Technische Universität Hamburg-Harburg
Arbeitsbereich Softwaresysteme

Studienarbeit

Entwicklung und Evaluierung einer Test-Infrastruktur für den Racer-Server

Keno Selzer
Matrikel-Nr.: 13075
Studiengang Informatik-Ingenieurwesen
Technische Universität Hamburg-Harburg

Betreuung:

Prof. Dr. rer. nat. Ralf Möller
M. Sc. Atila Kaya
Arbeitsbereich Softwaresysteme
Technische Universität Hamburg-Harburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Textverzeichnis

Danksagung

Ich bedanke mich bei Herrn Prof. Dr. rer. nat. Ralf Möller für das interessante Thema und die Betreuung dieser Arbeit.

Mein weiterer Dank gilt Herrn M. Sc. Atila Kaya für die Geduld während der ganzen Arbeit und seine hervorragende Unterstützung.

Kapitel 1

Einleitung

Das Internet ist in den letzten Jahren erheblich gewachsen und erfreut sich zunehmender Beliebtheit.
Aufgrund der großen Anzahl von Internetseiten und unzureichender Strukturierung des Internets in Bezug auf Informationen sind Suchmaschinen unabdingbar geworden. Trotz deren Vielzahl haben sie alle das gravierende Problem, dass sie keine Hintergrundinformationen über den Inhalt der Internetseiten besitzen und somit eventuell irrelevante Seiten als Suchergebnis ausgeben.
Wenn z.B. ein Geschäftsmann eine Person sucht, die er letztes Jahr auf einer Konferenz getroffen hat und sich nur an den Namen „Ms. Cook“, dass sie für einen seiner Kunden arbeitet und dass ihre Tochter auf seine ehemalige Schule geht, erinnern kann [dl-semweb], dann würde eine Suche nach „Cook“ wahrscheinlich eher auf eine Internetseite eines Kochs (engl: cook) verweisen, als das gewünschte Ergebnis zu erreichen.
Auch die bekannten Zusatzinformationen dürften nicht konstruktiv zum Suchergebnis beitragen, sondern womöglich gar keinen Treffer mehr ergeben.
Hier wäre ein semantisches Netz nützlich.
Ein semantisches Netz besitzt Hintergrundinformationen über die Daten.
Damit könnte eine Suchmaschine Treffer in Bezug auf Köche, Cook Island etc. ignorieren, die Seiten der Unternehmen seiner Kunden finden, private Internetseiten durchforsten, überprüfen, ob die Tochter noch zur Schule geht und einen Abgleich der Internetseite der ehemaligen Schule treffen [dl-semweb].
Somit könnte ohne Probleme die betreffende Person ausfindig gemacht werden, unter der Voraussetzung, dass genügend Informationen im Internet vorhanden sind.
Eine solche Umsetzung eines semantischen Netzes wird vom Racer-Server [Racer04] unterstützt, der die Beschreibungslogik $\mathcal{ALCQHI}_{\mathcal{R}^+}$ implementiert, bekannt auch unter $\mathcal{SHIQ}$ [Horrocks et al. 2000].

1.1 Ziel der Arbeit

Ziel der vorliegenden Arbeit ist es, eine einfache Entwicklungsumgebung für Benchmarks, die zum Messen verschiedener Funktionen/Aspekte des Racer-Servers dienen, zu schaffen. Bisher gibt es keine Möglichkeit, Benchmarks ohne Hintergrundwissen einer Programmiersprache im Racer-Kontext zu erstellen.
Dabei sollen in dieser Entwicklungsumgebung sowohl bereits vorhandene, in einer Programmiersprache enstandene Benchmarks verwaltet und gestartet werden, sowie diejenigen, die mittels der Umgebung entstanden sind.
Außerdem sollen verschiedene Racer-Server verwaltet werden und auf ihnen die Benchmarks angewendet werden können, um eine Vergleichbarkeit dieser herzustellen.

1.2 Gliederung der Arbeit

In Kapitel 2 erfolgt eine kurze Einführung in die Beschreibungslogik, außerdem eine Erklärung der Racer-Server und der Benchmarks.
In Kapitel 3 wird dann das in der Studienarbeit entstandene Programm „Benchee“ in seiner Funktionsweise dargestellt.
Kapitel 4 bietet dazu eine nähere Erläuterung des Dateiformates, das durch die Entstehung eines Benchmarks in Benchee definiert wird und einen genaueren Blick auf die Architektur des Programmes.
Abschließend enthält das fünfte Kapitel eine Zusammenfassung dieser Arbeit und einen Ausblick auf die zukünftige Erweiterung von Benchee.

Kapitel 2

Einführung

2.1 Kurze Einführung in die Beschreibungslogik

Die Beschreibungslogik (engl. d*escription logic*) ist entwickelt worden, um eine Wissensbasis zu repräsentieren und verwendet hierfür eine formale Semantik [dl-semweb].
In der tatsächlichen Welt gibt es Individuen z.B. im englischen Königshaus Elizabeth und Charles und Beziehungen zwischen diesen Individuen z.B. *has_child*, also hat Elizabeth ein Kind namens Charles.
Das Wissen wird hierbei von zwei disjunkten Alphabeten repräsentiert, der Rolle (engl. *role*) und dem Konzept (engl. *concept*).
Die Rolle präsentiert die Beziehungen zwischen den Individuen, also z.B. *has_child(x,y)*, während das Konzept eine Beschreibung des Individuums angibt, also z.B. $parent(x) \equiv person(x) \wedge \exists y : (has\_child(x, y) \wedge person(y))$.
Umgangssprachlich würde das Beispiel ein Elternteil (engl. *parent*) beschreiben, das eine Person *x* ist und ein Kind *y* besitzt, welches auch eine Person sein muss.
Die Standard-Beschreibungslogik heißt $\mathcal{ALC}$. Hier würde das Konzept *parent* als $parent(x) \equiv person \cap \exists has\_child.parent$ geschrieben werden.
Somit können nun alle für das Beispiel benötigten Konzepte beschrieben werden, um eine Familienstruktur zu definieren.
Eine Ansammlung von Konzepten wird eine TBox (**T**erminological **Box**) genannt.
Mit Konzepten kann man also in der Beschreibungslogik die Hintergrundinformationen darstellen.
In Abbildung 2.1 wird dargestellt, welche Mutterkonzepte es gibt. So wird definiert, dass eine weibliche Person, also eine Frau, zur Mutter wird, wenn sie gleichzeitig Elternteil ist usw.
Weiterhin muss nun in der Beschreibungslogik dargestellt werden können, dass ein Individuum einem Konzept zugehört und eine bestimmte Rolle erfüllt, z.B. dass Elizabeth eine Mutter ist. Dieses geschieht mit Hilfe von erklärenden Axiomen (engl. *assertional axioms*), die das Konzept (engl. *concept assertion*) und die Rollen (engl. *role assertion*) definieren.

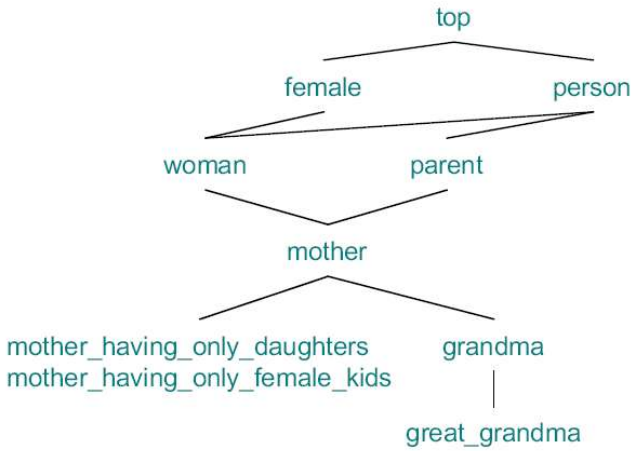

Abbildung 2.1: Beispiel einer TBox (aus [dl-semweb])

In der *concept assertion* wird einem Individuum ein Konzept zugewiesen, so z.B. Elizabeth dem Mutterkonzept (in $\mathcal{ALC}$: elizabeth:mother).

In der *role assertion* geschieht dieses in Bezug von Individuen auf deren Rolle, so z.B. Elizabeth und Charles der Rolle has_child (in $\mathcal{ALC}$: (elizabeth,charles):has_child).

