Umgang mit Daten in Form von Objekten. Zu Objekten zählen Konzepte, Klassen, Instanzen, Attribute und Relationen. Man kann Konzepte in Klassen einteilen. So ist es beispielsweise möglich, eine Klasse „mann“ zu definieren. Ein Konzept, das zu dieser Klasse „mann“ gehört, nennt man eine „Instanz“ oder „Subklasse“ dieser Klasse. Eine Relation drückt gewisse Beziehungen zwischen Konzepten aus. So kann man zum Beispiel eine Relation „vater“ auf die Klasse „mann“ definieren. Wendet man nun diese Relation auf eine Instanz oder Subklasse dieser Klasse an, so ist es beispielsweise möglich, das Konzept zu ermitteln, das vorher als „vater“ der Instanz oder Subklasse definiert wurde.

Beispielprogramm mit Abfragen

Um einen ersten Eindruck von F-Logik zu bekommen, geben wir zunächst ein F-Logik-Programm sowie eine Abfrage an. Dazu soll hier und auch im folgenden der Familienstammbaum der Simpsons dienen:

```
// Fakten
abraham : mann.
Penelope : frau.
Homer : mann[vater->abraham, mutter->penelope].
Bart : mann [vater->homer, mutter -> marge : frau].
Lisa : frau [vater->homer, mutter->marge].
Maggie : frau[vater->homer, mutter->marge].
// Regeln
X[sohn->>{Y}] :- Y : mann, Y[vater->X].
X[sohn->{Y}] :- Y : mann, Y[mutter->X].
X[tochter->{Y}] :- Y : frau, Y[vater->X].
X[tochter->{Y}] :- Y : frau, Y[mutter->X].
```

// Abfrage
X :- Y : homer[tochter->X].

Abbildung 4-1: Ein Beispiel von F-Logik
Bei der Syntax eines F-Logik-Programms gilt es zu beachten, dass jedes Faktum, jede Regel und jede Abfrage durch einen Punkt beendet wird.

Fakten

Das eigentliche F-Logik-Programm besteht aus den beiden Teilen „Fakten“ und „Regeln“. Die Fakten bestehen aus grundlegenden Informationen, die man dem Programm mitteilt:

abraham : mann.
homer : mann [vater -> abraham, mutter -> penelope].

Hier zum Beispiel ist die Information, dass Abraham und Homer Instanzen der Klasse „mann“ sind und Homer Abraham als Vater und Penelope als Mutter hat. In dem Teil des Programms, das „Fakten“ genannt wird, legt man also den expliziten Teil der Wissensbasis ab.

Regeln

Mit Hilfe von Regeln kann zusätzliches, nicht explizit formuliertes Wissen, sogenanntes implizites Wissen aus den Fakten abgeleitet werden. So wird dem Programm zunächst zum Beispiel mit der Fakten-Zeile
Bart : mann [vater -> homer, mutter -> marge : frau].
mitgeteilt, dass Homer der Vater von Bart ist. Im zweiten Programm-Teil kann dann mit der Regel
X[sohn ->> Y] :- Y : mann [vater -> X].
ermittelt werden, dass Bart der Sohn von Homer ist. Dies wollen wir nun genauer betrachten. Der rechte Teil der Regel, also der Teil, welcher rechts von „:-„ steht, wird Regelkörper, der linke Teil Regelkopf genannt. Man liest eine Regel wie folgt:
Ist der Regelkörper erfüllt, so gilt auch der Regelkopf.
Man liest also sozusagen „von rechts nach links“. In diesem Beispiel bedeutet dies:
Ist Y ein Mann, dessen Vater X ist, so ist Y Sohn von X.
Bezogen auf das Faktum, dass Homer Barts Vater ist, heißt das laut die obengenannte Regel, dass Bart der Sohn von Homer ist. Die Information, dass Bart der Sohn von Homer ist, wenn Homer Vater von Bart ist, ist für uns Menschen selbstverständlich, da wir mit diesen Beziehungen vertraut sind. Das Programm kennt diese Beziehungen natürlich nicht und muss sie deswegen mitgeteilt bekommen. Sehr elegant gelingt dies wie oben gesehen mit Hilfe von Regeln, da das Eintippen einer Großzahl von Familienbeziehungen nun überflüssig wird und daher Aufwand erspart werden kann.
Durch Regeln kann man also implizites Wissen ermitteln. Es werden dadurch neue Relationen geschaffen. So taucht die Relation „sohn“ in unserem Beispiel-Programm das erste Mal nicht in den Fakten, sondern in den Regeln auf. Die Relation wird dem Programm also nicht explizit durch die Fakten mitgeteilt, sie wird allgemein durch die Regeln definiert und somit „erschaffen“. Regeln dienen also der Inferenz von Wissen aus den vorhandenen Fakten und somit der Erweiterung der Wissensbasis.

