



Experimente auf Multicoretechnologie mit Racer und Racermanager

Christian Neuenstadt
5. Juni 2009

Bachelorthesis im Studiengang
Informationstechnologie

Betreut von Prof. Dr. Ralf Möller

Technische Universität Hamburg-Harburg [TUHH]
Arbeitsbereich Software Systems Group [STS]

Danksagung

Ich möchte Prof. Ralf Möller danken für das interessante Thema und für die Beratung und Gespräche parallel zur Erstellung der Arbeit trotz Zeitmangels.

Insbesondere möchte ich mich bedanken bei Atila Kaya für seine intensive persönliche Betreuung und die anregenden Diskussionen. Vorallem für seine Geduld während der Arbeit entstandene Probleme zu lösen, was die spätere Bearbeitung erst möglich gemacht hat.

Ferner möchte ich auch allen weiteren Mitarbeitern des Arbeitsbereiches danken, die mich während meiner Ausarbeitung unterstützt haben.

Nicht unerwähnt möchte ich meinen Kommilitonen Christian Feist lassen. Danke für die gute Zusammenarbeit und die vielen Tipps während der Erstellung der Arbeit und des Studiums.

Darüber hinaus danke ich meinen Freunde und meiner Familie für Ratschläge, Hinweise und Geduld.

Inhaltsverzeichnis

Danksagung	i
1. Einleitung	2
1.1. Aufgabenstellung	3
1.2. Inhaltlicher Aufbau	4
2. Grundlagen	5
2.1. Multicoretechnologie	5
2.1.1. Das Testmodell	6
2.2. Grundlagen des Semanticweb	6
2.2.1. Beschreibungslogik	6
2.2.2. Ontologie	8
2.2.3. RDF	11
2.2.4. OWL	12
3. Analyse	15
3.1. Inferenzsysteme	15
3.2. Racer	15
3.3. nRQL	16
3.4. RacerManager	17
3.5. OWL-QL	19
4. Durchführung	22
4.1. Cliententwicklung	22
4.1.1. Aktuelles System	22
4.1.2. Anforderungen an die Software	23
4.2. Der Client	24
5. Experiment	27
5.1. Generelle Vorüberlegungen	27
5.1.1. Probleme im Vorfeld	28
5.2. Tests auf kleinen ABoxen	29
5.3. Test auf großen ABoxen	30
5.4. Zusammenfassung der Experimente	31
6. Zusammenfassung und Ausblick	33

A. Anhang	34
A.1. LUBM Queries in OWL QL	34
Literaturverzeichnis	41
Eidesstattliche Erklärung	v

Abbildungsverzeichnis

1.1. Semantic Web Layercake	3
2.1. Modell der im Experiment verwendeten CPU	6
2.2. Eine TBox zur Definition einer Familie	7
2.3. Eine Family ABox	8
2.4. semiotisches Dreieck [Sure]	9
2.5. Beispiel einer Ontologie	10
2.6. Beispiel eines einfachen RDF-Graphen mit Literal	12
2.7. Syntax des RDF-Graphen in XML	12
2.8. Beispiel einer Ontologie in OWL	14
3.1. Anfrage in nRQL	16
3.2. Antwort in nRQL	16
3.3. Conjunctive-Query in nRQL	17
3.4. Schematische Darstellung eines Proxyservers	17
3.5. OWL-QL Dialog Schema [Galinski]	19
3.6. Beispiel eines Querypattern	20
3.7. Beispiel eines Continuationtokens	21
4.1. Schematische Darstellung des RacerManager [Doshi]	23
4.2. vereinfachtes Klassendiagramm des Clienttools myClient	25
5.1. Verarbeitungszeit bei einfachen Anfragen und kleiner ABox in Millisekunden	30
5.2. Verarbeitungszeit bei aufwändigen Anfragen und kleiner ABox in Millise-	
kunden	31
5.3. Verarbeitungszeit bei aufwändigen Anfragen und großer ABox in Sekunden	32
A.1. LUBM Query 01	35
A.2. LUBM Query 02	36
A.3. LUBM Query 05	37
A.4. LUBM Query 6	38
A.5. LUBM Query 10	39
A.6. Query mit besonders hohem Rechenaufwand	40

Abstract

Das Internet bietet eine schier unendliche Menge von Dokumenten und Informationen. Das Filtern dieser Informationen ist noch größtenteils dem Menschen selbst überlassen, indem er die Informationen nach speziellen Stichwörtern durchsucht.

Im Semantic Web soll diese Suche durch den Einsatz von Metadaten und durch deren Verarbeitung durch Computer vereinfacht werden. Dafür werden sogenannte Reasoner eingesetzt.

Ein solches System ist RACER. In dieser Arbeit soll untersucht werden, inwiefern Multicoreprozessoren ein Racersystem bei der Verarbeitung von Anfragen unterstützen können.

Kapitel 1.

Einleitung

Auf den vielen Millionen Internet-Hosts weltweit befindet sich eine riesige unüberschaubare Menge ständig vorhandener Daten. Diese Datenansammlung hat aber auch einen entscheidenden Nachteil. Um diese Daten zu durchsuchen, bedient sich der Mensch persönlichen Stichwörtern und Schemata. Der Mensch ist zwar in der Lage, Informationen zu erfassen und untereinander in Bezug zu setzen, eine Maschine kann das allerdings nicht. Diese haben von dem eigentlichen Inhalt der Dokumente meist keine Kenntnis und können sie nur anhand von Stichworten untereinander vergleichen.

Sucht man beispielsweise nach dem Begriff „Müller“, so erhält man Informationen über einen berühmten Fußballspieler, einen bekannten Trinkjoghurthersteller oder auch über den Berufszweig des eigentlichen Müllers. Ähnliches passiert wenn man nach dem Begriff „Fischer“ sucht. Dort findet man Daten zu einem Spielzeugwarenhersteller, einem berühmten Politiker oder aber das Fischereihandwerk.

Darauf aufbauend gründet die Vision des Erfinders des World Wide Web Tim Berners-Lee, der als erstes den Begriff des Semantic Webs prägte. Im Web der nächsten Generation soll das Web mit semantischen Informationen, den sogenannten Metadaten, angereichert werden und es so Programmen ermöglichen, die intendierte Bedeutung der Daten zu erfassen. Heutzutage wird das Semantic Web vor allem durch die Standards des W3C¹ geprägt.

Um dieses Ziel zu verwirklichen, muss eine einheitliche Sprache entwickelt werden, mit der die unterschiedlichen Systeme kommunizieren können.

Mit der Einführung von XML² können mittlerweile freie Informationsstrukturierungen genutzt werden. In Verbindung mit dem vom W3C standardisierten RDF³ werden Objekten eine Bedeutung zugeordnet, die daraufhin von Maschinen verarbeitet werden können. Um die Bedeutung der Informationen zu vereinheitlichen, ist das Nutzen von sogenannten Ontologien notwendig. Eine solche Ontologiesprache ist OWL⁴, die eine Erweiterung

¹World Wide Web Consortium, www.w3.org

²Extensible Markup Language

³Ressource Description Framework

⁴Web Ontology Language

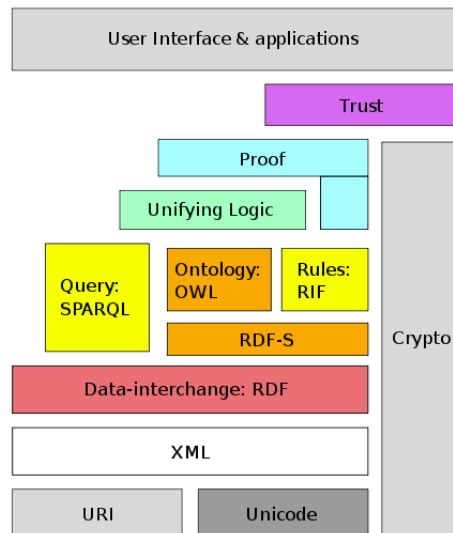


Abbildung 1.1.: Semantic Web Layercake

zu XML und RDF darstellt und ebenfalls vom W3C im Jahre 2004 standardisiert wurde. Die durch diese Standards bereitgestellten Informationen müssen von sogenannten Reasonern verarbeitet werden.

Diese Arbeit soll einen Beitrag leisten, die effiziente Verarbeitung der Daten in einem bestehenden Reasonersystem über offene Standards weiter zu verbessern.

1.1. Aufgabenstellung

Das Ziel dieser Arbeit ist es, Racer in Zusammenhang mit dem Betriebssystem Linux Ubuntu und der CPU INTEL XEON X3320 auf dessen Multiprozessorfähigkeit zu untersuchen.

Dabei sollen mehrere Aspekte untersucht werden, die sich auf die Performance auf einer Multicore-CPU auswirken könnten. Zum einen der Datenfluss zum Server, der über einen vorgeschalteten Proxyserver geregelt wird. Hierbei ist zu berücksichtigen, ob alle Daten den einzelnen Racern in ausreichender Geschwindigkeit zur Verfügung gestellt werden können. Zum anderen die CPU, deren Cores einen gemeinsamen L2Cache nutzen, daraufhin ob dieser die einzelnen CPUs behindert. Und zu guter Letzt, ob das Betriebssystem alle Racerprozesse so auf den Cores organisieren kann, dass auch diese sich nicht gegenseitig behindern.

1.2. Inhaltlicher Aufbau

Die vorliegende Arbeit gliedert sich in sechs Kapitel. Im Anschluss an die Einleitung wird zunächst ein kurzer Einblick in die Multicoretechnologie gegeben, danach werden die formalen Grundlagen des Semantic Web erläutert.

Anschließend gibt es einen Überblick über die technische Umsetzung des Semantic Web am Beispiel des Racersystems und des Racerproxies. In den späteren Teilen wird ein Entwurf für die Experimente auf dem Racersystem und deren Umsetzung in Form eines durch den Autor entwickelten speziellen Clients vorgestellt. Daran schließt sich eine ausführliche Analyse der Tests und der daraus folgenden Ergebnisse an.

Im letzten Kapitel soll eine abschließende Betrachtung sowie Zusammenfassung der Arbeit und entstandene Probleme besprochen werden.

Anmerkung Die Arbeitssprache der Informatik ist Englisch. Um die Internationalität des Themas und die Vergleichbarkeit von Begriffen zu berücksichtigen, wird in dieser Arbeit darauf verzichtet, Begriffe, die auch in deutschsprachigen Veröffentlichungen oft in ihrer englischen Form verwendet werden, ins Deutsche zu übersetzen.

Kapitel 2.

Grundlagen

Um das verwendete Racersystem zu verstehen, sollen zunächst einige Grundlagen im Bereich des Semantic Webs betrachtet werden.

2.1. Multicoretechnologie

Mehrprozessorsysteme und Parallelrechner sind nicht neu, man kennt sie bereits seit Jahrzehnten als sogenannte „Supercomputer“, die vor allem für besonders anspruchsvolle Rechenaufgaben in der Wissenschaft eingesetzt werden.

Bis heute bestehen diese Rechner aus Mehr-Prozessor-Systemen. Seit 2005 nutzen auch fast alle aktuellen Prozessoren in Personal Computern zwei oder mehr Kerne.

Im Gegensatz zu „Supercomputern“ wird die Parallelisierung allerdings direkt in einem Prozessor vorgenommen. Es werden in einen Prozessor mehrere Kerne eingebaut, diese Prozessoren werden auch als Multicore-Prozessoren bezeichnet.

