Universität Zürich
Institut für Informatik

Dynamic and Distributed Information Systems

Semantic Crystal. Ein End-User-Interface zur Unterstützung von Ontologie-Abfragen mit SPARQL

Diplomarbeit 12. Sept. 2006

Beat Sprenger
Geburtsort: Wil SG

Matrikelnummer: 99-912-792
bespr@gmx.ch

Betreuerin: Esther Kaufmann

Prof. Dr. Abraham Bernstein
Institut für Informatik
Universität Zürich
http://www.ifi.unizh.ch/ddis

Abstract

Although standards for structuring semantic information have been available for a long time (XML, RDF, OWL, SPARQL), most typical users have not yet encountered the semantic web. There are mainly two reasons for this: There are not many semantic knowledge bases available which could be interesting to query, and there is not much software to support such queries neither. Therefore this research paper presents *Semantic Crystal*, a prototype of such a query software. *Semantic Crystal* displays classes and relationships from OWL knowledge bases graphically and supports the setup of a query – graphically as well. Although the query eventually exists in the intricate language SPARQL, it will be comprehensible by a broad number of users due to its graphical representation.

Zusammenfassung

Obwohl Standards für die Strukturierung von semantischen Informationen längst vereinbart worden sind (XML, RDF, OWL, SPARQL), hat der typische Anwender noch keine Berührungen mit dem *Semantic Web* machen können. Einerseits liegen nur wenige semantische Wissensbasen vor, die zu durchforsten von Interesse wären, andererseits liegt auch nur wenig Software vor, welche solche Durchforstungen unterstützt. Diese Arbeit beschreibt *Semantic Crystal*, einen Prototyp einer solchen Software. Er zeigt Klassen und Verbindungen aus OWL-Wissensbasen graphisch dar und unterstützt – ebenfalls graphisch – den Aufbau einer Abfrage. Obwohl die Abfrage schliesslich in der komplexen Sprache SPARQL vorliegt, wird sie durch die graphische Darstellung so repräsentiert, dass sie für einen breiten Benutzerkreis verständlich und leicht bedienbar ist.

Inhaltsverzeichnis

1

Einführung

1.1 Vision und Entstehung des Semantic Web

Die Vision des *Semantic Web* klingt vielversprechend. Informationen, welche über das Internet verfügbar sind, sollen nicht nur für Menschen nutzbar sein, sondern auch für Maschinen. «Web-Erfinder» Tim Berners-Lee [1] erschuf diese Vision und löste damit Forschungs- und Entwicklungsschübe aus. Das *Semantic Web* soll etappenweise realisiert werden, wobei die ersten Etappen längst genommen sind.

Auch dieser Text beschäftigt sich mit einer dieser *Semantic-Web*-Etappen. Anhand des Software-Prototypen *Semantic Crystal* wird gezeigt, wie Computeranwender ohne ausgeprägtem Spezialwissen bestehende Web-Wissensbasen bzw. Ontologien abfragen können.

XML und RDF gehören zu denjenigen Elementen, die in der Welt des *Semantic Web* bereits akzeptiert sind und rege benutzt werden. OWL als Sprache für Web-Ontologien[1] ist auf gutem Wege zu solcher Akzeptanz und Benutzung.

Seit April 2006 ist SPARQL eine *W3C Candidate Recommendation*, also ein Kandidat für eine W3C-Empfehlung. SPARQL ist eine Abfragesprache für OWL-Wissensbasen und orientiert sich stark an der Datenbankabfragesprache SQL. Deshalb ist sie zwar logisch und konsistent, aber für den Alltagsgebrauch eher ungeeignet.

Die Software *Semantic Crystal* bietet Unterstützung im Formulieren solcher SPARQL-Abfragen, indem Wissensbasen durch graphische Darstellungen zugänglich gemacht werden, und indem die Elemente einer Abfrage einzeln zusammengestellt werden können.

1.2 Sinn und Zweck von Abfragen

Warum hat ein Mensch überhaupt ein Interesse daran, eine Wissensbasis abzufragen? Sollten diese Aufgabe nicht *Semantic-Web*-Maschinen automatisch für uns Menschen erledigen?

Diese kritischen Fragen sind berechtigt. Ist das *Semantic Web* nach der eingangs erwähnten Vision erst einmal entwickelt, werden solche Abfragen nur noch von Software-Agenten und einigen wenigen Experten durchgeführt. Für den Fall aber, dass das *Semantic Web* nicht nach dem «Masterplan» fertig gestellt wird, und immer erst dann mit der nächsten Entwicklungsetappe begonnen wird, wenn sich die vorhergehenden Techniken etabliert haben, dann könnten solche Abfragen für durchschnittliche Benutzer von grosser Wichtigkeit sein.

Dieser Zusammenhang gleicht beinahe einem «fortschrittshemmenden Teufelskreis». Momentan gibt es erst wenige OWL-Wissensbasen, und die wenigen, die vorhanden sind, wurden meist

[1] Seit Februar 2004 ist OWL eine W3C-Empfehlung [2] für Web-Ontologiesprachen.

nur zu Testzwecken erstellt oder behandeln ein Spezialgebiet. Da es an Wissensbasen mangelt, gibt es auch keine Eile, Software zur Ontologiedarstellung und Abfrageunterstützung zu entwickeln.

Der Sachverhalt kann aber auch positiv formuliert werden: Sobald eine gewisse Schwelle erreicht ist, bei welcher genügend Wissensbasen und Unterstützungssoftware vorhanden ist, entsteht eine Dynamik, die sofort unzählige weitere Wissensbasen und Software nach sich zieht. Um näher an diese Schwelle zu gelangen, wurde im Rahmen dieser Diplomarbeit die Software *Semantic Crystal* und eine Film-Wissensbasis entwickelt.

Anhand einer fiktiven Beispielsgeschichte möchte ich eine Anwendungsmöglichkeit für die Software *Semantic Crystal* aufzeigen.

> Ruth ist eine begeisterte Kinogängerin. Bevor sie sich einen Film anschaut, möchte sie sich darüber informieren. Dazu ging sie bisher auf Webseiten von Kinomagazinen und las darin kreuz und quer, bis sie genügend Informationen gesammelt hatte, um sich für oder gegen einen Kinobesuch zu entscheiden. Eines Tages hörte sie von ihrer Kollegin Doris, dass es in Zukunft nicht mehr nötig sein werde, selber auf Webseiten herumzusurfen, da diese Arbeit von Software-Agenten über sogenannte *Semantic-Web*-Wissensbasen erledigt würden. Doris nannte eine URL, wo eine solche Film-Wissensbasis zu finden sei.
>
> Freudig tippte Ruth die URL in ihren Webbrowser ein. Doch mit dem Resultat, einer riesigen XML-Datei, war sie nicht zufrieden. Zwar ist sie theoretisch mit XML vertraut und erkennt, dass zwischen den einzelnen XML-Tags viele Kino-Informationen stecken, doch wie sollte sie damit nur schneller zu ihren Information kommen als über die herkömmlichen Webseiten?
>
> Sie fragte also ihre Kollegin, wo sie denn einen dieser *Semantic-Web*-Software-Agenten herkriege. Diese meinte, dass sie das auch nicht wüsste, und führte ihr stattdessen *Semantic Crystal* vor. Ruth sah wie die kryptische XML-Datei graphisch dargestellt wurde. Ein Rechteck hiess *Movie*, ein anderes *Actor*, und die waren miteinander verbunden. Doris klickte auf *Actor*, gab den Namen von Ruths Lieblingsschauspieler ein und schon extrahierte die Software diejenigen Filme, in denen der besagte Schauspieler mitspielt. «Das ist ja wie eine Datenbank», sagte Ruth erfreut. «Ja», sagte Doris, «nur kann ich damit beliebige Wissensbasen im Web laden». Und schon zeigte die Software Informationen über Fussballspieler an.

Obwohl man mit solchen Beispielen leicht zu verschönerten Realitäten kommen kann, zeigt das obige Beispiel das Anwendungsgebiet von *Semantic Crystal* auf. Die Software soll einem Internetbenutzer helfen, sich leicht einen Überblick über eine erstmals gesehene Ontologie zu verschaffen. Zudem soll sie Unterstützung bieten beim Erstellen von Abfragen.

Nicht ohne Grund war im Beispiel von einer Film-Wissensbasis die Rede. Im Rahmen der Diplomarbeit wurde eine Film-Ontologie und die dazugehörigen Instanzen erstellt, welche in diesem Text immer wieder als Beispiel herhalten müssen.

1.3 Aufbau dieses Textes

Dieses Kapitel bietet eine kurze Einleitung, die zum Weiterlesen motivieren soll. Im 7. und letzten Kapitel befindet sich ein Resümee des gesamten Textes. Die dazwischenliegenden fünf Hauptkapitel sind wie folgt aufgebaut.

In Kapitel 2 werden die angesprochenen Elemente von *Semantic Web* wie XML, RDF, OWL und SPARQL und die Beispiels-Wissensbasis detaillierter beschrieben.

In Kapitel 3 werden theoretische Ausführungen über SPARQL-Abfragen gemacht. Abfragen werden analysiert, die Elemente einzeln angeschaut und daraus das TORC-Prinzip mit den vier Grundstrukturen einer Abfrage abgeleitet. Diese finden sich im Software-Prototyp *Semantic Crystal* wieder, welcher in Kapitel 4 vorgestellt wird.

In Kapitel 5 werden die Resultate der Evaluation von *Semantic Crystal* besprochen. Ausserdem findet sich dort ein theoretischer Exkurs über *Usability* und *Iterative Design*, zwei Softwareentwicklungsprinzipien, nach welchen *Semantic Crystal* erstellt worden ist.

Kapitel 6 widmet sich schliesslich den bestehenden Problemen und Herausforderungen, die sich einerseits auf mangelhafte oder für unsere Zwecke ungeeignete Ontologien beziehen, sich aber andererseits auch mit einem Ausbau von *Semantic Crystal* eliminieren liessen. Ausserdem wird auf zukünftige Forschungsfelder in diesem Gebiet vorausgeschaut.

1.4 Schreibstil und Konventionen

Obwohl dieser Text in deutscher Sprache verfasst ist, werde ich nicht darum herumkommen, englische Begriffe zu verwenden. Ich ziehe es vor, zu sagen, «ein RDF-*Statement* hat eine *Property*» als «eine RDF-Aussage hat eine Eigenschaft», da es sich bei den Termen *Statement* und *Property* um Fachbegriffe handelt, deren Charakteristik bei einer Übersetzung ins Deutsche verloren ginge. Ausserdem sind diese Fachbegriffe teilweise noch sehr jung, so dass es keine konventionalisierte Wortwahl im Deutschen dafür gibt, und deshalb das Risiko besteht, einem Gegenstand einen deutschen Begriff zu geben, der in fünf Jahren nicht mehr in der Bedeutung gebraucht wird. Die englischsprachigen Fachbegriffe sind aber hervorgehoben.

Ausnahmen bilden Wörter, die auf deutsch praktisch gleich lauten wie auf englisch, und bei welchen auch keine Bedeutungsunterschiede bestehen. Statt «wir beschreiben *concepts* und *resources*» schreibe ich «wir beschreiben Konzepte und Ressourcen».

Wie üblich werden Hervorhebungen *kursiv* geschrieben und Computercode in `Schreibmaschinenschrift`.

Bezeichnungen wie «Benutzer» oder «Entwickler» sind generische Maskulina und schliessen jeweils männliche und weibliche Personen ein.

1.5 Inhalt der beiliegenden CD

Die beiliegende CD enthält folgendes:

- Im Ordner `semcrys` ist die mit *Java Runtime Environment 5* lauffähige Software *Semantic Crystal*. Windows-Benutzer bringen das Programm mit der Batch-Datei `semcrys.bat` zum Laufen. Benutzer anderer Betriebssysteme führen dazu folgenden Befehl in der Konsole aus, um `semcrys.jar` mit der entsprechenden Umgebung zu starten:

  ```
  java -Djava.ext.dirs=lib -jar semcrys.jar
  ```

- Im Ordner `source` befindet sich der Quellcode des gesamten Programms
- Im Ordner `doc` befindet sich die Dokumentation des Quellcodes im *Javadoc*-Format
- Im Ordner `ontology` befindet sich die OWL-Wissensbasis `movie2005.owl`
- Im Hauptverzeichnis befinden sich die von der Universität Zürich vorgeschriebenen Dateien `zusfg.pdf` (Kurzfassung in deutscher Sprache), `abstract.pdf` (*Abstract* in englischer Sprache) und `thesis.pdf` (vorliegender Text)

Bitte beachten Sie, dass die Universität Zürich Eigentümerin der im Rahmen dieser Diplomarbeit erstellten Software und Wissensbasis ist. Die beiden Produkte sind deshalb nicht quelloffen im Internet verfügbar. Falls dennoch Interesse an der Software oder der Film-Wissensbasis besteht, kontaktieren Sie bitte Esther Kaufmann oder eine andere Person der DDIS-Gruppe des Instituts für Informatik an der Universität Zürich (`http://www.ifi.unizh.ch/ddis`).

Ich wünsche viel Vergnügen beim Lesen dieses Textes.

2

Grundlagen für Semantic-Web-Abfragen

2.1 Die Schichten des Semantic Web

Das *Semantic Web* setzt sich aus aufeinander aufbauenden Technikelementen zusammen. Abbildung 2.1 zeigt diese Zusammensetzung anhand des Schichtenmodells, welches Tim Berners-Lee in seinen Vorträgen benutzt hat.

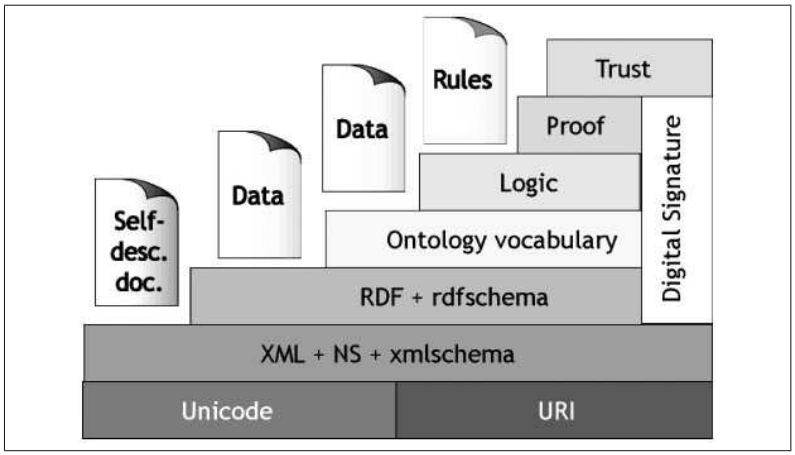

Abbildung 2.1: Die Schichten des Semantic Web. Die oberen Schichten bauen auf den unteren auf. Unicode, URIs, XML und XML-Schema sind ausgereifte Technologien, die heute in diversen Anwendungsgebieten eingesetzt werden. Je weiter oben eine Technologie angesiedelt ist, desto weniger ist sie standardisiert und zur Benutzung ausgereift. Ein Grossteil des vorliegenden Textes befasst sich mit der Schicht der Ontologie. (Quelle: [3] S.18)

Da jede Schicht auf der darunterliegenden Schicht aufbaut, werden die obenliegenden Elemente chronologisch später realisiert. Zum jetzigen Zeitpunkt haben sich die unteren Schichten (bis und mit RDF) bereits etablieren können. Sie werden in diversen Anwendungen rege benutzt, auch wenn viele dieser Anwendungen nicht direkt unter dem Mantel des *Semantic Web* stehen.

Die oberen Schichten (*Logic* und höher) stehen zum jetzigen Zeitpunkt noch am Anfang, und verschiedene Technikentwürfe, Prototypen und Sprachversionen stehen sich gegenüber. Dabei

ist unklar, ob diese Schichten überhaupt je nach dem «Masterplan der *Semantic-Web*-Vision» realisiert werden können, oder ob sich hier ganz andere Schichten auftun werden. Die Zukunft wird zeigen, ob die geleistete Vorarbeit in den Bereichen Logik, Beweisführung und Vertrauensforschung für das *Semantic Web* übernommen werden kann.

Der Entwicklungsgrad der Schichten zeigt sich schon darin, dass für die unteren Schichten Namen vergeben worden sind, die konkrete Techniken und Sprachen bezeichnen («URI» statt «Identifizierung», «XML» statt «Datenformat», «RDF» statt «Metadaten») während die oberen Schichten (noch) nach abstrakten Sammelbegriffen benannt sind («Ontology», «Logic», «Proof», «Trust»). Heute könnte man die Ontologie-Schicht gut mit «OWL» beschriften. Dies ist ein Zeichen dafür, dass sich die *Semantic-Web*-Vision am Verwirklichen ist.

Auch diese Arbeit bewegt sich vorwiegend in der Ontologieschicht. Da diese wie eben erklärt auf den unteren Schichten aufbaut, wird hier noch kurz auf die Fundamentalschichten eingegangen.

2.1.1 Unicode und URIs

Unicode ist eine Zeichencodierung, die «alle sinntragenden Zeichen dieser Welt»[1] abzudecken vermag. Damit wird das Problem beseitigt, dass z. B. deutsche Umlaute auf französischen Rechnern falsch dargestellt werden. Da das *Semantic Web* global einsetzbar sein wird, ist die Verwendung der Unicode-Zeichencodierung zwingend.

Ressourcen-URI	Ressourcen-Beschreibung
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	Die Eigenschaft *type*, die in RDF definiert wird.
urn:isbn:0552149519	Das Buch *The Da Vinci Code* von Dan Brown. Bestehende Codierungssysteme wie die *International Standard Book Number* (**ISBN**) werden mit vorangestelltem `urn` zu URIs. **URN** steht für *Uniform Resource Name*
http://www.ifi.unizh.ch/ddis/ont/movieont.owl#m8403	Der Film *The Da Vinci Code* in der Film-Wissensbasis.
http://www.ifi.unizh.ch/ddis/ont/movieont.owl#Jolie1a	Die Schauspielerin Angelina Jolie.

Tabelle 2.1: Beispiele von URIs und Ressourcen

Mit **URI**s (*Uniform Resource Identifier*) werden Ressourcen eindeutig identifiziert. Eine Ressource kann alles Mögliche sein: eine Webseite, ein real existierendes Objekt oder auch ein abstrakter Sammelbegriff. Beispiele von URIs sind der Tabelle 2.1 zu entnehmen.

Die bekannteste Form von URIs sind URLs (*Uniform Resource Locator*), welche Ressourcen nicht nur identifizieren, sondern auch einen Zugriff darauf bieten. Weil URIs als Bezeichner relativ lang und für den Menschen nicht sehr praktisch sind, wird oft auf eine Präfix-Schreibweise ausgewichen. Dabei ersetzt ein Präfix den für viele Ressourcen gleichbleibenden Teil der URI. So wird für `http://www.w3.org/1999/02/22-rdf-syntax-ns#` meistens das Präfix `rdf` gewählt, so dass z. B. `rdf:type` mit relativ wenig Zeichen geschrieben werden kann. Jedoch muss einer Präfixschreibweise immer eine Präfixdefinition vorangehen.

[1] Diese Information ist der freien Online-Enzyklopädie Wikipedia[4] entnommen worden, obwohl man in wissenschaftlichen Publikationen nicht aus Wikipedia zitieren sollte. Ich denke aber, dass man dann den Informationen aus Wikipedia vertrauen darf, wenn es sich um Artikel von grosser Tragweite handelt, und der Autor den Artikel skeptisch gegengeprüft hat. Beides ist hier der Fall.

Betrachtet man die Beispiel-URIs in Tabelle 2.1, kann man auf den Gedanken kommen, dass `http://www.ifi.unizh.ch/ddis/ont/movieont.owl#Jolie1a` unmöglich die offizielle URI für die bekannte Schauspielerin sein kann. Das ist richtig. Es handelt sich nicht um eine *offizielle* URI. Wenn der Fall eintreffen sollte, dass im *Semantic Web* andere URIs über die gleiche Ressource sprechen, dann können diese beiden URIs mit der Eigenschaft versehen werden, dass sie auf die gleiche Ressource verweisen. Dies wird mit dem `sameAs`-Schlüsselwort in OWL gemacht (Siehe Abschnitt 2.1.4).

2.1.2 XML und XML Schema

XML (*Extensible Markup Language*) ist eine Metasprache zur Definition anderer Sprachen. Die Charakteristiken von XML sind die öffnenden und schliessenden *Tags*, zwischen welchen die eigentlichen Informationen stecken (`<name>Ruth</name>`), die Attribute, die innerhalb dieser *Tags* definiert sind (`<name lang="de">Ruth</name>`) und die baumstrukturartige Verschachtelung (`<pers><name>Ruth</name><age>12</age></pers>`). Auf diese Art sind unzählige Datenformate und -sprachen definiert. Das allgegenwärtige HTML ist nur ein Beispiel davon.[2] Daneben existieren unzählige andere XML-konforme Sprachen. Dies ist deshalb möglich, da sich mit **XML Schema** die Syntax solcher Sprachen genaustens festlegen lässt. Auch die *Semantic-Web*-Techniken OWL und RDF treten meist in einer XML-konformen Schreibweise auf, obwohl sie auch in anderen Sprachen auftreten können.

2.1.3 RDF und RDF Schema

Mit **RDF** (*Resource Description Framework*) werden Ressourcen, die über eine URI identifiziert sind, näher beschrieben. Dies geschieht jeweils über Tripel, also über drei Elemente, wovon das erste als *Subjekt*, das zweite als *Prädikat* und das dritte als *Objekt* bezeichnet wird. Subjekt und Prädikat sind immer URIs während Objekte entweder URIs oder Literale sein können. Literale sind finale, für den Menschen interessante Werte wie z. B. Texte (`string`) oder Zahlen (`integer`, `float`, ...).

Einer mittels URI identifizierten Ressource wird also eine ebenfalls mittels URI identifizierte Eigenschaft zugeteilt. Diese wird mit einem Wert versehen. Ein Beispiel eines solchen RDF-Tripels: `bsp:hans bsp:vorname "Hans"`.[3] Dabei wird der Ressource `hans` der `vorname` Hans zugewiesen. Da als Objekte wiederum URIs stehen können, werden somit ganze Graphe aufgebaut. RDF-Tripel bilden die Grundlage für die Wissensbasen, die mit *Semantic Crystal* abgefragt werden.

Damit Web-Wissensbasen nicht nur in sich selbst durch eigene Prädikate definiert, sondern auch an gemeinsamen Elementen festgemacht sind, sind solche allgemeine Eigenschaften in RDF und vor allem in **RDF Schema** definiert. Die wichtigsten davon sind in Tabelle 2.2 aufgelistet.

Abbildung 2.2 aus Antonious Buch [3] zeigt eine graphische Darstellung einer Ontologie und von Instanzen. Die beiden Bereiche sind sehr schön voneinander getrennt, jedoch etwas unglücklich beschriftet. Die Trennung zwischen RDF und RDF Schema ist nämlich nicht so deutlich, wie dies die Abbildung suggeriert. Dennoch dient die obere Hälfte als Vorlage für diejenige graphische Ontologie-Repräsentation, die in *Semantic Crystal* verwendet wird (vgl. Abbildung 2.3). Zu-

[2] Da HTML historisch gewachsen ist, entspricht es nicht exakt der korrekten XML-Syntax. Beispielsweise wird der *Tag* des Zeilenumbruchs (`<br>`) nicht wieder geschlossen oder gewisse Attribute stehen nicht in Anführungszeichen (`<input type="text"readonly>`). Die XML-konforme Version heisst XHTML (`<br/>`, `<input type="text"readonly="readonly">`).

[3] Das genannte Beispiel eines RDF-Tripels ist in der sogenannten N3-Syntax verfasst. Diese ist leserfreundlicher als die XML-Syntax. Die Tripel sind dabei durch Leerschläge getrennt und enden mit einem Punkt. Weitere syntaktische Möglichkeiten können bei Berners-Lee [5] nachgelesen werden.

Präfix-URI	Beschreibung
`rdfs:Class`	Eine Klasse ist eine Sammlung von verschiedenen Instanzen. Alle Instanzen einer Klasse weisen die Eigenschaften auf, die für die Klasse definiert worden sind.
`rdf:type`	Definiert Klassen und ordnet Instanzen einer Klasse zu. Bsp: `bsp:mensch rdf:type rdfs:class.` (Mensch wird als Klasse definiert), `bsp:hans rdf:type bsp:mensch.` (Hans ist eine Instanz der Klasse Mensch).
`rdfs:subClassOf`	Bildet eine Taxonomie von Klassen. Bsp: `bsp:informatiker rdfs:subClassOf bsp:mensch.` Ein Informatiker ist auch ein Mensch, d. h. die Eigenschaften, die für die Klasse Mensch definiert worden sind (z. B. Vor- und Nachname), sind auch für die Subklasse Informatiker gültig.
`rdf:Property`	Definiert Eigenschaften. Eigenschaften sind diejenigen Ressourcen, die in RDF-Tripels typischerweise als Prädikate auftreten. Bsp: `bsp:vorname rdf:type rdf:Property.`
`rdfs:domain`	Ordnet einer Eigenschaft eine *Domain* zu. Das ist diejenige Klasse, die als Subjekt für diese Eigenschaft in Frage kommt. Bsp: `bsp:vorname rdfs:domain bsp:mensch.`
`rdfs:range`	Ordnet einer Eigenschaft einen Wertebereich (*Range*) zu. Das ist diejenige Klasse, die als Objekt für diese Eigenschaft in Frage kommt oder ein Datentyp, falls das Objekt ein Literal ist. Bsp: `bsp:vorname rdfs:range xsd:string.`
`rdfs:Resource`	Die oberste aller Ressourcen. Jede Ressource ist auch eine `rdfs:Resource`

Tabelle 2.2: Auswahl von wichtigen Ressourcen, die in RDF und RDF Schema definiert sind

dem wird in dieser Abbildung OWL nicht angezeigt, mit welcher sich erst eine richtige Ontologie beschreiben lässt.

