Projektangebot

Universität Leipzig – Softwarepraktikum 2014

Semantic Chess

Jonas Herrig

Hanno Krümpelmann

Nathalie Bargenda

Duc Hieu Nguyen

Stefan Süssmeier

Lisa Höncke

Inhaltsverzeichnis

1. Projektvision

Schach ist eines der bekanntesten und am besten dokumentierten Spiele der Welt, jedoch sind bei mehr als 3 Millionen verfügbaren Partien die Suche nach einer bestimmten Partie für einen Laien eine wahre Herausforderung. Daher besteht unsere Vision darin ein benutzerfreundliches Softwarepaket zu erstellen, das in der Lage ist, einfache Benutzeranfragen in Form von natürlicher Sprache zu analysieren. Diese sollen anschließend in Suchanfragen umgewandelt werden und effizient in dieser großen Datenbank passende Ergebnisse ermittelt und dem User präsentiert werden. So kann der User zum Beispiel in das vorgefertigte Webinterface folgende Anfrage eingeben: „Finde eine Partie mit Großmeister X“ und das Programm gibt im Interface alle entsprechenden Partien dieses Großmeisters aus.

2. Voraussetzungen

Das Softwarepaket setzt voraus, das der User eine Suchanfrage in natürlich englischer Sprache stellt. Die Schwierigkeit der Entwicklung besteht darin nicht nur die Wörter und deren semantische Bedeutung zu erkennen sondern auch die Syntax, also die Struktur der Zusammenhänge zu erkennen und richtig umzusetzen, welches durch die Stanford JavaNLP API ermöglicht wird. Die Ergebnisse werden anschließend in ein entsprechendes Pattern eingeordnet werden und zu einer SPARQL-Querie umgewandelt werden. Diese Anfrage soll dann in einer bereits existenten Virtuoso Triple-Store-Datenbank die Daten in RDF-Format finden und an das Userinterface als „Hyperlinks“ zu den jeweiligen Objekten zurückgeben.

3. Designübersicht und Funktionalität

Die einzige für den User sichtbare graphische Benutzeroberfläche stellt ein Webinterface dar, das durch eine URL im Web aufgerufen werden kann. Dieses Webinterface fordert den User auf eine Angabe in natürlicher Sprache zu machen und gibt Beispiele für eventuelle Eingaben.Während der Eingabe durch Benutzer, könnte das Interface weitere Möglichkeiten vorschlagen oder Korrekturmöglichkeiten bei eventuell aufgetretenen Fehlern präsentieren.

Darunter befindet sich ein Platzhalter, in dem später die Treffer in einer Tabelle präsentiert werden. Der User kann durch Klicken der Hyperlinks zu näheren Informationen des jeweiligen Spiels gelangen.

Durch ein Hyperlink „Admin“ gelangt man auf ein neues Interf1ace, bei dem Username und Passwort erfragt werden um den jeweiligen Nutzer als Admin zu autorisieren. Ist diese Autorisation erfolgreich, gelangt er zu einem neuen Interface in dem in der Lage ist Daten zu aktualisieren.

4. Nutzerszenarien

4.1 User

1. Öffnen der Website:
   Der User öffnet die Homepage und erhält die Möglichkeit die Anfrage in eine Eingabezeile einzugeben.

2. Eingabe:
   Die Eingabe erfolgt durch den User. Dieser ist in der Lage seine Anfrage in natürlicher Sprache zu stellen. Wir beschränken uns dabei auf die englische Sprache, da sie heute die allgemein anerkannte Universalsprache ist und verglichen zur deutschen Sprache weniger komplex ist.
   Die Fragen könnten mit einigen der typischen W-Fragen beginnen (Wer, Wann, Wo, Welche). Zusätzlich könnten die Anfragen auch imperativer Natur sein wie „Gib mir...“, „Suche nach ...“ oder „Zeige...“. Danach würde der bedeutende Teil, der die Bedingungen der zu suchenden Daten angibt, anfangen: „Spiele, die nach 15 Zügen enden“ oder „Spieler, dessen Elo höher als 2000“.

3. Ausgabe:
   Die Ergebnisse werden als Tabelle zurückgegeben.

4. Zusätzliche Details:
   Durch anklicken auf einer Person oder einer Partie aus der erhaltenen Liste bekommt der User mehr Informationen über das Spiel oder den Spieler. (z.B. Datum des Spiels)

4.2 Administrator

1. Einloggen und Ausloggen:
   Der Administrator soll Rechte erhalten, die der normale User nicht hat. Damit dies der Fall bleibt, muss der Administrator sich anmelden.
   Das Ausloggen dient der allgemeinen Sicherheit, sodass die Administrator-Rechte erneut aufgerufen werden müssen, wenn eine andere Person Zugang zum selben Rechner haben sollte.