Das gesamte Potenzial mehrerer Kerne kann jedoch nur ausgenutzt werden, wenn mehrere Anwendungen oder Prozesse gleichzeitig auf unterschiedlichen Prozessorkernen laufen können. Um eine einzelne Anwendung von vielen Kernen profitieren zu lassen, muss auch diese in unterschiedliche Prozesse beziehungsweise Threads aufgeteilt werden. Das heißt, eine Anwendung muss bereits vom Programmierer in mehrere parallel ablauffähige Programmstränge unterteilt werden. Diese Prozesse können vom Betriebssystem nicht mehr unterteilt werden, sie laufen also immer nur auf einem Kern. Dadurch wird die Verarbeitungsgeschwindigkeit des einzelnen Prozesses auch wieder maßgeblich von der Taktrate des Kerns bestimmt.

Bei aktuellen Multicore-Systemen teilen sich die Prozessorkerne üblicherweise den Arbeitsspeicher, einen Teil des Caches sowie den System Bus. Daraus folgt, dass je mehr Kerne ein Prozessor hat, desto größer auch der Speicher und die Bandbreite zum Speicher selbst sein muss.

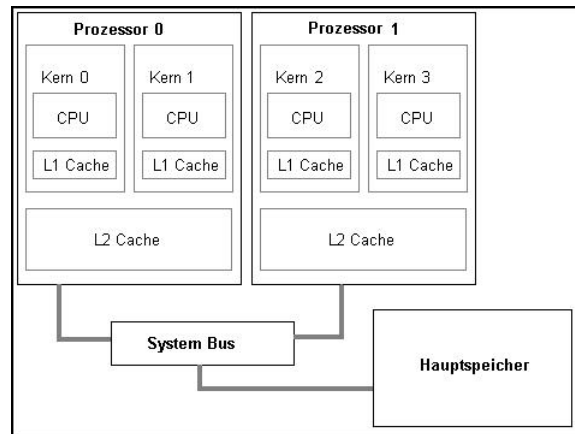


Abbildung 2.1.: Modell der im Experiment verwendeten CPU

2.1.1. Das Testmodell

Beim im Experiment eingesetzten Prozessor handelt es sich um einen Intel Xeon Prozessor des Modells X3320. Dieser Typ ist ein Quadcore-Prozessor, der kein echter Quadcore ist, sondern aus zwei auf einem Chip implementierten Dualcore-Prozessoren. Dieses hat zur Folge, dass sich, wie in Abbildung 2 dargestellt, jeweils zwei Kerne einen 3 MB großen L2Cache teilen müssen. Bei dem später folgendem Experiment soll unter anderem untersucht werden, ob diese Speicherteilung bei zu großen Datenmengen zu einem Nachteil bei der Serverauslastung führen kann.

2.2. Grundlagen des Semanticweb

Wie zu Anfang erwähnt werden im Semantic Web zusätzliche Informationen als sogenannte Metadaten hinterlegt. Diese Daten müssen aufbereitet und ausgewertet werden. Dies passiert über spezielle Beschreibungssprachen, welche auf die Beschreibungslogik aufbauen.

2.2.1. Beschreibungslogik

Die Beschreibungslogik (engl. Description Logic) dient der Wissensrepräsentation und geht aus der Prädikatenlogik hervor [DLHB]. Sie verwendet dabei eine formale Semantik aus der Aussagenlogik, die um parametrisierte Aussagen erweitert worden ist. So können Aussagen wie $p(x)$ oder $q(x,y)$ unter Verwendung der Quantoren $\forall$ („für alle“) und $\exists$ („es existiert“) quantifiziert werden.

Terminological Box

Eine Wissensbasis (engl. knowledgebase), die durch Beschreibungslogiken repräsentiert wird, wird in der Regel formal in zwei Teile eingeteilt. Zunächst muss die Gesamtheit aller Begriffe definiert und beschrieben werden, man nennt dies auch die Terminologie. Dazu müssen Objekte, die in der Beschreibung auftreten können, als sogenannte atomare Konzepte definiert werden. Diese bilden die Grundlagen für alle weiteren Konzepte, die sich von ihnen ableiten lassen. Atomare Konzepte können neue Konzepte beinhalten oder bilden. Sie haben die Form

$$A \sqsubseteq B \text{ oder } A \equiv B \quad (2.1)$$

wobei A und B Konzepte sind. So ist es zum Beispiel auch möglich, aus den Konzepten 'Person' und 'weiblich' das neue Konzept 'Frau' zu entwickeln.

$$Frau \equiv Person \sqcap weiblich \quad (2.2)$$

Neben den Konzepten besteht das Wissen zusätzlich aus Rollen, welche die Beziehungen zwischen den Individuen repräsentieren. So könnte die Rolle $\text{hatKind}(y)$ zur Beschreibung des Konzeptes 'Eltern' dienen. In der Standardbeschreibungslgik $\mathcal{ALC}^1$ würde all dies mit Hilfe des Konzeptes 'Person' dargestellt werden als

$$Eltern \equiv Person \sqcap \text{hatKind}.Person \quad (2.3)$$

Die Gesamtheit der Grundkonzepte stellt die Terminologie der Wissensbasis und wird auch als TBox (**T**erminological **B**ox) bezeichnet. Eine vollständige TBox ist in Abbildung

$$\begin{aligned} Frau &\equiv Person \sqcap weiblich \\ Mann &\equiv Person \sqcap \neg weiblich \\ Mutter &\equiv Frau \sqcap \text{hatKind}.Person \\ Vater &\equiv Mann \sqcap \text{hatKind}.Person \\ Eltern &\equiv Mutter \sqcup Vater \\ Oma &\equiv Mutter \sqcap \text{hatKind}.Eltern \\ Opa &\equiv Vater \sqcap \text{hatKind}.Eltern \end{aligned}$$

Abbildung 2.2.: Eine TBox zur Definition einer Familie

2.2 dargestellt. Ohne dass es explizit angegeben wird, kann nun natürlich auch sofort gefolgert werden, dass Oma und Opa eine Person, bzw. eine Frau oder ein Mann sind.

¹Attributive Language with Complements

Assertional Box

Der zweite Teil der Wissensbasis und damit das Gegenstück zur TBox bildet die ABox (engl. **A**ssertional **B**ox). In der ABox werden nun nicht mehr Konzepte, sondern tatsächliche Individuen definiert und beschrieben, indem sie den Grundkonzepten der TBox zugewiesen werden. Die Individuen dieser Weltbeschreibung kann man ähnlich wie die Instanzen einer Klasse verstehen. Man bezeichnet diese Zuweisung auch als 'concept assertion'. Ein Beispiel für eine solche Zuweisung wäre

$$Elisabeth : Frau \quad (2.4)$$

Mittels der sogenannten 'role assertion' werden Individuen einer Rolle zugewiesen, dies geschieht auf gleiche Art.

$$(Elisabeth, Charles) : hatKind \quad (2.5)$$

Die Gesamtheit aller concept und role assertions bildet die ABox der Individuen. Abbildung 2.3 zeigt das Beispiel einer ABox als Assertion zur vorangegangenen TBox.

$$\begin{aligned} & Queen_{mum} : Frau \\ & (Queen_{mum}, Elisabeth) : hatKind \\ & Elisabeth : Frau \\ & (Elisabeth, Charles) : hatKind \\ & Charles : Mann \\ & (Charles, Andrew) : hatKind \end{aligned}$$

Abbildung 2.3.: Eine Family ABox

2.2.2. Ontologie

Der Ontologiebegriff stammt ursprünglich aus dem Bereich der Philosophie. Er leitet sich aus dem griechischen Wort 'Ontos', für 'das Seiende', sowie 'Logos' die Lehre, ab und ist somit auch im Duden Fremdwörterbuch als 'Lehre des Seienden' definiert. Untersucht werden grundlegende Fragen der Existenz. Es werden Beziehungen zwischen Objekten analysiert und beschrieben, allerdings geht man in der Lehre der Ontologie nur von einer derartigen Beschreibung aus.

In der Informatik jedoch wird von vielen Beschreibungen ausgegangen, hier kommen

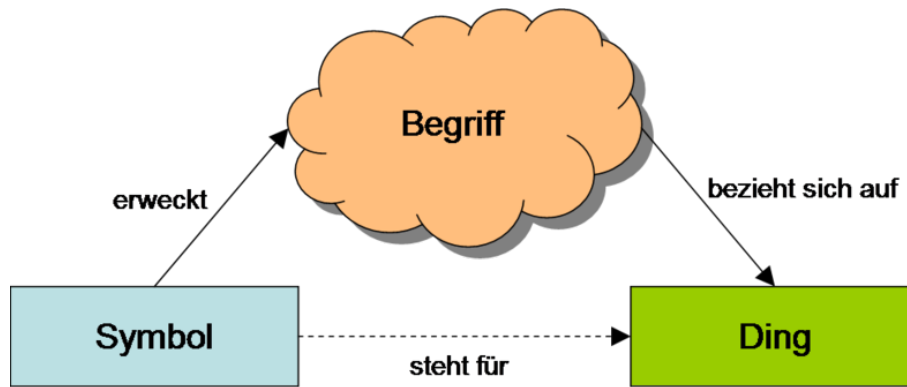


Abbildung 2.4.: semiotisches Dreieck [Sure]

Ontologien in einer Vielzahl vor. An diesem Thema wird beonders im Bereich der künstlichen Intelligenz und des Semantic Webs geforscht.

Eine der häufigsten Definitionen stammt von Gruber:

„An ontology is an explicit specification of a conceptualization.“[Gruber]

Ontologien dienen der Beschreibung von Beziehungen zwischen Symbolen, Begriffen und Objekten, deren Zusammenhang sich im sogenannten semiotischen Dreieck darstellt. Das semiotische Dreieck ist in Abbildung 2.4 dargestellt.

Dabei soll die Mehrdimensionalität von Begriffen, Zeichen und Gegenständen bildlich veranschaulicht werden. So haben Symbole sowohl semantische, als auch syntaktische Bedeutung. Die Beziehung zwischen Symbol und Gegenstand besteht allerdings nicht direkt, sondern wird nur über eine sogenannte 'Vermittlungsinstanz' erzeugt und wird deshalb mittels einer gestrichelten Linie dargestellt [Sure].

Aufbauend auf die vorher erläuterte Beschreibungslogik, stellen sich Ontologien als eine gemeinsame Terminologie dar, in der Konzepte und Begriffe formal beschrieben sind. Somit wird die Bedeutung der Konzepte erst deutlich, wenn auch die dazugehörige Ontologie bekannt ist. Ontologien werden oft in hierarchischen Beziehungen dargestellt. Das Beispiel in Abbildung 2.5 soll dies näher erläutern.

Im Beispiel lassen sich mehrere Klassen erkennen, dargestellt als grüne Kästen. Verbunden sind sie mit Pfeilen, die deutlich machen, dass es sich bei einem „Radfahrer“ um eine „Person“ und bei einem „Radrennen“ um einen „Wettbewerb“ handelt. Da „Personen“ an „Wettbewerben“ teilnehmen, lässt sich sofort folgern, dass auch „Radfahrer“ an „Radrennen“ teilnehmen können. Daneben gibt es noch konkrete Individuen. „Max“ ist ein Individuum des Typs „Radfahrer“ und die „Tour de France“ eine des Typs „Radrennen“. Unsere Ontologie besteht also in einer grafischen Darstellung, welche die Interpretation der darin erhaltenen Daten ermöglicht.