2.1.4 Ontologien

Über den bisher genannten Schichten liegt die Ontologieschicht. Was mit RDF Schema schon ansatzweise möglich ist, wird mit OWL, dem Standard für Web-Ontologien [2], vervollständigt. Mit OWL-Tripeln lassen sich nach Belieben Wissensrepräsentationen erstellen. Solche Wissensrepräsentationen gab es zwar schon lange vor OWL, jedoch sind nun erstmals über das Web zugängliche Wissensbasen nutzbar. Damit rückt der Traum näher, dass Software-Agenten eigenständig auf Informationssuche gehen können. Abschnitt 2.2.1 geht näher auf den Begriff und die Problematik von Ontologien ein. Tabelle 2.3 ergänzt die Tabelle 2.2 um die wichtigsten OWL-Konzepte.

Mit OWL lässt sich auch auf Ontologieebene die wichtige Unterscheidung zwischen *ObjectProperties* und *DatatypeProperties* vornehmen. Während *ObjectProperties* eine URI als Wertebereich haben, haben *DatatypeProperties* ein Literal. Literale sind im Graphen, der durch die Tripel aufgespannt wird, immer Endknoten. Sie werden deshalb in *Semantic Crystal* gar nicht erst in die graphische Darstellung aufgenommen.

Eine OWL-Wissensbasis weist gewisse Ähnlichkeiten zu einer Datenbank auf. Eine dieser Parallelen ist die Ähnlichkeit der Abfragesprachen. Bei relationalen Datenbanksystemen hat sich SQL als De-facto-Standard etabliert, so dass die Hersteller von Abfragesprachen für OWL-Ontologien

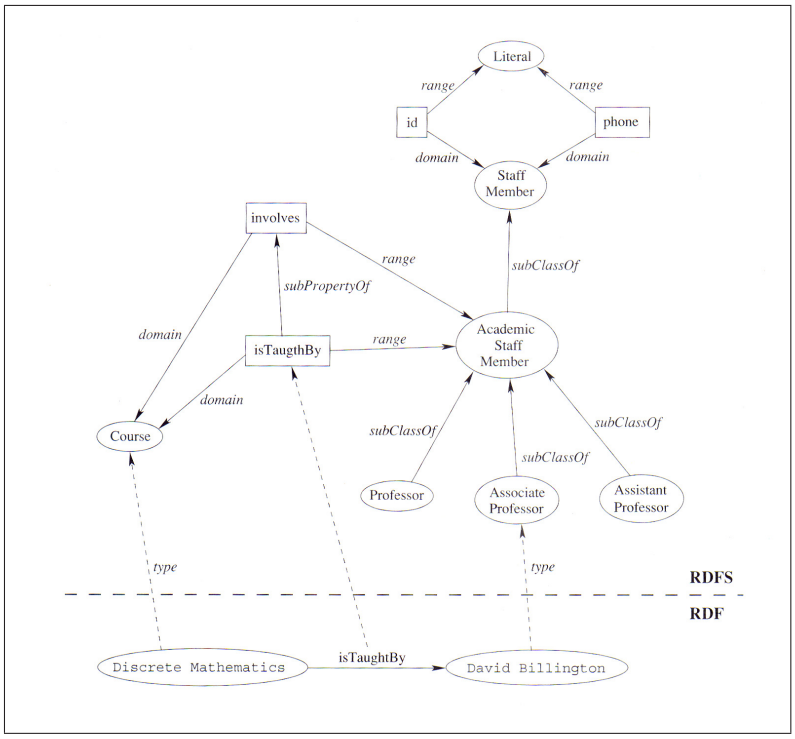

Abbildung 2.2: Graphische Darstellung eines RDF(S)-Graphen. [3] S.84

gut daran taten, eine SQL-ähnliche Syntax zu wählen. Diese Abfragesprache heisst SPARQL. Sie ist zwar noch keine W3C-Empfehlung[4], aber auf bestem Weg dorthin [6]. Eine detailliertere Analyse von SPARQL-Abfragen folgt in Kapitel 3.

2.1.5 Logic, Proof

Da sich der vorliegende Text vor allem um Ontologien, Wissensbasen und deren Abfrage dreht, will ich nicht zu viele Worte über diese zukünftigen *Semantic Web*-Schichten verlieren. Zu den **Logic** und **Proof**-Schichten gehören Logik-Regeln, die schon früh in der philosophischen Disziplin[5] erkannt und formalisiert worden sind.

Eine sehr einfache Logik-Regel ist beispielsweise die Bedingung $A \rightarrow B$. Eine sehr einfache Schlussfolgerung ist der sogenannte *Modus ponens*, der aus der Tatsache, dass A wahr ist und aus der genannten Bedingungsregel schlussfolgern kann, dass auch B wahr sein muss. $(A \rightarrow B), A \Rightarrow B$.

Die Besonderheit liegt hier aber nicht darin, diese Regeln und Schlussfolgerungsmechanismen in eine Software zu implementieren, sondern darin, sie in eine allgemeine Form zu bringen und somit auch austauschbar zu machen. *Semantic-Web*-Agenten sollen nämlich nicht nur lokal rechnen, sondern auch mit anderen Agenten kommunizieren und interagieren können. Auch Handelsgeschäfte zwischen Agenten sollen einmal möglich sein.

[4]Stand August 2006

[5]Der griechische Philosoph Aristoteles (384 – 322 v. Chr.) gilt als Vater der Logik. Er hat als einer der ersten die logischen Zusammenhänge systematisch untersucht. Der deutsche Philosoph Gottlob Frege (1848 – 1925) hat die Logik formal ausgedrückt und damit den Grundstein für die mathematische und informationstechnische Logikverarbeitung gelegt.

Präfix-URI	Beschreibung
`owl:Class`	Eine Subklasse von rdfs:Class.
`owl:sameAs`	Drückt aus, dass zwei verschiedene URIs die gleiche Ressource beschreiben.
`owl:differentFrom`	Drückt explizit aus, dass zwei verschiedene URIs zwei verschiedene Ressource beschreiben.
`owl:ObjectProperty`	Subklasse von `rdf:Property`. *ObjectProperty* hat als Wertebereich (`rdfs:range`) eine Klasse.
`owl:DatatypeProperty`	Subklasse von `rdf:Property`. *DatatypeProperty* hat als Wertebereich (`rdfs:range`) ein Literal.
`owl:inverseOf`	Definiert zwei *ObjectProperties* als zueinander invers. Die Eigenschaft `hasActor` ist z. B. invers zur Eigenschaft `isActorOf`.
`owl:maxCardinality`	Definiert Kardinalitäten. Für eine Instanz der Klasse *Mensch* soll z. B. die Eigenschaft *hatArme* genau zweimal vorhanden sein, d. h. die Kardinalität ist zwei. Neben `maxCardinality` gibt es selbstverständlich auch `minCardinality`.
`owl:disjointWith`	Drückt die Disjunktheit aus. Zwei Klassen sind disjunkt, wenn eine Instanz nicht zu beiden Klassen gleichzeitig gehören kann. In der Film-Wissensbasis sind beispielsweise `AnimatedMovie` und `FilmedMovie` disjunkt.

Tabelle 2.3: Auswahl von wichtigen Ressourcen, die in OWL definiert sind

Um solch komplexe Vorgänge durchzuführen, müssen *Semantic-Web*-Agenten entweder mit menschenähnlicher Intelligenz bestückt werden, oder wir Menschen müssen diese Vorgänge auf eine weniger komplexe Stufe herunterbrechen. Es müssen also Standards dieser Regelaustausch-, Kommunikations- und Handlungsprotokolle geschaffen werden.

Ansätze dazu sind *RuleML* [7], eine Sprache für Logik-Regeln in XML-Syntax und *Common Logic* [8], eine Sprache für den Austausch von Wissen, welche sich als Nachfolger des *Knowledge Interchange Format KIF* versteht.

2.1.6 Trust, Digital Signatures

Da die Struktur des *Semantic Web* sehr offen ist, muss bei heiklen und riskanten Daten- und Anwendungen sichergestellt werden, dass sowohl die herangezogenen Wissensbasen als auch die involvierten Agenten vertrauenswürdig sind. Hier werden zentrale Zertifizierungsstellen unumgänglich sein, welche die Vertrauenswürdigkeit einzelner Akteure fortlaufend überprüfen und publizieren.

Diese Arbeit benutzt als Beispiel eine Film-Wissensbasis. Ich kann Ihnen versichern, dass sie ausschliesslich korrekte Informationen enthält. Mehr an vertrauenswürdigen Massnahmen als mein Wort, kann ich zum jetzigen Zeitpunkt hierzu nicht bieten.

2.2 Begriffserklärung

Nachdem die *Semantic-Web*-Schichten beschrieben worden sind, sind noch die beiden Begriffe «Ontologie» und «Wissensbasis», welche im vorliegenden Text sehr häufig auftreten, präzise einzuführen und gegeneinander abzugrenzen.

2.2.1 Ontologie

«Was ist eine Ontologie?» Viele Informatiker antworten auf diese Frage mit der Definition von Tom Gruber: «Eine Ontologie ist eine Spezifikation einer Konzeptualisierung» [9]. Zwei Einwände kann man gegen diese Definition ins Feld führen. Erstens die philosophisch-belehrende, die sagt, dass es nur eine Ontologie gäbe, welches die Disziplin des Wesens alles Seienden sei. Und zweitens die skeptische, die sich nicht zufrieden gibt mit der Ersetzung des unverständlichen Begriffes *Ontologie* durch die zwei neuen «unverständlichen» Begriffe *Spezifikation* und *Konzeptualisierung*. Auf diese beiden Einwände wird in den nächsten Abschnitten eingegangen.

Ontologie bei den Philosophen

Die Philosophie kennt den Begriff *Ontologie* seit Anfang des 17. Jh.[6] und bezeichnet damit die Lehre des Seienden. Bei den alten Griechen war diese Disziplin auch als *Metaphysik* bekannt.

Ontologie als Seinswissenschaft gibt es deshalb nur eine. Die Ontologie untersucht das Wesen des Seienden, also dasjenige, das all demjenigen gemeinsam ist, welches ist bzw. existiert. Wir Informatiker können auch sagen, die klassischen Ontologen untersuchen das Wesen dessen, was wir `owl:thing` nennen. Natürlich wurde die Möglichkeit, klassische Ontologie überhaupt betreiben zu können, oft bestritten. Auch im 20. Jh. herrschte diese Meinung vor, und nur wenig wurde unter dem Namen Ontologie publiziert. Auch wenn die moderne Philosophie die klassische Ontologie nicht mehr aktiv betreibt, bleibt der Begriff unter Philosophen dennoch noch Jahrzehnte in seiner Ursprungsbedeutung vorhanden.

Ontologien bei Informatikern

Die Mathematiker und Informatiker bedienten sich aus der Terminologiekiste der Philosophen, und gaben dem Begriff «Ontologie» die Bedeutung einer *formalen Wissensrepräsentation* (vgl. Antoniou [3] S. 10). Wahrscheinlich wurde der Begriff als Metapher übernommen, da «Formale Wissensrepräsentation» weniger Pepp als «Ontologie» aufweist. Der Unterschied zur philosophischen Auffassung ist die Tatsache, dass mehrere Ontologien koexistieren und sich gar ergänzen können.

Eine grosse Frage ist aber auch, ob es grundsätzlich möglich ist, eine einzige, weltumfassende Ontologie zu bilden, in welcher das gesamte formal beschreibbare Wissen kohärent zusammengetragen ist. Nachdem in früheren Jahren diese Meinung existiert hatte und als Voraussetzung dafür genommen worden ist, dass Maschinen überhaupt menschenähnliche Intelligenz haben können, ist man von dieser Vorstellung wieder abgerückt.

Vor dem Hintergrund, dass die verschiedenen Ontologien nebeneinander existieren können, ist diese Tatsache vernachlässigbar. Natürlich ist es entscheidend, dass man Informationen aus verschiedenen Quellen zusammentragen kann, jedoch muss nicht zwingend jede Quelle kohärent eingebunden werden können. Widersprüchliche Angaben stellen also nicht gleich ganze Ontologien in Frage.

So wird beispielsweise in Abschnitt 2.4 eine Film-Ontologie vorgestellt. Sie beschreibt den Zusammenhang zwischen Filmen, Zeichentrickfilmen, Schauspieler, Ländern, Regisseuren und Verleihfirmen. Nun könnten mir fremde Menschen auf ihren Webseiten Informationen über Filme anbieten, die in meiner Wissensbasis nicht vorhanden sind. Zwei Informationsquellen liessen sich somit zu einer einzigen vereinen.

[6] Alle *philosophischen Facts* im vorliegenden Text sind dem Philosophie-Lexikon [10] des Rowohlt-Verlages entnommen. Ich erlaube mir die Freiheit, die darin enthaltenen Texte ohne Konsultierung der Originalschriften wiederzugeben, da philosophische Texte für einen Informatiker doch bisweilen in zu komplexer Sprache gehalten sind.

Ebenso könnte jemand eine andere, bessere Film-Ontologie erstellen, und Informationen über Filme in dieser neuen Ontologie darstellen. Auch in diesem Fall stehe ich nicht vor unlösbaren Schwierigkeiten. Ein einmaliges Mapping zwischen diesen beiden Filmontologien stellt sicher, dass sich Informationen aus der einen in die andere Ontologie übertragen lassen.

Damit wurde gezeigt, dass das *Semantic Web* so ausgelegt wird, dass verschiedene Ontologien zum gleichen oder ähnlichen Gegenstandsbereichen existieren können, und dass es nicht eine *richtige* Ontologie pro Gegenstandsbereich gibt, wie es auch keine allumfassende Welt-Ontologie geben wird. Es wäre beispielsweise sehr komplex oder gar unmöglich, diese Film-Ontologie in eine alles umfassende Welt-Ontologie einzugliedern. Da würden rasch Widersprüche auftauchen.

Betrachten wir noch kurz die Definition von Tom Gruber, nach welchem eine Ontologie eine *Spezifikation einer Konzeptualisierung* sei. Die Konzepte dieser *Konzeptualisierung* sind also Klassen und Beziehungen zwischen Klassen. Mit *Spezifikation* ist die formale und explizite Beschreibung gemeint. Statt *Spezifikation einer Konzeptualisierung* könnte man auch sagen, eine Ontologie sei die Struktur von Klassen und deren Beziehungen und Eigenschaften.

2.2.2 Wissensbasis

Eine Ontologie im engen Sinn bezeichnet ausschliesslich die Struktur von Konzepten und deren Beziehungen und Eigenschaften. Falls eine Ontologie aber noch mit Daten zu konkreten Instanzen angereichert ist, sprechen wir von einer Wissensbasis (engl. *Knowledge Base*).

In diesem Text wird deshalb das Wort *Wissensbasis* benutzt, wenn man davon ausgehen kann, dass auch Instanzen vorhanden sind. Mit dem Wort *Film-Ontologie* ist also nur die Struktur gemeint. Möchte ich betonen, dass daneben noch 1000 Filme und 3000 Schauspieler in OWL-Form angegeben sind, dann spreche ich von der *Film-Wissensbasis*. Das Wort Ontologiestruktur ist demzufolge ein Pleonasmus.

2.3 Voraussetzungen für Semantic Crystal

Für Abfragen mit der Software *Semantic Crystal* ist es notwendig, dass sich alle Informationen, sowohl Ontologie- als auch Instanzdaten, in einer einzigen Datei befinden. Dies widerspricht natürlich dem Ziel des *Semantic Web*. Statt dass ein Mensch in einer einzigen Datei auf Informationssuche geht, sollte ein Agent an verschiedenen Orten nach Informationen suchen. Für das Filmbeispiel könnte man sich vorstellen, dass die Ontologie an einem Ort definiert ist, und die einzelnen Filmehersteller bzw. -verleiher jeweils Informationen für ihre Filme bereitstellen, indem sie Bezug auf die Ontologie nehmen. Und dort würde dann der *Semantic-Web*-Agent fündig.

Doch bevor die Filmverleiher genug motiviert sind, Informationen dezentral zu erstellen und zu publizieren, müssen Benutzer vorhanden sein, die nach solchen Informationen suchen, und diese benötigen Tools wie *Semantic Crystal*, um bestehende Web-Wissensbasen abzufragen.

Es gibt noch einen anderen Grund, weshalb *Semantic Crystal* nicht fähig sein muss, Informationen aus mehreren Dateien zusammenzusuchen. Denn sobald sinnvolle dezentrale Informationsquellen vorhanden sind, werden auch Dienste entstehen, die diese Informationen aggregieren und neue, umfassendere Wissensbasen zur Verfügung stellen. Diese sind wiederum ideal für den Gebrauch mit *Semantic Crystal*.

2.4 Beispiel: Film-Wissensbasis

Im Rahmen dieser Diplomarbeit wurde eine Film-Wissensbasis erstellt, die im Text oft als Beispiel herangezogen wird und welche in *Semantic Crystal* auch standardmässig geladen wird.[7]

2.4.1 Warum eine neue Ontologie?

Im Web sind zwar einige Ontologien vorhanden, allerdings weisen die meisten gewisse Unzulänglichkeiten für den Gebrauch mit *Semantic Crystal* auf. Der verbreitetste Mangel ist das Fehlen von Instanzen. Das soll nicht heissen, dass Ontologien ohne zugehörige Instanzen unbrauchbar wären.[8] Solche *Struktur-Ontologien* können durchaus sinnvoll sein, allerdings nicht für unsere Zwecke. Im Rahmen dieser Arbeit betrachten wir nur Abfragen von Ontologien, welche mit Instanzen angereichert sind.

Neben dem Fehlen von Instanzen sind auch andere Unzulänglichkeiten anzutreffen. Oft sind die Daten sehr semi-strukturiert, d. h. einige Instanzen weisen Eigenschaften auf, die in der Ontologie nicht definiert worden sind. Auf die Probleme, die deswegen auftreten können, möchte ich im Abschnitt 6 eingehen.

Andere Wissensbasen zeichnen sich durch einen extremen Umfang aus. Einige sind nur zu Testzwecken erstellt worden, und umfassen nur sehr wenige Klassen und Instanzen, andere wiederum umfassen Tausende von Klassen und Hunderttausende von Instanzen. Wiederum andere beschränken sich auf ein Spezialgebiet, welches den meisten Menschen nicht zugänglich ist. Mit einer Wissensbasis über die Gene einer Maus, können beispielsweise nur Experten etwas anfangen. Deshalb wurde für *Semantic Crystal* eine Wissensbasis eines Gebietes gewählt, mit welchem die meisten Leute vertraut sind, und so können sie auch das Verhalten und die Resultate von *Semantic Crystal* besser verifizieren.

Natürlich ist diese Film-Wissensbasis nicht für den alleinigen Gebrauch in *Semantic Crystal* gedacht. Sie kann gerne für andere Zwecke verwendet werden.

2.4.2 Der Aufbau der Film-Ontologie

Die gesamte Film-Ontologie ist im XML-Format mit ein paar Beispielsinstanzen im Anhang B zu finden. Die Film-Ontologie besteht aus den zwei Taxonomien `Movie` und `Person`. `Movie` besitzt die Subklassen `FilmedMovie` und `AnimatedMovie`. Der Unterschied zwischen diesen beiden liegt darin, dass ein `FilmedMovie` zusätzlich zu den geerbten Attributen von `Movie` noch Schauspieler hat, welche bei `AnimatedMovie` selbstverständlich fehlen. `Person` besitzt die Subklassen `Actor` und `Director`.

Zusätzlich gibt es folgende Klassen: `Distributor`, `MovieGenre`, `Country`. Diese Klassen sind untereinander mit *ObjectProperties* verbunden, wie in Abbildung 2.3 ersichtlich ist.

Die Wissenbasis enthält 997 Instanzen von `FilmedMovie` und 25 Instanzen von `Animated-Movie`. Es sind dies diejenigen Filme, die im Jahr 2005 erschienen sind, im Jahr 2005 erstmals in den Schweizer Kinosälen gezeigt worden sind oder solche, die einen der folgenden Schauspieler oder Regisseuren enthalten:

- Angelina Jolie

[7] Die Daten für die Film-Wissensbasis wurden von der Firma Cinergy AG (`http://www.cinergy.ch`) zur Verfügung gestellt.

[8] In vielen Fällen kann es sogar sein, dass Instanzen unnötig sind, und nur die Struktur wichtig ist. Beispielsweise könnte in einer (zugegeben etwas einfachen) Anatomie-Ontologie definiert sein, dass jeder Mensch zwei Arme, und jeder Arm fünf Finger hat. Mensch, Arm und Finger seien dabei Klassen. Zumindest für die Klassen Arm und Finger sind Instanzdaten sinnlos.

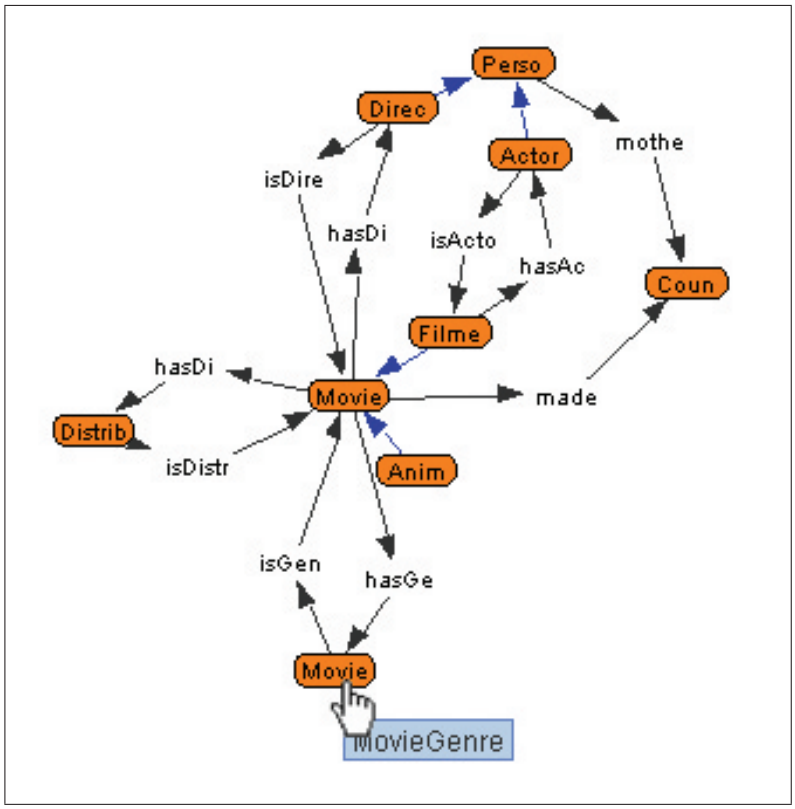

Abbildung 2.3: Graphische Darstellung der Film-Ontologie. Orange Knoten stellen OWL-Klassen dar, blaue Pfeile Subklassen-Beziehungen und schwarze Pfeile ObjectProperties. (Quelle: Semantic Crystal)

- Audrey Tautou
- Bruno Ganz
- Franka Potente
- Gérard Depardieu
- Jean Reno
- Johnny Depp
- Monica Bellucci
- Nicole Kidman
- Roberto Benigni

- David Lynch
- Fatih Akin
- Lars von Trier
- Quentin Tarantino
- Spike Lee

- Steven Spielberg
- Pedro Almodóvar

Mit dieser Auswahl wollte ich nicht (nur) einige Schauspieler und Regisseure hervorheben, sondern sinnvolle Abfragen ermöglichen, wenn man nach allen Filmen eines bestimmten Schauspielers suchen möchte. Zum Beispiel bringt die Suche nach allen Filmen von Jack Nicholson keinen einzigen Treffer hervor, da keiner seiner Filme im Jahre 2005 gemacht worden ist, und ich seine Filme im Unterschied zu den aufgelisteten Personen nicht speziell berücksichtigt habe. Deshalb sucht man mit Vorteilen nach Filmen von einer der oben aufgelisteten Personen.

Neben den *ObjectProperties* haben die Instanzen aus der Film-Wissensbasis auch *DatatypeProperties*, also Eigenschaften, die ein Literal als Wert haben.