2. Datenmanipulation:
   Dem Administrator ist es erlaubt neue Schachdaten zur Datenbank hinzuzufügen. Sollten einige Daten korrupt sein und die Suchanfrage erschweren, so ist er in der Lage diese zu bearbeiten oder gegebenenfalls zu löschen.

3. Chronik:
   Die Chronik soll dem Administrator zeigen wann welche Daten bearbeitet wurden. Dies soll ihm die Fehlersuche vereinfachen.

4. Arbeitspakete

4.1 Übersicht der Arbeitspakete

Mussziele

Oberfläche (Seite mit Suchfeld)	10%
Serverprozess	10%
Question Parser	30%
SPARQL Query Generator	30%
Schnittstelle zu Triplestore	10%
Benchmark	10%
Gesamt:	**100%**

Kannziele

Autovervollständigung	15%
Administrationsoberfläche	20%
Erweiterung der Schnittstelle für Triplestores	10%
gesamt:	**145%**

4.2 Mussziele

4.2.1 Oberfläche

Damit die User die Schachdaten auf möglichst einfache Art und Weise durchsuchen können, ist es erforderlich, eine einfach zu bedienende (graphische) Oberfläche für die QA-Engine zu entwickeln. Für diesen Zweck ist es geplant, dies mittels einer Weboberfläche zu realisieren. Diese soll (wie bei allen Suchmaschinen üblich) aus einer einfachen (zentrierten) Eingabezeile für die Frage bestehen. Des Weiteren sollen die aus der Frage resultierenden Ergebnisse in einer ansprechenden Art und Weise dargestellt werden.

4.2.2 Serverprozess

Um die vom User über die Weboberfläche übermittelte Frage entgegen zunehmen und die nötigen Schritte zur Weiterverarbeitung einzuleiten, soll als Teil der eigentlichen QA-Engine ein Serverprozess implementiert werden. Dieser soll entweder direkt als eigener Webserver arbeiten (eher optional...) oder zumindest als Schnittstelle zu einem bereits vorhandenen Webserver dienen.

4.2.3 Question Parser

Der Question Parser ist ein zentraler Bestandteil der QA-Engine und für den ersten Schritt in der Verarbeitung der natürlichsprachigen Frage hin zu einer SPARQL-query verantwortlich. Dieser soll die einzelnen Worte der Frage zuerst mittels eines Part-of-speech (POS) Taggers einzelnen Wortarten zuordnen und nachfolgend anhand von zwei Lexika ein SPARQL Query-template erzeugen. Dieses gibt die Struktur der Frage wieder.

4.2.4 SPARQL Query Generator

Der SPARQL Query Generator ist der zweite wesentliche Bestandteil der QA-Engine und für die Umwandlung des Query-templates aus dem Question Parser hin zu einer tatsächlichen SPARQL-Query erforderlich. Hierfür muss das Template mit den entsprechenden Ausdrücken bzw. deren URIs gefüllt werden. Außerdem muss der Generator im Falle mehrerer möglicher Resultate diese entsprechend ihrer Relevanz bewerten. Alle Ergebnisse werden dann gegen den Triplestore getestet.

4.2.5 Schnittstelle zu Triplestore

Die Schnittstelle zum Triplestore muss entwickelt werden, damit die vom SPARQL Query Generator erzeugten Queries an den Triplestore weitergeleitet werden bzw. um die Antworten des Triplestores wieder zurückzuleiten. Ein wesentlicher Vorteil der Schnittstelle besteht darin, dass sie erweiterbar ist, da (optional) die Unterstützung für verschiedene Triplestores implementiert werden kann, welche unter Umständen auch auf entfernten Rechnern eingerichtet sein können. Die Schnittstelle bietet somit eine einzige, abstrakte Zugriffsmöglichkeit, unabhängig vom verwendeten Triplestore und ist dadurch auch auf andere Probleme übertragbar. Sie soll bereits im Vorprojekt umgesetzt werden.

4.2.6 Benchmark

Für eine QA-Engine ist es äußerst wichtig, dass die in ihr verwendeten Algorithmen möglichst effizient arbeiten, weil der User in der Regel nicht sehr lange auf eine Antwort warten will. Deswegen muss ein Benchmark implementiert werden, welcher vorgefertigte Fragen aus einer Datei an die QA-Engine stellt und die Zeit bis zum Eintreffen der Antwort(en) misst. Das soll dabei helfen, die Software bereits in frühen Stadien ausgiebig zu testen um eventuell auftretenden Problemen bei der Performance rechtzeitig entgegenwirken zu können. Daher sollte der Benchmark implementiert werden, sobald die Schnittstellen zu der QA-Engine spezifiziert sind.