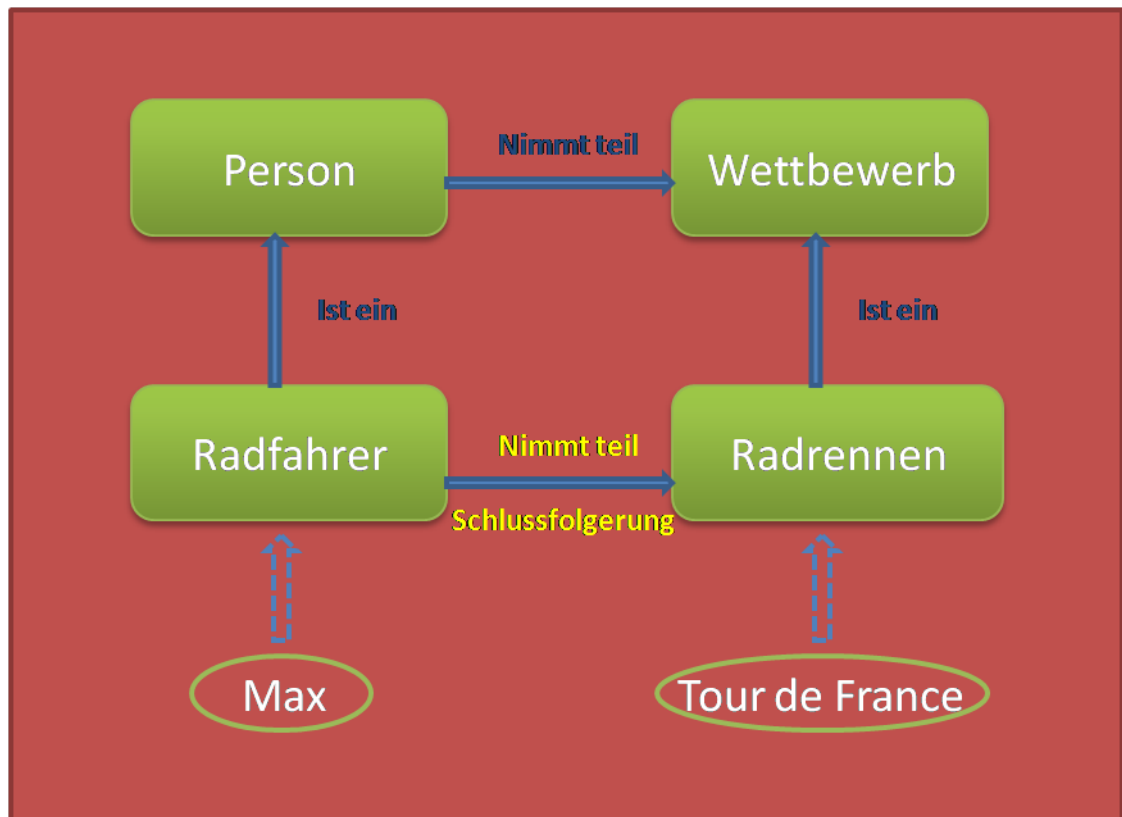


Abbildung 2.5.: Beispiel einer Ontologie

2.2.3. RDF

Das *Resource Description Framework* kurz RDF ist eine formale Sprache zur Beschreibung von strukturierten Informationen. Durch die Nutzung von RDF soll es möglich werden, Daten im Web auszutauschen, ohne dass dadurch eine Bedeutung verloren gehen kann. Somit geht es im Gegensatz zu HTML und XML nicht mehr nur um die Darstellung, sondern um die Weiterverarbeitung und den Erhalt der Dokumente und Daten.

Daher wird die Sprache RDF oft als das grundlegende Darstellungsformat des Semantic Web angesehen [Hitzler]. In solchen Metadaten könnten zum Beispiel Informationen über Autor, Titel oder Verlag abgelegt und mit einem Buch verknüpft sein. Die Beschreibung von Aussagen in RDF findet dabei aber nicht hierarchisch, sondern in Form eines Graphen aus Knoten und Kanten statt, sowie in der RDF/XML-Syntax.

Das RDF-Modell besteht aus drei unterschiedlichen Objekttypen: den Ressourcen, Eigenschaften und Werten, man bezeichnet sie zusammen auch als Triple. Der Wert solch einer Aussage kann wiederum die Ressource einer weiteren Aussage sein. Durch die Verknüpfung von Aussagen ergibt sich ein gerichteter Graph aus Ressourcen und ihren Eigenschaften. Zur eindeutigen Bezeichnung von Ressourcen und Eigenschaften werden URIs² verwendet. Durch URIs bekommen alle Bezeichnungen eine Verankerung im Web [Birkenbiehl].

In der Spezifikation zu RDF wird eine Formulierung in Satzform der Art Subjekt, Prädikat, Objekt vorgeschlagen. Wobei die Ressource die Rolle des Subjektes, die Eigenschaft die des Prädikats und der Wert die des Objektes einnimmt. Ein sich daraus ergebender RDF-Graph ist in Abbildung 2.6 dargestellt.

Sie zeigt einen einfachen RDF-Graphen und beschreibt in diesem Fall die Beziehung zwischen dem „Buch1“ und dem „Verlag1“ bei dem dieses erschienen ist. Subjekt, Prädikat und Objekt werden durch Bezeichner, den URIs, repräsentiert. Zusätzlich ist das Subjekt des Buches mit einem Literal verknüpft. Es enthält lediglich die Information über den Autor als String und kann im Gegensatz zu den Objekten nicht wieder als Subjekt verwendet werden. Literale werden als Rechteck dargestellt.

Möchte man den Graphen in der Syntax von RDF/XML darstellen, dann entspricht der reine Quelltext dem in Abbildung 2.7.

Die Darstellung erfolgt geordnet nach dem Subjekt, in diesem Fall hat es die Bezeichnung „http://www.Beispiel.de/Buch1“. Der Identifier von Subjekt und Objekt wird durch „rdf:about“ gekennzeichnet, das Prädikat wird ohne direkten Identifier angegeben. Die Namespaces „rdf“ und „bsp“ sind beliebig.

²Uniform Resource Identifier

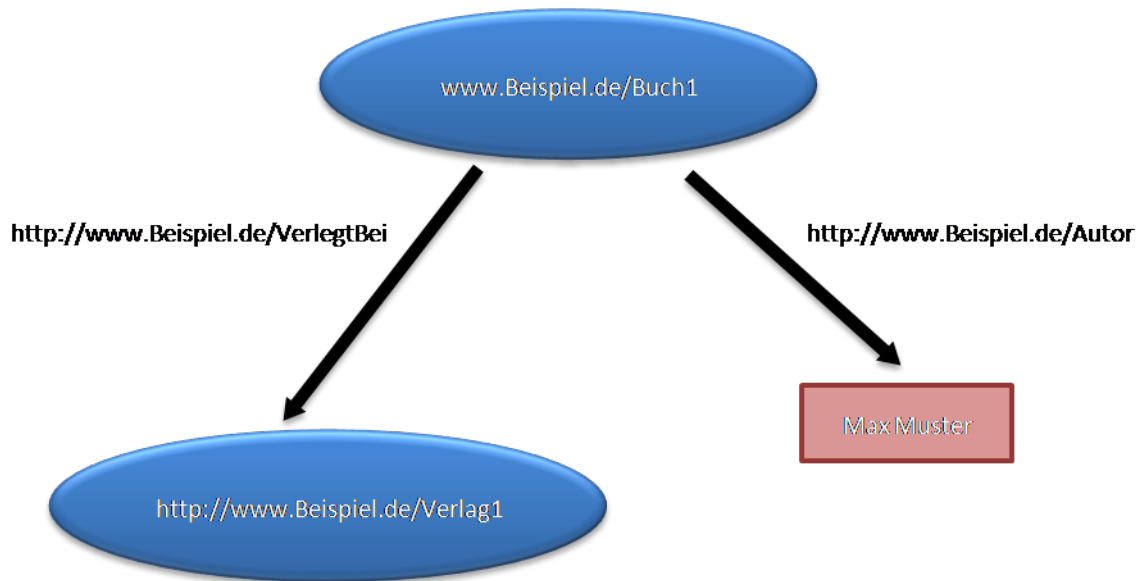


Abbildung 2.6.: Beispiel eines einfachen RDF-Graphen mit Literal

```

1 <rdf:Description rdf:about=" 'http://www. Beiispiel .de/
  Buch1" '>
2 <bsp:Author>Max Muster</bsp:Author>
3 <bsp:VerlegtBei>
4   <rdf:Description rdf:about=" 'http://www.
     Beispiel.de/Verlag1" ' />
5 </bsp:VerlegtBei>
6 </rdf:Description>

```

Abbildung 2.7.: Syntax des RDF-Graphen in XML

2.2.4. OWL

RDF ist zur Modellierung einfacher Ontologien geeignet, wie wir im vorherigen Kapitel gezeigt haben. Allerdings hat RDF nur beschränkte Ausdrucksmittel und ist somit für die Darstellung komplexerer Zusammenhänge nicht mehr ausreichend. So ist es zum Beispiel nicht möglich, eine Aussage wie „Jedes Projekt hat mindestens einen Mitarbeiter“ in RDF zu formulieren.

Zur Darstellung solch komplexen Wissens verwendet man ausdrucksstarke Repräsentationssprachen, die - wie auch RDF - auf formaler Logik basieren. Durch logisches Schlussfolgern wird dadurch auch der Zugriff auf impliziertes Wissen ermöglicht. [Hitzler] Zur Darstellung solcher Konstrukte wurde die Ontologiesprache OWL³ 2004 vom W3C entwickelt.

³<http://www.w3.org/2004/OWL/>

Die Abkürzung OWL steht für Web Ontology Language und hat sich vor allem im Bereich des Semantic Web schnell durchsetzen können. OWL erweitert die Sprache RDF unter anderem mit der Darstellung von:

- Kardinalität (Mengenangabe von Objekte z.B. „mindestens Eins“)
- Klassenbeziehungen (zB. Schnittmengenbildung und Vereinigung von Klassen)
- Beziehungen zwischen Individuen (z.B. gleiche Bezeichner für ein Individuum durch *owl:sameAs*)

Da es in der Praxis zu verschiedenen Anforderungen im Sinne von Komplexität und Entscheidbarkeit kommt, wurden für den Anwender drei unterschiedliche Teilsprachen von OWL entwickelt. OWL-Lite, OWL-DL und OWL-Full:

OWL-Lite – ist die syntaktisch einfachste Sprache.

- ist Teilsprache von OWL-DL, sowie OWL-Full und entscheidbar
- ist allerdings wenig ausdrucksstark.

OWL-DL – ist bereits wesentlich ausdrucksstärker.

- enthält OWL-Lite und ist Teilsprache von OWL-Full.
- ist genauso entscheidbar wie OWL-Lite
- wird von aktuellen Softwarewerkzeugen fast vollständig unterstützt

OWL-Full – enthält OWL-DL und OWL-Lite und als einzige der drei Varianten ganz RDFS⁴

- ist sehr ausdrucksstark und unentscheidbar
- wird durch aktuelle Software aber nur bedingt unterstützt.