Ein `Movie` hat folgende *DatatypeProperties*: `movieTitle`, `movieRating` (wobei nur ein Teil der Filme ein Rating haben) und `publYear`. Eine `Person` hat folgende *DatatypeProperties*: `firstName` und `familyName`. Zudem besitzen die Instanzen von `Country` einen `countryName` und die `Distributors` einen `companyName`.

3

Die Elemente einer SPARQL-Abfrage

In diesem Kapitel wird die OWL-Abfragesprache SPARQL analysiert. Ziel ist es, abstrakte Elemente aus typischen Abfragen zu extrahieren, welche in die Software *Semantic Crystal* eingebaut werden können. Diese Elemente werden dem Benutzer die Formulierung solcher Abfragen erleichtern.

3.1 Anforderungen an die Verfasser von SPARQL-Abfragen

OWL-Wissensbasen werden mit der Sprache SPARQL abgefragt. Das folgende Beispiel zeigt, dass die Formulierung einer solchen Abfrage nicht trivial ist. Wenn man die Film-Wissensbasis nach allen Filmtiteln durchsuchen möchte, in welchen Angelina Jolie mitspielt, dann würde folgende SPARQL-Abfrage zum gewünschten Resultat führen.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX film:<http://www.ifi.unizh.ch/ddis/ont/movieont.owl#>
SELECT ?title
WHERE {
  ?movie rdf:type film:FilmedMovie .
  ?movie film:movieTitle ?title .
  ?movie film:hasActor ?actor .
  ?actor film:familyName ?name .
  FILTER regex(?name, "Jolie") .
}
```

Eine solche in SPARQL formulierte Abfrage ist zwar einigermassen lesbar und verständlich, aber deren Formulierung verlangt höchste Anforderungen. Ein Verfasser solcher SPARQL-Abfragen muss insbesondere über zwei wichtige Anforderungen verfügen.

- **SPARQL-Syntaxkenntnisse.** Die Tatsache, dass URIs in *grösser-als*-Klammern stehen, Variablen mit einem Fragezeichen[1] beginnen, Abfragetripel mit einem Punkt enden und die Restriktion mit den Stichwörtern `FILTER` und `regex` eingeleitet werden, ist Grund dafür, dass solche Abfragen nur von SPARQL-Experten auf Anhieb korrekt formuliert werden können.

[1] Alternativ zum Fragezeichen kann auch das Dollarzeichen zur Auszeichnung von SPARQL-Variablen benutzt werden.

- **Kenntnisse über vorhandene Ontologie.** Menschen, die zum ersten Mal mit einer bestimmten Ontologie in Berührung kommen, müssen sich zuerst Kenntnisse über deren Namen verschaffen. Dass in unserem Beispiel der Titel eines Filmes durch die Eigenschaft `movieTitle` gefunden werden kann, ist weder Naturgesetz noch Konvention. Stattdessen könnte die Eigenschaft auch `hasFilmTitle` oder `leTitreDe` heissen.

Semantic Crystal soll einem Benutzer die Möglichkeit bieten, Web-Wissensbasen abzufragen ohne über ausgeprägte Kenntnisse in der SPARQL-Syntax oder in der vorhandenen Ontologie zu verfügen.

3.2 Grundelemente in der Bedingungsliste

Im SPARQL-Beispiel des letzten Abschnitts ist zu erkennen, dass ein Grossteil der Abfrage zwischen den geschwungenen Klammern nach dem Stichwort `WHERE` formuliert ist. Zwischen diesen Klammern werden verschiedene Bedingungen für die Abfrage aufgestellt. Ich werde die typischen dieser Bedingungen einzeln auflisten und zeigen, was für eine Charakteristik sie aufweisen, um zu den gesuchten Grundelementen zu kommen.

3.2.1 Typeinschränkung, ?x rdf:type ?y

Mit der Typeinschränkung `?x rdf:type ?y` (Bsp: `?movie rdf:type film:FilmedMovie`) wird sichergestellt, dass als ?x nur Elemente der Klasse ?y in Betracht gezogen werden. Eine solche Typeinschränkung muss in fast jeder Abfrage vorgenommen werden, obwohl es bei kleinen Wissensbasen oft ohne Konsequenzen bleibt, wenn die Typeinschränkung weggelassen wird. Wenn man z. B. nach Ressourcen sucht, die den Namen «Jolie» enthalten, wird bei der relativ kleinen Film-Wissensbasis sowieso nur ein Eintrag zur Schauspielerin Angelina Jolie gefunden, doch wenn ich den gleichen Vorgang mit einer grösseren Wissensbasis und einer komplexeren Ontologie ausführe, könnten Ressourcen anderer Klassen auftauchen, die den Namen «Jolie» tragen, wie beispielsweise eine Blume, ein Politiker oder eine Stadt.

Von grosser Wichtigkeit wird die Typeinschränkung auch bei einer Agentensuche, der nicht nur eine Ontologie durchsucht, sondern das ganze *Semantic Web*. Falls sich der Agent für eine Aufgabe ausreichend Zeit nehmen darf, gleicht das Resultat der Abfrage ohne Typeinschränkung (also z. B. nur `?a rdfs:label "Jolie"`) der heutigen Volltextsuche. Denn es werde alle möglichen Ressourcen gefunden, welche das Label «Jolie» tragen. Die Stärke des *Semantic Web* liegt also darin, dass man Typeinschränkungen vornehmen kann und deshalb bessere Resultate erhält als die einer Volltextsuche.

3.2.2 Variablennamen, Mehrfachvorkommen von Variablen

Wenn wir Typeinschränkungen vornehmen, wird eine Variable auch mehrmals vorkommen.

```
?movie rdf:type film:FilmedMovie .
?movie film:movieTitle ?title .
```

Diese Feststellung ist ein wichtiger Punkt für die Zusammenstellung der Abfrage. Weder darf man in der zweiten Zeile des obigen Beispiels die Variable anders nennen, noch darf man im umgekehrten Fall – wenn man also zwei verschiedene Instanzen des gleichen Typs abfragen möchte – den gleichen Variablennamen benutzen.

Auch die Namenswahl der Ausgabevariablen ist von grosser Wichtigkeit. Im obigen Beispiel wurde dazu `?title` genommen, was dazu führt, dass die Resultate mit diesem Namen betitelt werden, sofern überhaupt welche gefunden werden.

Die Vergabe der Variablennamen ist eine für den Menschen äusserst leichte Aufgabe, nicht aber für eine Software. Während es bei der Eigenschaft `movieTitle` noch einfach wäre, zum passenden Variablennamen `?movieTitle` zu gelangen, ist es bereits schwieriger, wenn die Eigenschaft keinen passenden Namen vorgibt. Zum Beispiel kann aus der Allerweltseigenschaft `rdfs:label` kein passender Variablenname herausgepickt werden, ohne in Konflikt mit einer anderen Variable zu geraten, die sich ebenfalls auf eine `label`-Eigenschaft bezieht, aber von einer anderen Klasse ausgeht.

Wenn also eine Software einen Variablenname automatisch bestimmen soll, sollte dieser aus dem Klassennamen und aus dem Eigenschaftsnamen bestehen, oder zumindest aus wiedererkennbaren, eindeutigen Teilen davon.

3.2.3 Filter und Regex

```
FILTER regex(?name, "Jolie") .
```

Diese Zeile zeigt eine weitere wichtige Einschränkung. Obwohl das *Semantic Web* zwar erweiterte Suchmöglichkeiten neben der Volltextsuche bietet, bleibt die Suche nach Wörtern bzw. Teilen von Wörtern ein sehr wichtiger Bestandteil.

Ein normales Filtern (z.B. `FILTER (?age > 18)`) kann nur für boolsche Bedingungen angewendet werden, also für Bedingungen, die einen boolschen Operator wie = oder < enthalten und entweder *true* oder *false* zurückliefern.

Beim Filtern nach Teilen von Wörtern kommt das *Pattern Matching* mit *Regular Expressions* (kurz: *Regex*) ins Spiel. Dabei übernimmt *Regex* die Funktion des boolschen Operators. Er liefert für einen Eintrag in der Wissensbasis dann *true* zurück, falls dieser dem Muster entspricht, das der Benutzer vorgibt (im obigen Beispiel also `"Jolie"`).

In der weiten Welt der Informatik gibt es verschiedene Definitionen, wie genau *Regex*-Muster zu formulieren sind. SPARQL greift auf diejenige von X-Path[2] zurück. Das Muster `"Jolie"` würde unter anderem bei folgenden Werten *true* liefern: `"Jolie"`, `"Jolieqpqp"` oder auch `"KurtJolie"`. Es müssen also keine *Wildcards* eingegeben werden, um SPARQL mitzuteilen, dass hinter und vor dem Muster noch weitere Zeichen stehen dürfen. Dennoch gibt es viele solche Sonderzeichen. Einige davon sind in Tabelle 3.1 eingetragen.

Muster	Bemerkungen	Beispiele gültiger Zeichenfolgen
`^muster`	nur am Beginn der Zeichenfolge	`"muster", "mustermann"`
`muster$`	nur am Ende der Zeichenfolge	`"muster", "strickmuster"`
`m{2}?`	Zeichen kommt genau 2mal vor	`"mm", "kamm", "hammer"`
`m{2, 3}?`	Zeichen kommt mind. 2mal, max. 3mal vor	`"mm", "mmm", "kammmass"`
`m*?`	Zeichen kommt nie oder mehrmals vor	`"schere", "kamm"`
`m+?`	Zeichen kommt einmal oder mehrmals vor	`"dem", "kamm"`

Tabelle 3.1: Auswahl wichtiger Regex-Muster, die in SPARQL angewendet werden können

[2]Es wird die Funktion `fn:matches($input, $pattern)` aufgerufen. Siehe *http://www.w3.org/TR/xpath-functions*

3.2.4 Optionale Bedingungen

Auch dem unscheinbaren Parameter *optional* kommt eine grosse Bedeutung zu. Angenommen wir wollen in der Film-Wissensbasis von denjenigen Filmen, in welchen Angelina Jolie mitspielt, den Titel und das Rating abfragen. Auf eine Abfrage mit `SELECT ?title ?rating` würden dabei nur die Angaben zu denjenigen Filmen geliefert, die auch ein solches Rating haben. SPARQL ist nämlich auf die Art ausgerichtet, dass alle RDF-Tripel, nach denen eine Bedingung gestellt wird, auch als Tripel vorkommen müssen. In einer Datenbank wäre im Feld Rating bei einem Film ohne Bewertung dennoch ein Wert eingetragen (z.B. `"0"`, `"No Rating"` oder `null`). Bei RDF-Tripeln fehlt dieser Eintrag.

Deshalb sind solche `OPTIONAL`-Bedingungen von grossem Vorteil. Falls ein RDF-Tripel vorhanden ist, das die gestellte Bedingung erfüllt, wird das Resultat ausgegeben. Existiert kein solches RDF-Tripel, wird ein leeres Feld zurückgegeben.

```
?movie film:movieTitle ?title .
OPTIONAL{ ?movie film:movieRating ?rating . }
```

Bei dieser Abfrage könnte folgendes Resultat herauskommen.

```
?title          ?rating
Walk the Line   43.557
Cool Money
Shopgirl
Match Point     40.763
```

Wäre die Rating-Bedingung nicht `OPTIONAL`, fielen die beiden Filme ohne Bewertung («Cool Money», «Shopgirl») aus der Liste.

3.3 Grundelemente ausserhalb der Bedingungsliste

Neben den SPARQL-Elementen innerhalb der Bedingungsklammern, gibt es auch ausserhalb davon einige wichtige Elemente.

3.3.1 Projektion, SELECT

Unmittelbar nach dem Stichwort `SELECT` folgen diejenigen Variablen, die im Resultat ausgegeben werden sollen. Sollen alle Variablen ausgegeben werden, die in der Abfrage vorkommen, reicht dazu ein «`SELECT *`». Jedoch treten typischerweise auch Variablen auf, die lediglich dazu verwendet werden, bestehende RDF-Tripel zu verbinden (vgl. 3.2.2). Deshalb ist es sinnvoll, nur diejenigen Variablen auszugeben, nach denen explizit gefragt ist.

Im Datenbank-Jargon wird dieser Vorgang *Projektion* genannt. Statt alle Informationen auszugeben, werden nur gewisse Spalten projiziert.

3.3.2 Reihenfolge festlegen, Resultate beschränken

Häufig sollen die Suchresultate sortiert sein. Dies wird in SPARQL mit den `ORDER-BY`-Stichwörtern gemacht. Dabei wird die Variable, nach welcher sortiert werden soll, zusammen mit dem Stichwort `ASC` bzw. `DESC` für auf- bzw. absteigende Sortierfolge mitgegeben.

Mit dem Stichwort `LIMIT` wird die Anzahl Resultate beschränkt. Wird z. B. in der Einwohnerdatenbank nach Einwohnern der Schweiz gesucht, so spuckt SPARQL eine Liste mit sieben Millionen Einträgen aus. Ist man allerdings nur an den zehn höchstgewachsenen interessiert, reicht es, das Limit auf 10 zu setzen.

Das Gegenstück zu `LIMIT` ist `OFFSET`. Damit wird festgelegt, ab dem wievielten Resultat SPARQL zu zählen beginnt. Wird bei der Suche nach den höchstgewachsenen Schweizern `LIMIT 10 OFFSET 5` angegeben, erscheinen die Leute auf den Rängen 6 bis 15, da die ersten 5 ausgelassen werden. Diese Funktion ist bei der *Blätternfunktion* wichtig, wenn nur einige Resultate anzeigt werden und auf die Ausgabe der darauffolgenden Resultate gesprungen werden kann.

3.3.3 Verschiedene Formen von Abfrageresultaten

Bisher haben wir nur Abfragen vom Typ `SELECT` in Betracht gezogen. Daneben erlaubt SPARQL noch drei Typen von Abfragen, die alle nach dem jeweiligen Abfrage-Keyword benannt werden können: `CONSTRUCT`, `ASK` und `DESCRIBE`.

Bei `SELECT`-Abfragen ist das zurückgelieferte Resultat tabellenförmig. In einer Zeile steht jeweils eine Lösung, in einer Spalte alle Werte für eine bestimmte SPARQL-Variable.

Es lohnt sich, einen Blick auf die anderen drei Abfragearten zu werfen, um festzustellen, was neben `SELECT` auch noch möglich ist, und deren möglichen Anwendungsgebiete anzusprechen.

CONSTRUCT-Abfragen

Bei `CONSTRUCT`-Abfragen gibt der Benutzer eine Graph-Vorlage mit, mit welcher SPARQL einen neuen Graph konstruiert. Damit lassen sich z. B. Mappingvorgänge durchführen. Das folgende Beispiel zeigt, wie aus Inhalten der Film-Wissensbasis und einer `CONSTRUCT`-Abfrage eine neue Wissensbasis mit neuen URIs erstellt werden kann.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.ifi.unizh.ch/ddis/ont/movieont.owl#>
PREFIX neu: <http://www.ifi.unizh.ch/neu#>
CONSTRUCT {
  ?movieURL neu:titel ?title .
  ?movieURL neu:jahr ?year .
}
WHERE {
  ?movieURL rdf:type :Movie.
  ?movieURL :publYear ?year .
  ?movieURL :movieTitle ?title .
}
```

`CONSTRUCT` ist eine interessante Nutzungsmöglichkeit von SPARQL. Um einen neuen RDF-Graphen zu konstruieren, muss nicht der Umweg über `SELECT`-Abfrage und Herbeiziehung einer anderen Programmiersprache eingeschlagen werden.

Die `CONSTRUCT`-Möglichkeiten gehen aber über die Anforderungen für *Semantic Crystal* hinaus. Wir wollen schliesslich Wissensbasen abfragen und dazu reicht `SELECT` aus. Ausserdem stehen momentan zuwenig solche Wissensbasen zur Verfügung. Der Wunsch, Informationen zusammenzutragen und zu neuen gesamtheitlichen Informationen zu machen, wird erst dann sinnvoll, wenn auch genügend primäre Informationsquellen zur Verfügung stehen.

ASK-Abfragen

Abfragen des Typs `ASK` sehen wenig nützlich aus. Als Resultat wird das Wort *yes* zurückgeliefert, wenn mindestens eine Abfragelösung existiert, das Wort *no*, wenn keine davon existiert. Ansonsten ist die Syntax gleich wie diejenige des Bedingungsteils von `SELECT`-Abfragen.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.ifi.unizh.ch/ddis/ont/movieont.owl#>
ASK {
  ?movie rdf:type :Movie.
  ?movie :publYear ?year .
  FILTER (?year = 1984) .
}
```

Das obige Beispiel gibt *yes* zurück, falls ein Film aus dem Jahre 1984 existiert, und ansonsten *no*. Dies ist aus *Performance*-Gründen eine wichtige Funktion, da die SPARQL-Maschine bei der ersten gefundenen Lösung die Suche abbrechen kann.

Für unsere Zwecke muss aber auch dieser Typ von Abfrage nicht benutzt werden. Wir können weiterhin `SELECT`-Abfragen machen und je nachdem, wieviele Abfragelösungen gefunden worden sind, als Mensch erkennen, ob die nachgefragten RDF-Tripel vorhanden sind oder nicht.

DESCRIBE-Abfragen

«Das `DESCRIBE`-Merkmal von SPARQL ist ein absichtlich uneingeschränktes Abfragemerkmal», schreibt die *RDF Data Access Working Group* des *W3C*, die SPARQL entwickelt [6]. Damit ist gemeint, dass innerhalb von SPARQL nicht definiert ist, wie ein Resultat einer `DESCRIBE`-Abfrage aussehen soll. Der jeweils eingesetzten SPARQL-Prozessor entschiedet über das Aussehen des Resultats einer solchen Abfrage.

Zurückgegeben wird aber auf jeden Fall ein Graph, der eine Ressource beschreibt. Ein Beispiel:

```
PREFIX film: <http://www.ifi.unizh.ch/ddis/ont/movieont.owl#>
DESCRIBE film:m8403
```

Damit wird der SPARQL-Prozessor angehalten, die Ressource «m8403» zu beschreiben, was dem Film *The Da Vinci Code* entspricht. Wie das Resultat genau aussieht, kann nicht vorhergesagt werden. Jedenfalls sieht es ähnlich aus wie das Resultat der folgenden `SELECT`-Abfrage.

```
PREFIX film: <http://www.ifi.unizh.ch/ddis/ont/movieont.owl#>
SELECT ?predicate ?object
WHERE{
  film:m8403 ?predicate ?object
}
```

Es können aber auch weitere Informationen zurückgegeben werden, die über die erste Kante des Graphen hinausgehen. Auch diese vierte Abfragemöglichkeit bleibt wegen ihrer Undefiniertheit zumindest vorerst für *Semantic Crystal* uninteressant. Wir beschränken uns von dieser Stelle an nur auf `SELECT`-Abfragen.

3.4 TORC: Die Grundelemente von SPARQL-Abfragen

SPARQL-Abfragen können aus vier Grundelementen zusammengestellt werden: *Token*, *Output*, *Restriction*, *Connection*. Diese Nomenklatur hat sich bei der Entwicklung von *Semantic Crystal* ergeben. Nicht alle möglichen SPARQL-Abfragen können auf diese vier Elemente zurückgeführt

werden, aber zumindest ein Grossteil der typischen Abfragen. Die vier Grundelemente können auch TORC genannt werden, was sich aus ihren Anfangsbuchstaben herleitet.

Nachfolgend werden die TORC-Elemente einzeln beschrieben.

3.4.1 Token

Tokens sind OWL-Klassen, die für eine Abfrage benutzt werden. Wichtig ist aber die Unterscheidung zwischen Klassen und *Tokens*. Falls Informationen über Instanzen einer Klasse abgefragt werden wollen, wird aus dieser Klasse ein *Token* gemacht. Pro Klasse können auch mehrere *Tokens* existieren, nämlich dann, wenn zwei verschiedene Instanzen derselben Klasse gleichzeitig in die Abfrage übernommen werden. An dieser Stelle ein erhellendes Beispiel: Wir wollen die Nachnamen aller Schauspieler abfragen, welche schon mit Monica Bellucci zusammengearbeitet haben.[3]