Eine Ansammlung von *assertion axioms* wird ABox (**A**ssertional **Box**) genannt.

Abbildung 2.2 zeigt ein Beispiel einer ABox, die die Beziehungen im englischen Königshaus darstellt.

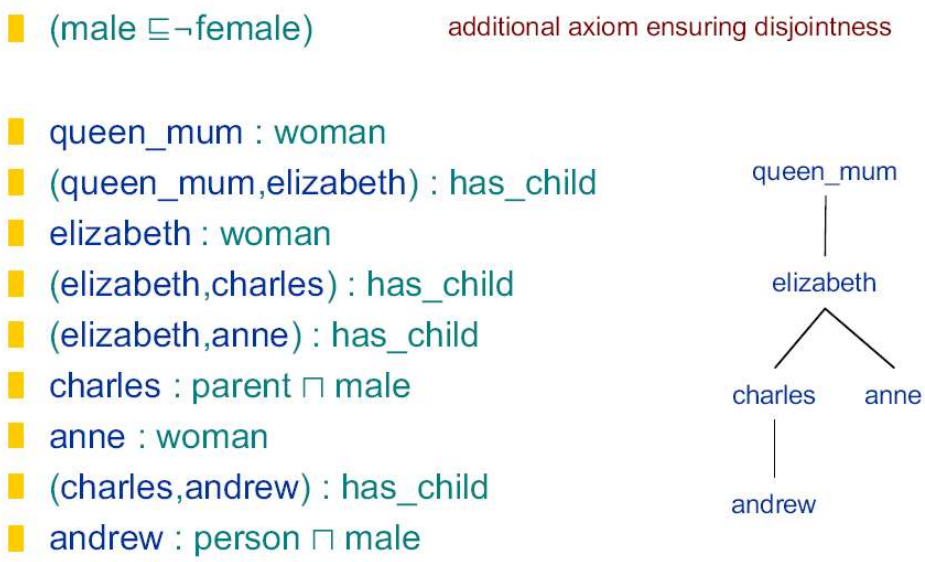

Abbildung 2.2: Beispiel einer ABox (aus [dl-semweb])

2.2 Einführung in Racer

Racer steht für „**R**enamed **A**Box and **C**oncept **E**xpression **R**easoner“ und ist ein wissenpräsentierendes System, dass die Beschreibungslogik $\mathcal{ALCQHI}_{R^+}$ implementiert, welches auch unter $\mathcal{SHIQ}$ [Horrocks et al. 2000] bekannt ist.

$\mathcal{SHIQ}$ ist eine Erweiterung von $\mathcal{ALC}$ in Bezug auf die Anzahl der Beschränkungen, der Rollen Hierarchien, der inversen Rollen und der transitiven Rollen [racer-manual].

Racer ist somit ein System, das das in der Einleitung beschriebene Problem unter der Voraussetzung, dass die benötigten Hintergrundinformationen vorhanden sind, lösen kann.

Es benutzt dabei T- sowie A-Boxen.

Racer ist also eine Umsetzung einer Folgerungsmaschine auf dem semantischen Netz, welches Ontologien entwickeln kann und Suchanfragen über RDF Dokumente stellen kann, wobei RDF (engl. für **R**esource **D**escription **F**ramework) ein universelles Format für Daten im Netz darstellt [RDF].

Außerdem wird neben RDF noch DAML-OIL [DAML-OIL] und OWL (Web Ontology Language) [OWL] untersützt.

Racer implementiert den HTTP-basierenden quasi-standard DIG (Description Logic Implementation Group) für die Kommunikation des Beschreibungslogik-Systems und den Anwendungen, die ein XML-basiertes Protokoll benutzen[Bechhofer 02] .

Außerdem implementiert Racer die meisten Funktionen, die im älteren KRSS (Knowledge Representation System Specification [Patel-Schneider and Swartout 93]) spezifiziert wurden [racer-manual].

Racer lässt sich neben dem semantischen Netz z.B. noch für Aufgaben aus dem Bereich der medizinischen Informatik, Process-Engineering, Software-Engineering usw. anwenden.

Weitergehende Informationen über das Racer-System können im Racer Handbuch [racer-manual] gefunden werden.

2.3 Einführung in Benchmarks

Benchmarks werden verwendet, um eine Messung der Leistungsfähigkeit von Systemen, bestehend aus Hardware und/oder Software, vorzunehmen.
Hierbei können verschiedene Werte gemessen werden. Bei Prozessoren-Benchmarks z.B. werden die FLOPS (Floation Point Operations per Second) gemessen oder im Fall der Beschreibungslogik die Zeit, die das System braucht, um eine oder mehrere Suchanfragen zu bearbeiten.
Meistens nimmt die Komplexität innerhalb eines Benchmarks stetig zu, so dass der Server immer mehr Zeit benötigt, die Anfragen zu beantworten.
Um eine Vergleichbarkeit zwischen unterschiedlichen Versionen oder Ausführungen eines Systems zu garantieren, sollten gleiche Systembedingungen geschaffen werden.
Dies würde bei der Beschreibungslogik einem Computer mit gleicher Leistungsfähigkeit entsprechen, auf dem z.B. der Racer-Server läuft.
Ein Benchmark in der Beschreibungslogik definiert üblicherweise als erstes einen Datenbestand (ABox und Tbox), der dem Server übermittelt wird, um anschließend eine oder mehrere Suchanfragen auszuführen.
Häufig werden die Suchanfragen und/oder der Datenbestand im Laufe des Benchmarkes komplexer und werden solange ausgeführt, bis eine Zeitschranke für die Beantwortung der Suchanfrage überschritten wird.
Anstatt der Suchanfrage kann es auch interessant sein, die Leistung einiger Befehle zu messen, wie z.B. die Konsistenzüberprüfung der ABox und der Leistungsvergleich der Suchanfragen mit oder ohne Optimierungen. Deswegen ist es nicht immer nötig, die TBoxen bzw. ABoxen jedes Mal zu löschen und neu zu laden, sondern die Suchanfragen auf den gleichen Datenbestand auszuführen.
Diese Benchmarks der Beschreibungslogik werden hauptsächlich von drei Benutzergruppen verwendet: Anwender, die das System bewerten und die Konfiguration optimieren wollen, Wissenschaftler, die das Verhalten erforschen und mit anderen vergleichen und Entwickler, die das System zur Ermittlung von Weiterentwicklungspunkten analysieren.

Kapitel 3

Vorstellung des Programmes

Im Rahmen dieser Arbeit ist ein Programm namens „Benchee“ entstanden, welches die Verwaltung von vorhandenen Benchmarks für den Racer-Server bietet und eine einfache Erstellung von neuen Benchmarks erlaubt.
Außerdem können unterschiedliche Racer-Server in unterschiedlichen Versionen verwaltet werden die aus dem Programm, genauso wie die Benchmarks, heraus gestartet und beendet werden können.
Dabei werden die Ergebnisse der über Benchee erstellten Benchmarks in einer Datei gespeichert, die mittels Gnuplot [gnuplot] grafisch dargestellt werden kann.
Vorhandene Benchmarks können in der Programmiersprache Java geschriebene Programme (verpackt in *java archive repositories* (jar-Dateien)) oder jegliche per Kommandozeile ausführbare Dateien sein.
Das Programm selbst wurde, um eine möglichst vom Betriebssystem unabhängige Implementierung zu erreichen, in Java [Java] geschrieben.
Benchee ist ein selbstständig laufendes Programm, das die Rückmeldungen und Fehlermeldungen in die Konsole ausgibt. Nach dem Starten des Programmes können Benchmarks ohne Hilfe von weiteren Anwendungen erzeugt, verwaltet und durchgeführt werden.

3.1 Erstellen eines Benchmarks

Ein Benchmark besteht in der Regel aus einem Datenbestand (ABoxen und/oder TBoxen), einer Suchanfrage und Befehlen, die den Racer-Server vorbereiten oder optimieren.
So ist es z.B. sinnvoll, die ABoxen bzw. TBoxen vor dem Laden von neuen Daten erst einmal zu löschen, damit die Zuverlässigkeit des Benchmark-Ergebnisses gesichert werden kann.

Benchee unterstützt die Erstellung von neuen Benchmarks durch eine einfach gehaltene Entwicklungsumgebung, die in Abbildung 3.1 dargestellt ist.