OWL-DL ist die wohl am weitesten verbreitete der drei Varianten und wird von aktuellen Softwarewerkzeugen fast vollständig unterstützt, zum Beispiel auch vom RACER System, das in einem der folgenden Kapitel vorgestellt wird. Dahingegen wird OWL-Full von aktueller Software nur bedingt unterstützt [Hitzler].

⁴Resource Description Framework Schema

```

1 <?xml version="1.0" encoding="UTF-16"?>
2 <rdf:RDF xmlns:owl = "http://www.w3.org/2002/07/owl#"
3     xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns"
4     xmlns:rdfs = "http://www.w3.org/1999/02/22-rdf-schema#">
5
6 <owl:Class rdf:about="Radfahrer">
7     <rdfs:subClassOf><owl:Class rdf:ID="Person" />
8 </rdfs:subClassOf>
9 </owl:Class>
10 <owl:Class rdf:ID="Radrennen">
11     <rdfs:subClassOf><owl:Class rdf:ID="Wettbewerb" />
12 </rdfs:subClassOf>
13 </owl:Class>
14 <owl:ObjectProperty rdf:ID="nimmtteil">
15 <rdfs:domain rdf:resource="#Person" />
16 <rdfs:range rdf:resource="#Wettbewerb" />
17 </owl:ObjectProperty>
18 </rdf:RDF>

```

Abbildung 2.8.: Beispiel einer Ontologie in OWL

Wir wollen uns in einem kleinen Beispiel noch einmal die Syntax der OWL-Sprache verdeutlichen. Dafür nehmen wir das Ontologiebeispiel aus dem vorherigen Kapitel. In der OWL-Syntax stellt es sich dar wie in Abbildung 2.8.

In diesem Fall wird die „ist ein“-Beziehung durch eine Definition als Unterklasse mit Hilfe des subClassOf-Bezeichners realisiert. Die Relation „nimmt teil“ wird dadurch definiert, dass als Subjekt nur Individuen des Typs „Person“ und als „Objekt“ nur Individuen des Typs „Wettbewerb“ zugelassen sind.

Kapitel 3.

Analyse

3.1. Inferenzsysteme

Inferenzsysteme können durch Schlussfolgerungen neues Wissen aus zuvor definierten Wissensbasen ableiten. Zur Auswertung von Anfragen auf Wissensbasen werden sogenannte Reasoner eingesetzt. Mit ihnen ist es möglich, impliziertes Wissen aus explizitem Wissen zu folgern.

Ein besonderer Inferenzdienst ist zum Beispiel *Subsumption*. Durch Subsumption lässt sich ermitteln, ob ein Konzept C ein allgemeineres ist als das Konzept D. Woraus man implizit folgern könnte, dass Instanzen von D automatisch Instanzen vom Typ C sind. Eine mögliche Fragestellung für Subsumption wäre zum Beispiel: "Ist eine Mutter immer eine Frau?"

3.2. Racer

Ein Reasonersystem welches die Auswertung von OWL-Daten ermöglicht ist RACER¹. RACER ist eine Software der Racer Systems GmbH und Co. KG², die seit einigen Jahren von Prof. Ralf Möller, Prof. Volker Haarslev und Dr. Michael Wessel weiterentwickelt wird. Für Ontologien auf OWL oder RDF Basis bietet RACER unter anderem den folgenden Service an [Racer-User-Guide]:

- Konsistenzüberprüfung einer OWL Ontologie
- Finden von impliziten Beziehungen aufgrund von Unterklassendeklaration in der Ontologie.

¹Renamed ABox and Concept Expression Reasoner

²<http://www.racer-systems.com>

- Finden von Synonymen für Ressourcen
- Inkrementelles Beantworten von Anfragen. Der Nutzer kann selber entscheiden, ob er alle Antworten oder nur einen Teil und die restlichen später abfragen möchte.

Für weitergehende Informationen möchte der Autor an dieser Stelle auf den ausführlichen Racer-User-Guide verweisen [Racer-User-Guide].

3.3. nRQL

Um Schlussfolgerungen über das in A- und T-Boxen auf einem Racerserver hinterlegte Wissen zu erhalten, müssen Anfragen über eine Schnittstelle gestellt werden. Diese Anfragen nennt man aus dem Englischen auch Queries. Die Anfrageschnittstelle ist im Falle von Racer die nRQL³. Sie ist eine mächtige Anfragesprache auf A-Boxen, OWL und RDF-Dokumente, die im folgenden Teil kurz vorgestellt werden soll. Eine Query in nRQL besteht aus einem Queryhead und einem Querybody. Im Headabschnitt befindet sich die Anfrageoperation, sowie die Variable auf die sie sich bezieht. Im Bodyabschnitt besteht sie aus der Variable und den Bedingungen, welche die ausgewählten Individuen erfüllen sollen. Da Racer auf Lisp basiert, werden alle Anfragen in Klammern geschrieben. In Abbildung 3.1 ist eine Retrieve-Operation im Beispielcode angegeben.

$$(RETRIEVE (?X) (?X\ KIND)) \quad (3.1)$$

Abbildung 3.1.: Anfrage in nRQL

Eine Übersetzung in unsere Sprache könnte lauten: “Suche alle Individuen, die eine Instanz des Konzeptes KIND sind“. Eine mögliche Antwort findet man in Abbildung 3.2

$$(((?X\ MAX)) ((?X\ MORITZ))) \quad (3.2)$$

Abbildung 3.2.: Antwort in nRQL

Als Antwort gibt der Racerserver eine Liste von Individuen zurück, welche die vorher definierten Anfragebedingungen erfüllen. Das „X“ ist dabei die Variable, auf die sie sich beziehen.

Ferner sind natürlich auch kompliziertere Anfrageformen möglich, welche zum Beispiel die Erfüllung von mehreren Bedingungen erfordern. Ein solches Beispiel ist in Abbildung 3.3 dargestellt.

³new Racer Query Language

(RETRIEVE (?X) ((AND (?X WEIBLICH(?X HATKIND.PERSON)) (3.3)

Abbildung 3.3.: Conjunctive-Query in nRQL

Diese Art der Anfrage wird auch „Conjunctive-Query“ genannt, da die Antwortbedingung eine Konjunktion aus mehreren Bedingungen ist. In dem abgebildeten Beispiel wird implizit nach allen Individuen gesucht, die gleichzeitig eine Mutter sind. Für eine genauere Erläuterung wird wieder auf den Racer-User-Guide verwiesen [Racer-User-Guide].

3.4. RacerManager

Der Racermanager ist ein um mehrere Funktionen erweiterter Proxyserver, der für das Racersystem entwickelt wurde. Racer war ursprünglich nicht multithreadingfähig, das bedeutet, dass der Server nur eine Anfrage zur Zeit beantworten konnte und während der Bearbeitungszeit alle weiteren zurückweisen musste.

Ein Proxyserver ist dabei eine Art Vermittler zwischen Client und Server. Er nimmt die Anfrage des Clients in Empfang und leitet diese an einen oder mehrere Server weiter. Für den Client ist dabei häufig nicht ersichtlich ob er direkt mit dem Server oder nur mit einem vorgeschalteten Proxy kommuniziert.

Der Racerproxy wurde mit der Zielsetzung entwickelt, Anfragen von mehreren Clients anzunehmen und auf verschiedene Racerserver zu verteilen, um somit eine Art künstliches Multithreading zu ermöglichen.

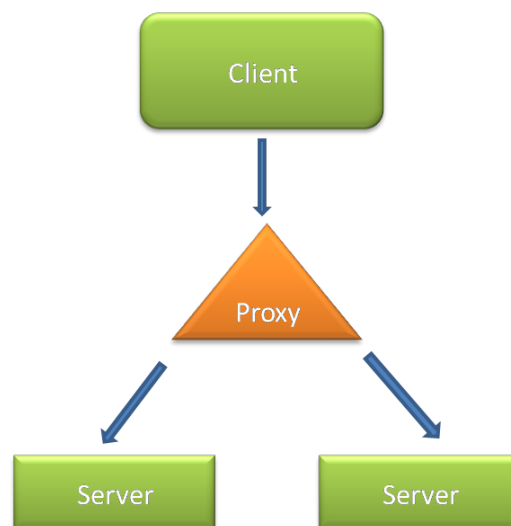


Abbildung 3.4.: Schematische Darstellung eines Proxyservers

Es wurden dazu alle Anfragen in einer einfachen FIFO⁴-Queue gespeichert und nacheinander abgearbeitet. Dadurch wurde es möglich bei einem Ausfall eines Racerserver das Routing auf andere zu verlagern und, was der größte Nutzen war, man konnte nun mehrere Anfragen parallel stellen.

Auf dem Proxy war dabei eine einfache Round-Robin-Struktur implementiert, die eine Anfrage jeweils einfach an den nächsten zu erreichenden Server weiter gab. Zu diesem Zeitpunkt mussten noch alle Queries, die von einem Client kamen, in nRQL gestellt werden.

Nach der Entwicklung von OWL durch das W3C im Jahre 2004 wurde Racer mit der Version 1.8.x im darauffolgenden Jahr auf ein System umgestellt, das die Vorteile dieser neuen Beschreibungssprache nutzen sollte. OWL ermöglichte unter anderem interaktive Anfragen in Zusammenhang mit der neuen Anfragesprache OWL-QL. In diesem Modus war es nun möglich, gezielt eine bestimmte Anzahl von Antworten zu erhalten. So zum Beispiel nur die ersten Fünf von einhundert Antworten und im Anschluss erst mit weiteren Anfragen dann die restlichen Individuen. Dieser Modus wurde von den Racer-programmierern „Tuple-at-a-time“-Modus genannt.

Um diese neuen Funktionen allerdings auch auf dem Racermanager zu implementieren, mussten diesem mehrere Erweiterungen hinzugefügt werden.

Der Proxyserver wurde dabei zunächst in der Kommunikation mit dem Racerserver grundlegend verändert. So mussten Übergabehandles geschaffen werden, mit denen vereinbart werden konnte, wie viele Antworten angefordert und welche schon abgefragt wurden. Dadurch kam es zur Entwicklung von Dialogobjekten auf dem Proxy, in denen zum Beispiel festgelegt wird, welcher Racerserver zuvor die Anfrage beantwortet hat, da auch nur dieser anschließend in der Lage ist, weitere Antworten zu versenden.

Diese Erweiterungen wurden entwickelt von Jog Ratna Maharjan und Tejas Doshi. Für nähere Informationen zu diesen Arbeiten wird an dieser Stelle auf die Veröffentlichungen [Maharjan] und [Doshi] verwiesen.

Auf der anderen Seite war es auch nötig, die Kommunikation zwischen Client und Racermanager zu reorganisieren. So sollte nun anstatt einer einfachen Übergabe der nRQL-Anfrage ein Webservice implementiert werden, der in der Lage ist Queries in OWL-QL zu verarbeiten und gleichzeitig eine Übersetzung der beiden Anfragesprachen zwischen Client und Proxy leisten kann.

Diese Funktionen implementierte Jan Galinski. Dazu veröffentlichte er eine Publikation im Jahre 2005 [Galinski]. Weitere Funktionen, die realisiert wurden, sind z.B. ein XML-Validator und ein Cachesystem auf dem Proxy, das alle Antworten zu den Queries anhand eines Dialogs speichert und bei Übereinstimmung sofort die Antwort liefern kann.