```
SELECT ?name
WHERE{
  ?actor1 film:familyName ?name
  ?actor1 film:actorOf    ?movie
  ?actor2 film:actorOf    ?movie
  ?actor2 film:familyName "Bellucci"
}
```

Dieses vereinfachte[4] Beispiel zeigt auf, dass wir zwei *Tokens* für Schauspieler haben müssen. Das *Token*, das sich in der Variable `?actor2` zeigt, wird so eingeschränkt, dass es auf den Namen «Bellucci» hört und dass es im Film `?movie` mitspielt. Das andere *Token*, das sich in der Variable `?actor1` verbirgt, ist ebenfalls von der Klasse `Actor` und es spielt ebenfalls im Film `?movie` mit. Von diesem *Token* `?actor1` wird nun der Name gesucht.

Wir brauchen also zwei *Tokens* der Klasse `Actor`, nämlich `?actor1` und `?actor2`. Sie repräsentieren pro Resultat zwei verschiedene Instanzen. Von der Klasse `Movie` hat es in diesem Beispiel ebenfalls ein *Token*. Allerdings reicht hier ein einziges.

3.4.2 Output

Output ist erwartungsgemäss dasjenige Element, das den Benutzer eigentlich interessiert. Konkret handelt es sich dabei um Werte von *DatatypeProperties* (also um Literale). Im obigen Beispiel verbirgt sich hinter `?name` ein solches *Output*-Element.

3.4.3 Restriction

Restriction ist das Schwesternelement von *Output*, denn es handelt sich dabei ebenfalls um Werte von *DatatypeProperties*. Jedoch möchte der Benutzer diese Werte nicht anzeigen, sondern einschränken. Im obigen Beispiel war die Einschränkung des Namens auf «Bellucci» eine solche *Restriction*.

Von derselben Eigenschaft können gleichzeitig *Output* und *Restriction* zum Einsatz kommen. Beispielsweise könnte ein Benutzer an den Titeln aller Filme interessiert sein, in welchen das Wort

[3] Folgende Schauspieler der Film-Wissensbasis haben schon mit Monica Bellucci zusammengearbeitet: Vincent Cassel, James Caviezel, Gérard Depardieu, Morgan Freeman, Gene Hackman, Bruce Willis und 19 weitere.

[4] Die Vereinfachung soll dem besseren Verständnis dienen. Für eine korrekte Abfrage müssten auch die Präfixdefinitionen und ev. die Typeinschränkungen miteinbezogen werden.

«Love» vorkommt.[5] Dabei benutzt er die Eigenschaft `movieTitle` sowohl als *Output* als auch als *Restriction*.

3.4.4 Connection

Das letzte Element heisst *Connection* und hält die *Tokens* mit *ObjectProperties* zusammen. `actorOf` ist im obigen Beispiel ein solches *ObjectProperty*, das als *Domain* die Klasse `Actor` und als *Range* die Klasse `FilmedMovie` hat. *ObjectProperties* werden in Abfragen nicht dazu benutzt, um etwas auszugeben. Schliesslich bestehen die Werte von *ObjectProperties* nur aus URIs.

3.4.5 Das TORC-Fazit

Mit den vier TORC-Elementen kann ein breiter Benutzerkreis leicht und intuitiv SPARQL-Abfragen zusammenbauen. Wie das graphisch aussehen kann, wird in Kapitel 4 erläutert. Anzufügen ist aber auch, dass SPARQL mehr zu leisten imstande ist, als sich durch diese vier Elementen ausdrücken lässt. Z. B. kann ein Informatiker an URIs interessiert sein, und nachfragen: Welche URI hat der Film «King Kong»[6]. Oder er interessiert sich für die Klasse, die zu einer URI gehört.[7] Diese Art von Abfragen sind zwar typisch für Experten, nicht aber für normale Benutzer. Deshalb sind die vier TORC-Elemente doch sehr brauchbar als Baukastenelemente für SPARQL-Abfragen.

[5] In folgenden Filmen der Film-Wissensbasis kommt das Wort «Love» vor: Must Love Dogs, My Summer of Love, A Lot Like Love, Love and Hate.

[6] Folgende Abfrage gibt die URI von «King Kong» zurück: `SELECT ?x WHERE ?x film:movieTitle „King Kong“ .`

[7] Folgende Abfrage gibt eine Liste aller URIs und den dazugehörigen Klassen wieder: `SELECT ?uri ?class WHERE ?uri rdf:type ?class . ?class rdf:type owl:class .`

4

Die Semantic-Crystal-Software

In diesem Kapitel wird die Software *Semantic Crystal* vorgestellt. Zuerst wird kurz die Zielgruppe skizziert, danach ausführlich auf die sichtbaren Merkmale der Software eingegangen, und schliesslich werden auch einzelne Kernpunkte angeschnitten, die sich dem Benutzer nicht direkt zeigen, aber dennoch einige Denkarbeit und Systematik beinhalten.

Dieses Kapitel ist das erste von dreien, welche sich explizit der Software *Semantic Crystal* widmen. In Kapitel 5 wird *Semantic Crystal* auf Usability untersucht, das Design-Paradigma *Iterative Design Approach* vorgestellt und über die Evaluation berichtet. Kapitel 6 geht schliesslich auf bestehende Herausforderungen ein und blickt auf mögliche zukünftige Forschungsthemen.

4.1 Zielpublikum von Semantic Crystal

Bevor *Semantic Crystal* vorgestellt wird, soll das Zielpublikum umrissen werden. Ohne Kenntnisse der Zielgruppe kann wenig über Stärken und Schwächen einer Software ausgesagt werden, denn den Bedürfnisse *aller* Anwendergruppen kann *Semantic Crystal* sicher nicht gerecht werden.

Das Zielpublikum von *Semantic Crystal* sind Benutzer, die am Inhalt einer Wissensbasis interessiert sind, diese aber noch nicht kennen. Sie bringen zwar ein Vorwissen für die Gegenstandsdomäne mit (sie sind beispielsweise Filmfans), wissen aber nicht, wie die Ontologie aufgebaut ist, und wie die einzelnen Klassen genau heissen.

Zudem bringt der Zielnutzer von *Semantic Crystal* Grundwissen von OWL, Ontologien und *Semantic Web* mit. Wörter wie *Klasse*, *ObjectProperty* oder *RDF-Tripel* schrecken ihn nicht ab. Er weiss, dass sich hinter diesen Wörtern einfache informationstechnische Konzepte verstecken, und dass diese in ein paar Jahren dem Otto Normalbenutzer genauso alltäglich vorkommen werden, wie dies heute Fachwörter früherer Zeiten wie *Browser*, *Spam*, *WLAN* oder *Link* sind.

Aus der Menge des Zielpublikums ausgeschlossen sind also Anfänger, welche die Technologien des *Semantic Web* vollständig verborgen haben wollen. Ein gutes Programm, das diese Benutzergruppe abdeckt, ist *Ginseng* [11], das an der gleichen Universitätsabteilung wie *Semantic Crystal* entwickelt worden ist. *Ginseng* vereinigt die Techniken des *Natural Language Processing* mit denjenigen des *Semantic Web* auf geschickte Art. In *Ginseng* ist eine natürlichsprachliche Grammatik in klassischer Weise aufgebaut, doch dem Benutzer werden nur solche Buchstaben- und Wortkombinationen ermöglicht, welche die Grammatik einhalten. «What is the height of mount marcy?», ist beispielsweise ein solcher Satz. Der Benutzer bekommt das Gefühl, echtes Englisch zu schreiben, dabei ist er an sehr starre Grammatik und Wortregeln gebunden. Im Unterschied zu *Ginseng* widmet sich *Semantic Crystal* aber nicht den Benutzern, die nicht verstehen, was sich im Innern der Software abspielt, sondern denjenigen, die die ganze Möglichkeitspalette von SPARQL nützen wollen.

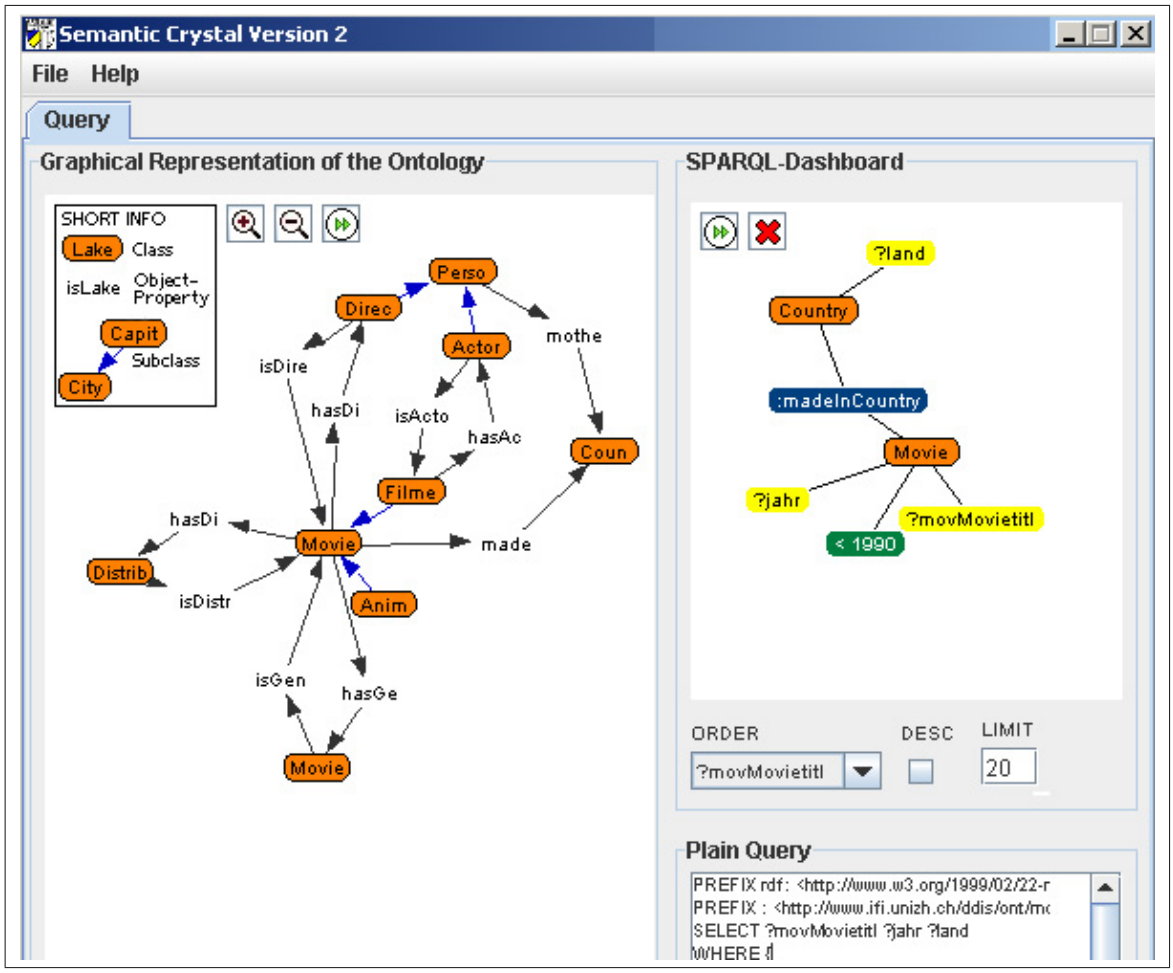

Abbildung 4.1: Screenshot der Software Semantic Crystal.

4.2 Aufgaben von Semantic Crystal

Semantic Crystal hat drei Hauptaufgaben: Das Anzeigen einer Ontologie in graphischer, intuitiver Art und Weise, die graphische Darstellung der Abfrage, welche der Benutzer mit den vier TORC-Elementen zusammensetzt, sowie die Präsentation der Abfrageresultate. Diesen drei Hauptaufgaben ist nachfolgend je ein Unterkapitel gewidmet.

4.2.1 Ontologien laden und graphisch darstellen

Ontologie laden

Semantic Crystal ist imstande, sowohl Ontologien darzustellen, die auf dem lokalen Rechner gespeichert sind, als auch solche, die im Web liegen und über eine URL zugreifbar sind. Die Implementierung ist für diese zwei Varianten praktisch dieselbe. Die Wissensbasis aus dem Web wird zuerst als temporäre Datei gespeichert, danach wird gleich fortgefahren, wie dies mit einer lokalen Ontologie getan wird.

Ontologien aus dem Web zu benutzen, ist ein wichtiges Merkmal dieser Software. Schliesslich soll *Semantic Crystal* wie ein Webbrowser benutzt werden können, in welchem man nach Lust und Laune Wissensbasen durchforsten kann. Natürlich ist es vorstellbar, dass in *Semantic Crystal* eine *Bookmark*-Liste eingeführt wird, mit welcher die wichtigsten *Semantic-Web*-Wissensbasen aufzufinden sind. Allerdings sind momentan erst wenige OWL-Wissensbasen bekannt, so dass Entwickler und Betatester die gewünschten Dateien sowieso lokal gespeichert haben.

Darstellung

Wenn eine Ontologie geladen worden ist, sollte sie sich dem Benutzer graphisch darstellen. Eine erste Möglichkeit dazu war in Abbildung 2.2 aus dem Buch von Antoniou [3] ersichtlich, anhand welcher RDF Schema vorgestellt worden ist. In *Semantic Crystal* wurde eine ähnliche Darstellung gewählt, allerdings etwas übersichtlicher, um auch über grössere Ontologien einen Überblick zu bieten. *ObjectProperties* werden so dargestellt, wie es Antoniou vorschlägt, *DatatypeProperties* werden aber (zumindest in dieser ersten Überblicksdarstellung) weggelassen, da diese sowieso immer Endknoten des Graphen wären und somit zur eigentlichen Struktur nicht viel beisteuern.

Die Abbildung 2.3, die ebenfalls schon in Kapitel 2 benutzt worden ist, zeigt, wie *Semantic Crystal* die Darstellung einer Ontologie realisiert. Dabei sind alle Klassen Knoten eines Graphes. *ObjectProperties* sind aber keine Graphenkanten, sondern ebenfalls Knoten. Dies hat einen pragmatischen und einen pädagogischen Grund.

Der Pragmatische zuerst: *ObjectProperties* müssen beschriftet sein, denn der Name einer solchen Beziehung ist genauso wichtig wie die *Domain* bzw. der *Range*. Werden aber *ObjectProperties* als Kanten dargestellt, müssen diese Kanten beschriftet sein, was aber die benutzte Graphikbibliothek nicht elegant beherrscht. Deshalb liess ich die *ObjectProperties* als Zwischenknoten darstellen.

Entscheidend ist aber der pädagogische Grund: Alles, was mit einer URI repräsentiert wird, soll auch als Knoten dargestellt werden. Und neben den Klassen sind natürlich auch die *ObjectProperties* durch eine URI repräsentiert, weshalb sie es verdienen, durch Knoten dargestellt zu werden. Durch die unterschiedlichen Farben und Rahmen können Klassen aber dennoch leicht von *ObjectProperties* unterschieden werden.

Ein wichtiges Element fehlt aber noch: Die Subklassen-Beziehung. Abbildung 4.2 zeigt, wie der Graph ohne Berücksichtigung dieser Beziehung aussieht. Klassen, die nur über ihre Elternklassen Beziehungen zu anderen Klassen führen, sind mit dem Rest des Graphen nicht verbunden. Jeder `Movie` ist beispielsweise in einem `Country` gemacht worden. Diese Eigenschaft `madeInCountry` wird an alle Subklassen vererbt, also müsste sie auch bei einem `Animated-Movie` ersichtlich sein. Diese Klasse hat sich aber von den anderen abgespaltet, weil sie keine unmittelbaren *ObjectProperties* zu andern Klassen besitzt. In *Semantic Crystal* wurde nun der Weg gegangen, dass die Subklassen-Beziehung auch eingezeichnet werden (Abbildung 2.3 aus Kapitel 2). Zur Unterscheidung von anderen Beziehungen sind die Subklassen-Beziehungen durch blaue und kürzere Kanten repräsentiert und haben keine Zwischenknoten.[1]

Statt die Subklassenbeziehungen einzuzeichnen, hätte auch alle Eigenschaften der Elternklassen auf die Subklassen übertragen werden können. Dann wäre in Abbildung 4.2 nicht nur zwischen `Movie` und `Country` eine `madeInCountry`-Beziehung zu sehen, sondern auch zwischen `AnimatedMovie` und `Country` bzw. zwischen `FilmedMovie` und `Country`. Bei Wissensbasen mit verschiedenen Klassenhierarchien gäbe diese Variante aber eine enorme Vervielfachung dieser Beziehung, so dass die klassische Darstellung der Subklassen-Beziehung doch vorgezogen wurde.

[1] Theoretisch müssten auch für Subklassen-Beziehungen Zwischenknoten geführt werden, denn auch diese haben eine URI, nämlich die folgende: `http://www.w3.org/2000/01/rdf-schema#subClassOf`. Da aber alle Subklassen-Zwischenknoten immer gleich angeschrieben wären, wurden sie weggelassen.

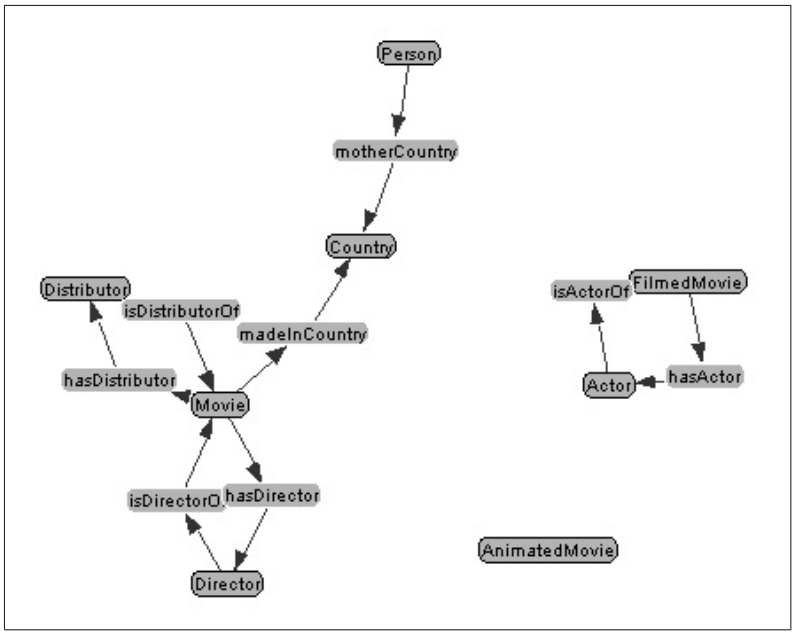

Abbildung 4.2: Die Film-Ontologie ohne Berücksichtigung der Subklassen-Beziehung. Klassen und ObjectProperties werden dargestellt, allerdings fehlen die Vererbungsdarstellungen. Ein `FilmedMovie` erbt die Eigenschaften ihrer Superklasse `Movie`, d. h. auch ein `FilmedMove` besitzt die ObjectProperty `madeInCountry`. (Quelle: Semantic Crystal)

Physikalisch simulierte Knotenanordnung

Die benutze Graphikbibliothek *Prefuse* (mehr dazu in Abschnitt 4.4.2) erlaubt es, den Graphen physikalisch zu simulieren. Dabei verhalten sich Knoten wie abstossende Magnete und die Kanten wie Federn. Abstossungskraft und Federkonstante lassen sich dabei frei definieren. Damit lässt sich auf einfache Weise einen Zustand einstellen, der den Graphen möglichst übersichtlich darstellt. Da die Graphstruktur im Gegensatz zur Baumstruktur Zyklen enthält, existiert kein Wurzelknoten, welcher an prominenter Stelle dargestellt werden könnte. Alle Knoten sind sozusagen gleich wichtig, so dass die genannte physikalische Simulation eine bestmögliche Darstellung der Ontologie ist.

Da die Länge der Knotenbeschriftungen sehr variiert, und Knotenüberlappungen die Physiksimulation zunichte machen, sind die Namen gekürzt worden. Statt die ganze (unlesbare) URI anzuzeigen, wird nur der eigentliche Name nach dem Fragmentzeichen (#) angezeigt. Falls dieser Name noch immer zu lang ist, wird er nach ein paar Buchstaben gekappt.[2]

Andere Darstellungsarten

Ein sehr guter Ansatz, Ontologien graphisch darzustellen, wird auch von der Software *Growl* verfolgt [12]. Allerdings kommt es bei *Growl* zu Problemen, weil eine zu grosse Informationsflut auf den Benutzer fällt oder das System aus *Performance*-Gründen mit der Darstellung nicht

[2]Der ganze Name ist als Tooltip-Text sichtbar, wenn man mit der Maus über einen Knoten fährt. Diese Funktion ist sehr hilfreich, was auch bei der Film-Ontologie ersichtlich ist: Die Klasse «MovieGenre» wird nämlich ausgerechnet zu «Movie» abgekürzt.

mehr nachkommt. *Growl* zeigt nicht nur Klassen, *ObjectProperty*-Beziehungen und Subklassen-Beziehungen an, sondern auch alle OWL-Einschränkungsmöglichkeiten und Instanzen. Instanzen werden zwar in anderer Farbe gehalten als Klassen, jedoch haben sie die gleiche Form, was bei hunderten von Instanzen grosse Verwirrung stiften kann.

Dafür hat *Growl* die interessante Eigenschaft, Elemente, die von wenig Interesse sind, interaktiv auszublenden. Falls der Benutzer also beispielsweise auf ein Land klickt, werden alle Filme aus diesem Land angezeigt, diejenigen aus anderen Ländern aber ausgeblendet.

Growl ist wie *Semantic Crystal* ein Forschungsprototyp und demzufolge nicht gegen alle Bugs und Hacks gefeit. Allerdings zeigt es einige sehr gute Ansätze, wie verschiedenartige und komplexe Ontologien graphisch zu verallgemeinern sind. Die Interaktion in *Growl* führt dazu, dass beim Auswählen einzelner Elemente andere Elemente ausgeblendet werden. Auch bei *Semantic Crystal* ist es so, dass weiterführende Informationen und Aktionsmöglichkeiten nur möglich sind, wenn man eine bestimmte Klasse auswählt.

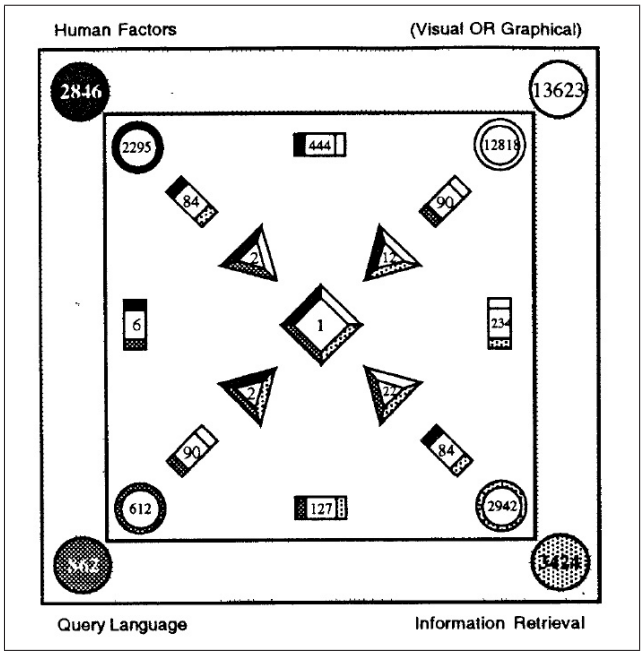

Abbildung 4.3: Anselm Spoerris «InfoCrystal». Vier Suchbegriffe sind angezeigt, sowie die Anzahl Treffer die sich für alle Kombinationen ergeben, wenn man sie mit AND-Verknüpfungen zusammensetzt. Die Form eines Elementes repräsentiert die Anzahl verknüpfter Begriffe (Kreis: 1 Begriff; Dreieck: 3 Begriffe; ...). Bsp: Nur 1 Dokument enthält alle Begriffe, 127 Dokumente enthalten die Begriffe «Query Language» und «Information Retrieval». (Quelle: [13] S. 14)

Daneben möchte ich noch auf die Arbeit von Anselm Spoerri verweisen [13]. Spoerri entwickelte bereits 1993 mit *InfoCrystal* ein Konzept einer visuellen Abfragesprache. Er ging von einem Venn-Diagramm aus, welches Mengen mit Kreisen symbolisiert und sie dann so überlappen lässt, dass alle möglichen Mengen-Kombinationen repräsentiert sind. Venn-Diagramme funktionieren nur bis zu drei Mengen. Mit *InfoCrystal* lassen sich dagegen mehrere Mengen darstellen, wie die Abbildung 4.3 zeigt.

InfoCrystal geht das Problem an, dass wenige Suchbegriffe zuviele sinnlose Resultate nach sich ziehen bzw. viele Suchbegriffe wenige oder gar keine. Die Gewichtung der Suchbegriffe kann mit Hilfe von *Slidebars* eingestellt werden.

Der Name *Semantic Crystal* ist von *InfoCrystal* abgeleitet. Auch *Semantic Crystal* verfolgt das Ziel, Informationen graphisch intuitiver erfassbar zu machen. In der Umsetzung unterscheidet es sich jedoch markant von Spoerris Vorschlag.

Umwandlung von RDF- in Graph-Daten

Lädt der Benutzer eine OWL-Datei, liest *Semantic Crystal* diese RDF-Tripel ein und macht bereits die ersten Abfragen, um die Ontologie überhaupt darstellen zu können. Zuerst werden alle RDF-Tripel gesucht, in welchen OWL-Klassen definiert sind. Damit werden Knoten erstellt. Jeder Knoten ist intern durch seine URI identifiziert. Danach werden *ObjectProperties* gesucht, und von diesen die jeweilige *Domain* und den jeweiligen *Range*. Die Struktur des Graphen wird nun durch Kanten ergänzt, die von den URIs von Klassen über URIs von *ObjectProperties* zu URIs von anderen Klassen führen.

Wenn zu einer *ObjectProperty* ein *Range* angegeben ist, der nicht als Klasse definiert ist, dann tauchen Probleme auf. Wie sollte *Semantic Crystal* Beziehungen zu nicht existierenden Klassen darstellen? Wenn Ontologien mit unvollständigen Angaben ausgestattet sind, kann es zu Problemen bei der Umwandlung von RDF- zu Graph-Daten kommen und somit zu Ladefehlern im Programm.

Gleich wie die *ObjectProperties* werden auch die Subklassen-Beziehungen in der OWL-Datei gesucht und anschliessend in eine Form gebracht, die sich für eine graphische Interpretation eignet.

Interaktion mit dem Graphen

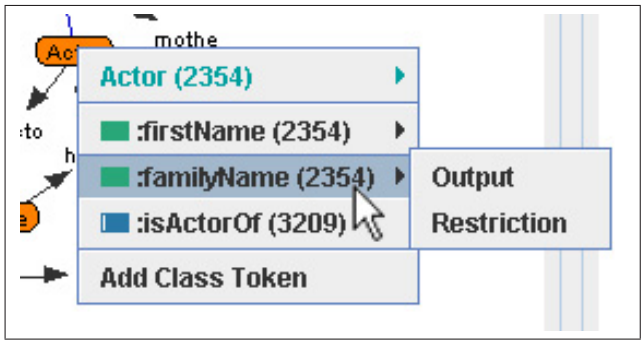

Abbildung 4.4: Popup-Menü eines Klassenknotens. Wenn der Benutzer auf einen Knoten klickt, erscheinen Informationen zur Klasse wie die URI, die Anzahl Instanzen und die Attribute in einem Popup-Menü. (Quelle: Semantic Crystal)

Der Benutzer kann auf zwei Arten mit dem Graphen interagieren. Einerseits kann er die Knoten manuell platzieren, falls er mit der Anordnung durch die physikalische Simulation nicht zufrieden ist. Andererseits kann er über ein Popup-Menü zusätzliche Informationen über die ausgewählte Klasse anzeigen lassen. In Abbildung 4.4 sind diese Informationen ersichtlich. Hinter dem Namen steht in Klammern die Anzahl Instanzen, die für die ausgewählte Klasse vorhanden sind. Ein wichtiges Kriterium für die Entscheidung, ob überhaupt eine Abfrage erstellt werden soll. Hat die Klasse keine Instanzen, steht dort eine Null. Als Submenü versteckt kann auch die volle URI der Klasse angezeigt werden, was dann sinnvoll ist, wenn verschiedene Klassen den

gleichen Namen[3] haben.

Weiter unten im Popup-Menü befindet sich die Attributsliste, die für jede Klasse verschieden lang sein kann.[4] *DatatypeProperties* sind mit einem grünem *Icon* dargestellt, *ObjectProperties* mit einem blauen. Dies ist die gleiche Symbolik, die der Ontologie-Editor *Protégé* [14] benutzt. Auch hinter dem Namen eines Attributs steht in Klammern die Anzahl Vorkommnisse des gewählten Attributs. Diese Zahl muss nicht zwingend identisch sein mit der Anzahl Instanzen. Denn eine Instanz kann mehrmals das gleiche Attribut mit verschiedenen Werten aufweisen oder ein Attribut entbehren.

Neben dem blossen Anzeigen von Informationen können mit diesem Popup-Menü auch SPARQL-Abfragen zusammengestellt werden. Mehr dazu folgt im nächsten Abschnitt.

4.2.2 Elementarer Aufbau einer Abfrage

In Kapitel 3 sind die vier TORC-Grundelemente einer SPARQL-Abfrage hergeleitet worden. Diese wurden in die Software *Semantic Crystal* eingebaut.

Jede *DatatypeProperty* (das sind in Abbildung 4.4 die mit den grünen *Icons*) kann als *Output* oder als *Restriction* in die Abfrage übernommen werden. Dies wählt der Benutzer in demjenigen Popup-Menü aus, das bereits vorgestellt worden ist. Sobald der Benutzer seine Auswahl getroffen hat, ist sie im sogenannten *Dashboard* ersichtlich. *Dashboard* wird derjenige Bereich genannt, in welchem die Abfrage graphisch dargestellt ist. Eine *ObjectProperty* kann nur als *Connection* in die Abfrage übernommen werden.

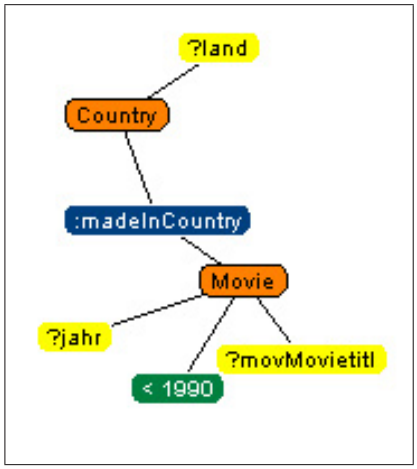

Abbildung 4.5: Darstellung einer Abfrage im Dashboard. (Quelle: Semantic Crystal)

Abbildung 4.5 zeigt wie eine Abfrage im Dashboard dargestellt wird. Die beiden Klassen `Movie` und `Country` sind als *Token* vertreten und durch die *Connection* `madeInCountry` verknüpft. Gelbe Knoten repräsentieren *Output*-, grüne Knoten *Restriction*-Elemente.

[3] Mit Name einer Klasse meine ich denjenigen Teil, der hinter dem Fragmentzeichen # einer URI steht.

[4] Ein Attribut einer Klasse ist nichts anderes als eine OWL-*Property*, die als *Domain* eben diese Klasse hat. Je nachdem, ob der *Range* dieser *Property* ein Literal ist oder ein andere Klasse, handelt es sich um eine *Datatype*- bzw. *ObjectProperty*.

Weitere Interaktionen

Im Dashboard kann die Abfrage ebenfalls via Popup-Menü editiert werden. Dabei wird für jedes TORC-Element ein verschiedenes Popup-Menü angezeigt. Das Popup-Menü für *Output*-Elemente ermöglicht beispielsweise, eine Variable umzubenennen. In Abbildung 4.5 sehen wir das Element `?movMovietitl`, welches nicht umbenannt worden ist. Dieser unpassende Variablenname muss offensichtlich von der Software automatisch vergeben worden sein. Wie wir im Abschnitt 3.2.2 gesehen haben, ist die automatische Variablennamenvergabe nicht unproblematisch. Die Möglichkeit, *Output*-Elemente manuell umzubenennen, ist deshalb ein wichtige Funktion. Die beiden anderen *Output*-Elemente wurden bereits umbenannt.

Bei *Restriction*-Elementen kann bzw. muss selbstverständlich die gewünschte Einschränkung vorgenommen werden können. In unserem Beispiel wurde das Publikationsjahr (das übrigens als `?jahr` auch ausgegeben wird) auf unter 1990 eingeschränkt.

Das Popup-Menü von *Tokens* unterscheidet sich nur minim von denjenigen, die aus der Ontologie-Darstellung (Abbildung 4.4) bereits bekannt sind. Damit lässt sich bequem von einem bereits für die Abfrage bestimmten *Token* ein neues Attribut hinzufügen, ohne zur Ontologie-Darstellung zurückkehren zu müssen.

Mehrere Tokens derselben Klasse

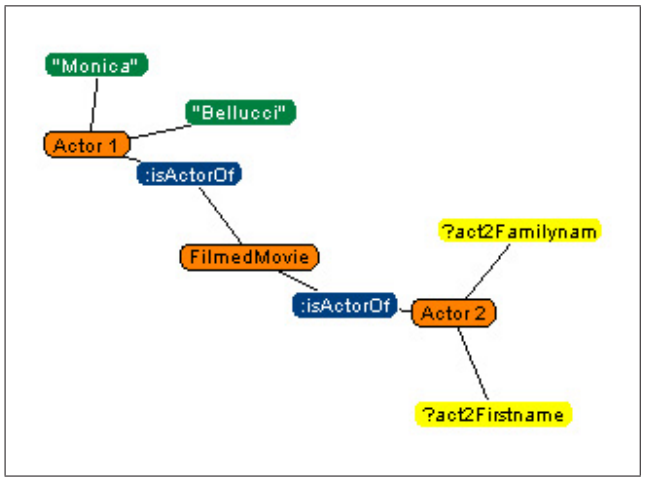

Abbildung 4.6: Darstellung einer Abfrage mit zwei Tokens derselben Klasse. (Quelle: Semantic Crystal)

Wird ein Attribut ausgewählt, dessen (*Domain*)-Klasse noch nicht als *Token* im *Dashboard* vorhanden ist, wird automatisch ein solches *Token* erstellt. Wird ein Attribut ausgewählt, dessen (*Domain*)-Klasse bereits vorhanden ist, so wird das neue Attribut ans bestehende *Token* angeschlossen.

Es besteht aber auch die Möglichkeit, zwei Tokens derselben Klasse zu benutzen. Dazu befindet sich zuunterst im Popup-Menü (Abbildung 4.4) der Menüpunkt «Add Token». Das in Abschnitt 3.4.1 gemacht Beispiel, in welchem nach allen Schauspielern gesucht wird, die bereits mit Monica Bellucci zusammengearbeitet haben, benötigt zwei Tokens der Klasse `Actor`. Abbildung 4.6 zeigt, wie dies im Dashboard aussieht.

Die SPARQL-Abfrage nicht verbergen

Aus zwei Gründen soll der Kern einer Software dem Benutzer nicht verborgen werden: Erstens gehören zur Zielgruppe keine «Anfänger» (vgl. 4.1). Zweitens widerspricht dies Erkenntnissen aus der Usability-Forschung (vgl. 5.2.3). Der Kern von *Semantic Crystal* ist die automatisch generierte SPARQL-Abfrage. Aus den genannten Gründen wurde diese nicht versteckt, sondern absichtlich angezeigt. Abbildung 4.7 zeigt den Abfragetext, den Semantic Crystal durch die Einstellungen im Dashboard der Abbildung 4.5 generiert hat.

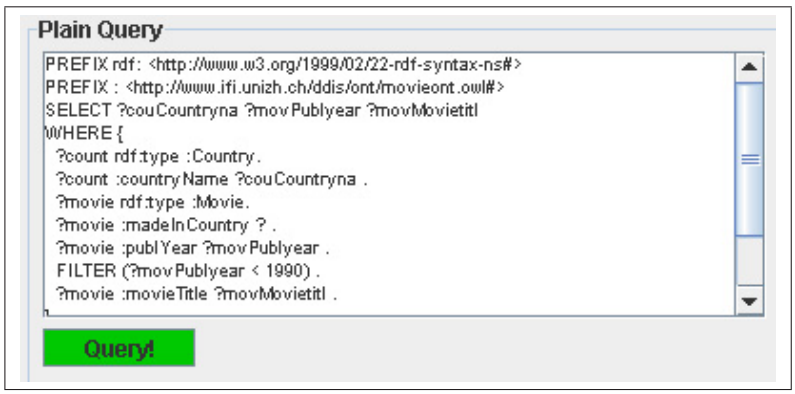

Abbildung 4.7: Generierte SPARQL-Abfrage. (Quelle: Semantic Crystal)

Der Benutzer kann so die generierte Abfrage kopieren und in einer anderen Anwendung wiederverwenden. Oder er kann den Abfragetext innerhalb von *Semantic Crystal* bearbeiten und damit eine Abfrage ausführen, die er selber formuliert hat. Allerdings wird dies bei der nächsten Dashboard-Mutation wieder überschrieben.

4.2.3 Abfrageresultat anzeigen

In Abbildung 4.8 ist die Resultatetabelle zu sehen, die aus der Abfrage aus den Abbildungen 4.5 und 4.7 entstanden ist. Es stellt sich nun die Frage, ob das Resultat nicht auch anders dargestellt werden könnte.

Datentyp und Sprache anzeigen

Eine Möglichkeit wäre, die Zusatzinformationen über den Datentyp und über die Sprache anzuzeigen. Viele Wissensbasen haben für einen Wert den Datentyp und/oder bei Texten Informationen über die benutzte Sprache mitgespeichert. Das folgende Beispiel zeigt zwei Möglichkeiten auf, wie solche Zusatzinformationen den eigentlichen Informationen mitgegeben werden kann.