Abfragen

Abfragen gehören zwar nicht zu einem F-Logik-Programm, sind aber trotzdem von großer Bedeutung. Sie dienen dazu, Informationen aus dem Programm herauszuziehen. Das Wissen liegt in Form von Fakten vor, wobei diese Wissensbasis noch durch Inferenzmechanismen (also durch Regeln) erweitert wird. Mit Hilfe von Abfragen holt man sich dann die gewünschten Informationen aus dem F-Logik-Programm. Könnte man nicht gezielt mit Abfragen auf die gesammelten Informationen zugreifen, so wäre diese Wissensansammlung zwecklos. Die oben angegebene Beispiel-Abfrage:

X :- homer [tochter ->> X].

liefert als Ergebnis alle Töchter von Homer. In dieser Abfrage wird nach allen Belegungen der Variablen „X“ gefragt, welche die rechte Seite der Regel erfüllen, also Töchter von Homer sind.

Kommentare

Die Beschriftungen „Fakten“, „Regeln“, „Abfragen“ im oben angegebenen Programmtext ist kein eigentlicher Code, sondern nur Kommentare. Kommentare können in F-Logik auf drei verschiedene Arten ins Programm eingebracht werden. Die Zeichen „//“ und „%“ bedeuten, dass der Rest der Zeile ein Kommentar ist, genauso wie der Text, der sich zwischen den Zeichen „/*“ und „*/“ befindet. Diese Möglichkeit der Erläuterung im Code erhöht die Lesbarkeit und Verständlichkeit des Programms und sollte deshalb auch zur Dokumentation genutzt werden.

Literatur

Tim Berners-Lee, James Hendler, Ora Lassila, *Scientific American: The Semantic Web*, http://www.scientificamerican.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21, 17.05.2001

Tim Berners-Lee, *Standards, Semantics and Survival*, http://www.w3.org/2003/Talks/01-siia-tbl/Overview.html, 2003

Bertelsmann, Das moderne Lexikon, *Ontologie*, Band 13, Seite 388, 1972

Harold Boley, Stefan Decker, Michael Sintek, *Tutorial on Knowledge Markup and Resource Semantics*, http://www.dfki.uni-kl.de/km/kmrs/, 20.08.2001

Dan Brickley, R. V. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema*, http://www.w3.org/TR/rdf-schema/, 23.01.2003

Dean Connoly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, *DAML+OIL (March 2001) Reference Description*, http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218, 18.12.2001

Diverse, *What is Ontology? Definitions by leading philosophers*, http://www.formalontology.it/section_4.htm, 2003
Dieter Fensel, Ora Lassila, Frank van Harmelen, Ian Horrocks, James Hendler, Deborah L. McGuinness, *The Semantic Web and its languages*, http://www.computer.org/intelligent/ex2000/pdf/x6067.pdf, 12.2000

Yolanda Gil, Varun Ratnakar, *A Comparison of (Semantic) Markup Languages*, http://trellis.semanticweb.org/expect/web/semanticweb/flairs02.pdf, 2000

May Wolfgang, *How to write F-Logik Programms in Florid*
Universität Freiburg, 1999

Patrick Hayes, *RDF Semantics*, http://www.w3.org/TR/rdf-mt/, 23.01.03
[Hef03] Jeff Heflin, *Web Ontology Language (OWL) Use Cases and Requirements*, http://www.w3.org/TR/webont-req/, 31.03.2003

Graham Klyne, Jeremy J. Carroll, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, http://www.w3.org/TR/rdf-concepts/, 23.01.2003

Ora Lassila, Ralph R. Swick, *Resource Description Framework (RDF) Model and Syntax Specification*, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 22.02.1999