Oben links befinden sich drei Felder und deren Knöpfe, die zur Einbindung der Bestandteile des Benchmarks dienen.

Darunter befindet sich das Benchmark-Fenster, welches das zu erzeugende Benchmark enthält bzw. über das es bearbeitet werden kann.

Eine genaue Beschreibung der Semantik eines Benchmarks und das Aussehen der Datenbestands- bzw. Suchanfragendatei werden in Kapitel 4.1 näher erläutert.

Im Command-Feld darüber wird ein Befehl eingegeben, der über den „Add Command"-Knopf in den zu erstellenden Benchmark an der markierten Stelle eingefügt wird.

Ähnlich wird mit den Feldern, die den Datenbestand bzw. die Suchanfragen (engl. *query*) enthalten, verfahren. Es können entweder direkt die Pfade zu den betreffenden Dateien angegeben werden oder über den „Open..."-Knopf, der ein Datei-Dialog öffnet, ausgesucht werden.

Auch diese werden wieder über den entsprechenden „Add"-Knopf eingefügt.

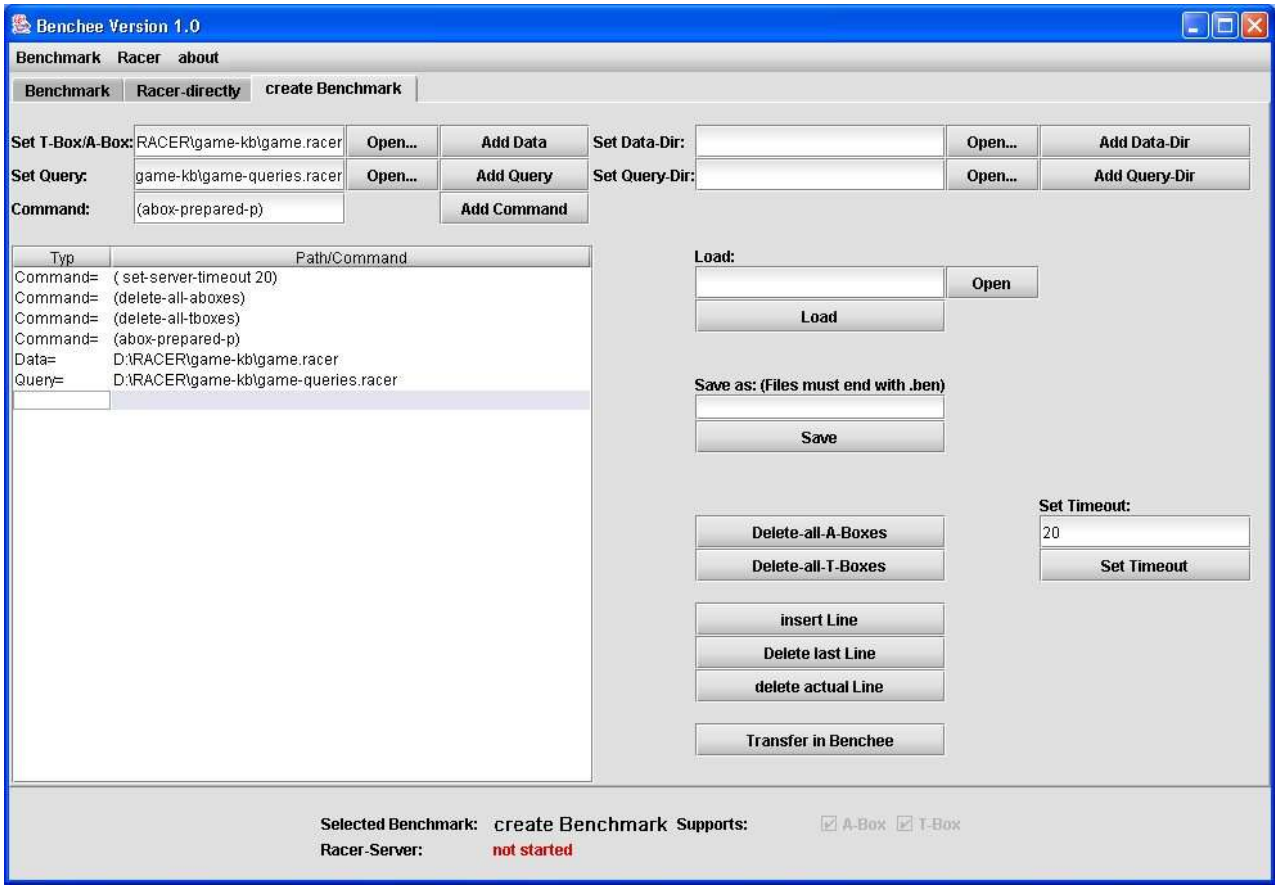

Abbildung 3.1: Programmbildschirm für das Erstellen eines Benchmarks

Dabei enthalten die Dateien, auf die die angegebenen Pfade verweisen, entweder Suchanfragen für den *Query*-Typ oder TBoxen und/oder ABoxen für den *Data*-Typ.

Neben dem Benchmark-Fenster sind außer den Lade- und Speicheroptionen einige häufig verwendete Befehle, wie das Löschen der verschiedenen Datenbestände, das Setzen der Zeitüberschreitung (engl. *Timeout*) und auch einige Bearbeitungsmöglichkeiten des zu bearbeitenden Benchmarks, wie z.B. das Einfügen einer neuen Zeile in das Programm, zu finden.

Mit dem Speichern (engl. s*ave)*-Knopf kann der im Benchmark-Fenster definierte Benchmark in einer Datei gesichert werden.

Diese Datei muss mit der Endung „.ben“ versehen sein, damit Benchee es als Benchee-konformen Benchmark erkennen kann und beim Starten entsprechende Methoden verwendet.

Außerdem bietet sich die Möglichkeit den aktuellen Benchmark direkt zu übernehmen, und so über den Benchmark-Karteireiter (siehe Kapitel 3.3) zu starten.

Benchmarks haben öfter in ihrem Ablauf eine Steigerung der Komplexität, so dass nach dem ersten Laden eines Datenbestandes und der entsprechenden Durchführung der Suchanfrage und der Vorbereitungs- und Optimierungsbefehle derselbe Ablauf noch einmal mit gesteigerter Such- bzw. Datenkomplexität ausgeführt wird.

Da es sehr aufwendig wäre sich wiederholende Muster einzugeben, gibt es eine einfache Methode in Benchee, um Schleifen zu definieren.

Die Schleifen werden ähnlich den normalen Benchmarks definiert, wobei allerdings nicht die Suchanfragen und die Datenbestände direkt über die entsprechenden „Set Query“- und „Set T-Box/A-Box“-Felder eingebunden werden, sondern Dateien, die die entsprechenden Pfadangaben enthalten. Dies geschieht über die oben rechts befindlichen Felder (*Set Data-Dir* und *Add Query-Dir*).

Die dort anzugebenden Dateien sind einfache Textdateien, die zeilenweise eine Pfadangabe zu einer Datei mit einer Suchanfrage bzw. einem Datenbestand enthalten.

Hierbei ist zu beachten, dass die Anzahl der Pfadangaben in den entsprechenden Dateien übereinstimmen, da über die Zeilennummer der entsprechende Datenbestand für die Suchanfrage geladen wird.

D:\RACER\KAON\dct_10_50\test.racer
D:\RACER\KAON\dct_15_50\test.racer
D:\RACER\KAON\dct_20_50\test.racer
Text 3.1: Beispiel einer Datei mit Datenbestandspfadangaben

D:\RACER\KAON\dct_10_50\test_query.racer
D:\RACER\KAON\dct_15_50\test_query.racer
D:\RACER\KAON\dct_20_50\test_query.racer
Text 3.2: Beispiel einer Datei mit entsprechenden Suchpfadangaben

Sollten jeweils dieselbe Suchanfrage oder derselbe Datenbestand benutzt werden, so muss die jeweilige Datei wiederholt angegeben werden.

Ein Beispiel für diese Art von Dateien zeigt Text 3.1 und Text 3.2.

Eine Schleife erstreckt sich von einer Datei (Query-Dir), die Suchanfragen enthält, bis zur nächsten, bzw. vom Anfang des Benchmarks bis zur Query-Dir-Datei.