4.3 Kannziele

4.3.1 Autovervollständigung

Der Komfort kann deutlich erhöht werden, indem eine Funktion zur automatischen Vervollständigung des vom User eingetippten Textes implementiert wird. Hierfür benötigt der Serverprozess einen Cache, in dem häufig gestellte Fragen gespeichert sind sowie ein Lexikon mit im Triplestore vorkommenden Ausdrücken.

4.3.2 Administrationsoberfläche

Eine Administrationsoberfläche kann implementiert werden, um etwa das Einpflegen neuer Datensätze in den Triplestore zu vereinfachen. Hierfür müssten dann auch der Serverprozess sowie die Schnittstelle zum Triplestore umfangreich erweitert werden.

4.3.3 Erweiterung der Schnittstelle für Triplestores

Die Schnittstelle kann, wie bereits erwähnt, für die Interaktion mit anderen Triplestores erweitert werden.

5. Erster Meilenstein -Vorprojekt

Abgabe: 7.4.

In einem ersten Meilenstein wird das Team bereits eine rudimentäre Version der Suche auf der Schachdatenbank erstellen. Dabei wird noch keine Freitexteingabe möglich sein. Stattdessen werden einige vorgefertigte Suchanfragen zur Auswahl stehen. Von den oben genannten Arbeitspaketen wird also bereits “Oberfläche” größtenteils umgesetzt. Auch der Serverprozess und die Schnittstelle zum Triple-Store werden bereits durch das Vorprojekt realisiert.

Der erste Meilenstein wird jedoch offensichtlich die großen Pakete “Question Parser” und “SPARQL Query Generator” unerfüllt lassen.

6. Qualitätssicherung

Qualitätsanforderungen

Produktqualität	Sehr gut	Gut	Normal	Nicht relevant
Funktion	X			
Zuverlässigkeit		X		
Benutzbarkeit				X
Effizienz	X			
Änderbarkeit			X	
Übertragbarkeit			X	

Bei diesem Projekt steht vor allem die Funktion und die Effizienz im Vordergrund. So sollten möglichst viele Anfragen in einer möglichst kurzen Zeit beantwortet werden können. Dabei kann die Zuverlässigkeit nicht vernachlässigt werden, so sollten zwei Anfragen, die sich sehr ähnlich sind, gleich beantwortet werden. Weniger wichtig hingegen sind die Änderbarkeit und Übertragbarkeit, weil es sehr schwer wäre die Funktion für andere Triplestores sicherzustellen ohne kritische Teile zu ändern. Jedoch wäre es wünschenswert eine gewisse Übertragbarkeit bzw. Änderbarkeit zu verwirklichen, aber es sollte nicht im Hauptfokus stehen. Nicht relevant ist die Benutzbarkeit, weil von vornherein durch eine Webseite sehr viele Systeme unterstützt werden, und es auch nur ein einziges Formular gibt was man bedienen kann. So sollte es für jeden User eindeutig sein, wie es zu bedienen ist.

7. Glossar

Client-Server-Architektur

Mit Client-Server-Architektur ist gemeint, dass eine Aufgabe im Netzwerk auf Client und Server verteilt wird. Die verwendete Software unterteilt dabei in Client und Server. Der Client fordert dabei vom Server einen Dienst an, welcher entsprechend die Anforderung beantwortet. Client- und Serverteil eines Programms sind meist auf verschiedenen Rechnern eines Netzwerks.

RDF

Das Ressource Description Framework ist ein Modell zum beschreiben von Aussagen bzw. Wissen. Genauer handelt es sich um Tripel der Form Subjekt – Prädikat – Objekt, welche eine Relation zwischen Subjekt und Objekt repräsentieren. Eine Menge solcher Tripel beschreibt somit einen gerichteten Graphen, wobei Subjekte und Objekte die Knoten und Prädikate die Kanten (von Subjekt zu Objekt) bezeichnen

SPARQL

Die SPARQL Protocol And RDF Query Language ist eine Abfragesprache für RDF. Hierfür wird eine Engine benötigt, welche die queries verarbeitet. Engines sind z.B. in Apache Jena oder Virtuoso enthalten.

Triple Store

Ein Triple-Store ist eine Datenbank, in der Daten in Form von Triples gespeichert sind und entsprechend für die Abfrage durch Abfragesprache bereit stehen. Dabei hat ein Tripel immer die Form Subjekt-Prädikat-Objekt.

Virtuoso

Virtuoso ist ein „Universaler Datenserver“, der verschiedene Modelle unterstützt. Unter anderem kann Virtuoso als Triplestore für RDF-Daten genutzt werden. An diesen können SPARQL-queries gestellt werden. Im Unterschied zu Jena ist Virtuoso eine eigenständige Serveranwendung.