```
<!-- Example from the movie-ontology -->
<publYear rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
  2004
</publYear>

<!-- Example from a geo-ontology -->
<rdfs:label xml:lang="en">
  big stone lake
</rdfs:label>
```

Results

?couCountryna	?movPublyear	?movMovietitl
United States	1979	1941
United States	1989	Always
France	1976	Barocco
France	1977	Baxter, Vera Baxter
United States	1986	Blue Velvet
France	1988	Camille Claudel
United States	1977	Close Encounters of the Third Kind
Australia	1989	Dead Calm
United States	1989	Dead Calm
France	1977	Der Amerikanische Freund
Germany	1977	Der Amerikanische Freund
Switzerland	1980	Der Erfinder
Germany	1987	Der Himmel über Berlin

Abbildung 4.8: Resultate. (Quelle: Semantic Crystal)

Diese Zusatzinformation könnten angezeigt werden. Sicher ungeeignet ist es, die Informationen in Textform ans eigentliche Resultat anzuhängen, wie eine Vorversion von *Semantic Crystal* deutlich gezeigt hatte. Besser wäre, mit Hilfe von Icons den Datentyp und die Sprache zu kennzeichnen.

XML-Export statt tabellenförmige Anzeige

Ein SPARQL-Resultat muss nicht tabellenförmig sein. Das W3C hat ein XML-Format definiert, in welchem SPARQL-Resultate zurückgegeben werden [15]. Die erste Zeile des Resultats aus Abbildung 4.8 wird wie folgt zurückgegeben.

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="couCountryna"/>
    <variable name="movPublyear"/>
    <variable name="movMovietitl"/>
  </head>
  <results>
    <result>
      <binding name="couCountryna">
        <literal>United States</literal>
      </binding>
      <binding name="movPublyear">
        <literal>1979</literal>
      </binding>
      <binding name="movMovietitl">
        <literal>1941</literal>
      </binding>
   </result>
  </results>
</sparql>
```

Die Variante, das Resultat per XML zurückzugeben ist aber nur interessant, wenn es von einer Maschine weiterbearbeitet werden soll. Für den Menschen ist diese Darstellung als reine Lösungsansicht ungeeignet. Sie kann deshalb als Zusatz angeboten werden, aber die tabellenförmige Darstellung nicht ersetzen.

Natürlich sollen aber in der XML-Darstellung möglichst viele Informationen reinfliessen wie Datentyp und Sprache, die wir in der tabellenförmigen Ansicht weggelassen haben. Eine bessere Lösung wäre daher:

```
...
  <binding name="movPublyear">
    <literal datatype="http://www.w3.org/2001/XMLSchema#int">
      1979
    </literal>
  </binding>
  <binding name="movMovietitl">
    <literal xml:lang="en">1941</literal>
  </binding>
...
```

Interaktive Anzeige

Einige der Tester von *Semantic Crystal* haben bei der Resultateansicht gestaunt, dass kein Weitersurfen möglich ist. Sie hätten gerne auf einen Filmtitel geklickt und dann in einem Popupfenster alle möglichen Detailinformationen zum entsprechenden Film zu erblicken gehofft. Dies würde der Absicht die *Describe*-Funktion von SPARQL nahe kommen (vgl. 3.3.3).

Jedoch ist dies ein Zusatzfeature, das später zu realisieren ist. Dieses «Weitersurfen» wäre wie eine innere Applikation. Auch innerhalb dieses Popupfensters wird sich der Benutzer sicherlich wünschen, wiederum auf Elemente zu klicken, und sich damit eben «durchzusurfen». Dieser Surfwunsch wird bei *Semantic Crystal* (noch) nicht unterstützt.

4.3 Gelöste Herausforderungen

Die «Herausforderungen», die in diesem Abschnitt erwähnt werden, wurden von *Semantic Crystal* ganz oder zumindest grösstenteils gelöst. Daneben existieren andere, komplexere Herausforderungen, die in *Semantic Crystal* nicht berücksichtigt sind. Oft ist dabei die Materie so komplex oder noch nicht weit genug entwickelt, dass sich ein Engagement kaum gelohnt hätte, alle diese Punkte in *Semantic Crystal* gebührend zu berücksichtigen. Nichtsdestotrotz sind diese bestehenden Herausforderungen nicht uninteressant. Sie werden in Kapitel 6 genauer beschrieben.

4.3.1 Einseitigkeit bei inversen Eigenschaften

OWL-Wissensbasen sind ähnlich wie Datenbanken aufgebaut. Ein `Movie` hat beispielsweise bei der Eigenschaft `hasActor` den Wert `#Pitt1a`, was nichts weiter ist als ein *Fremdschlüssel* zur Ressource Brad Pitt. In der Ontologie sind diese Eigenschaften auch entsprechend deklariert: `hasActor` hat die *Domain* `Movie` und den *Range* `Actor`.

In der Ontologie ist noch eine weitere wichtige Information enthalten. Die Eigenschaft `isActorOf` ist als *invers* zur Eigenschaft `hasActor` definiert. Invers bedeutet, dass diejenige Klasse, die bei der einen Eigenschaft als *Domain* gewirkt hatte, bei der inversen Eigenschaft nun als *Range*

wirkt und umgekehrt. Hat also der Film «Fight Club» für die Eigenschaft `hasActor` den Wert «Brad Pitt», hat umgekehrt auch «Brad Pitt» für die Eigenschaft `isActorOf` den Wert «Fight Club».

In den meisten Wissensbasen sind diese Werte nicht zweimal explizit vorhanden, weil sonst der Speicherplatz nur unnötig vergrössert würde. Inverse Eigenschaften können aber hergeleitet werden.

In *Semantic Crystal* wurde ein *Reasoner* eingebaut, der beim Laden der Wissensbasis diese herleitbaren RDF-Tripel explizit ins Datenmodell aufnimmt. Die SPARQL-Abfrage wird anschliessend auf dieses aufgeblasene Datenmodell angewendet. Nur dank der Vorarbeit des *Reasoners* werden auch implizit vorhandene Tripel gefunden.

Reasoner ist ein Oberbegriff für alle Programme, die aus bestehenden Tatsachen und Regeln neue Tatsachen schlussfolgern können.

Warum muss diese Vorarbeit geleistet werden? Ist denn SPARQL nicht «intelligent» genug, um selber diese Schlussfolgerungen vorzunehmen? Eine Antwort darauf ist auf der Webseite von ARQ (vgl. Abschnitt 4.4.3) zu finden:

> SPARQL ist eine Abfragesprache und damit «datenorientiert», d. h. es werden nur Informationen abgefragt, die in den Modellen enthalten sind; es werden keine Schlussfolgerungen in der Abfragesprache selber durchgeführt. Natürlich könnte das Jena Modell «klug» sein und den Eindruck vermitteln, dass gewisse Tripel existieren, indem es sie *on-demand* mittels OWL-*Reasoning* kreiert. SPARQL macht nichts anderes als die Beschreibung in der Form einer Abfrage zu nehmen, und die Informationen zurückzugeben in der Form einer Menge von Bindungen oder eines RDF-Graphen.

Es ist also tatsächlich so, dass SPARQL das *Reasoning* nicht vorsieht.

4.3.2 Reasoner für Subklasseninstanzen

Auch wegen der Subklassenbeziehung musste *Reasoner*-Vorarbeit geleistet werden. Gehen wir von folgenden Beispielstripeln der Film-Wissensbasis aus.

```
[1]  film:Person  rdf:type        owl:Class
[2]  film:Actor   rdf:type        owl:Class
[3]  film:Actor   rdfs:subclass   film:Person
[4]  film:Jolie1a rdf:type        film:Actor
[5]  film:Jolie1a film:firstName  "Angelina"
[6]  film:Jolie1a film:familyName "Jolie"
```

In der 4. Zeile ist Jolie als Instanz des Typs `Actor` eingetragen. Aus Zeile 3 ist zu entnehmen, dass ein `Actor` auch eine `Person` ist, und dass ergo Jolie vom Typ `Person` sein muss. Dies scheint trivial. Es ist aber wichtig, dass der *Reasoner* hier schon Vorarbeit leistet und die entsprechenden Tripel ins Datenmodell aufnimmt, auf welches die Abfragen gestellt werden. Folgende SPARQL-Abfrage liefert ohne diese Vorarbeit nämlich kein Resultat zurück.

```
SELECT *
WHERE {
  ?person rdf:type :Person .
  ?person :familyName "Jolie" .
}
```

4.3.3 Indizierung und Joins

Wie bereits erwähnt gleichen Web-Wissensbasen den herkömmlichen Datenbanken. In der Datenbanktechnik haben sich vor allem relationale Datenbanksysteme durchgesetzt. Solche Datenbanken sind auf gute Abfrage-*Performance* ausgelegt. Dies wird unter anderem mittels Indizierung getan, d. h. Inhalte werden in einer sortierten Liste nachgeführt, was zwar das Einfügen neuer Datensätze erschwert, die Abfragen jedoch beschleunigt.

Das wird aber zum Problem bei Web-Wissensbasen. Da werden die Strukturen nicht in einem einmaligen Vorgang fix angelegt, und kein menschlicher Techniker kann bestimmen, welche Werte indiziert werden sollen. Dies führt dazu, dass solche Indizierungen ziemlich schwierig, wenn nicht gar unmöglich werden.

Ganz besonders zum Tragen kommt dies bei grossen Wissensbasen. Ausserdem kommt diese Problematik zum Vorschein, wenn der Benutzer den «Fehler» in der Abfrage macht, keine Verknüpfung zwischen zwei Klassen zu setzen, und somit die Lösungsmenge ungewollt vervielfacht. Im folgenden Beispiel wird dieser Sachverhalt dargestellt. Gesucht *wären* alle Filme mit dem Schauspieler Johnny Depp.

```
SELECT ?title
WHERE {
  ?actor rdf:type :Actor
  ?actor :familyName "Depp"
  ?film rdf:type :Movie
  ?film :movieTitle ?title
}
```

Bei dieser Abfrage ging der Verknüpfungs-Tripel `?actor :isActorOf ?film` vergessen, so dass die SPARQL-*Engine* «glaubt», man suche nach allen Schauspielern mit dem Namen «Depp» *und* nach allen Filmen, unabhängig davon, ob Johnny Depp mitspielt oder nicht. SPARQL gibt deshalb als Resultat alle Kombinationen von Filmtiteln und Schauspieler mit dem Namen «Depp» zurück.[5]

Versuche bei Testpersonen haben ergeben, dass diese Verknüpfungen tatsächlich häufig vergessen gehen.[6] In einer Urversion von *Semantic Crystal* bleib bei jeder Abfrage die Applikation solange blockiert, bis das Resultat zurückgegeben worden ist. Technisch ausgedrückt: Die Software bestand nur aus einem einzigen *Thread*. Das hatte zur Folge, dass der Tester zwar nach fünf Sekunden seinen Fehler bemerkt hatte, dann aber trotzdem fünf Minuten warten musste, bis der Rechner das unsinnige Resultat ausgespuckt hatte.

Bei *Semantic Crystal* wurde schliesslich ein separater *Thread* für die Suchabfrage eingerichtet (vgl. 4.4.4). Dabei wird der *Button*, mit welchem sich die Abfrage starten lässt, während einer Abfrage durch einen «Abbrechen»-*Button* ersetzt. Somit kann der Benutzer jederzeit einen laufenden Abfrageprozess beenden.

4.3.4 Nur Literale als Output ermöglichen

Das Objekt eines RDF-Tripels ist entweder ein RDF-Knoten (Ressource) oder ein Literal. Mit SPARQL können alle möglichen Tripel durchsucht werden, jedoch sind für Standard-Abfragen

[5] Glücklicherweise gibt es nur einen Schauspieler mit dem Nachnamen «Depp», so dass die Anzahl Lösungen nur Anzahl Filme (1022) mal Anzahl Schauspieler mit dem Namen «Depp» (1) beträgt. Wäre der Schauspielername weniger eingeschränkt käme man rasch auf eine gigantische Lösungsmenge.

[6] Allerdings wurden diese Versuche noch mit einem nicht-graphischen Dashboard gemacht (siehe Abschnitt 5.1.1). In der jetzigen Version sollten diese Verknüpfungen weniger häufig vergessen gehen, da der Benutzer das Fehlen einer Verknüpfung unmittelbar sieht.

eher Literale gedacht. Bevor das Konzept der vier TORC-Elemente entwickelt worden ist, konnte man nicht nur die literalen Werte der *DatatypeProperties* ausgeben lassen, sondern auch die Werte von *ObjectProperties*, welche eben URIs sind.

Testpersonen waren oft verwirrt, wenn sie die Schauspieler eines Filmes suchen wollten und statt des Namens des Schauspielers seine URI erhalten haben. Für die Anwendung *Semantic Crystal* hatte das zur Folge, dass *Output*-Elemente nur noch bei *DatatypeProperties* möglich waren. Dabei wurde die Mächtigkeit der Anwendung natürlich beschnitten, jedoch konnte damit ein grosser *Usability*-Vorteil errungen werden.

4.3.5 Boolscher Operator oder Regex?

Bei *Restriction*-Elementen können Einschränkungen gemacht werden. Einschränkungen sollten sowohl numerisch als auch als *Regex-Pattern-Matching* (vgl. 3.2.3) möglich sein. Beide Varianten sind mit SPARQL möglich, und beide haben ihre Vorteile. Beispiel:

```
[1]  FILTER (?jahreszahl < 1990) .
[2]  FILTER regex(?name, "Spielb") .
```

Die beiden Varianten unterscheiden sich in SPAQRL durch das Stichwort `regex`. Auch hier möchte ich von einer Vorversion von *Semantic Crystal* berichten. Ursprünglich musste der Benutzer selber entscheiden, ob er *Regex* anwenden möchte oder nicht, was vielen Testpersonen nicht leicht fiel. Die Software kann jedoch diese Entscheidung recht zuverlässig abnehmen.

Semantic Crystal benutzt die numerische Variante falls der eingegebene Einschränkungstext mit einem boolschen Operator (=, >, <, <=, >=, !=) beginnt. In sonstigen Fällen wird die *Regex*-Variante angewendet.

4.4 Benutzte Techniken

4.4.1 Java SE 5.0

Implementiert wurde *Semantic Crystal* in Java mit einem *JavaSE-5-Development-Kit* [16]. *Semantic Crystal* läuft leider nicht auf Rechnern mit einer Runtime Environment 4.0 oder tiefer.

4.4.2 Prefuse-Graphikbibliothek, GraphML

Die gewählte Graphikbibliothek *Prefuse* [17] erfüllte die gestellten Ansprüche bestens. Sie hält den Anforderungen der graphischen Darstellung stand, bietet eine exzellente Umsetzung der physikalischen Graphensimulation und ist frei verfügbar.

Semantic Crystal benutzt die Version *beta release 2006.07.15*. *Prefuse* wurde an der Universität Berkeley in Kalifornien entwickelt und wird unter der Leitung von Jeffrey Heer weitergeführt. Ein kleiner Schwachpunkt mag dabei ist die unvollständige Dokumentation sein. Durch die sehr schnelle Reaktion der *Prefuse*-Entwickler in ihrem Hilfeforum wird dies aber wettgemacht.

Prefuse bietet verschiedene Möglichkeiten für das Einlesen der Daten an. Das einfachste Format ist *GraphML* [18], eine XML-Sprache zur Graphbeschreibung.

4.4.3 Das Semantic-Web-Framework Jena

Jena [19] ist eine Java-Bibliothek für *Semantic-Web*-Aufgaben. *Semantic Crystal* benutzt die Version 2.3 von Oktober 2005. Auch *Jena* erfüllt die Anforderungen und ist frei verfügbar. Im Unterschied

zur Graphikbibliothek sind hier aber praktisch keine Alternativen vorhanden. Dies liegt einerseits daran, dass das Gebiet relativ jung ist und sich damit (noch) nicht viel Geld erwirtschaften lässt, andererseits aber auch daran, dass *HP Labs* sehr viele Ressourcen in die Entwicklung von *Jena* gesteckt hat. Zu *Jena* wird auch die SPARQL-Abfrage-*Engine ARQ* [20] mitgeliefert, mit welcher in *Semantic Crystal* die Abfragen ausgeführt werden.

4.4.4 Sonstige Techniken

Die Thread-Klasse SwingWorker

Da einzelne Abfragen eine sehr lange Bearbeitungszeit brauchen, ist es unumgänglich, diese separat auszuführen, um dem Benutzer in der Zwischenzeit die Benutzung der Software weiterhin zu ermöglichen. So kann der Benutzer sinnlose Abfragen auch abbrechen (vgl. 4.3.3).

Diese separate Abarbeitung wird in Java mit sogenannten *Threads* erledigt. Eine praktische Hilfsklasse, welche Arbeitsschritte in separaten Threads laufen lässt, heisst `SwingWorker`. Sie ist seltsamerweise nicht im Umfang von *Java SE 5* enthalten[7], obwohl sie von den offiziellen Java-Tutorials empfohlen wird.

SurveyLogger

Für die Evaluation benutzte ich *SurveyLogger*, eine Zusatzklasse, die für die Software *Ginseng* [11] entwickelt worden ist. *SurveyLogger* bewirkt, dass in einem zusätzlichen Fenster Text angezeigt wird, der in einer Konfigurationsdatei vorbereitetet worden ist. Damit lassen sich Aufgaben für Testpersonen anzeigen, aber auch Fragen und Antworten, welche den Fragebogen auf Papier ersetzen. Zudem misst *SurveyLogger* die Zeiten, die eine Testperson für eine Aufgabe benötigt.

