Entwurfsbeschreibung

Universität Leipzig – Softwarepraktikum 2015

Semantic Chess

Jonas Herrig

Hanno Krümpelmann

Nathalie Bargenda

Duc Hieu Nguyen

Stephan Süßmaier

Lisa Höncke

Inhaltsverzeichnis

1. Allgemeines....3
2. Produktübersicht....3
3. Grundsätzliche Struktur- und Entwurfsprinzipien....3
4. Struktur- und Entwurfsprinzipien....5
5.Datenmodell....6
6. Testkonzept....7
7. Glossar....7
8. Quellen....9

1. Allgemeines

In diesem Dokument soll das Konzept für die Realisierung einer Softwarelösung zum Beantworten von Fragen in natürlicher (englischer) Sprache mittels template based question answering beschrieben werden. Dabei gilt die Einschränkung auf einfache Fragen zu Schachspielen.

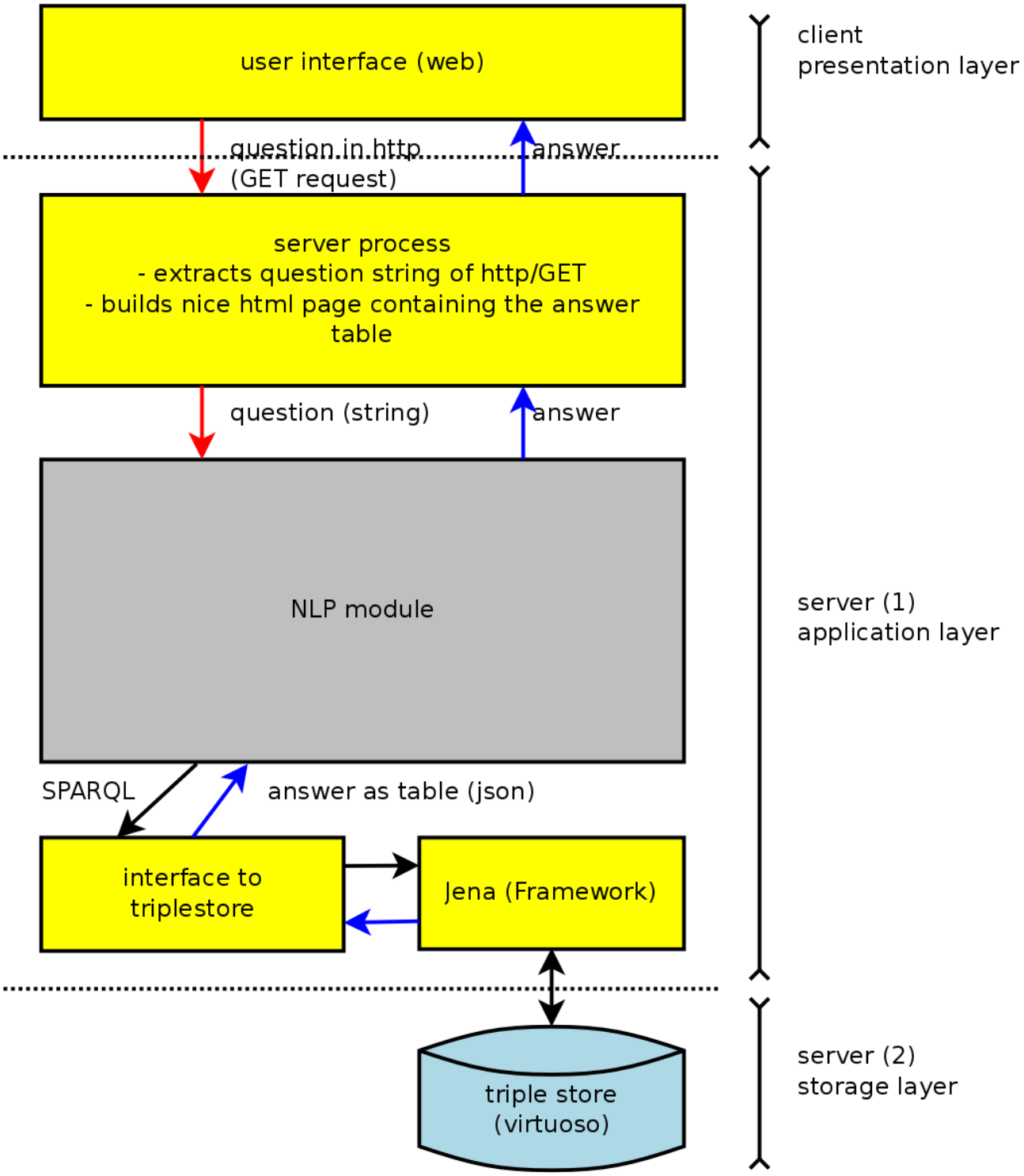
2. Produktübersicht

Die Software setzt im wesentlichen eine Funktion um: über eine simple Weboberfläche kann der User eine Frage stellen. Diese Frage wird serverseitig in eine ausführbare Anfrage überführt, an einen Virtuoso-Triplestore weitergeleitet und das