⁴First In First Out

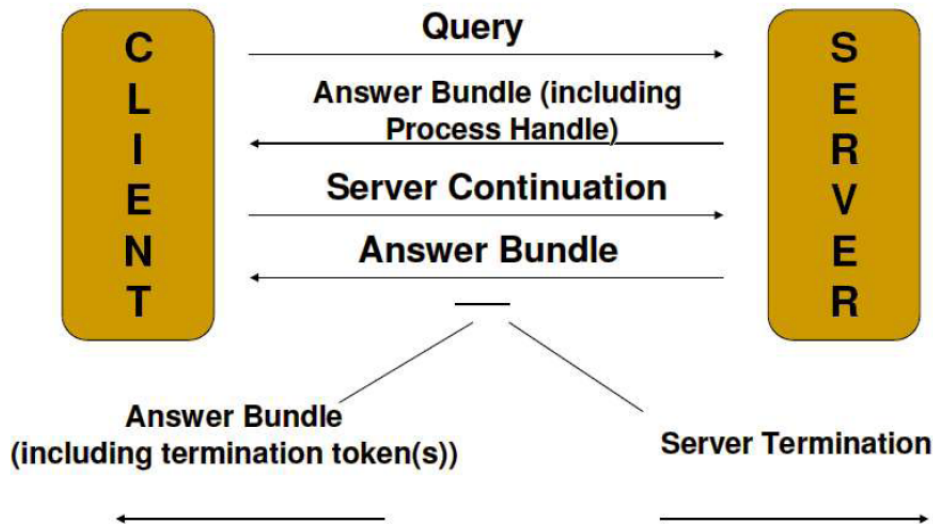


Abbildung 3.5.: OWL-QL Dialog Schema [Galinski]

3.5. OWL-QL

OWL-QL⁵ wurde vom W3C parallel zur Beschreibungssprache OWL entwickelt, sie soll den standardisierten Zugriff auf Wissensbasen und Reasoner Systeme ermöglichen. Seit 2007 ist eine Erweiterung zu OWL in der Entwicklung und mit ihr OWL 2 QL⁶.

In der Spezifikation der formalen Sprache werden Anfrage- und Antwortqueries anhand eines XML-Schemas definiert. Zudem wird ein spezielles Protokoll für den Dialogprozess vorgeschlagen.

Der Dialog

Eine Besonderheit, die mit OWL-QL implementiert worden ist, sind interaktive Abfragen. Dabei werden zunächst nur eine zuvor definierte Anzahl von Antworten vom Server zurückgeschickt. In einer weiteren Nachricht, die ein Continuationtoken enthält, können anschließend vom Client weitere Antworten angefordert werden, solange bis eine der beiden Seiten ein Terminationtoken sendet.

⁵Web Ontology Query Language

⁶www.w3.org/2007/OWL

Die Anfrage

Ein OWL-QL Dokument besteht aus einem XML-Schema, das wiederum aus mehreren unterschiedlichen Pattern besteht. Im Rahmen dieser Arbeit soll nun kurz auf einige Wichtige davon anhand eines Beispiels eingegangen werden.

```

1  <?xml version="1.0"?>
2  <query
3    xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
4    xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
5    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6    xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
7  <queryPattern>
8    <rdf:RDF>
9      <rdf:Description
10        rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
11        >
12        <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#GraduateStudent"/>
13      </rdf:Description>
14      <rdf:Description
15        rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
16        >
17        <uni:takesCourse rdf:resource="http://www.Department0.University0.edu/GraduateCourse0"/>
18      </rdf:Description>
19    </rdf:RDF>
20  </queryPattern>
21  <mustBindVars>
22    <var:x/>
23  </mustBindVars>
24  <answerKBPattern>
25    <kbRef rdf:resource="http://www.Beispiel.de/KnowledgeBase.owl"/>
26  </answerKBPattern>
27  <answerSizeBound>
28    1
29  </answerSizeBound>
30 </query>

```

Abbildung 3.6.: Beispiel eines Querypattern

Querypattern

Das Querypattern wird nur bei der ersten Abfrage übergeben, es enthält alle Bedingungen und Variablen. Die Form gleicht in ihrer Art der aus einer OWL-Wissensbasis, als Anordnung im Stile von RDF-Tripeln bei der die zu erfragenden Individuen durch Variablen ersetzt wurden. In Abbildung 3.6 ist ein solches Pattern dargestellt. Es besteht aus zwei Bedingungen, die erste besagt: „X enthält als Individuen alle Studenten“ und die Zweite fordert, dass in X nur Kursteilnehmer vom „GraduateCourse0“ enthalten sein dürfen.

Gebundene Variablen

Im Querypattern können beliebig viele Variablen verwendet werden. Um diese anschließend genauer zu spezifizieren, können sie auf verschiedene Arten gebunden sein

mustbind Variablen können zwingend gebunden werden. Das heißt, es werden nur Ergebnismengen ausgegeben, welche diese Bedingung erfüllen und die Variable enthalten. Es muss in jedem Query immer mindestens eine mustbind-Variable definiert werden.

maybind Variablen können, aber müssen nicht gebunden sein, das heißt es werden Individuen ausgegeben, unabhängig davon, ob die Variable definiert ist oder nicht. Wird nach Studenten gefragt, die Kurse besuchen, könnte in der TBox zum Beispiel definiert sein, dass jeder Student mindestens einen Kurs besucht, aber manchen Studenten in der ABox trotzdem keiner zugewiesen ist. Wenn die Variable „Kurs“ maybind gebunden ist, werden somit Studenten mit und ohne Kurs ausgegeben.

dontbind Variablen dürfen nicht gebunden sein. In einer Ausgabe würden dann z.B. Studenten in jedem Fall ohne Kursangabe ausgegeben werden.

Weitere Pattern

In unserem Beispiel enthält das Query noch zwei weitere Patternteile. Unter „answerKB-pattern“ wird der Pfad zur Wissensbasis festgelegt, auf dem die Anfrage getätigt werden soll. Diese kann vom Racer dann direkt geladen werden. Unter „answerSizeBound“ wird zusätzlich die Anzahl der verlangten Antworten angegeben.

Continuationtoken Falls vom Racer noch nicht alle Antworten verschickt worden sind, wird in der Antwort ein sogenanntes Processhandle mit verschickt. Mit dem kann der Client anschließend weitere Antworten anfordern, ohne noch einmal ein weiteres Querypattern schicken zu müssen. Er schickt dazu das Processhandle innerhalb eines Continuationtokens an den Server.

```

1  <serverContinuation>
2  <processHandle>31327050#14775623</processHandle>
3  </serverContinuation>
4

```

Abbildung 3.7.: Beispiel eines Continuationtokens

Mit Hilfe des Processhandles kann der Client so lange weitere Antworten verlangen, bis er entweder ein Terminationtoken vom Server erhält oder selbst eines verschickt.

Kapitel 4.

Durchführung

4.1. Cliententwicklung

Die Zielsetzung dieser Arbeit soll es, so wie in der Einleitung beschrieben, sein, das Verhalten von RacerManager und RacerServer in Zusammenhang mit Multicore CPUs zu untersuchen. Dafür soll in diesem Kapitel zunächst das aktuelle System genauer betrachtet werden, um dann zu entscheiden, welche Anforderungen eine eventuelle Testsoftware erfüllen muss und wie die mögliche Implementierung aussehen könnte.

4.1.1. Aktuelles System

Das Racermanagersystem¹ ist opensource und auf Sourceforge.net verfügbar, in einer ersten Version an der Fachhochschule Wedel entstanden und wurde in einem Projekt am Institut für Softwaresysteme² an der technischen Universität Hamburg-Harburg weiterentwickelt.

Auf der Clientseite verhält sich der Racermanager wie ein OWL-QL Webservice, der seine Querys an einen entfernten DL-Reasoner weiterleitet. Im vorliegenden System handelt es sich dabei um den, ebenfalls vom Institut entwickelten, Reasoner RacerPro³. Um den OWL-QL Webservice mit der nRQL-Schnittstelle des Racers zu verbinden, wurde zudem ein Übersetzungsmodul entwickelt, das die Syntax beider Sprachen versteht.

Die gesamte Programmierung des Proxyserverns erfolgte in Java und wird als mehrschichtige Applikation auf einem Apache Tomcatserver [Apache Tomcat] ausgeführt, die als Modell in Abbildung 4.1 dargestellt ist. Für die eigentliche Implementierung wurden

¹<http://racerproject.sourceforge.net/>

²www.sts.tu-harburg.de

³www.racer-systems.com

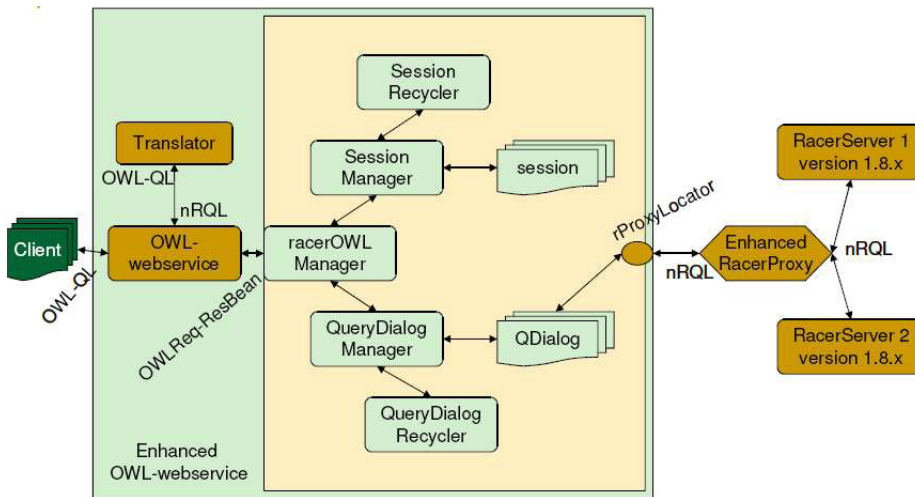


Abbildung 4.1.: Schematische Darstellung des Racermanager [Doshi]

weitere Frameworks verwandt, so zum Beispiel das Apache Axis Web Services Framework [Apache Axis] zum Verschicken der Querys und bündeln in einer SOAP⁴-Message.

4.1.2. Anforderungen an die Software

Das vorliegende Racersystem war in der Lage, mit Hilfe eines eigens für den Racetproxy entwickelten OWL-QL Clients einfache Anfragen auf eine OWL-Wissensbasis aus TBox und ABox zu beantworten. Zur Performancemessung war dieser Client allerdings nicht geeignet.

Ein gängiges Tool für Geschwindigkeitsmessungen auf Webservern ist das OpenSource-Werkzeug JMeter [Apache JMeter]. Dieses wurde zum Beispiel auch bereits von Tobias Berger in seiner Arbeit über das Load Balancing auf Racetservern genutzt [Berger].

Dieses Tool kann unter Verwendung von vorgefertigten „Testplans“ Querys an einen Server schicken und die Antwortzeiten ausgeben. Für ausführliche Tests und Dokumentation erwies es sich allerdings nach kurzer Zeit als zu unflexibel und fehlerbehaftet, was schließlich zu der Entwicklung eines eigenen Clients führte.