[7] Die Klasse `SwingWorker` wird aller Voraussicht nach im Gegensatz zu Java 5 in Java 6 enthalten sein.

5

Evaluation der Software

Ich konnte sieben Leute zur *Usability*[1] von Semantic Crystal testen. Sieben Leute mögen etwas wenig sein für eine statistisch repräsentative Grundmenge. Doch die quantitative Auswertung war nur ein Bestandteil der Evaluationsstrategie. Da die Software in einem *Iterative Design Approach* entwickelt worden ist, musste sie regelmässig von Testbenutzern geprüft werden. Ich kombinierte diese beiden Ansätze, d. h. die Testbenutzer waren für *Inputs* eines *Iterative-Design*-Zyklus verantwortlich und dienten gleichzeitig auch als Testperson, an welchen die Usability evaluiert wurde. Deshalb hatte der erste dieser sieben Probanden noch eine andere Software vor sich als der siebte. Die Evaluationsresultate müssen immer auch vor diesem Hintergrund betrachtet werden.

5.1 Was ist Iterative Design Approach?

Die klassische Vorgehensweise der Software-Entwicklung aus den Anfangszeiten der Informatik bestand aus folgenden Hauptschritten: Experten fertigten einen Entwurf an, Programmierer setzten diesen um und Tester wurden danach eingesetzt, um *Bugs* zu finden. Dabei tritt das Problem auf, dass die Befunde der Testpersonen nichts mehr am Entwurf ändern können, der von den Experten angefertigt worden ist.

Der *Iterative Design Approach* ist ein anderer Ansatz zur Software-Entwicklung: Nach einer Analyse und einem ersten Entwurf wird ein kleiner Prototyp angefertigt, welchen die Testnutzer bereits prüfen. Ein solcher Prototyp kann auch nur auf Papier bestehen, wobei die Interaktionen dann von Menschen improvisiert und simuliert werden. Dieser Vorgang wird iterativ fortgesetzt, d. h. die entdeckten Probleme werden erfasst und ein neuer Entwurf wird ausgearbeitet, welcher zu einem neuen Prototypen führt. Dieser Vorgang wiederholt sich, bis die Software den gestellten Ansprüchen gerecht wird (vgl. *Iterative Design* bei Lauesen [21]).

Natürlich muss auch *Iterative Design* systematisch vorgenommen werden. Das Problem der klassischen Software-Entwicklung ist schliesslich kein neues Phänomen. Dennoch ist *Iterative Design* gerade bei Neuentwicklungen im Forschungsbereich ein erfolgversprechender Ansatz, da keine hungrige Anwendergruppe mit grossen Erwartungen und ebenso grossem Anforderungskatalog darauf wartet, die Software benutzen zu dürfen.

[1] Das englische Wort *Usability* kann auf Deutsch sehr passend mit *Benutzerfreundlichkeit* übersetzt werden. Da es sich bei *Usability* aber um einen Fachterm handelt, der je nach Autor auch mehr oder anderes umfasst als nur gerade Benutzerfreundlichkeit, und da der Begriff der *Usabilität* doch etwas zu künstlich klingt, wird hier mit dem Begriff *Usability* gearbeitet, obwohl der Duden es (noch) nicht als deutsches Wort anerkannt.

5.1.1 Iterative Design bei Semantic Crystal

Der erste Designvorschlag für *Semantic Crystal* kam von der Aufgabenstellung des Institutes. Auf Papier fertigte ich die ersten Ideen an. Später konnte ich das Produkt meiner Projekt-Betreuerin vorführen. Es war zwar erst ein *Screenshot* auf Papier, weil die Funktionalität erst rudimentär funktionierte, dennoch konnten erste Schwachpunkte bereits ausgemerzt werden.

Auch die Testbenutzer machten mich auf Mängel aufmerksam, welche ich sofort änderte, um die *Usability* zu erhöhen. Ich nenne diese Befunde *Usability-Bugs*. *Usability-Bugs* unterscheiden sich von normalen *Bugs* dadurch, dass sie nicht die grundsätzliche Funktionalität beeinträchtigen, sondern nur die Benutzung erschweren. Entwickler einer Software gehören nach einer gewissen Zeit nicht mehr oder nur noch selten zu den Benutzern, die *Usability-Bugs* entdecken. Sie führen im Unterschied zu unbeschwerten Erstbenutzern in der Regel keine Vorgänge aus, welche zu *Usability-Bugs* führen.

Neben kleineren *Usability-Bugs* entdeckten die Testbenutzer auch grössere Problembereiche. Auf diese gehe ich in Kapitel 6 ein.

Einer dieser Usability-Bugs war die Blockierung des Systems bei fehlerhaften Abfragen ohne Verknüpfung (vgl. 4.3.3), ein anderer die Tatsache, dass dem Benutzer selber die schwierige Wahl überlassen wurde, sich für oder gegen *Regex* zu entscheiden (vgl. 4.3.5)

Das Graphische Dashboard

Ein Meilenstein meiner Arbeit war die Ersetzung des nicht-graphischen *Dashboards* durch ein graphisches. Abbildung 5.1 stellt eigentlich das gleiche dar wie die graphische Variante in Abbildung 4.5. Als Entwickler bemerkte ich nicht, dass die SPARQL-Abfrage ebenso wie die Ontologie als Graph dargestellt werden kann. Erst Professor Bernstein machte mich nach dem Software-Test darauf aufmerksam.

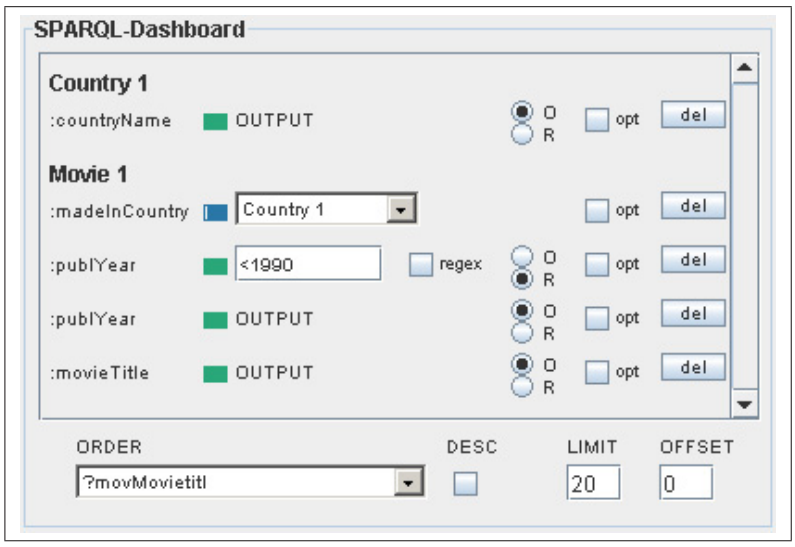

Abbildung 5.1: Das nicht-graphische Dashboard einer alten Programmversion. (Quelle: Semantic Crystal)

Ähnliches geschah mit den Informationen zu einer Klasse der Ontologie. In einer früheren Version wurden diese Informationen, welche nun in einem Popup-Menü direkt neben der entsprechenden Klasse verfügbar sind, in einem separaten Bereich der Benutzeroberfläche angezeigt.

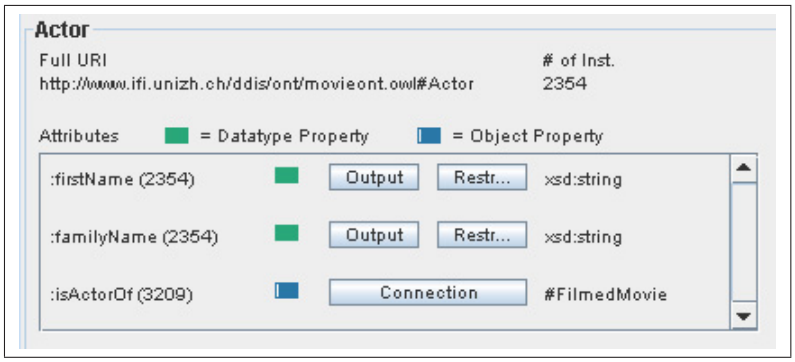

Abbildung 5.2: Anzeige der Klasseninformationen einer alten Programmversion. (Quelle: Semantic Crystal)

Abbildung 5.2 zeigt diesen Bereich, von wo aus der Benutzer auch die weiteren Einstellungen vorzunehmen hatte.

5.2 Was ist Usability?

5.2.1 Sechs Faktoren der Usability und ihre Messmethoden

Viele Autoren teilen *Usability* in die Faktoren Funktionalität, Effizienz und subjektive Zufriedenheit ein. Lauesen [21] zählt sogar sechs Faktoren auf: Funktionalität, Lernfreundlichkeit, Effizienz, Erinnerungsfreundlichkeit, Subjektive Zufriedenheit und Verständlichkeit. Wenn alle diese Faktoren optimal sind, dann hat das Programm auch die höchste *Usability*. Das Problem liegt aber darin, dass diese Faktoren zumindest teilweise schwer zu messen sind.

Häufige Messmethoden sind Messung der Zeit, die ein Benutzer für eine Aufgabe benötigt, Meinungsumfragen mit Fragebogen oder Zählen der vorkommenden Probleme, die der Benutzer beim sogenannten *Laut-Denken-Test* nennt.

Daneben existieren auch die bekannten heuristische Evaluationsmethoden. Dabei betrachten Usability-Experten das Programm und stellen ein Gutachten zusammen, in welchem die Probleme des Programms aufgelistet sind. Das Problem dabei ist nach Lauesen, dass heuristische Evaluation nur eine 50%-Trefferquote aufweisen. Die Hälfte aller vermeintlichen Probleme sind gar keine Probleme. D. h. obwohl Experten einen Sachverhalt als problematisch eingestuft haben, finden sich Benutzer damit gut zurecht. Da man aber nicht weiss, welche der genannten Probleme echte, und welche nur vermeintliche sind, lohnt sich der Aufwand kaum, Probleme aus heuristischen Evaluationen zu beheben.

Funktionalität

Der *Usability*-Faktor **Funktionalität** zeigt sich in der Frage, ob das Programm fähig ist, bestimmte Aufgaben durchzuführen. Beim Test zu *Semantic Crystal* sind die Aufgaben allerdings bereits so formuliert, dass sie durch das Programm durchführbar sind. Zeitmessung und Laut-Denken-Tests sind hier nicht sehr sinnvoll, da sie nichts über die Funktionalität aussagen. Meinungsumfragen sind etwas besser.

Lernfreundlichkeit

Das Mass der **Lernfreundlichkeit** zeigt sich darin, wie leicht die Bedienung des Programms zu erlernen ist. In diesem Zusammenhang ist vor allem interessant, wie verschiedene Benutzergruppen die Lernfreundlichkeit beurteilen. Hier sind Zeitmessung und Laut-Denken-Tests geeignet. Allerdings müssen die gleichen Aufgaben wiederholt werden oder in ähnlicher Form vorkommen, damit schlüssige Aussagen gemacht werden können. Meinungsumfragen sind überraschenderweise ungeeignet. Die Empirie zeigte, dass die Antworten der Benutzer und die tatsächlich gemessene Zeit nicht korrelieren (vgl. Lauesen [21] S. 33).

Effizienz

Wie **effizient** ein Programm ist, lässt sich durch die Frage bestimmen, wieviele Ressourcen der regelmässige Benutzer aufwenden muss, um eine Aufgabe zu erfüllen. Je mehr Ressourcen aufgewendet werden müssen, desto ineffizienter ist das Programm. Zeitmessung wäre nur sinnvoll, wenn Experten des Programms getestet würden. Meinungsumfragen sind aus dem selben Grund wie beim Testen der Lernfreundlichkeit ungeeignet.

Erinnerungsfreundlichkeit

Die Frage nach der **Erinnerungsfreundlichkeit** lautet: Wie leicht ist es für gelegentliche Benutzer, sich an die Benutzung des Programms zu erinnern. Wenn Tests nach gewissen Zeitabständen wiederholt werden können, ist Zeitmessung eine gute Testmethode, wegen den Wiederholungen aber auch eine aufwändige. Ein bisher nicht genannte Methode ist die Überprüfung, ob Richtlinien eingehalten worden sind.

Richtlinien (engl. *Guidelines*, auch *Entwurfsregeln* genannt, engl. *design rules*) sind eine Sammlung von Empfehlungen, wie eine Benutzeroberfläche aussehen sollte. Es gibt sehr unterschiedliche Richtlinienkataloge. Die einen beschränken sich auf grundsätzliche Empfehlungen[2], andere geben detaillierte, dafür auch zahlreiche Empfehlungen.[3]

Die Überprüfung, wie sehr sich eine Benutzeroberfläche an vorgegebene Richtlinien hält, kann Aufschluss darüber geben, wie erinnerungsfreundlich die Oberfläche ist.

Subjektive Zufriedenheit

Subjektive Zufriedenheit ist ein wichtiger Faktor für *Usability*. Allerdings lässt sich subjektive Zufriedenheit schlecht objektiv messen. Meinungsumfragen scheinen auf den ersten Blick ideal zu sein. Lauesen [21] weist aber auf das Problem hin, dass die Benutzer in Fragebogen nicht das sagen, was sie wirklich denken. Die gleichen Fragen auf verschiedene Arten zu stellen, kann hier weiterhelfen. Daneben taucht aber ein weiteres Problem auf, das auch das *zweite Usability-Gesetz* genannt wird (zumindest von Lausesen. Vgl. 5.2.5): Wenn die subjektive Zufriedenheit hoch ist, muss nicht zwingend auch die Ausführungsleistung hoch sein. Doch dies ist schliesslich das Ziel von *Usability*.

Andere Autoren sehen das weniger kritisch. John Brooke [23] entwickelte einen für alle Benutzeroberflächen einheitlichen Kurzfragebogen, um die subjektiven Meinungen zu sammeln. Brooks schreibt seinem Fragebogen selber die Eigenschaften «quick and dirty» zu und gab ihm

[2] Søren Lauesen ([21], S.169) gibt 8 Entwurfsregeln vor, nach welchen die Benutzeroberflächen auf Papier entworfen werden sollen. Dabei sind keine konkrete Angaben darunter, sondern eher grundsätzliche wie z. B., dass der Benutzer auf möglichst wenige Fensterinstanzen zugreifen müssen soll, um eine Aufgabe zu erfüllen.

[3] Jakob Nielsen [22] erwähnt vier Richtliniensammlungen, von welchen der umfangreichste 944 Richtlinien umfasst, der am wenigsten umfangreiche immerhin noch deren 162.

den Namen *SUS*, was für *System Usability Scale* steht. Auch *Semantic Crystal* wurde mit *SUS* ausgewertet, deshalb ist es sinnvoll, genauer auf die Motivation Brooks einzugehen, die zur Entwicklung dieses Fragebogens führte. Dies wird in Abschnitt 5.2.4 nachgeholt.

Verständlichkeit

Ein letzter Faktor ist die **Verständlichkeit**. Dem Benutzer sollte das Verhalten des Programms verständlich sein. Entscheidend ist dies vor allem bei ungewolltem Verhalten. Fehlermeldungen sollen Aufschluss darüber geben, inwiefern das Programm sich nicht wie gewünscht verhält. Auch hier ist Zeitmessung nicht sinnvoll, Lautes-Denken-Tests sind aufschlussreicher, aber leider kein eigentliches Mass.

5.2.2 Zusammenhang der Usability-Messmethoden

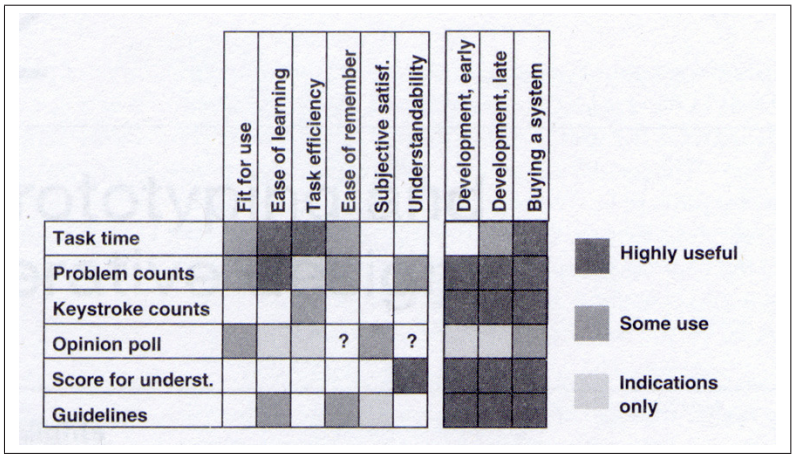

Abbildung 5.3: Messtechniken für Usability. Nicht jede Messtechnik ist für jeden Usability-Faktor gleich geeignet. Die Tabelle gibt einen Überblick darüber.
Legende der Messmethoden: Task time ≻ Messung der Zeit pro Aufgabe; Problem counts ≻ Zählen der Probleme mittels Laut-Denken; Keystroke counts ≻ Zählen der Tastaturanschläge; Opinion poll ≻ Meinungsumfragen; Score for underst. ≻ Verständistest; Guidelines ≻ Überprüfen der Einhaltung von Richtlinien.
Legende der Faktoren: Fit for use ≻ Funktionalität; Ease of learning ≻ Lernfreundlichkeit; Task efficiency ≻ Effizienz; Ease of remember ≻ Erinnerungsfreundlichkeit; Subjective satisf.≻ Subjektive Zufriedenheit; Understandability ≻ Verständlichkeit (Quelle: Lauesen [21] S. 39)

Abbildung 5.3 gibt Aufschluss über die Eignung verschiedener Messmethoden für die einzelnen Faktoren. Zeitmessung ist bei vier von sechs Faktoren zumindest eine brauchbare Methode. Auch das Zählen von Problemen durch Laut-Denken kommt bei vier von sechs Faktoren gut weg. Das zeigt auf, dass die beiden beliebten Messtechniken zwar im Vergleich zu Alternativen gut sind, jedoch nicht alle Faktoren vollständig messen können.

5.2.3 Das mentale Modell

Der Benutzer hat jeweils ein «Bild im Kopf», wie eine Software bestehende Informationen abspeichert. Ein Benutzer hat beispielsweise eine Eingabemaske vor sich, in welche er zehn Angaben zu einer Person machen muss. Er füllt diese zehn Felder aus und drückt auf «Speichern», worauf die Eingabemaske verschwindet. Der Benutzer hat nun das Gefühl, dass diese Eingabemaske genauso gespeichert wird, wie er sie zuletzt gesehen hat. Im Usability-Jargon wird dies *mentales Modell* genannt. In Wirklichkeit kann es sein, dass nicht alle dieser zehn Angaben miteinander am gleichen Ort gespeichert werden (vgl. [21] S. 105ff).

Der Usability-Tipp lautet aber: Gestalte ein Programm so, dass es mit dem mentalen Modell der meisten Benutzer übereinstimmt. Also so, dass die Software tatsächlich das macht, was der Benutzer erwartet.

Dieser Tipp hat mich veranlasst, die durch den Benutzer im Dashboard zusammengestellte Abfrage als reines SPARQL in einem Textfeld anzuzeigen (vgl. 4.2.2). Auch wenn die Abfrage nirgends als SPARQL ersichtlich wäre, würde der Benutzer erwarten, dass im «Innern des Programms» eine solche SPARQL-Abfrage existiert. Zumindest die meisten Benutzer hätten dieses mentale Modell. Da nun diese Abfrage tatsächlich als SPARQL angezeigt wird, hat der Benutzer bei ungewollten Ergebnissen die Möglichkeit, direkt im SPARQL-Feld nach dem Fehler zu suchen. Umgekehrt kann er bei einer besonders gelungen Abfrage den SPARQL-Code ansehen und weiterbenutzen.

5.2.4 SUS-Test

Für die subjektive Meinung nach der *Usability* wurde bei Semantic Crystal der SUS-Test von John Brooke benützt. Dieser beinhaltet zehn standardisierte Fragen, die der Testbenutzer beantworten muss. Die zehn Fragen im kompletten Wortlaut und die vollständigen Werte des Experimentes sind im Anhang A aufgelistet. Eine Interpretation der Auswertungen wird im Abschnitt 5.3 vorgenommen.

Wie Søren Lauesen weiss auch John Brooke [23], dass es für Usability kein absolutes Mass gibt, sondern dass deren Güte immer auch vom Zweck des Programms abhängig ist. Dennoch besteht nach Brooke eine Nachfrage nach einem allgemeinen Mass, mit welchem *Usability* auch zwischen verschiedenen Systemen verglichen werden kann. Ausserdem sollte ein solches Mass rasch einsetzbar sein und nicht viel kosten.

Aus diesen Gründen entwickelte John Brooke den SUS-Test. Die zehn Fragen sind so ausgewählt, dass alle Faktoren von *Usability* genügend repräsentiert sind. Die Fragen sind in der bekannten *Likert*-Skala angeordnet (5 Werte zwischen «Volle Zustimmung» und «Volle Ablehnung»). Mit nur zehn Fragen, die erst noch mit Likert-Werten beantwortet werden sollen, beschäftigt man eine Testperson nicht allzu lange. So wird die Gefahr tief gehalten, dass die Antworten falsch ausfallen, weil sich die Testperson langweilt. Brooke musste aber sicherstellen, dass die zehn Fragen die richtigen zehn sind. Dazu hat er mehrere Fragen pro Themenbereich notiert. Nur diejenigen zehn Fragen, welche die grössten Ausschläge provoziert hatten, fanden den Weg in den endgültigen Fragebogen.

Der Test kann sogar mit einer vorgegebenen Summenformel auf eine einzige Zahl reduziert werden, welche ein Mass für *Usability* sein soll. Hier ist aber doch Vorsicht geboten, denn ich denke, dass viele Leute Fragebogen generell gut bzw. schlecht bewerten. Der eine verteilt seine Werte auf der Skala von 0 – 4 meistens zwischen 2 und 4, während sich ein anderer eher an den Bereich zwischen 0 und 2 hält. Aussagekräftiger als die aufsummierte Schlusszahl dünkt mich, festzustellen, bei welchen dieser zehn Fragen die Tester in der Gesamtheit eine schlechte bzw. gute Bewertung gegeben haben. Und für welche dieser Fragen die grösste Einigkeit unter den Testpersonen geherrscht hat.

5.2.5 Lauesens Usability-Gesetze

Lauesen ([21] S. 19, 34, 44) hebt drei *Usability*-Befunde hervor und proklamiert diese als die *drei Usability-Gesetze*.

1. **Heuristische Evaluationen haben nur eine Trefferquote von 50%**. Wird eine Benutzeroberfläche von einem Usability-Experten begutachtet, werden zwar viele Probleme entdeckt, jedoch sind ungefähr die Hälfte davon keine eigentliche Probleme. Es hat sich gezeigt, dass viele Benutzer keine Schwierigkeiten mit diesen vermeintlichen Problemen haben.

2. **Subjektive Zufriedenheit korreliert nur wenig mit objektiver Ausführungsleistung**. Dies betrifft vor allem die Meinungsumfragen nach subjektiver Zufriedenheit. Die mit solchen Umfragen messbare Zufriedenheit hängt stark von anderen Einflüssen ab, wie der momentanen Laune und der Stimmung im Betrieb.

3. **Je mehr Effort Entwickler für die Erstellung eines Prototyps aufgewendet haben, desto weniger sind sie gewillt, ihn zu ersetzen**. Dies ist das Credo des *Iterative Design Approach* und hat vor allem zwei Gegebenheiten zur Folge. (1) Nach einer Entwicklerrunde muss eine Testrunde eingelegt werden, bevor die Entwickler weiterarbeiten. (2) Für jede Entwicklerrunde muss klar sein, auf welcher Stufe sich der Prototyp befindet. Die Tester dürfen nicht die Funktionalität bemängeln, die noch nicht vorgesehen ist.

5.3 Die Evaluation bei Semantic Crystal

5.3.1 Messung der Zeit und Anzahl Versuche

Für jede der vier Aufgaben wurde im Schnitt mehr Zeit benötigt als für die jeweils vorangegangene, jedoch nahm auch der Schwierigkeitsgrad der Aufgaben zu. So wurde für die erste Aufgabe im Schnitt 2:06 gebraucht, für die letzte 7:53. Das sind eher hohe Zahlen, die jedoch deshalb relativiert werden müssen, weil alle Testpersonen zum ersten Mal *Semantic Crystal* benutzt haben.

Eine Tatsache kann hervorgehoben werden: Für die dritte und vierte Aufgabe wurde im Schnitt fast gleichviel Zeit verwendet, jedoch ist für die vierte Aufgabe die Standardabweichung einiges höher. Dieser gravierende Unterschied deutet darauf hin, dass einigen Testpersonen die Benutzung von *Semantic Crystal* schon bei der vierten Aufgabe ziemlich klar war, und sie die Aufgabe zügig erledigen konnten (der Rekord liegt bei 3:26). Andere hatten auch bei der vierten Aufgabe noch einige Schwierigkeiten (der Minusrekord liegt bei 11:19). Auf diesen Schluss deuten auch die Rückmeldungen hin. Die Evaluation war ja gleichzeitig ein Laut-Denken-Test, und einige Testpersonen meinten nach Beendigung der vierten Aufgabe, dass sie erst jetzt wirklich begreifen, wie die Software zu bedienen sei.