Die zusammengehörigen Schleifeninhalte werden in Benchee farblich hervorgehoben (s. Abb. 3.2).

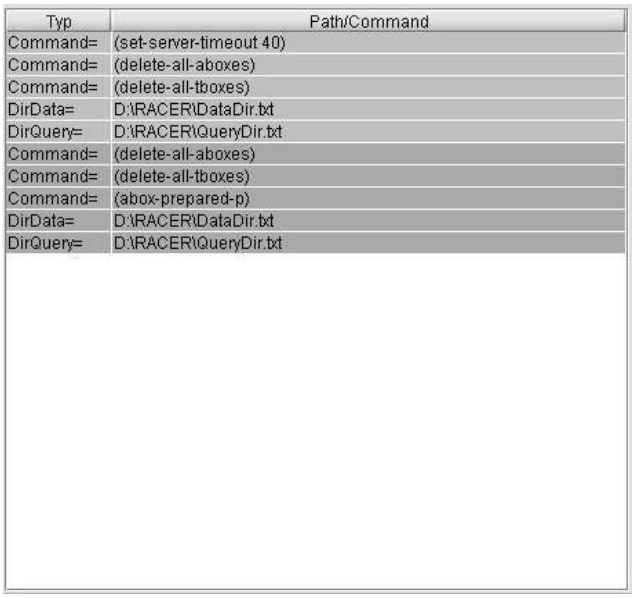

Abbildung 3.2: Darstellung des Schleifenverhältnisses in Benchee

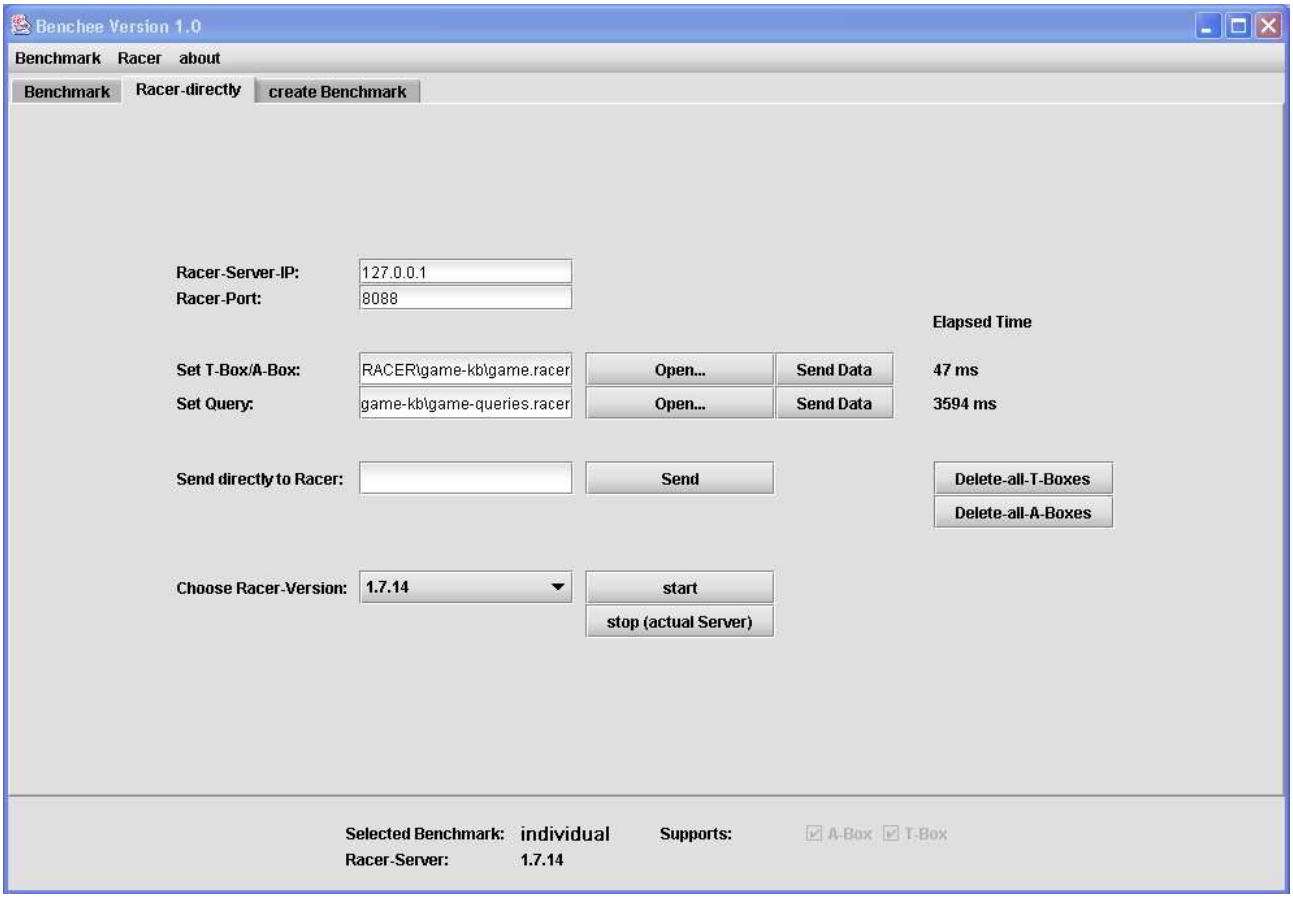

Abbildung 3.3: Bildschirm für die direkte Kommunikation mit dem Racer-Server

3.2 Direkte Kommunikation mit dem Racer-Server

Unter dem Karteireiter „Racer-directly“ ist eine direkte Kommunikation mit dem Racer-Server möglich (s. Abb. 3.3).
Hierbei können Datenbestände sowie Suchanfragen durch das Drücken auf den jeweiligen *Send-Data*-Knopf gesendet werden, wobei die benötigte Zeit rechts davon ersichtlich ist.
Diese Zeiten werden nur ausgegeben und nicht mitprotokolliert, da diese Funktion von Benchee nur für zwischenzeitliche Kommunikation mit dem Racer-Server oder zum Testen von Suchanfragen oder Datenbeständen gedacht ist.
Die Befehle, die unter „Send directly to Racer“ eingegeben werden können bzw. die vordefinierten Befehle „Delete-all-T-Boxes“ und „Delete-all-A-Boxes“, unterliegen keiner Zeitmessung.
Außerdem ergibt sich die Möglichkeit, einen lokalen Racer-Server zu starten bzw. ihn wieder zu beenden, wobei nur Server beendet werden können, die auch über Benchee gestartet wurden.
Selbstverständlich können auch nicht über Benchee gestartete Racer-Server angesprochen werden, wenn die IP-Adresse bzw. der entsprechende Port angegeben wird.
Standardmäßig wird hierbei ein lokaler Server unter dem Standardport 8088 angesprochen.

3.3 Starten eines Benchmarks

Unter dem Karteireiter „Benchmark“ (s. Abb. 3.4) können Benchmarks mit verschiedenen Optionen gestartet und ähnlich wie bei der direkten Kommunikation mit Racer (Karteireiter „Racer-directly“) ein Racer-Server gestartet bzw. dieser auch wieder beendet werden.

„Benchee“ kann unterschiedliche Benchmarks verwalten:

Einerseits können natürlich die in Benchee erstellten Benchmarks gestartet werden, die über den „Transfer in Benchee“-Knopf unter dem Karteireiter „create Benchmark“, der in Kapitel 3.1 beschrieben wurde, oder über die Menüleiste im Menü „Benchmark“ unter dem Menüpunkt „Add/Change Benchmark“ hinzugefügt werden können (siehe Kapitel 3.4).

Andererseits sind aber auch externe Benchmarks erlaubt, die entweder als compiliertes Kommandozeilenprogramm oder als Java-„Jar“-Archiv vorliegen.

Zum Starten eines Benchmarks, stehen abhängig von der Art des Benchmarks einige Optionen zur Verfügung, die sowohl den Racer-Server als auch den Benchmark betreffen.