Die Idee eines neuen Clients bestand darin, den oder die Racetserver möglichst flexibel auslasten zu können. Erste Tests ergaben, dass einfache Anfragen auf einer LUBM⁵-Wissensbasis in Zusammenhang mit einem Intel Xeon Prozessor nur wenige Millisekunden benötigten. Daher war es nötig, nicht einzelne sondern sehr viele Queries zu verschicken, um alle Kerne voll auslasten zu können, was zu folgendem Anforderungsprofil führte.

⁴stand ursprünglich für „Simple Object Access Protocol“, da dieser Ausdruck allerdings nicht der eigentlichen Bedeutung entspricht, wird diese Bezeichnung offiziell seit 2003 nicht mehr als Akronym verwendet.

⁵Lehigh University Benchmark

- flexible Anzahl von Queries durch Simulation verschiedener Clients, die ihre Anfragen parallel oder nacheinander verschicken können. Dies dient vor allem dem Untersuchen der Verarbeitungsgeschwindigkeit der Anfragen insgesamt.
- flexible Anzahl parallel verschickbarer Anfragen als eine Art Belastungstest für den Proxy und insbesondere dessen Queue.
- Überprüfen der korrekten Antwort vom Server. Insbesondere nötig für den Belastungstest.
- Verwalten unterschiedlicher Queries in verschiedenen Dateien.
- Speicherung aller Messdaten in einer Ausgabedatei.
- Verwaltung von Konfigurationsdateien.

Die Entwicklung des Clients erfolgte ebenso wie die Programmierung des RacerManagers in Java. Umsetzung und Programmstruktur sollen in dem folgenden Abschnitt kurz vorgestellt werden.

4.2. Der Client

Zur Erläuterung der einzelnen Programmteile betrachten wir uns zunächst einmal das Klassendiagramm des Clientgenerators, der im Rahmen dieser Arbeit entstanden ist, in Abbildung 4.2.

Das Diagramm stellt nur eine Zusammenfassung der Klassen dar. Unterfunktionen, die für das eigentliche Verständnis nicht wichtig sind, wurden bewusst weggelassen.

Die Hauptfunktionalität besteht darin, eine bestimmte Anzahl Clients zu generieren, Queries an den Racerproxy zu versenden, anschließend die Antwort aufzufangen und die gebrauchte Zeit, sowie dessen Inhalt auszuwerten.

In der folgenden Übersicht wird die Funktionsweise der einzelnen Klassen näher erläutert.

MyClientFrame beinhaltet die Mainfunktion und ruft das Userinterface auf. Das Interface ist schlicht gehalten und bietet nur 2 Funktionen, die Eingabe der Anzahl der generierten Clients und die Anzahl der Queries, die insgesamt verschickt werden, sowie einen Button, um das Programm zu starten. Außerdem wird hier bereits eine Instanz der Klasse „Config“ erzeugt, um die Standardeinstellungen zu laden und Informationen an die Clients weiterzuleiten.

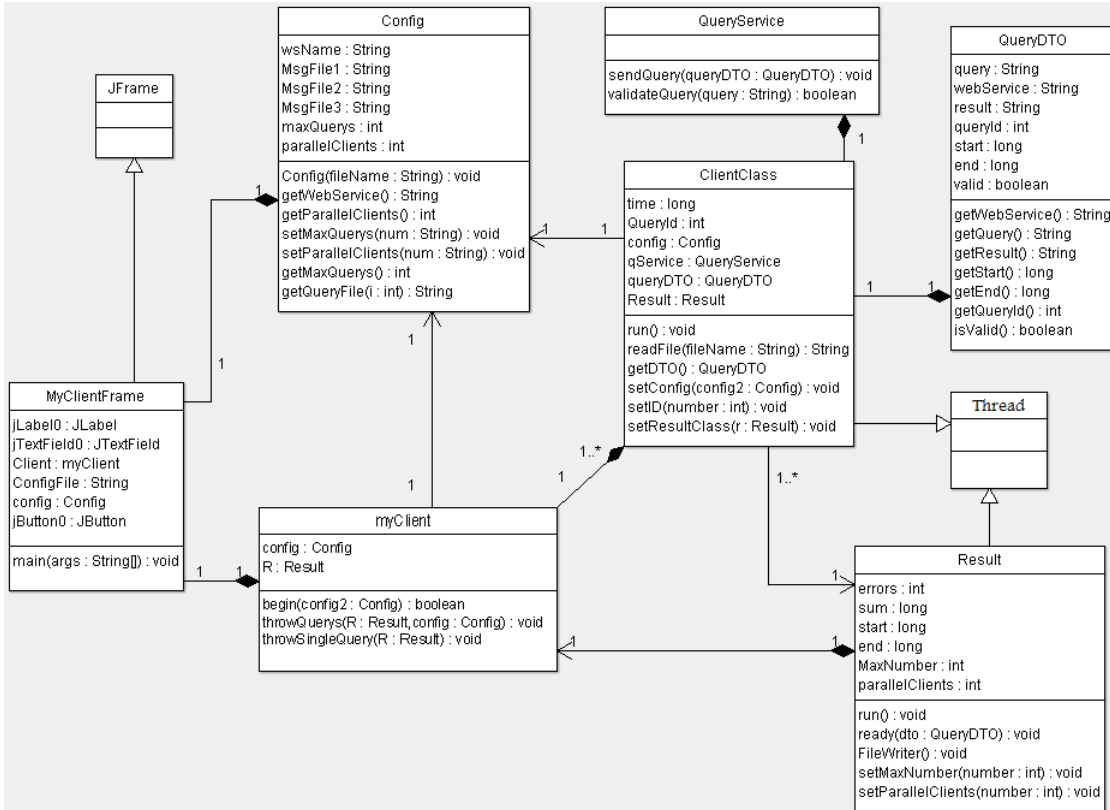


Abbildung 4.2.: vereinfachtes Klassendiagramm des Clienttools myClient

myClient dient zum Erzeugen der Threadklassen. Es gibt so viele QueryKlassen wie vom Benutzer vorgegeben und eine Ergebnisklasse. Diese kann später weitere Queries aufrufen, bis die maximale Anzahl erreicht ist.

ClientClass jede Instanz dieser Klasse symbolisiert einen Client und ist die erste Threadklasse. Nach dem Empfangen einer Antwort wird diese weitergeleitet an die Resultklasse.

QueryDTO das Datatransferobject verwaltet alle Informationen rund um ein Query. Zum Beispiel Inhalt, Antwort, sowie Antwortszeit.

QueryService hier passiert das eigentliche Verschicken, nachdem die nötigen Informationen dafür im Transferobject von der Clientklasse übergeben wurden.

Result die Ergebnisklasse zeichnet alle Ergebnisse aus den Queries auf und entscheidet selbstständig, ob alle Antworten eingetroffen sind und genug Queries versand wurden. Wartet die Resultklasse auf fehlende Antworten, so weist sie die „myClient“-Klasse an weitere Clients zu generieren. Sind alle Antworten eingetroffen, wird die Auswertung der Antworten in eine Textdatei geschrieben.

Config in diesem Objekt sind alle Informationen aus der Datei „config.cfg“ enthalten, die sich im Verzeichnis vom „myClient“ befinden muss. Die Daten werden automatisch bei der Generierung eingelesen.

Kapitel 5.

Experiment

Nach der theoretischen Analyse des vorliegenden Systems und der softwaretechnischen Implementierung des Versuchs in den Kapiteln drei und vier soll nun im fünften Kapitel der Versuch selbst betrachtet werden.

Nach Klärung der Rahmenbedingungen folgt die Beschreibung der einzelnen Tests, sowie eine kurze Analyse der Ergebnisse im späteren Teil.

5.1. Generelle Vorüberlegungen

Die Durchführung der Versuche erfolgte auf einem Server mit Intel Xeon X3320 Prozessor. Um zusätzlichen Overhead durch eine Netzwerkverbindung zu vermeiden, erfolgte die Implementierung sowohl des Proxyserver, als auch des Racerservers komplett auf diesem System. Nur der Client befand sich davon unabhängig auf einer zweiten Recheneinheit und schickte seine Anfrage über das Netzwerk der Uni an den Proxyserver.

Alle verwendeten Wissensbasen waren ABoxen aus dem LUBM¹-Projekt. Dieses Projekt stellt automatisch generierte ABoxen bereit, welche die Organisationsstruktur einer oder mehrerer Universitäten enthalten soll. Größe und Umfang der Universität sind dabei frei definierbar, so dass größere oder kleinere ABoxen generiert werden. Für nähere Informationen über das Projekt und den Generator selber soll auf [LUBM] verwiesen werden.

Die Bearbeitungszeit eines Queries kann von vielen Faktoren abhängen. Zum Einen vom System selbst, in der effektiven Implementierung der Software, als auch in der Gesamtanordnung, wie in diesem Fall zum Beispiel die Anzahl der verwendeten Reasoner. Zum Anderen aber auch vorallem vom Versuch selbst. Das heißt, insbesondere die Art der Anfrage und somit, zum Beispiel, wie viele Bedingungen das gesuchte Individuum einschränken. Ebenso die Wissensbasis, bei ihr ist vorallem die Größe entscheidend und wie viel Speicherplatz sie dabei einnimmt.

¹Lehigh University Benchmark

All diese Umstände haben Einfluss auf die Testergebnisse und sollen in den folgenden Versuchen berücksichtigt werden. Einen besonderen Einfluss hat natürlich auch die verwendete Hardware, welche aber im Rahmen der Arbeit keine weitere Rolle spielen wird.

5.1.1. Probleme im Vorfeld

Erste Experimente in der angegebenen Umgebung ergaben recht schnell, dass der Racermanager in seiner gegenwärtigen Verfassung nicht für Multithreading und insbesondere nicht für die innerhalb des Experimentes an ihn gestellten Aufgaben geeignet war. Sichtbar wurde dies durch eine hohe Fehlerwahrscheinlichkeit nach parallelem Verschicken von bereits wenigen Nachrichten. Woraufhin die Antworten an den Racerproxyserver nicht mehr die erwartete Syntax aufwiesen.

Nach einer näheren Untersuchung des Systems ergab sich ein Problem beim Balancing der Anfragen auf die unterschiedlichen Reasoner. Die Verteilung geschieht anhand einer Round-Robin Methode, bei der das Query an den nächsten Server weitergeleitet wird, der gerade keines bearbeitet. Sollte einmal gar keiner zu Verfügung stehen, wurde einfach an den ersten Server in der Schlange geroutet. Da der Racermanager aber nur eine Anfrage pro Reasoner zur Zeit bearbeiten kann, führte dies unumgänglich zu Synchronisationsproblemen und die Nachrichten waren nicht mehr korrekt.

Um dieses Problem zu umgehen, mussten Reservierungs- und Wartemechanismen geschaffen werden, die ein entsprechendes Query immer so lange warten lassen, bis auch wirklich ein Server zur Bearbeitung frei wird.

Nach weiteren Versuchen stellte sich die Implementierung des Cache auf dem Proxyserver ebenso als Hindernis heraus. Zum einen, weil eine Beantwortung aus dem Cache ähnliche Queries umgeht und eine Bearbeitung auf dem Racer verhindert und zum Anderen, da das Speichern und der Abgleich der einzelnen Anfragen in Stringform zumindest in der aktuellen Variante fehlerhaft ist und dazu führen kann, dass die Überprüfung fehlschlägt und eine „Nullpointerexception“ zurück an den Client geschickt wird.