Eine ausführlichere Interpretation der Zeitmessungen erachte ich nicht als sehr aussagekräftig. Für weitere Interpretationen befinden sich alle Messzahlen in Anhang A.2.

Was die Anzahl gemessener Abfrageversuche betrifft, so ist eine Interpretation nicht sinnvoll. Es ist nicht das Ziel von *Semantic Crystal*, dass dem Benutzer Abfragen auf den ersten Versuch gelingen. Das Gegenteil ist der Fall: Mit *Semantic Crystal* lassen sich rasch Teilabfragen ausführen, was jeweils sinnvolle Rückschlüsse darüber erlaubt, ob man mit der Erstellung einer Abfrage auf dem richtigen Weg ist. 8 der 25 gemessenen Aufgaben wurden in einem Versuch korrekt gelöst. Einmal aber benötigte jemand für eine Aufgabe neun Versuche. Da die Anzahl Versuche keine Rolle spielen, möchte ich nun zur sinnvolleren Auswertung des Fragebogens übergehen.

5.3.2 Fragebogen

Zum Einstieg möchte ich die wichtigsten Erkenntnisse in Ranglistenform präsentieren. Die einzelnen Fragen sind hier auf sinngemässe Titel gekürzt. Die Nummer in Klammer verweist auf die Liste in Anhang A.1.3, in welcher die Fragen in dem Wortlaut stehen, wie sie den Testpersonen gestellt wurden.

Die positivsten Punkte (hoher Mittelwert)

1. Funktionen gut eingebunden (5)
2. Vermittelt Zuversicht (9)
3. Keine Inkonsistenzen (6)

Die negativsten Punkte (tiefer Mittelwert)

1. Tiefe Lernfreundlichkeit (7)
2. Schlechte Bedienbarkeit (3)
3. Kein Wunsch zur regelmässigen Benutzung (1)

Die eindeutigsten Punkte (tiefe Standardabweichung)

1. Einbindung der Funktionen (5)
2. Schwerfälligkeit der Benutzung (8)
3. Inkonsistenzen (6)

Die uneindeutigsten Punkte (hohe Standardabweichung)

1. Lernfreundlichkeit (7)
2. Wunsch zur regelmässigen Benutzung (1)
3. Komplexität (2)
4. Bedarf nach Unterstützung durch Techniker (4)

Interpretation dieser Ranglisten

Grundsätzlich sind diejenigen Punkte eher positiv bewertet worden, die die technischen Aspekte betreffen. Funktionalität bzw. deren Einbindung in die Oberfläche wurde dann auch am besten bewertet (3.29 auf einer Skala von 0 – 4). Auch dass wenig Inkonsistenzen (3.14) auftreten, rechne ich zur technischen Seite. Die dritte der positiv bewerteten Fragen, die nach der Zuversicht, passt nicht ins Schema. Vielleicht wäre die Zuversicht schlechter beantwortet worden, wenn die Testpersonen auch nachher mit dieser Software arbeiten müssten. Denn diese Resultate sind auch immer vor dem Hintergrund zu sehen, dass die Testpersonen aus *Goodwill* bei den Tests mitgemacht haben und daraus keine direkten Folgen für ihren Alltag befürchten müssen.

Die Frage, die am stärksten verneint wurde, lautete: «Ich stelle mir vor, dass die meisten Leute sehr schnell lernen werden, wie dieses Interface zu bedienen ist.» Zwar ist die Verneinung nicht allzu stark (1.57), und gleichzeitig ist dies auch der Punkt mit der höchsten Standardabweichung (1.29), jedoch zeigt er die Problematik von *Semantic Crystal* auf. *Semantic Crystal* kann ohne Einarbeitung oder Einführung nur schwer benutzt werden. Dies liegt daran, dass der Benutzer die

geladene Ontologie erst kennenlernen muss, und dass ihm viele Interaktionsmöglichkeiten geboten werden. Jeder Knoten hat ein Popup-Menü mit je ca. 5 – 15 Auswahlmöglichkeiten.

Die weiteren negativen Punkte zeigen in gleiche Richtung. Die Bedienbarkeit sei nicht sehr hoch und somit bestehe auch nicht der Wunsch, das Programm regelmässig benutzen zu wollen. Um definitivere Aussagen über die Lernfreundlichkeit oder die Bedienbarkeit machen zu können, müssten aber die Testpersonen ein zweites und ein drittes Mal diesen ca. 45minütigen Test durchführen. Erst dann liesse sich sagen, ob sie diese Punkte, die ja eher nach einer gewissen Erfahrung beantwortet werden können, gleich bewerten würden oder anders.

Gesamtheitlich wurde die Usability von *Semantic Crystal* als *knapp «eher gut»* bewertet, wie der Mittelwert der *SUS*-Totalzahl von 65 (auf einer Skala von 0 – 100) nahelegt (vgl. 5.2.4 bezüglich der Aussagekraft dieser Zahl).

6

Bestehende Herausforderungen und künftige Forschungsschwerpunkte

In diesem Kapitel werden diejenigen «Herausforderungen» beschrieben, die den Weg in die Software *Semantic Crystal* nicht gefunden haben. Es handelt sich dabei nicht unmittelbar um Sachverhalte, die wegen mangelnder Programmierfähigkeiten oder zuwenig Engagement nicht gebührend behandelt wurden, sondern um solche, die tiefergreifende Analysen benötigen, weshalb diese «Herausforderungen» auch mit dem Ausblick auf die zukünftige Forschung zusammenfallen.

6.1 Ontologie-Design

6.1.1 Semistrukturiertheit

Bei OWL-Wissensbasen haben wir es einerseits mit einer Ontologie zu tun, welche die Struktur der Instanzen beschreibt, andererseits mit RDF-Tripel, die Informationen nach dieser Ontologie bereitstellen. Inwiefern können wir darauf vertrauen, dass jede Instanz wirklich so ausgestattet ist, wie dies die Ontologie vorschreibt?

Dabei spreche ich nicht (nur) fehlerhaft geführte Wissensbasen an. Es kann z. B. vorkommen, dass einer Instanz eine Eigenschaft zugeordnet wird, die sie gemäss der Ontologie gar nicht haben müsste. In der Film-Wissensbasis könnte für die Instanz «Bruno Ganz» des Typs `Actor` die Zusatzinformation stehen, dass dieser einen Sohn habe. Die Tatsache, dass keine vollständige Strukturiertheit vorliegt, wie dies bei herkömmlichen Datanbanksystemen der Fall ist, nennt man *Semistrukturierung*.

Eine anderer Fall der Semistrukturierung liegt vor, wenn in der Ontologie absichtlich oder unabsichtlich keine vollständige Angabe über einen Sachverhalt gemacht wird. In der Film-Ontologie wird über den Filmtitel beispielsweise nur folgende Aussagen gemacht.

```
film:movieTitle rdf:type   owl:DatatypeProperty .
film:movieTitle rdf:domain film:Movie .
film:movieTitle rdf:range  xsd:string .
```

Es wird also formal beschrieben, dass ein `Movie` einen `movieTitle` haben kann. Über die Kardinalität ist aber nichts gesagt, obwohl dies mit OWL möglich wäre. Kann es auch Filme ohne

Titel geben, oder solche mit zwei?[1] Es ist oft nicht möglich oder zumindest nicht wünschenswert, eine exakte Strukturierung vorzugeben.

```
film:Movie rdfs:subclassOf _:a .
_:a        rdf:type        owl:restriction .
_:a        owl:onProperty  film:movieTitle .
_:a        owl:cardinality "1"^^xsd:int .
```

Die hier aufgelisteten Tripel definieren mit OWL, dass ein `Movie` exakt einen Filmtitel aufweisen muss. Diese Strukturierung hat Vor- und Nachteile. Den Nachteil haben wir bereits angesprochen: Die Realität ist oft nicht in die starren Strukturen abzubilden, welche Informatiker die Arbeit stark vereinfacht. Ein Vorteil einer starren Struktur ist die Möglichkeit, Wissensbasen automatisch zu validieren. Falls zum Beispiel `Movie`-Instanzen ohne Titel auftreten, dann können diese entfernt werde, so dass die Wissensbasis *valid* ist.

Da die Entwickler von Wissensbasen aber wissen, dass bei starrer Strukturierung zuviele Instanzen den Validitätstest nicht bestehen, werden sie die Ontologie möglichst wenig einschränken.

Ein Ansatz von *Semantic Crystal* ist es, die Instanzen einzeln zu betrachten, um die Feinheiten der Semistrukturierung zu erkennen. Das Popup-Menü (Abb. 4.4) gibt z. B. die Anzahl Vorkommnisse eines Attributes wieder. Diese Angabe ist eine quantitative Anzeige, wieviele der Instanzen mit solchen Angaben versehen sind. Dies ist sinnvoller als nur eine strukturelle Angabe aus der Ontologie wiederzugeben.

Die Semistrukturierung ist eine der grossen Stärken des *Semantic Web*. Jedoch kann sie auch zu Problemen führen, die bei Vollstrukturierung nicht auftreten.

6.1.2 «Saubere» Wissensbasen

Für die Software *Semantic Crystal* ist es wichtig, dass viele OWL-Wissensbasen zur Verfügung stehen. Wie bereits beklagt (vgl. 2.4.1), gibt es davon momentan nicht sehr viele.

Solche Ontologien und Wissensbasen werden oft mit dem Ontologie-Editor *Protégé* [14] oder mit ähnlichen Editoren erstellt. Dabei wird die Ontologie innerhalb von *Protégé* graphisch elegant angeordnet, allerdings geschieht dies nicht mit der erstellten Wissensbasis als solches. Klassen- und Instanzdefinitionen mischen sich in der OWL-Datei anscheinend völlig willkürlich untereinander und mit Referenzen zu bereits eingeführten Instanzen.

Solange man solche Wissensbasen nur mit *Protégé* weiterbearbeitet, tauchen dabei auch keine Probleme auf. Wenn sich aber eine Wissensbasis in *Protégé* nicht laden lässt, oder man aus einer bestehenden Wissensbasis nur Teile davon extrahieren möchte, scheint mir eine «saubere» Strukturierung in einer OWL-Datei doch sinnvoll.

Als Beispiel soll die Gegenüberstellung eines OWL-Abschnitts aus Protégé mit einem selbst erstellten dienen.

OWL aus *Protégé*:

```
<Actor rdf:ID="pitt1">
  <firstName>Brad</firstName>
  <familyName>Pitt</familyName>
  <motherCountry>
    <Country rdf:ID="USA">
```

[1] Beides ist in der realen Welt möglich. Wenn man in einer Kunstschule nachfragt, ob schon jemand die Idee hatte, einen Film ohne Titel zu machen, bejahen dies sicherlich einige. Und natürlich sind auch mehrere Titel für einen Film möglich. «The Da Vinci Code» ist schliesslich der gleiche Film wie «Sakrileg»

```
      <countryName>United States</countryName>
    </Country>
  </motherCountry>
</Actor>
<Actor rdf:ID="jolie1">
  <firstName>Angelina</firstName>
  <familyName>Jolie</familyName>
  <motherCountry rdf:resource="#USA"/>
</Actor>
```

Protégé definiert die Instanz USA der Klasse `Country` innerhalb der Definition der Instanz Brad Pitt der Klasse `Actor`. Auf scheinbar gleiche Art und Weise wie die Instanz Brad Pitt wird auch die Instanz Angelina Jolie definiert. Allerdings wird bei der Eigenschaft des Herkunftslandes keine neue `Country`-Instanz mehr definiert, sondern auf die bereits definierte referenziert. Wieso findet also die USA-Definition bei Brad Pitt und nicht bei Angelina Jolie statt? Dafür gibt es keinen plausiblen Grund.

Falls ein Mensch diese Wissensbasis manuell weiterverarbeiten möchte oder muss, dann ist ihm mit einem strukturierten OWL-Code weit mehr gedient.

Strukturiertes OWL-Fragment:

```
<!-- Countries -->
<Country rdf:ID="USA">
  <countryName>United States</countryName>
</Country>

<!-- Actors -->
<Actor rdf:ID="pitt1">
  <firstName>Brad</firstName>
  <familyName>Pitt</familyName>
  <motherCountry rdf:resource="#USA"/></Actor>
<Actor rdf:ID="jolie1">
  <firstName>Angelina</firstName>
  <familyName>Jolie</familyName>
  <motherCountry rdf:resource="#USA"/>
</Actor>
```

6.1.3 Berechnete Ontologien

OWL bietet einige Möglichkeiten, Begebenheiten exakt zu definieren, die aber in dieser Arbeit nur ansatzweise zur Sprache gekommen sind (vgl. 2.1.4).

Die bisher vorgestellten Klassen waren immer direkt und bedingungslos definiert. Allerdings liessen sich Klassen auch über mathematische Konstrukte definieren. So könnte beispielsweise die Klasse `FilmedMovie` so definiert sein, dass eine Instanz von `Movie` automatisch auch zur Klasse `FilmedMovie` gehört, wenn sie mindestens einen Schauspieler aufweisen kann.

Damit lassen sich verschiedenste Konzepte definieren, ohne für jedes Konzept die Instanzen einzeln aufzulisten. Die Subklassen-Beziehung, die in *Semantic Crystal* zum Einsatz kommt, ist eine dieser Möglichkeiten, Ontologien und Zusammenhänge zu «berechnen». Doch schon bei Subklassen war *Reasoner*-Arbeit erforderlich (vgl. 4.3.2), ausserdem musste eine Lösung konzipiert werden, wie dieser Subklassen-Zusammenhang graphisch intuitiv darzustellen ist.

Könnten wir nun weitere dieser mathematischen Konstrukte bei *Semantic Crystal* anzeigen, wäre dies einerseits ein Gewinn. Andererseits führte das aber auch zu einer Informationsflut. Der Ontologie-Browser *Growl* ([12], vgl. 4.2.1) bietet für viele OWL-Konstrukte Darstellungsmöglichkeiten und hat dementsprechend mit der Informationsüberflutung zu kämpfen.

6.2 Reasoner-Arbeiten

6.2.1 Vollständiger Reasoner

Da sich SPARQL-Abfragen nur direkt auf ein vorliegendes Datenmodell beziehen, muss dem SPARQL-Prozessor notwendigerweise eine Reasonertätigkeit vorangehen. Ideal wäre, einen *vollständigen Reasoner* einzusetzen, welche alle möglichen herleitbaren Tripel ins Datenmodell aufnimmt. Dies führt aber zu einer enormen Datenmenge, welche ein System bereits bei mittelgrossen Wissensbasen überfordern kann.

Ein anderer Ansatz ist, SPARQL-Abfragen an einen *Reasoner* weiterzuleiten, welcher bei jeder Abfrage das Datenmodell nur soweit ergänzt, wie es die gestellte Abfrage erfordert. Ein solcher *Reasoner* wäre z. B. *Pellet* der *Mindswap*-Gruppe [24].

Einige dieser *Reasoner*-Arbeiten, die das Datenmodell ergänzen, sollten aber schon beim Laden der Wissensbasis durchgeführt werden, und nicht erst bei der Ausführung einer SPARQL-Abfrage. Denn die Auswirkungen sollten schon bei der Darstellung der Ontologie sichtbar sein.

Hier tauchen zahlreiche Knackpunkte auf, wovon ich einen exemplarisch erwähnen möchte. In der Geographie-Ontologie von *Ginseng* [11] befinden sich folgende Informationen: Jeder Fluss fliesst durch einen oder mehrere Staaten, was durch eine oder mehrere `runsThrough`-Eigenschaften gegeben ist. Die Eigenschaft `runsThrough` ist in der Ontologie korrekt deklariert: Als *ObjectProperty* mit der *Domain* `River` und dem *Range* `State`. Daneben ist auch die Eigenschaft `hasRiver` deklariert, für die es sich umgekehrt verhält: Die *Domain* ist `State` und der *Range* ist `River`. Die Eigenschaft `hasRiver` ist als *invers* zur Eigenschaft `runsThrough` deklariert.

```
[1]  rioGrande runsThrough newMexico .
[2]  newMexico hasRiver rioGrande .
```

Die Instanzen sind natürlich nur einmal eingetragen. In der Geographie-Wissensbasis ist nur der erste der beiden RDF-Tripel explizit vorhanden, der zweite lässt sich nur durch die Angabe herleiten, dass `hasRiver` zu `runsThrough` invers ist. Dies soll ein *Reasoner* machen.

Ein primitiver *Reasoner* führt dazu typischerweise folgende Schritte aus:

1. Er findet die *ObjectProperty* `runsThrough`.
2. Er testet, ob `runsThrough` zu etwas invers ist.
3. Weil in der Geographie-Ontologie `runsThrough` nicht als invers zu `hasRiver` deklariert ist (sondern nur `hasRiver` zu `runsThrough`), bricht er hier das *Reasoning* für die *ObjectProperty* `runsThrough` ab.
4. Er geht zur nächsten *ObjectProperty*. Diesmal ist es `hasRiver`.
5. Er testet, ob `hasRiver` zu etwas invers ist.
6. Er findet eine inverse Eigenschaft, nämlich `runsThrough`.

7. Er nimmt alle Tripel von `hasRiver` (`A hasRiver B`) und fügt die inverse Version dem Modell hinzu (`B runsThrough A`). In der Geographie-Wissensbasis sind aber keine Tripel des Typs «`A hasRiver B`» eingetragen, sondern eben nur solche des Typs «`B runsThrough A`», so dass der *Reasoner* auch hier ohne Wirkung bleibt.

Natürlich ist in diesem konkreten Untersuchungsgebiet der *Reasoner* von *Semantic Crystal* so angepasst worden, dass er auch inverse Tripels ins Datenmodell aufnimmt, wenn nur bei derjenigen Eigenschaftsdeklaration eine «Invers»-Angabe steht, für welche keine expliziten Tripel vorhanden sind. Dieser Vorfall dient aber dennoch als Beispiel dafür, wie viel verzwickte Arbeit bei unterschiedlichem Ontologieaufbau auf einen *Reasoner* zukommen kann.

6.2.2 Mehrere Quellen und dynamisches Nachladen

Es ist nicht die Idee des *Semantic Web*, dass alle ihre eigenen Film- bzw. Geographie-Wissensbasen zusammenstellen. Vielmehr sollen sie sich zu einer umfassenderen Wissensbasis zusammenschliessen. Eine Software wie *Semantic Crystal* könnte dies unterstützen, indem sie neue Teile von Ontologien und Wissensbasen fortlaufend nachlädt.

Wenn die Ontologien zu gross werden, nimmt einerseits die *Performance* des Systems ab, andererseits wird die Darstellung durch eine zu grosse Informationsflut unübersichtlich. Wenn Elemente aus einer Ontologie sich wieder auf Definitionen in anderen Ontologien beziehen, können diese zusätzlichen Informationen erst dann nachgeladen werden, wenn sie tatsächlich gebraucht werden.

6.3 Graphische Darstellung

6.3.1 Blank Nodes

Da mit RDF nur Informationen im Tripel «Subjekt Prädikat Objekt» gespeichert werden können, wird oft auf *Blank Nodes* zurückgegriffen. *Blank Nodes* dienen zur Strukturierung komplexerer Strukturen. Sie treten wie normale Ressource auf, haben aber keine URI, um auf sie zu verweisen, da dies auch nicht nötig ist.

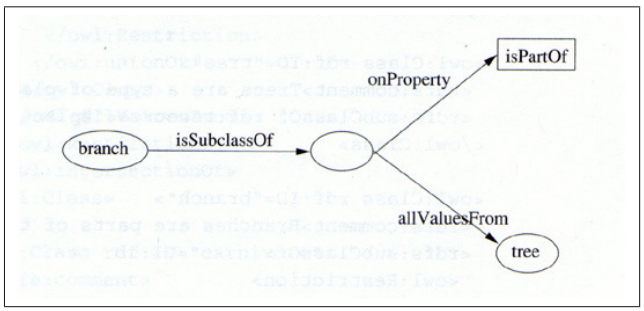

Abbildung 6.1: Beispiel eines Blank Node zu Strukturierungszwecken. (Quelle: [3] S.129)

Auch bei OWL treten *Blank Nodes* häufig auf. Vor allem bei den zahlreichen Strukturierungsmöglichkeiten (vgl. 6.1.1) sind *Blank Nodes* unumgänglich. Abbildung 6.1 zeigt einen *Blank Node*, wie er zur Strukturierung der Eigenschaft `isPartOf` für Instanzen der Klasse `Branch` benutzt wird.

Solche *Blank Nodes* können die Darstellung in *Semantic Crystal* in die Irre führen, deshalb werden sie nicht angezeigt. Beim Beispiel aus Abbildung 6.1 kann dieser anonyme Knoten (so werden *Blank Nodes* bisweilen auch genannt) getrost weggelassen werden, denn die Klassen `Branch` und `Tree` sowie die Eigenschaft `isPartOf` sind genügend durch andere Tripel beschrieben. Auch die Beschränkung, dass als Werte für `isPartOf` nur Instanzen der Klasse `Tree` benutzt werden dürfen, ist unter Umständen nicht wichtig.

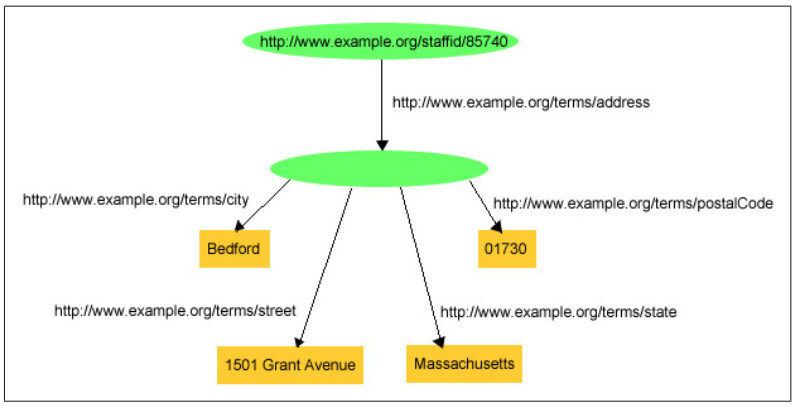

Abbildung 6.2: Beispiel eines Blank Node zu Datengruppierungszwecken. (Quelle: RDF-Primer des W3C, http://www.w3.org/TR/rdf-primer)

Diese Entfernung von *Blank Nodes* bei Ontologie-Beschränkungen ist sicherlich sinnvoll. Allerdings können *Blank Nodes* auch für andere Zwecke benutzt werden, wie Abbildung 6.2 zeigt. Hier ist ein *Blank Node* der Wert der Eigenschaft `address` und führt die thematisch zusammengehörigen Eigenschaften wie `city` oder `street` mit. Falls eine Ontologie eine solche Struktur vorsieht, kann sie mit *Semantic Crystal* nicht adäquat dargestellt werden.

6.3.2 (Un)bekannte Domain

Jede *ObjectProperty*, die mit *Semantic Crystal* angezeigt werden kann, muss über eine bekannte *Domain* und über einen bekannten *Range* verfügen. Ist dies nicht der Fall, kann die Verknüpfung der *ObjectProperty* nicht dargestellt werden. Ein Problem taucht auf, wenn eine Eigenschaft für mehrere Eigenschaften gültig sein soll. Beispiel: Schauspieler kommen aus einem Land, und Filme sind in einem Land produziert worden. Nun erlaubt es OWL nicht, zwei Eigenschaften mit der gleichen URI zu definieren, das eine mal mit `Actor`, das andere mal mit `Movie` als *Domain*.

Eine mögliche Lösung ist, die Eigenschaft auf der tiefsten gemeinsamen Elternklasse anzusetzen, doch das wäre bei `Actor` und `Movie`, die oberste OWL-Klasse überhaupt, nämlich `owl:Class`. Es ist aber nicht sinnvoll, alle Aussagen so allgemein wie möglich zu halten.

Eine andere Lösung sieht vor, die zwei Eigenschaften mit verschiedenen URIs zu versehen. Bei der Film-Ontologie gibt es zum Beispiel die `motherCountry`-Eigenschaft für Schauspieler und die `madeInCountry`-Eigenschaft für Filme. Auch diese Lösung scheint nicht vollumfänglich zufriedenstellend. Man kann schliesslich nicht beliebig viele Namen für ähnliche Eigenschaften erfinden.[2]

Dies kann dazu führen, dass gewisse *ObjectProperties* in Ontologien ohne *Domain*-Angaben definiert werden, was wiederum dazu führt, dass *Semantic Crystal* keine Verknüpfungen machen

[2] Auch die Lösung mit `rdfs:subPropertyOf` ist für dieses Problem nicht zufriedenstellend. Bei den spezialisierten Subeigenschaften müssen genauso neue Namen erstellt werden.

kann.