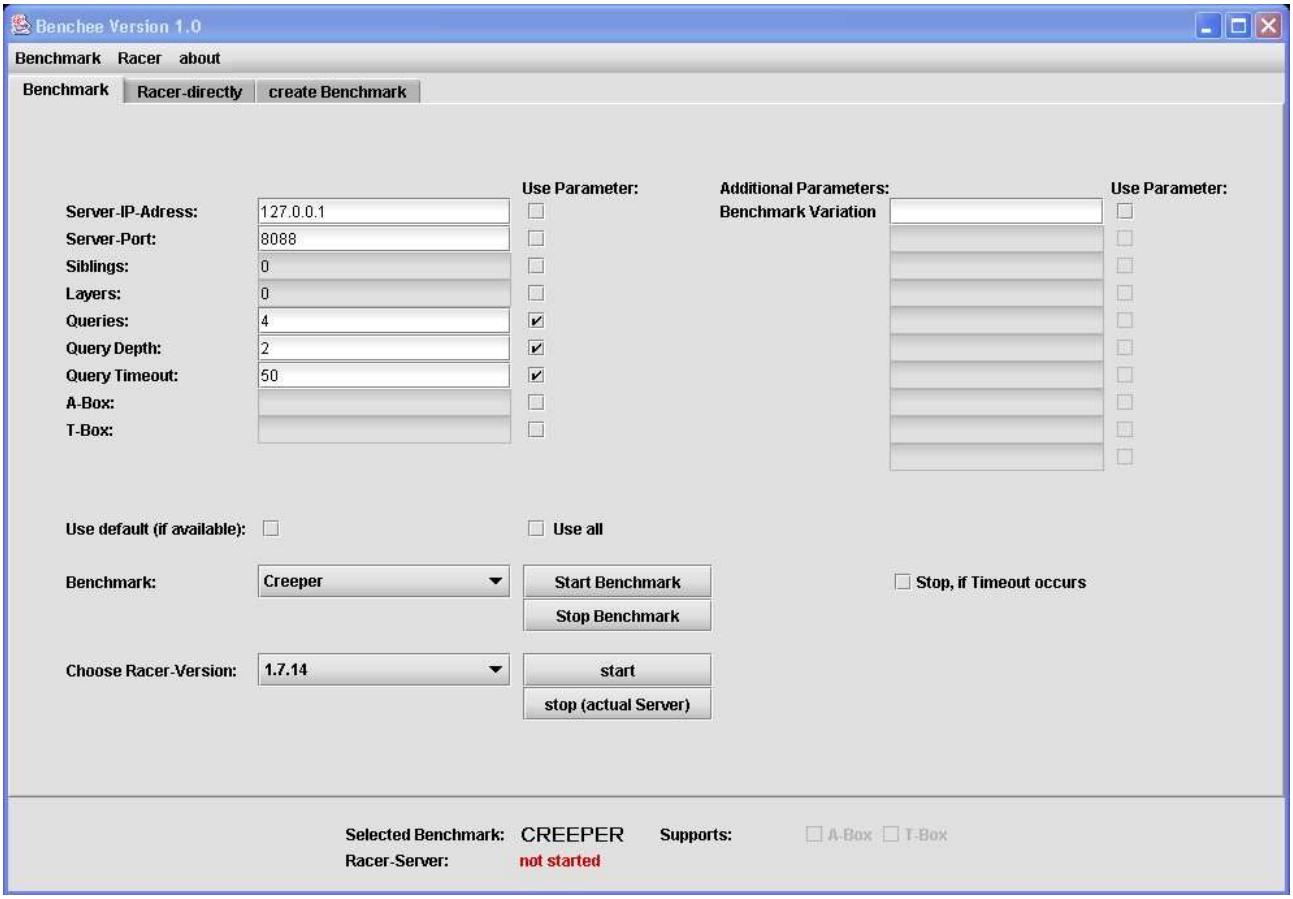

Abbildung 3.4: Programmbildschirm zum Starten eines Benchmarks

Bei einem in Benchee erstellten Benchmark werden lediglich die Racer-Server-IP sowie der Racer-Port als Optionen angeboten, da die restlichen Parameter, wie z.B. Anzahl der Suchanfragen etc., innerhalb des eigentlichen Benchmarks definiert werden.
Die Racer-Optionen werden benötigt, wenn ein externer Racer-Server angesprochen werden soll.
Für die Kommandozeilenprogramme bzw. die Java-Archive können, je nach Art des Benchmarks, die gewünschten Parameter als Argumente übergeben werden.
Die betreffenden Argumente werden in der Benchee-Benchmark-Verwaltung eingegeben und stehen dann beim Starten des Benchmarks als Option zur Verfügung.
Es gibt auf der linken Seite des Bildschirmes einige vordefinierte Optionen, die durch die frei definierbaren Parameter auf der rechten Seite ergänzt werden (s. Abb. 3.4).
Dabei werden die Parameter nur dann berücksichtigt bzw. freigegeben, dargestellt durch ein editierbares Textfeld, wenn sie auch wirklich vom jeweiligen Benchmark unterstützt werden.
Die Verwendung des Parameters wird durch einen Haken hinter dem Textfeld bestimmt, wobei es die Möglichkeit gibt, alle unterstützten Parameter mittels entsprechenden Hakens auszuwählen.
Wenn ein Standardparameterprofil (engl.: *default*) existiert, lässt sich dieses ebenso auswählen.
Der Benchmark selber kann, genauso wie der Racer-Server, über das entsprechende Feld ausgewählt werden, wobei es bei in Benchee erstellten Benchmarks die Möglichkeit gibt, bei einer Zeitüberschreitung abzubrechen (*Stop, if Timeout occurs*).
Dabei ist zu beachten, dass die Timeoutfunktion erst ab Racer-Version 1.8 unterstützt wird.
Bei anderen Arten von Benchmarks kann dieses innerhalb des Programmes definiert werden.
In diesem Fall wird es nicht über den betreffenden Haken, sondern nur über das Timeout-Parameterfeld ausgewählt.
Benchee-Benchmarks Ergebnisse werden in eine Gnuplot-konforme Datei gespeichert und können mit dem Visualisierungsprogramm Gnuplot [gnuplot] dargestellt werden (s. Abb. 3.5).
Alle Befehle und Anfragen, die der Benchmark ausführt, werden dabei mitprotokolliert.
Die Ausgabe externer Benchmarks wird nicht geändert, so dass sie genauso wie außerhalb von Benchee funktionieren.