Diese Funktion wurde deshalb für den Zeitraum des Experimentes außer Kraft gesetzt.

Reasoner Für die gesamte Testphase wurden mehrere Prozesse des Reasoners Racer-Pro in der aktuellen Version 1.9.3 beta verwandt.

Testläufe Aus Zeitgründen in Hinsicht auf die Arbeit wurde jeder Testlauf maximal drei mal durchgeführt, da alle Tests in sich geringe Abweichungen zeigten, wurde dieses vom Autor als hinreichend genug eingestuft.

5.2. Tests auf kleinen ABoxen

Der Aufbau der Tests soll in mehreren Stufen erfolgen. In der obersten Stufe werden verschiedene ABoxen eingesetzt, die sich vorallem in ihrer Größe unterscheiden. In der Stufe darunter wird anhand von unterschiedlichen Queries getestet, wie sich die Racer verhalten und in der untersten Stufen werden direkt verschiedene Racerkombinationen verglichen, ein bis vier Racer auf ein bis vier Kernen und in einem Spezialfall vier Racer auf einem Kern, da auch dieses Szenario sich unterschiedlich auswirken könnte.

Die Erwartungshaltung, welche sich aus dem Verdoppeln beziehungsweise dem Vervierfachen der verwendeten Prozessorkerne ergibt, ist die, dass sich die gebrauchte Rechenzeit antiproportional verhält. Und sich damit in diesem Fall halbieren oder vierteln würde. Diese Theorie soll in dem folgenden Experiment überprüft werden.

Erstes Testszenario: einfache Anfragen

Beim ersten Versuch soll zunächst untersucht werden, was passiert, wenn relativ schnell zu bearbeitende Queries auf eine kleine ABox an unterschiedlich viele Racer geschickt werden. Als Anfragen wurden dafür die Standardqueries des LUBM-Benchmark gewählt, die auf der Homepage des Projekts bereitgestellt werden. Hierbei handelt es sich um Anfragen vom Typ: "Liste mir alle Studenten auf". [LUBM] Im folgenden Diagramm werden die Antwortzeiten gegenüber gestellt.

Anhand des Diagramms ist leicht zu sehen, dass sich zwar bei einem Einsatz von mehr als einem Racer auf weiteren Prozessorkernen ein Performancezuwachs erzielen lässt, dieser aber weitaus geringer ausfällt, als man bei einer Verdopplung der Kerne zunächst denken würde. Bei Zuschalten von weiteren Racern ist der Unterschied vernachlässigbar gering. Zudem wirkt sich der Unterschied erst bei etwa eintausend Queries aus, bei einhundert beträgt er dagegen nur wenige Millisekunden. Dass sogar mehrere Racer auf einem Core schneller sind als ein Einzelner, ist ein Hinweis darauf, dass die Cores in diesem Szenario keinen Einfluss haben und vom System nicht ausgelastet werden können.

Zweites Testszenario: aufwändige Anfragen

Für das Szenario mit aufwändigen Anfragen wurde ein umfangreicheres Query entwickelt, dass alle Eigenschaften der ABox abfragt. Dabei werden sowohl Studenten, Kurse, einfache Dozenten, sowie Professoren und deren Publikationen berücksichtigt, was zu einem höheren Rechenaufwand auf den Racern führt. Alle verwendeten Queries sind im Anhang aufgeführt.

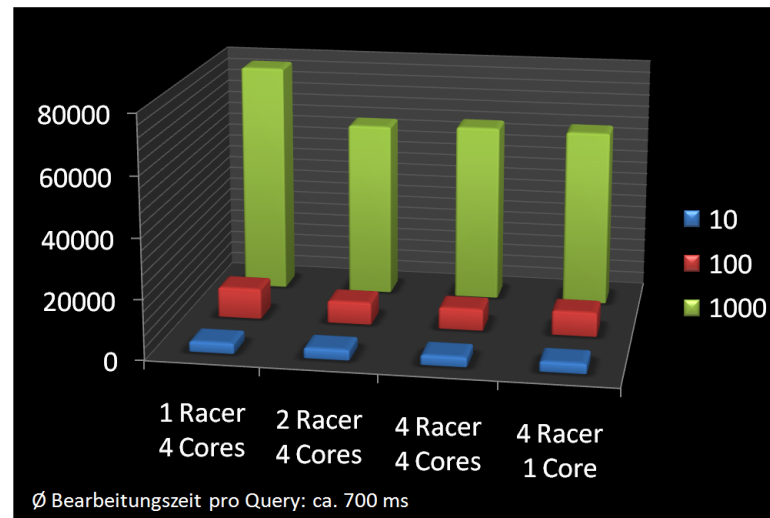


Abbildung 5.1.: Verarbeitungszeit bei einfachen Anfragen und kleiner ABox in Millisekunden

In diesem Szenario profitiert die Verarbeitungszeit zum ersten Mal spürbar von mehreren Cores. Der Performanceanstieg bei mehr als zwei Cores ist aber deutlich geringer, als von einem auf zwei Cores. Das bedeutet, dass auch in diesem Fall noch nicht alle Cores und Racer ausreichend schnell mit Daten und Anfragen versorgt werden können, um die volle Leistung auszunutzen, obwohl die Bearbeitungszeit auf den Servern bereits deutlich zugenommen hat.

5.3. Test auf großen ABoxen

Für das dritte und vierte Testszenario wurde eigens eine ABox generiert, welche die mehr als zehnfache Menge an Daten enthielt. Damit sollte die Bearbeitungszeit weiter erhöht und der Racerproxyserver etwas entlastet werden.

Drittes Testszenario: aufwändige Anfragen auf großen ABoxen

In diesem Test soll der prozentuale Anteil der Bearbeitung auf den Reasonern erhöht werden, wodurch die insgesamt gebrauchte Rechenzeit weiter zu nimmt. Die Ergebnisse

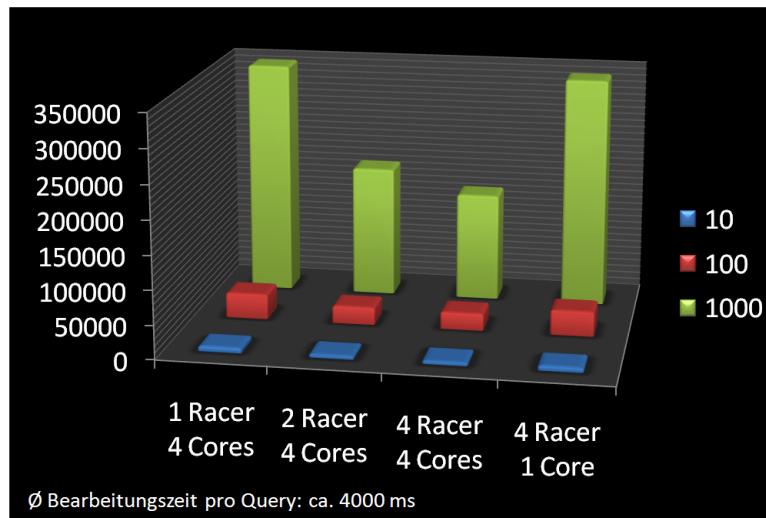


Abbildung 5.2.: Verarbeitungszeit bei aufwändigen Anfragen und kleiner ABox in Millisekunden

des Versuches sind auf Abbildung 5.3 in einem Diagramm dargestellt.

Hier können fast alle Kerne komplett ausgelastet werden. Bei vier Kernen beträgt die Verarbeitungszeit gegenüber einem Kern nur noch ein Drittel, bei zwei Kernen etwa zwei Drittel. Bei vier Racern auf einem Kern lassen sich keine merklichen Geschwindigkeitszuwächse feststellen.

5.4. Zusammenfassung der Experimente

Die drei vorhergehenden Versuche haben einen direkten Zusammenhang zwischen der Rechenzeit eines Querys auf den Reasonern und der Auslastung auf den Kernen deutlich gemacht. Im ersten Versuch wurden die Kerne so gering belastet, dass 4 Racer auf einem Kern sogar noch besser waren als ein Einzelner.

Später konnte mit zunehmender Rechendauer ein Großteil der Last auf die Reasoner verlagert werden, wodurch der Proxyserver mehr Zeit zum Verteilen der Querys bekam und seine Rolle als Flaschenhals ablegen konnte.

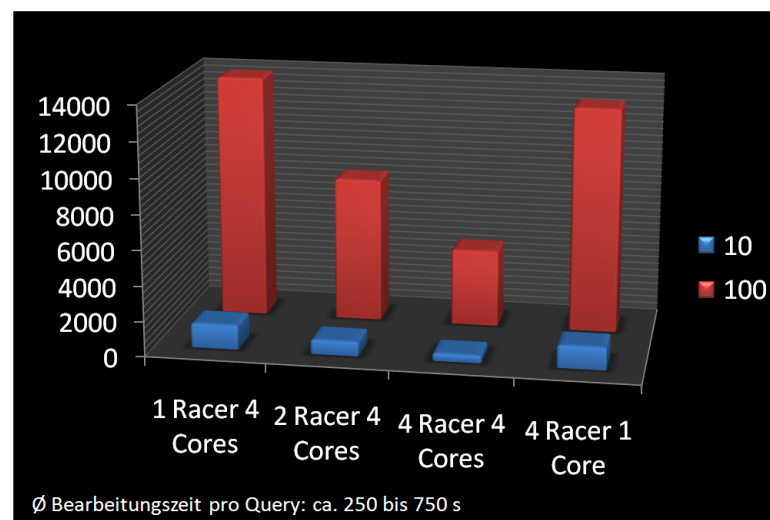


Abbildung 5.3.: Verarbeitungszeit bei aufwändigen Anfragen und großer ABox in Sekunden

Kapitel 6.

Zusammenfassung und Ausblick

Zu Beginn der Arbeit wurde die Frage aufgestellt: “Kann man mit Hilfe des Racermanagers die Singlecoreapplikation des RacerPro in eine Multicoreapplikation umwandeln?”. Diese Frage kann nach Betrachtung der Experimente nur unter sehr starken Einschränkungen mit „Ja“ beantwortet werden. Den wirklichen Nutzen des Mehrprozessorsystems kann man nur bei aufwändigen und länger dauernden Queries ziehen. Bei vielen schnellen Queries gibt es kaum einen nennbaren Effekt.

Der Grund dafür ist im Gesamtsystem zu suchen. Der Racer Manager wurde nicht entwickelt um schnellstmöglich vier CPU-Kerne mit Anfragen zu versorgen. Er wurde vor einigen Jahren in einem Projekt entwickelt, um, zum Beispiel, mehrere Anfragen in einer Warteschlange abwickeln zu können.

Die letzte große Aktualisierung fand im Jahre 2005 mit der Implementierung des Webservices und OWL-QL für die Racerversion 1.8 statt. Seitdem wurde RacerPro weiterentwickelt, RacerManager allerdings nicht. Die aktuelle Version ist 1.9.3 beta. Das Reasonersystem hat eine eigene Queue und einen eigenen Cache.

Aufgrund der ineffizienten Programmierweise wird der Racermanager zum Flaschenhals und ist nicht belastungsresistent gegen viele und schnelle Anfragen. Die Queue lässt nur wenig Platz. Bei zwanzig parallel versendeten Queries treten erste Fehler auf, bei einhundert verschickten Anfragen liegt die Fehlerrate bereits bei etwa 60 %. Zudem verursacht der Cache mit einem ineffizienten Stringvergleich weitere Fehler, die zu einer scheinbar zufällig auftretenden Exception bei zu vielen Anfragen führen.