6.3.3 Weitere offene Sachverhalte

Die Abfragen werden in *Semantic Crystal* mittels Popup-Menüs aufgebaut (vgl. 4.2.2). Intuitiver wäre eine Lösung, in welcher der Benutzer die Elemente aus der Ontologiedarstellung per *Drag and Drop* in die Abfrage übernehmen könnte. Zumindest die Token-Elemente sollten auf diese Weise abgefertigt werden können.

Zudem wurde das «Weitersurfen» bei den Resultaten gewünscht. Dies wurde bereits in 4.2.3 diskutiert und wird hier nur der Vollständigkeit halber erwähnt.

7

Schlussbemerkungen

Ein Browser, der Wissensbasen im Web zu durchsuchen hilft, ist notwendig. Ohne solche Browser werden keine Wissensbasen erstellt. RSS-Reader (News-Aggregatoren), die RSS-Feeds durchsuchen, sind bereits eine primitive Art eines solchen Browsers. Sie lassen sich aber nur für RSS-Wissensbasen anwenden, welche sich immer auf die gleichen wenigen Informationselemente wie Titel, Text, Datum, ... beschränken.

Natürlich können sich auch auf anderen Gebieten ähnlich wie bei RSS spezialisierte Browser durchsetzen. Jedoch wäre ein allgemeiner Browser, welche für Wissensbasen der verschiedensten Arten angewendet werden kann, ein wichtiger Schritt für die Verbreitung von Wissensbasen. Und je mehr solche Wissensbasen existieren, desto weiter ist auch das *Semantic Web*.

Ein HTML-Browser zeigt für Menschen gemachte Präsentationen. *Semantic Crystal* zeigt Strukturen von Wissensbasen an, die auch von Maschinen gelesen werden können. Das grosse Ziel des *Semantic Web* ist eine globale Vernetzung von maschinenlesbaren Informationen. Doch der Beginn liegt im Kleinen. Diese kleinen Wissensbasen lassen sich mit *Semantic Crystal* dank den vier TORC-Elementen bequem abfragen.

Die vier TORC-Elementen sind die folgenden: *Token*, *Output*, *Restriction* und *Connection*. Statt komplizierte SPARQL-Abfragen zu erstellen, stellt sich der Benutzer seine Abfrage mit den vier TORC-Elementen zusammen. Aus Klassen macht er die gewünschten *Tokens*. *DatatypeProperties* benutzt er entweder als *Output*-Variable, oder er nutzt sie um Einschränkungen (*Restriction)* vorzunehmen. Und die *ObjectProperties* nützt er für die Verknüpfung verschiedener *Tokens*.

Die Ontologie, aus welcher sich der Benutzer die Abfrage zusammenbaut, wird in *Semantic Crystal* so dargestellt, dass der Benutzer möglichst rasch, eine möglichst grosse Übersicht über eine Wissensbasis erhält. Zudem werden beim Laden der Wissensbasis *Reasoner*-Tätigkeiten ausgeführt, so dass auch Instanzen von Subklassen oder Beziehungsinstanzen aus inversen Eigenschaften angezeigt werden. Damit wird eine möglichst vollständige Darstellung der Ontologie gewährleistet.

Semantic Crystal kam als Prototyp bei den Testpersonen zwar grundsätzlich gut an, dennoch wurden einige Mängel konstatiert. In der Evaluation wurde u. a. als positiv bewertet, dass die Funktionen gut integriert seien und dass keine Inkonsistenzen und Widersprüche vorhanden seien. Als negativ wurden aber insbesondere die Punkte bewertet, dass das Programm weder lernfreundlich noch leicht zu bedienen sei.

Zum Abschluss kann gesagt werden, dass aus verschiedenen Gründen *Semantic Crystal* für eine Massenbenutzung zuwenig ausgereift ist, dass aber die Grundideen soweit umgesetzt sind, dass sich OWL-Wissensbasen bequem abfragen lassen.

7.1 Danksagung

Ein Dankeschön geht

- an die Teilnehmer der Evaluation (Andreas Brändle, Bartosz Wilczek, Cécile Oberholzer, Esther Kaufmann, Gian-Marco Laube, Ivo Engeler, Peter Vorburger)
- an den Entwickler des grossartigen Graphikframeworks *Prefuse* (Jeffrey Heer) für seine geleistete Arbeit an der Implementation aber auch im Hilfeforum
- an die Entwickler des ebenso grossartigen Semantic-Web-Frameworks *Jena*
- an die Universität Zürich und meine Projektbetreuung (Esther Kaufmann, Abraham Bernstein)
- an die Ingenieure meines Memory-Sticks, auf welchen ich in der Schlussphase dieser Arbeit stündlich Backups durchführte, für dessen Robustheit und Zuverlässigkeit
- und an alle, die sonst noch mitgeholfen haben

A

Fragebogen

Die sieben Testpersonen von *Semantic Crystal* haben in einem separaten Fenster verschiedene Aufgaben bzw. Fragen gestellt bekommen. Vier Aufwärmfragen zum Kennenlernen des Programms, vier «echte» Aufgaben, zehn *SUS*-Fragen und fünf persönliche Fragen. Die Resultate sind der Tabelle A.1 zu entnehmen.

A.1 Aufgaben und Fragen

A.1.1 Aufwärmfragen

- Aufwärmfrage 1 (von 4): Mit welchen Klassen ist die Klasse «Country» direkt verknüpft? Tipp: Gefragt ist nach Klassen, nicht nach ObjectProperties. Die korrekte Antwort lautet: Mit den Klassen «Movie» und «Person» ist die Klasse «Country» verknüpft.
- Aufwärmfrage 2 (von 4): Welches sind die Subklassen von «Movie»? Tipp: Wenn du mit der Maus über eine Klasse fährst, wird der ganze Name angezeigt. Die korrekte Antwort ist: «FilmedMovie» und «AnimatedMovie» sind die Subklassen von «Movie».
- Aufwärmfrage 3 (von 4): Wieviele Instanzen von «AnimatedMovie» gibt es? Welche Attribute weisen die Instanzen von «AnimatedMovie» auf? Welche davon sind Datatype-Properties? Welche davon sind Object-Properties? Die korrekte Antwort lautet: Es gibt 25 Instanzen von «AnimatedMovie». Folgende Attribute sind Datatype-Properties von «AnimatedMovie»: «movieTitle», «movieRating», «publYear». Folgende Attribute sind Object-Properties von «AnimatedMovie»: «hasDirector», «hasGenre», «madeInCountry»
- Aufwärmfrage 4 (von 4): Setze die Attribute «movieTitle» und «publYear» als «OUTPUT» ins Dashboard. Drücke unterhalb der SPARQL-Abfrage (unten rechts) auf «Query!». Interpretiere die Resultateseite. Eine korrekte Antwort lautet: Es erscheinen Namen und Erscheinungsjahre von Zeichentrickfilmen. Tipp: Offene Resultateregister kannst du über das Menü wieder schliessen («File» - «Close all result Tabs»)

A.1.2 Aufgaben

- Aufgabe 1 (von 4): Mach folgende Abfrage. Gesucht sind Titel und Jahreszahlen von «gefilmten» Movies (also Klasse «FilmedMovies», nicht «AnimatedMovies»). Gib nur diejenigen 40 Filme aus, deren Titel alphabetisch als erste kommen.

- Aufgabe 2 (von 4): Mach folgende Abfrage. Gesucht sind Titel und Jahreszahlen von «gefilmten» Movies (also Klasse «FilmedMovies», nicht «AnimatedMovies»). Gib maximal 20 Filme aus, die vor 1990 herausgekommen sind. Diese 20 Filme sollen wiederum alphabetisch nach dem Filmtitel sortiert sein. Tipp: Die Bedingung «publYear kleiner als 1990» lässt sich numerisch formulieren. Das Feld «regex» muss also nicht angekreuzt werden.
- Aufgabe 3 (von 4) Mach folgende Abfrage. Gesucht sind Titel und Jahreszahlen von «gefilmten» Movies und Schauspielernamen (Vor- und Nachname). Gib maximal 10 Filme aus, in welchen der Schauspieler deiner Wahl mitspielt. Sortiere die Filme nach der Jahreszahl. Wähle als Schauspieler jemanden der folgenden Liste: Angelina Jolie, Audrey Tautou, Bruno Ganz, Franka Potente, Gérard Depardieu, Jean Reno, Johnny Depp, Monica Bellucci, Nicole Kidman, Roberto Benigni. ACHTUNG: Gewisse Abfragen können lange dauern (bis zu 30 Sekunden). Nur Geduld!
- Aufgabe 4 (von 4): Mach folgende Abfrage. Gesucht sind Titel und Ratingzahlen der besten Movies des Jahres 2005 (also Klasse «Movie», nicht mehr Subklasse «FilmedMovie»). Zusätzlich soll der Vor- und der Nachname des Regisseurs («Director») und der «CompanyName» des «Distributors» ausgegeben werden. Die besten Filme kriegt man, indem man das Resultat nach dem «MovieRating» in absteigender Reihenfolge sortiert. Beschränke das Resultat auf 20 Datensätze. Und nicht vergessen: Nur die Movies des Jahres 2005 sind gefragt.

A.1.3 SUS-Fragen

Bei den zehn SUS-Fragen musste die Testperson jeweils eine der folgenden Antworten auswählen:

- Damit bin ich gar nicht einverstanden
- Damit bin eher nicht einverstanden
- Weder Zustimmung noch Ablehnung (Oder: Weiss nicht)
- Dem kann ich tendenziell zustimmen
- Dem kann ich sehr stark zustimmen

Die Fragen mit ungerader Nummer sind positiv formuliert, was zur Folge hat, dass starke Ablehung null Punkte gibt und starke Zustimmung vier. Bei Fragen mit gerader Nummer ist es gerade umgekehrt. Sie sind negativ formuliert, so dass starke Ablehnung die Maximalpunktezahl von vier ergeben.

- Abschlussfrage 1 (von 15): Ich denke, dass ich dieses Interface gerne regelmässig benutzen würde.
- Abschlussfrage 2 (von 15): Ich fand dieses Interface unnötig komplex.
- Abschlussfrage 3 (von 15): Ich denke, dass dieses Interface leicht zu bedienen ist.
- Abschlussfrage 4 (von 15): Ich denke, dass ich Unterstützung eines Technikers benötige, um dieses Interface zu benutzen.
- Abschlussfrage 5 (von 15): Ich finde, dass die verschiedenen Funktionen dieses Interface gut eingebunden sind.
- Abschlussfrage 6 (von 15): Ich denke, dass in diesem Interface zu viele Inkonsistenzen/Widersprüche vorhanden sind.

- Abschlussfrage 7 (von 15): Ich stelle mir vor, dass die meisten Leute sehr schnell lernen werden, wie dieses Interface zu benützen ist.
- Abschlussfrage 8 (von 15): Ich fand die Benützung dieses Interface sehr schwerfällig.
- Abschlussfrage 9 (von 15): Ich fühlte mich bei der Benutzung dieses Interface sehr zuversichtlich.
- Abschlussfrage 10 (von 15): Ich musste sehr viel lernen, bevor ich mit der Benützung dieses Interface habe loslegen können.

A.1.4 Persönliche Fragen

Bei den fünf persönlichen Fragen musste die Testperson jeweils eine der folgenden Antworten auswählen:

- (fast) nicht vorhanden
- lückenhaft
- eher gut
- gut
- expertenhaft ausgeprägt

Keine Kenntnisse wurden mit der Zahl 0 kodiert, expertenhaft ausgeprägte Kenntnisse mit der Zahl 4. Die anderen Werte liegen linear dazwischen.

- Abschlussfrage 11 (von 15): Meine Kenntnisse im benutzten Ontologie-Bereich (hier also Kinofilme und Schauspieler) sind ...
- Abschlussfrage 12 (von 15): Meine Kenntnisse in allgemeiner Computeranwendung sind ...
- Abschlussfrage 13 (von 15): Meine Kenntnisse im Bereich Semantic Web (bzw. Ontologie/Wissensrepräsentation) sind ...
- Abschlussfrage 14 (von 15): Meine Kenntnisse im Bereich SQL und Datenbanken sind ...
- Abschlussfrage 15 (von 15): Meine Kenntnisse im Bereich SPARQL sind ...

A.2 Resultate

In Tabelle A.1 sind die Resultate des Fragebogens ersichtlich. Die Punkte 1 – 10 beziehen sich auf den SUS-Test, Punkte 11 – 15 auf die persönlichen Fragen. Die Fragen des SUS-Tests, die eine gerade Zahl haben, sind zwar negativ formuliert, die Bewertung ist aber bereits umgerechnet. Wert 4 heisst also «volle Zustimmung» bei Frage 1 bzw. «volle Ablehnung» bei Frage 2.

Die aufsummierte SUS-Punktezahl multipliziert mit dem Faktor 2.5 ergibt das SUS-Total, das sich auf eine Skala von 0 bis 100 erstrecken kann.

Tabelle A.2 zeigt die benötigten Zeiten und die Anzahl Versuche für die vier Aufgaben an. Die Resultate wurden in Abschnitt 5.3 interpretiert.

Nr.	Beschreibung	$\overline{x}$	$\widetilde{s}$	x_{med}	$x_1, x_2, ..., x_7$
1	Gerne benutzen	2.29	1.28	2	1, 1, 3, 1, 4, 2, 4
2	Komplex	2.43	1.05	3	1, 3, 4, 1, 2, 3, 3
3	Leichte Bedienung	2.14	0.83	2	1, 2, 3, 1, 3, 2, 3
4	Techniker-Unterstützung	2.57	1.05	3	3, 3, 1, 1, 4, 3, 3
5	Funktionen gut eingebunden	3.29	0.45	3	3, 3, 4, 3, 4, 3, 3
6	Inkonsistenzen/Widersprüche	3.14	0.64	3	4, 3, 3, 3, 3, 2, 4
7	Lernfreundlich	1.57	1.29	1	3, 1, 1, 0, 0, 3, 3
8	Schwerfällige Benutzung	2.57	0.49	3	3, 2, 2, 2, 3, 3, 3
9	Zuversichtliches Gefühl	3.00	0.93	3	3, 3, 3, 3, 4, 1, 4
10	Hohe Vorbereitung	2.86	0.99	3	3, 4, 3, 1, 2, 3, 4
	Total SUS	65	13	63	63, 63, 68, 40, 73, 63, 85
11	Kenntnisse Kino	2.29	0.88	2	1, 3, 2, 4, 2, 2, 2
12	Kenntnisse Anwendungen	2.86	0.99	3	2, 3, 3, 4, 3, 1, 4
13	Kenntnisse Semantic Web	2.00	1.51	1	1, 1, 3, 4, 1, 0, 4
14	Kenntnisse SQL, Datenbanken	1.71	0.88	2	1, 2, 2, 3, 2, 0, 2
15	Kenntnisse SPARQL	1.00	1.31	0	0, 0, 3, 1, 0, 0, 3

Tabelle A.1: Resultate der Evaluation. $\overline{x}$ arithmetisches Mittel, $\widetilde{s}$ Standardabweichung, x_{med} Median

Nr. (Zeiten)	$\overline{x}$	$\widetilde{s}$	x_{med}		Nr. (Versuche)	$\overline{x}$	$\widetilde{s}$	x_{med}
1	2:06	0:45	2:00		1	1.8	0.9	1.5
2	2:20	0:49	2:03		2	2.2	1.3	2.0
3	7:09	2:28	7:45		3	3.9	2.3	3.0
4	7:53	3:03	9:09		4	3.2	2.5	2.0

Tabelle A.2: Resultate der Zeitmessung und der Anzahl Abfrageversuche. $\overline{x}$ arithmetisches Mittel, $\widetilde{s}$ Standardabweichung, x_{med} Median

B

Die Film-Ontologie

In diesem Anhang ist die im Rahmen dieser Arbeit erstellte Film-Ontologie in der XML-Syntax dargestellt. Im Anschluss daran befinden sich einige Instanzen.

```
<?xml version="1.0"?>


<!-- PREFIXES -->

<rdf:RDF
    xmlns="http://www.ifi.unizh.ch/ddis/ont/movieont.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xml:base="http://www.ifi.unizh.ch/ddis/ont/movieont.owl">


<!-- ONTOLOGIE META-DATA -->

  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      v1.6, 05.09.2006
    </owl:versionInfo>
  </owl:Ontology>


<!-- CLASSES -->

  <owl:Class rdf:ID="Person"/>
  <owl:Class rdf:ID="Actor">
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>
  <owl:Class rdf:ID="Director">
    <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>

  <owl:Class rdf:ID="Movie"/>
  <owl:Class rdf:ID="FilmedMovie">
    <rdfs:subClassOf rdf:resource="#Movie"/>
    <owl:disjointWith rdf:resource="#AnimatedMovie"/>
  </owl:Class>
  <owl:Class rdf:ID="AnimatedMovie">
    <owl:disjointWith rdf:resource="#FilmedMovie"/>
    <rdfs:subClassOf rdf:resource="#Movie"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="Distributor"/>

  <owl:Class rdf:ID="Country"/>

  <owl:Class rdf:ID="MovieGenre">
    <rdfs:subClassOf>
      <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
          <MovieGenre rdf:about="#GenreAction"/>
          <MovieGenre rdf:about="#GenreAdult"/>
          <MovieGenre rdf:about="#GenreAdventure"/>
          <MovieGenre rdf:about="#GenreChildren"/>
          <MovieGenre rdf:about="#GenreComedy"/>
          <MovieGenre rdf:about="#GenreCrime"/>
          <MovieGenre rdf:about="#GenreDocumentary"/>
          <MovieGenre rdf:about="#GenreDrama"/>
          <MovieGenre rdf:about="#GenreFantasy"/>
          <MovieGenre rdf:about="#GenreFilm-Noir"/>
          <MovieGenre rdf:about="#GenreHorror"/>
          <MovieGenre rdf:about="#GenreMusic"/>
          <MovieGenre rdf:about="#GenreMusical"/>
          <MovieGenre rdf:about="#GenreMystery"/>
          <MovieGenre rdf:about="#GenreRomance"/>
          <MovieGenre rdf:about="#GenreRomanceComedy"/>
          <MovieGenre rdf:about="#GenreRomanceDrama"/>
          <MovieGenre rdf:about="#GenreSci-Fi"/>
          <MovieGenre rdf:about="#GenreShort"/>
          <MovieGenre rdf:about="#GenreThriller"/>
          <MovieGenre rdf:about="#GenreWar"/>
          <MovieGenre rdf:about="#GenreWestern"/>
        </owl:oneOf>
      </owl:Class>
    </rdfs:subClassOf>
  </owl:Class>


<!-- OBJECT PROPERTIES -->

  <owl:ObjectProperty rdf:ID="isDirectorOf">
    <rdfs:domain rdf:resource="#Director"/>
    <rdfs:range rdf:resource="#Movie"/>
    <owl:inverseOf rdf:resource="#hasDirector"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasDirector">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="#Director"/>
    <owl:inverseOf rdf:resource="#isDirectorOf"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasGenre">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="#MovieGenre"/>
    <owl:inverseOf rdf:resource="#isGenreOf"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isGenreOf">
    <rdfs:domain rdf:resource="#MovieGenre"/>
    <rdfs:range rdf:resource="#Movie"/>
    <owl:inverseOf rdf:resource="#hasGenre"/>
  </owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:ID="hasActor">
    <rdfs:domain rdf:resource="#FilmedMovie"/>
    <rdfs:range rdf:resource="#Actor"/>
    <owl:inverseOf rdf:resource="#isActorOf"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isActorOf">
    <rdfs:domain rdf:resource="#Actor"/>
    <rdfs:range rdf:resource="#FilmedMovie"/>
    <owl:inverseOf rdf:resource="#hasActor"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasDistributor">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="#Distributor"/>
    <owl:inverseOf rdf:resource="#isDistributorOf"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isDistributorOf">
    <rdfs:domain rdf:resource="#Distributor"/>
    <rdfs:range rdf:resource="#Movie"/>
    <owl:inverseOf rdf:resource="#hasDistributor"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="madeInCountry">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="#Country"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="motherCountry">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Country"/>
  </owl:ObjectProperty>


<!-- DATATYPE PROPERTIES -->

  <owl:DatatypeProperty rdf:ID="familyName">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="firstName">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="birthday">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="movieTitle">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="publYear">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="movieRating">
    <rdfs:domain rdf:resource="#Movie"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#decimal"/>
  </owl:DatatypeProperty>
```

```
  <owl:DatatypeProperty rdf:ID="countryName">
    <rdfs:domain rdf:resource="#Country"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="companyName">
    <rdfs:domain rdf:resource="#Distributor"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>


<!-- RESTRICTIONS -->

  <owl:Class rdf:about="#Movie">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#movieTitle"/>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
          1
        </owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>


<!-- END ONTOLOGY, BEGIN KNOWLEDGE BASE INSTANCES -->

  <FilmedMovie rdf:ID="m5760">
    <movieTitle rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Pirates of the Caribbean: The Curse of the Black Pearl
    </movieTitle>
    <madeInCountry rdf:resource="#USA" />
    <hasGenre rdf:resource="#GenreAction" />
    <hasGenre rdf:resource="#GenreAdventure" />
    <publYear rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2003</publYear>
    <movieRating rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">
      44.822
    </movieRating>
    <hasActor rdf:resource="#Depp1a" />
    <hasActor rdf:resource="#Wilson2a" />
    <hasDirector rdf:resource="#Verbins1d" />
    <hasDistributor rdf:resource="#BuenaV1v" />
  </FilmedMovie>

  <Actor rdf:ID="Depp1a">
    <firstName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Johnny
    </firstName>
    <familyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Depp
    </familyName>
    <birthday rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
      1963-06-09
    </birthday>
    <motherCountry rdf:resource="#USA" />
  </Actor>

  <Director rdf:ID="Verbins1d">
    <firstName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Gore
    </firstName>
```

```
    <familyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Verbinski
    </familyName>
  </Director>

  <Country rdf:ID="USA">
    <countryName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      United States
    </countryName>
  </Country>

  <Distributor rdf:ID="BuenaV1v">
    <companyName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Buena Vista International
    </companyName>
  </Distributor>

</rdf:RDF>
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21, Mai 2001.

[2] W3C. *OWL Web Ontology Language. W3C Recommendation*. http://www.w3.org/TR/owl-features, 2004.

[3] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.

[4] *Wikipedia. Die freie Enzyklopädie*. http://www.wikipedia.org, 2006.

[5] Tim Berners-Lee. *Primer: Getting into RDF & Semantic Web using N3*. http://www.w3.org/2000/10/swap/Primer.html, 2000.

[6] W3C. *SPARQL Query Language for RDF. W3C Candidate Recommendation*. http://www.w3.org/TR/rdf-sparql-query, 2006.

[7] *The Rule Markup Initiative*. http://www.ruleml.org, 2006.

[8] *Common Logic Standard*. http://philebus.tamu.edu/cl, 2006.

[9] Tom Gruber. *A Translation Approach to Portable Ontology Specifications*, 5 edition, 1993.

[10] Anton Hügli and Poul Lübcke. *Philosophie-Lexikon. Personen und Begriffe abendländischer Philosophie von der Antike bis zur Gegenwart*. Rowohlt, 2000.

[11] Abraham Bernstein, Esther Kaufmann, and Christian Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. In *15th Workshop on Information Technology and Systems (WITS 2005)*, pages 45–50, Dezember 2005.

[12] Sergey Krivov. *Growl*. Department of Computer Science and Gund Institute for Ecological Economics, University of Vermont, http://www.uvm.edu/ skrivov/growl, 2006.

[13] Anselm Spoerri. Infocrystal: a visual tool for information retrieval & management. In *CIKM '93: Proceedings of the second international conference on Information and knowledge management*, pages 11–20, New York, NY, USA, 1993. ACM Press.

[14] *Protege. A free, open source ontology editor and knowledge-base framework*. http://protege.stanford.edu, 2006.

[15] W3C. *SPARQL Query Results XML Format. W3C Candidate Recommendation*. http://www.w3.org/TR/rdf-sparql-XMLres, 2006.

[16] Sun Microsystems. *Java 2 Platform Standard Edition (SE) 5.0*. http://java.sun.com/javase, 2006.

[17] Jeffrey Heer. *The Prefuse Visualization Toolkit. A Java-based toolkit for building interactive information visualization applications*. http://www.prefuse.org, 2006.

[18] *The GraphML File Format*. http://graphml.graphdrawing.org, 2006.

[19] *Jena. A Semantic Web Framework for Java*. http://jena.sourceforge.net, 2006.

[20] *ARQ. A SPARQL Processor for Jena*. http://jena.sourceforge.net/ARQ, 2006.

[21] Søren Lauesen. *User Interface Design. A Software Engineering Perspective*. Addison Wesley, 2005.

[22] Jakob Nielsen. *Usability Engineering*. Academic Press, Elsevier, 1993.

[23] Jon Brooke. SUS. a quick and dirty usability scale. *Usability Evaluation in Industry*, 1996.

[24] Maryland Information and Network Dynamics Lab Semantic Web Agents Project (Mindswap). *Pellet. Open-source Java based OWL DL reasoner*. University of Maryland, College Park, http://www.mindswap.org/2003/pellet, 2006.