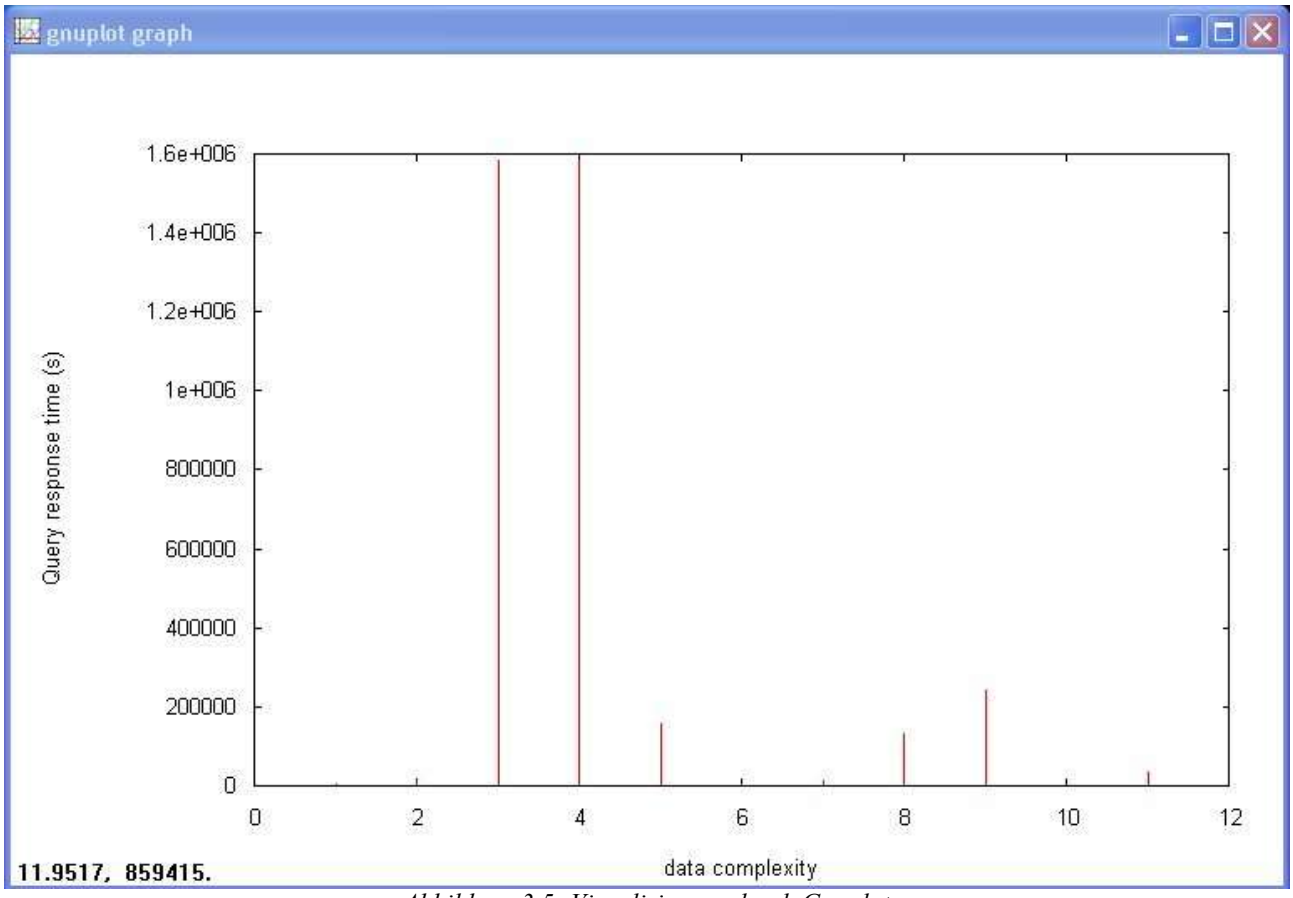

Abbildung 3.5: Visualisierung durch Gnuplot

Abbildung 3.6: Programmbildschirm für die Verwaltung von Racer-Servern

3.4 Verwalten der Benchmarks bzw. der Racer-Server

Unter den Menüpunkten „Benchmark“ bzw. „Racer“ lassen sich die Benchmarks und Racer-Server verwalten, d.h. hinzufügen, verändern und löschen.
In Abbildung 3.6 ist das Verändern bzw. das Hinzufügen eines Racer-Servers dargestellt.
Hierbei müssen die Pflichtfelder Version und Pfad (engl.: *path*) ausgefüllt und alternativ Optionen, deren Übergabe beim Starten des Servers über Benchee erfolgt, angegeben werden.
Um einen vorhandenen Server auszuwählen, muss zuerst der Server anhand seiner Versionsnummer ausgewählt werden. Danach können die Attribute bearbeitet werden.
Das Hinzufügen bzw. Verändern eines Benchmarks verhält sich ähnlich, es können jedoch noch zusätzliche Parameter angegeben werden (s. Abb. 3.8).
Als Pfadangabe können hier Java-„Jar“-Archive oder Benchee-Benchmarks, deren Dateien die Endung „.ben“ haben, eingefügt werden. Benchee-Benchmarks werden ohne weitere Parameter erkannt und entsprechend gestartet.
Außerdem können jegliche Art von auf dem Betriebssystem lauffähigen Programmen angegeben werden.
Die Parameter werden nur berücksichtigt, wenn sie ausgefüllt werden bzw. bei den frei definierbaren Parametern Name und Parameterangabe gesetzt wird.
Außerdem kann die Angabe erfolgen, ob ABoxen bzw. TBoxen vom Benchmark unterstützt werden, oder ob sie nur eine der beiden Möglichkeiten im Benchmark messen.
Das Löschen gestaltet sich bei Benchmarks sowie bei Racer-Servern gleich.
Es werden die zu löschenden Objekte ausgewählt und mit dem Lösch-Knopf (engl.:delete) gelöscht.

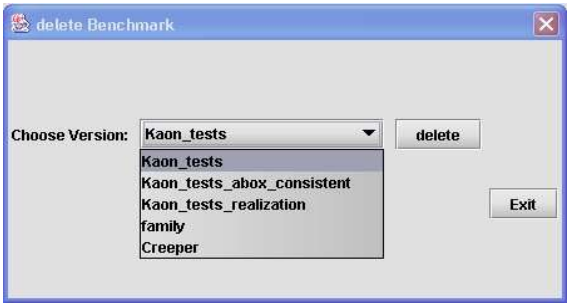

Abbildung 3.7: Löschen eines Benchmarks

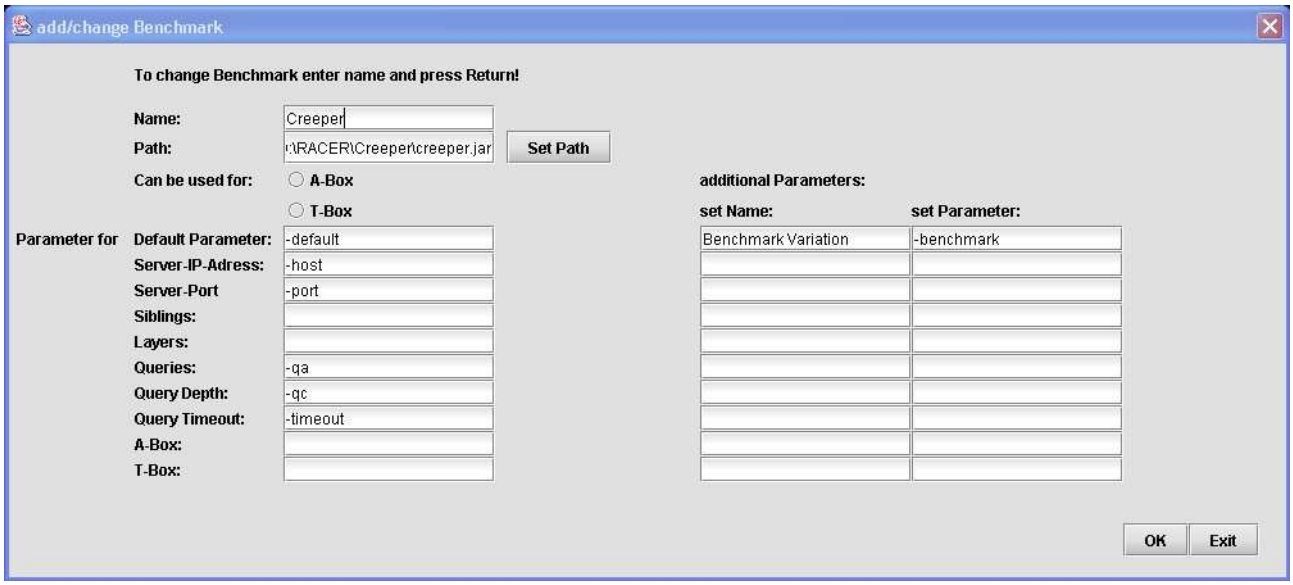

Abbildung 3.8: Programmbildschirm zur Verwaltung von Benchmarks

Kapitel 4

Technische Vorstellung des Programmes

4.1 Beschreibung des Benchee internen Benchmarks

Der Benchee-konforme Benchmark, der über den Karteireiter „create Benchmark“ erstellt werden kann, wird zeilenweise bearbeitet, d.h. in jeder Zeile steht eine Operation.
Als erstes wird der Operationstyp angegeben, gefolgt von einem Gleichheitszeichen und dem eigentlichen Befehl oder der Angabe des Pfades.
Die in Benchee erstellten Benchmarks werden in Text-Dateien gespeichert.
So werden die einzelnen Operationstypen zeilenweise und ebenfalls wie oben beschrieben, durch Gleichheitszeichen getrennt, abgespeichert.
Es gibt fünf Operationstypen: Data, Query, Command, DirQuery und DirData.
Der Operationstyp Data gibt eine einfache Text-Datei, die ABoxen und TBoxen in Racer-Semantik enthält, an und bildet die Daten für die spätere Suchanfrage.
Ein Beispiel für so eine Text-Datei bietet Text 4.1.
Hier wird eine TBox mit einer Familienstruktur definiert, auf die dann anschließend eine ABox mit einer fiktiven Familie aufgesetzt wird.
Im Query-Typ wird ebenfalls der Pfad einer einfachen Text-Datei angegeben, die allerdings eine oder mehrere Suchanfragen oder auch Optimierungsoperationen etc. enthält, je nachdem worüber die Antwortzeit gemessen werden soll.
Dabei wird die Zeit über alle in der Datei befindlichen Befehle gemessen. Für einzelne Messungen muss jeder Befehl in einer Text-Datei stehen.
Für das Familien-Beispiel wäre z.B. „(INDIVIDUAL-INSTANCE? DORIS WOMAN)“ eine Frage, ob *Doris* eine Instanz des Elementes *Woman* ist oder in die reale Welt übertragen, ob Doris eine Frau ist.
Beim Command-Typ werden keine Pfadangaben gemacht, sondern direkt ein Befehl eingegeben. Hierüber erfolgt keine Zeitmessung und dient daher eher der Vorbereitung bzw. der Optimierung des Racer-Servers.