Ferner sollte untersucht werden, ob der geteilte Cache auf dem Prozessor ein Ausbremsen der Kerne bei zu großen Datenmengen verursachen kann. Dies konnte im Rahmen der Arbeit nicht gezeigt werden. Allerdings ist es nicht auszuschließen, dass dieses Szenario bei weitaus größeren Wissensbasen auftreten kann.

Anhang A.

A.1. LUBM Queries in OWL QL

Einfache Anfragen

Die folgenden Queries unter den Abbildungen A.1, A.2, A.3, A.4, A.5 wurden von der LUBM Projektseite zur Verfügung gestellt und sind schnelle Anfragen zur Benchmarkerstellung. Alle wurden im Experiment als „einfache Anfragen“ verwendet.

Aufwändigere Anfragen

Das Query unter Abbildung A.6 wurde speziell in Hinblick auf die LUBM Wissensbasis entwickelt um die Rechenzeit auf einem Racerserver zu erhöhen und möglichst alle enthaltenen Individuen der ABox zu berücksichtigen.

```

1  <?xml version="1.0"?>
2  <query
3      xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
4      xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
7  <queryPattern>
8      <rdf:RDF>
9          <rdf:Description
10              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
11              >
12              <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#GraduateStudent"/>
13          </rdf:Description>
14          <rdf:Description
15              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
16              >
17              <uni:takesCourse rdf:resource="http://www.Department0.University0.edu/GraduateCourse0"/>
18          </rdf:Description>
19      </rdf:RDF>
20  </queryPattern>
21  <mustBindVars>
22      <var:x />
23  </mustBindVars>
24  <answerKBPattern>
25      <kbRef rdf:resource="http://www.tu-harburg.de/~secn1025/university0-0.owl"/>
26  </answerKBPattern>
27  <answerSizeBound>
28      400
29  </answerSizeBound>
30 </query>

```

Abbildung A.1.: LUBM Query 01

```
1
2 <?xml version="1.0"?>
3 <query
4   xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
5   xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
6   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7   xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
8 <queryPattern>
9   <rdf:RDF>
10     <rdf:Description
11       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
12       >
13       <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
14         /2004/0401/univ-bench.owl#GraduateStudent"/>
15     </rdf:Description>
16     <rdf:Description
17       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y"
18       >
19       <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
20         /2004/0401/univ-bench.owl#University"/>
21     </rdf:Description>
22     <rdf:Description
23       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#z"
24       >
25       <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
26         /2004/0401/univ-bench.owl#Department"/>
27     </rdf:Description>
28     <rdf:Description
29       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
30       >
31       <uni:memberOf rdf:resource="http://www.w3.org/2003/10/
32         owl-ql-variables#z"/>
33     </rdf:Description>
34     <rdf:Description
35       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#z"
36       >
37       <uni:subOrganizationOf rdf:resource="http://www.w3.org
38         /2003/10/owl-ql-variables#y"/>
39     </rdf:Description>
40     <rdf:Description
41       rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
42       >
43       <uni:undergraduateDegreeFrom rdf:resource="http://www.
44         w3.org/2003/10/owl-ql-variables#y"/>
45     </rdf:Description>
```

Abbildung A.2.: LUBM Query 02

```

1  <?xml version="1.0"?>
2  <query
3      xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
4      xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
7  <queryPattern>
8      <rdf:RDF>
9          <rdf:Description
10              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
11              >
12              <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
13                  /2004/0401/univ-bench.owl#Person"/>
14          </rdf:Description>
15          <rdf:Description
16              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
17              >
18              <uni:memberOf rdf:resource="http://www.Department0.
19                  University0.edu"/>
20          </rdf:Description>
21      </rdf:RDF>
22  </queryPattern>
23  <mustBindVars>
24      <var:x />
25  </mustBindVars>
26  <answerKBPattern>
27      <kbRef rdf:resource="http://www.tu-harburg.de/~secn1025/university0-0.
28          owl"/>
29  </answerKBPattern>
30  <answerSizeBound>
31      400
32  </answerSizeBound>
33  </query>

```

Abbildung A.3.: LUBM Query 05

```

1  <?xml version="1.0"?>
2  <query
3      xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
4      xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
7  <queryPattern>
8      <rdf:RDF>
9          <rdf:Description
10              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
11              >
12              <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
13                  /2004/0401/univ-bench.owl#Student"/>
14          </rdf:Description>
15      </rdf:RDF>
16  </queryPattern>
17  <mustBindVars>
18      <var:x />
19  </mustBindVars>
20  <answerKBPattern>
21      <kbRef rdf:resource="http://www.tu-harburg.de/~secn1025/university0-0.
22          owl"/>
23  </answerKBPattern>
24  <answerSizeBound>
25      400
26  </answerSizeBound>
27  </query>

```

Abbildung A.4.: LUBM Query 6

```

1  <?xml version="1.0"?>
2  <query
3      xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
4      xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
7  <queryPattern>
8      <rdf:RDF>
9          <rdf:Description
10              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
11              >
12              <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2
13                  /2004/0401/univ-bench.owl#Student"/>
14          </rdf:Description>
15          <rdf:Description
16              rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x"
17              >
18              <uni:takesCourse rdf:resource="http://www.Department0.
19                  University0.edu/GraduateCourse0"/>
20          </rdf:Description>
21      </rdf:RDF>
22  </queryPattern>
23  <mustBindVars>
24      <var:x />
25  </mustBindVars>
26  <answerKBPattern>
27      <kbRef rdf:resource="http://www.tu-harburg.de/~secn1025/university0-0.
28          owl"/>
29  </answerKBPattern>
30  <answerSizeBound>
31      400
32  </answerSizeBound>
33  </query>

```

Abbildung A.5.: LUBM Query 10

```

1  <?xml version="1.0"?>
2  <query
3
4      xmlns="http://www.w3.org/2003/10/owl-ql-syntax#"
5      xmlns:var="http://www.w3.org/2003/10/owl-ql-variables#"
6      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7      xmlns:uni="http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl#">
8  <queryPattern>
9      <rdf:RDF>
10         <rdf:Description
11             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#u">
12             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
13                 univ-bench.owl#Publication"/>
14         </rdf:Description>
15         <rdf:Description
16             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#v">
17             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
18                 univ-bench.owl#UndergraduateStudent"/>
19         </rdf:Description>
20         <rdf:Description
21             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#w">
22             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
23                 univ-bench.owl#Course"/>
24         </rdf:Description>
25         <rdf:Description
26             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#w2">
27             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
28                 univ-bench.owl#Course"/>
29         </rdf:Description>
30         <rdf:Description
31             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#x">
32             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
33                 univ-bench.owl#GraduateCourse"/>
34         </rdf:Description>
35         <rdf:Description
36             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y">
37             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
38                 univ-bench.owl#FullProfessor"/>
39         </rdf:Description>
40         <rdf:Description
41             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y2">
42             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
43                 univ-bench.owl#AssociateProfessor"/>
44         </rdf:Description>
45         <rdf:Description
46             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#z">
47             <rdf:type rdf:resource="http://www.lehigh.edu/%7Ezhp2/2004/0401/
48                 univ-bench.owl#GraduateStudent"/>
49         </rdf:Description>
50         <rdf:Description
51             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#z">
52             <uni:takesCourse rdf:resource="http://www.Department0.University0.
53                 edu/GraduateCourseABCD"/>
54         </rdf:Description>
55         <rdf:Description
56             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#z">
57             <uni:takesCourse rdf:resource="http://www.w3.org/2003/10/owl-ql-
58                 variables#x"/>
59         </rdf:Description>
60         <rdf:Description
61             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y">
62             <uni:teacherOf rdf:resource="http://www.w3.org/2003/10/owl-ql-
63                 variables#x"/>
64         </rdf:Description>
65         <rdf:Description
66             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y">
67             <uni:teacherOf rdf:resource="http://www.w3.org/2003/10/owl-ql-
68                 variables#w"/>
69         </rdf:Description>
70         <rdf:Description
71             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#v">
72             <uni:takesCourse rdf:resource="http://www.w3.org/2003/10/owl-ql-
73                 variables#w"/>
74         </rdf:Description>
75         <rdf:Description
76             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#v">
77             <uni:takesCourse rdf:resource="http://www.w3.org/2003/10/owl-ql-
78                 variables#w2"/>
79         </rdf:Description>
80         <rdf:Description
81             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#y2">
82             <uni:teacherOf rdf:resource="http://www.w3.org/2003/10/owl-ql-
83                 variables#w2"/>
84         </rdf:Description>
85         <rdf:Description
86             rdf:about="http://www.w3.org/2003/10/owl-ql-variables#u">
87             <uni:publicationAuthor rdf:resource="http://www.w3.org/2003/10/owl-
88                 ql-variables#y"/>
89         </rdf:Description>
90     </rdf:RDF>
91 </queryPattern>
92 <mustBindVars>
93     <var:u/>
94 </mustBindVars>
95 <answerKBPattern>
96     <kbRef rdf:resource="http://www.tu-harburg.de/~secn1025/university0-0.owl"/>
97 </answerKBPattern>
98 <answerSizeBound>
99     135
100 </answerSizeBound>
101 </query>

```

Abbildung A.6.: Query mit besonders hohem Rechenaufwand

Literaturverzeichnis

[DLHB] Franz Baader, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider. The description logic Handbook. 2003, Kapitel 1.3

[Gruber] Gruber, Thomas, R., A translation approach to portable ontology specifications. 1993, Academic Press, S. 199

[Sure] Sure, Y., Ontologien, Institut AIFB, Karlsruhe

[Hitzler] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure. Semantic Web Grundlagen. 2003, Kapitel 3.1

[Birkenbiehl] Klaus Birkenbiehl. Standards für das Semantic Web. 2006, S. 73-88

[Racer-User-Guide] elektronisches Dokument RacerPro-User- Guide-1-9-2-beta.pdf. Racer Systems GmbH & Co. KG, <http://www.racer-systems.com>. RacerPro User's Guide Version 1.9.2. 2007, Kapitel 1.1 , Seite 2

[Maharjan] Jog Ratna Maharjan. Reengineering of the RACER Proxy - Iterative Query Answering. Master Thesis, 2005

[Doshi] Tejas Doshi. Enhancement of the RacerProxy webservice Interface for Iterative Query Answering. Master Thesis, 2005

[Galinski] Jan Galinski. Einsatz von Web Services im SemanticWeb am Beispiel der RACER Engine und OWL-QL. Diplomarbeit, 2005

[Apache Tomcat] Apache Tomcat Application Server. Webseite: <http://jakarta.apache.org/tomcat/>.

[Apache Axis] Apache Axis Web Services Framework. Webseite: <http://ws.apache.org/axis/>.

[Apache JMeter] Apache JMeter. Webseite: <http://jakarta.apache.org/jmeter/usermanual/>.

- [Berger] Tobias Berger. Implementation, Test and Evaluation of Load Balancing Strategies for Multi-User Inference Systems. Master Thesis, 2007
- [LUBM] Lehigh University Benchmark. Webseite: <http://swat.cse.lehigh.edu/projects/lubm/>

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Hamburg, 5. Juni 2009

Unterschrift