Beispielsweise werden über „(delete-all-tboxes)“ alle T-Boxen gelöscht, um einen konsistenten Datenzustand zu haben.

Die beiden Typen DirData und DirQuery sind, wie bereits in Kapitel 3.1 beschrieben, für die iterative Ausführung der angegebenen Daten und Befehle verantwortlich.

Da Benchmarks oft in der Komplexität ansteigen, werden die gleichen Befehle mit immer komplexeren Daten und/oder komplexeren Suchanfragen ausgeführt.

Dieses wird mittels einer Pfadangabe auf eine Text-Datei, in der sich zeilenweise Pfadangaben für die entsprechenden Datenbestands- bzw. Suchanfragendateien befinden, die entsprechend der gewünschten Komplexität aufgebaut sind, realisiert.

Ein Beispiel wurde bereits in Text 3.1 und 3.2 gegeben.

Die Schleifen erstrecken sich dabei, wie bereits erwähnt, vom Anfang bis zum DirQuery bzw. vom DirQuery bis zum nächsten DirQuery und werden in Benchee farblich hervorgehoben.

```
(in-tbox family)


(signature :atomic-concepts (human person female male woman man
                             parent mother father
                             grandmother aunt uncle
                             sister brother
                             only-child)
           :roles ((has-descendant :transitive t)
                   (has-child :parent has-descendant
                              :domain parent
                              :range person)
                   (has-sibling :domain (or sister brother)
                                :range (or sister brother))
                   (has-sister :parent has-sibling
                               :range (some has-gender female))
                   (has-brother :parent has-sibling
                                :range (some has-gender male))
                   (has-gender :feature t))
           :individuals (alice betty charles doris eve))
(implies person (and human (some has-gender (or female male))))
(disjoint female male)
(implies woman (and person (some has-gender female)))
(implies man (and person (some has-gender male)))
(equivalent parent (and person (some has-child person)))
(equivalent mother (and woman parent))
(equivalent father (and man parent))
(equivalent grandmother
            (and mother
                 (some has-child
                       (some has-child person))))
(equivalent aunt (and woman (some has-sibling parent)))
(equivalent uncle (and man (some has-sibling parent)))
(equivalent brother (and man (some has-sibling person)))
(equivalent sister (and woman (some has-sibling person)))




(in-abox smith-family)

(instance alice mother)
(related alice betty has-child)
(related alice charles has-child)
(instance betty mother)
(related betty doris has-child)
(related betty eve has-child)
(instance charles brother)
(related charles betty has-sibling)
(related doris eve has-sister)
(related eve doris has-sister)
```

Text 4.1: Beispiel eines Datenbestandes (TBox und Abox) aus [Racer-Example]

4.2 Beschreibung der Architektur von „Benchee“

Die Abbildung 4.2 zeigt das Klassendiagramm des im Rahmen dieser Arbeit entstandenen Programmes Benchee. Dabei wird nur das wesentliche der Architektur dargestellt und so z.B. die Klassen für die grafische Ausgabe in der symbolischen Klasse „GUI“ (Abkürzung für Graphical User Interface) zusammengefasst.

Für das Starten des Benchee-Programmes wird die „Main“-Methode der Hauptklasse Benchee aufgerufen, die beim Starten des Benchee-Programmes als erstes ausgeführt wird und die grafische Ausgabe startet bzw. die Klassen für diese anspricht.

Über diese grafische Ausgabe werden die Klassen „Racer“ und „Benchmark“ angesprochen, die die verschiedenen Racer-Server bzw. Benchmarks verwalten, starten und beenden.

Dabei bietet die Klasse „Racer“ als Attribute Options, Path und Version an, welche die benötigten Angaben für den Racer-Server wiederspiegeln (s. Kapitel 3.4).

Außerdem stellt die Klasse noch, neben den verschiedenen Setz- bzw. Auslesemethoden der Attribute (engl.: *get* bzw. *set*), Methoden zum Laden- bzw. Abspeichern und Löschen der Server aus bzw. in die Konfigurationsdatei, die die Racer-Server beinhaltet.

Diese Konfigurationsdatei befindet sich genauso wie die entsprechende Datei für die Benchmarks im Hauptverzeichnis des Benchee-Programmes und wird unter dem Namen „RacerConfig.dat“ abgespeichert.

Die vergleichbare Datei der Benchmarks wird als „BenchmarkConfig.dat“ benannt.

Außerdem bietet die „Racer“-Klasse noch die Methoden StartRacer bzw. StopRacer über die die verwalteten Racer-Server gestartet bzw. beendet werden können.

Die Voraussetzung für das Beenden ist aber der Start über das Benchee Programm.

Die Klasse „Benchmark“ besitzt ebenfalls diverse Attribute, die neben dem Namen und dem Pfad des Benchmarks, vordefinierten Parametertypen (s. Kapitel 3.3), wie z.B. „ParameterQueryDepth“ das die Suchtiefe des Benchmarks definiert und auch die zusätzlichen frei definierbaren Optionen des Benchmarks enthalten.

Auch hierfür werden wieder die betreffenden Setz- bzw. Auslesemethoden zur Verfügung gestellt und durch die Abspeicherungs-, Lade- und Löschvorgänge, sowie die entsprechenden Start- bzw. Stopmethoden vervollständigt.

Ergänzt wird diese Klasse durch die Aggregation der „ProcORThread“-Klasse, die in erster Linie kontrolliert, ob der gestartete Benchmark als Prozess oder als Thread gestartet wurde.

Diese Unterscheidung ist erforderlich, da die Benchmarks entweder als Benchee konforme Ben-Datei, als Java-Archiv oder als auf der Betriebssystemplattform ausführbare Datei verwaltet werden. Für die letzten beiden Möglichkeiten wird ein neuer Prozess gestartet, während die Ben-Dateien in einem Thread laufen.

Da diese beiden Programmstartmöglichkeiten leicht unterschiedlich beendet werden, ist diese Klasse in der Programmstruktur notwendig.

Im Gegensatz zu den im Prozess laufenden Benchmarks wird zum Starten für die Benchee-Benchmarks eine zusätzliche Klasse namens „StartBenchThread“ benötigt, die die Ben-Dateien als Thread abarbeitet.

Die Kommunikation mit dem Racer-Server erfolgt dabei über den im Klassendiagramm nicht dargestellten JRacer [JRacer].

JRacer ist dabei eine Java API (engl. ***A****pplication* ***P****rogrammable* ***I****nterface*), die mittels Methodenaufrufe das Senden von Racer-Befehlen erlaubt.

Diese Benchmarks werden mittels der Klasse „GnuplotLog“, die aus dem Benchmark Creeper [Creeper] übernommen und entsprechend angepasst wurde, über die verschiedenen Methoden in eine Datei geschrieben, die mittels Gnuplot grafisch ausgewertet werden kann.

Die Datei, die diese Auswertungen enthält, wird dabei ins Verzeichnis von Benchee geschrieben und setzt sich aus Benchmarkname und Datum bzw. Uhrzeit zusammen.

Wie bereits erwähnt, hat Benchee keinen Einfluss auf die Auswertung bereits vorhandener externer Benchmarks, es wird die interne Aufbereitung der Daten dieser Benchmarks verwendet.

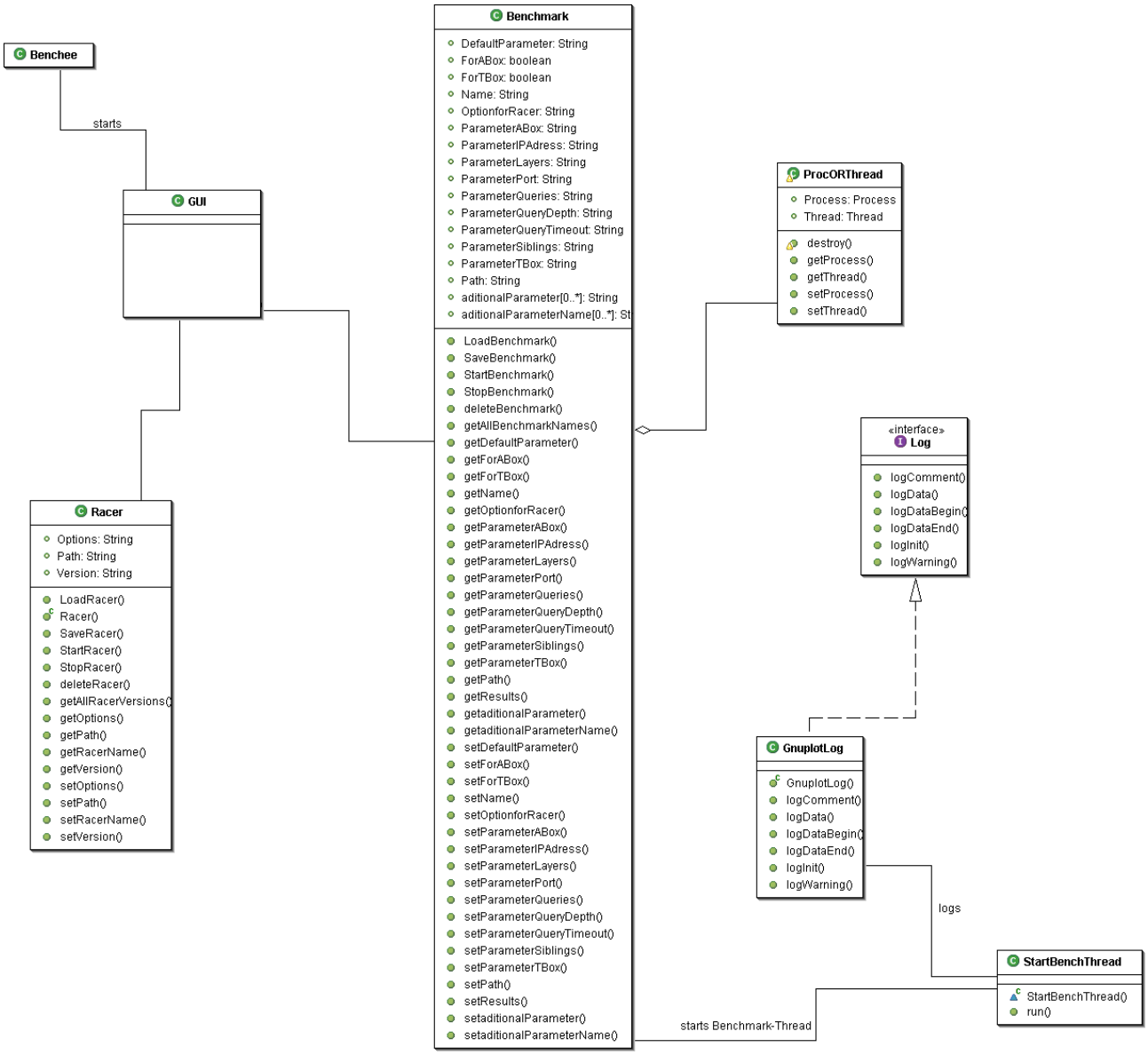

Abbildung 4.1: Klassendiagramm des Benchee-Programmes

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

In der Arbeit ist ein Benchmark-Verwaltungs- und Erstellungsprogramm entstanden, das den verschiedenen Benutzergruppen ermöglicht, ohne Vorkenntnisse einer Programmiersprache neue Benchmarks zu erstellen und vorhandene einzubinden.
Dabei ist das Programm, genannt „Benchee“, durch die Verwendung der Programmiersprache Java plattformunabhängig.
Als Ausgabe wurde dabei ein Dateiformat gewählt, welches durch das Visualisierungsprogramm Gnuplot dargestellt werden kann.
Außerdem können verschiedene Versionen des Racer-Servers verwaltet und somit eine Vergleichbarkeit hergestellt werden.

5.2 Ausblick

Das in dieser Arbeit entstandene Programm stellt eine erste Version dar und ließe sich in der Funktionalität noch erweitern.

So wurde zur Ergebnisspräsentation das Visualisierungsprogramm Gnuplot gewählt und entsprechend werden die Resultate konform von Benchee-internen Benchmarks abgespeichert.

Dabei lag nicht unbedingt der Fokus der Arbeit auf deren Auswertung, so dass in einer Weiterentwicklung eine bessere Auswertung und damit eine komfortablere und direkte Vergleichbarkeit der Ergebnisse integriert werden könnte, die auch eine bessere Nachvollziehbarkeit ermöglichen würde.

Dabei wäre vielleicht auch ein automatisches Starten der Benchmarks auf verschiedenen Versionen des Racer-Servers denkbar, damit man bei Weiterentwicklungen des Servers die unterschiedliche Antwortgeschwindigkeit ermitteln kann.

Diese Funktion müsste zur Zeit noch explizit von Hand aus mit dem Starten der verschiedenen Server und Benchmarks erledigt werden.

Die Richtigkeit der Ergebnisse wird auch nicht überprüft. Dieses könnte bei schnellen Fehlermeldungen des Systems, die als Antworten auf Suchanfragen interpretiert werden, zu falschen Schlussfolgerungen führen.

Außerdem ist Benchee sehr Racer-spezifisch entwickelt worden.

Eine Anpassung auf andere Beschreibungslogik-Systeme wäre noch eine sinnvolle Erweiterung.

Weiterhin wäre langfristig auch eine Entwicklung einer Benchmarksprache für die Beschreibungslogik vorstellbar, die nicht nur von Benchee verstanden wird, sondern einen allgemein gültigen Standard darstellt, der dann auch wieder in Benchee integriert würde.

Literaturverzeichnis

[dl-semweb]: Dr. rer.-nat. habil. Volker Haarslev, Description Logics:A Logical Foundation of theSemantic Web and its Applications, http://www.cs.concordia.ca/~haarslev/publications/dl-semweb.pdf

[Racer04]: Dr. rer.-nat. habil. Volker Haarslev, Prof. Dr. rer. nat. Ralf Möller, Racer-Server, http://www.sts.tu-harburg.de/~r.f.moeller/racer/index.html, 2004

[Horrocks et al. 2000]: Horrocks, I., Sattler, U., and Tobies, S., Reasoning with individualsfor the description logic SHIQ., 2000

[Bechhofer 02]: Bechhofer, The DIG Description Logic Interface:DIG/1.0, 2000

[Patel-Schneider and Swartout 93]: P.F. Patel-Schneider, B. Swartout, Description LogicKnowledge Representation System Specification from the KRSS Group of the ARPAKnowledge Sharing Effort, November 1993

[gnuplot]: Gnuplot Visualisierungsprogramm http://www.gnuplot.info/

[Racer-Example]: Racer Beispieldateien http://www.sts.tu-harburg.de/~r.f.moeller/racer/Examples.zip, 2004

[Creeper]: Jan Lipiski / Eno Cramer, Creeper Benchmark, 2003

[Java]: Java Programmiersprache, http://java.sun.com

[racer-manual]: Dr. rer.-nat. habil. Volker Haarslev, Prof. Dr. rer. nat. Ralf Möller, RACER User's Guide and Reference Manual, http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf, April 2004

[Jracer] Daniel Hartwig, Rene Weissmann, ein TCP-basierender Client für Racer, Univ. of Applied Sc., Wedel, http://www.sts.tu-harburg.de/~r.f.moeller/racer/JRacer2.zip, 2004

[RDF]: Frequently Asked Questions about RDF, http://www.w3.org/RDF/FAQ, 2004

[OWL]: OWL Web Ontology Language, http://www.w3.org/TR/owl-guide/

[DAML-OIL]: DAML-OIL Reference http://www.w3.org/TR/daml+oil-reference

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Hamburg-Harburg im November 2004