Ergebnis in Form einer einfachen Tabelle zurückgegeben. Die Antwort soll ohne größere Verzögerung erfolgen. Derzeit hat der User keine weiteren Interaktionsmöglichkeiten neben einem Textfeld für die Frage.

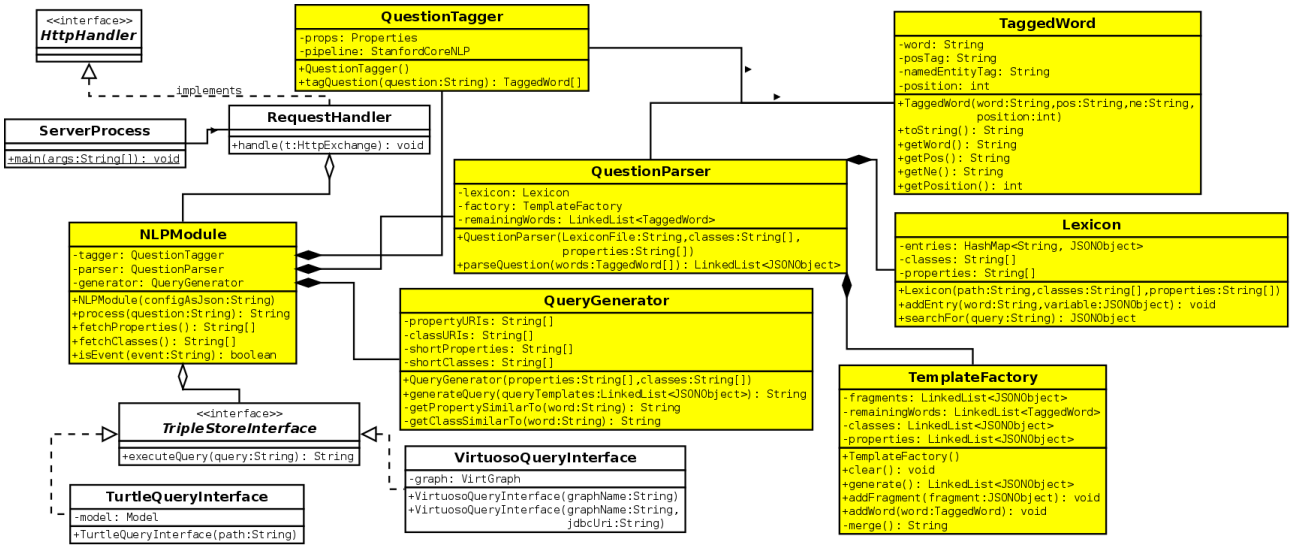
3. Grundsätzliche Struktur- und Entwurfsprinzipien

Da es sich bei der Software um eine Webanwendung handelt, die eine textuelle Benutzereingabe in sequenziellen Teilschritten in eine SPARQL-query umwandeln soll, die schließlich mittels einer query engine beantwortet werden kann, wurde eine Schichtenarchitektur gewählt. Diese untergliedert sich grob in 3 Schichten: Eine Clientschicht für die graphische Benutzerschnittstelle und 2 Serverschichten für die Anwendung selbst und den triple store. Diese 3 Schichten können auf 3 verschiedenen Rechnern ausgeführt werden, da die Kommunikation zwischen diesen Schichten über Netzwerkprotokolle erfolgt. Die Anwendungsschicht untergliedert sich ihrerseits wieder in 3 Schichten: Den Serverprozess, das NLP-modul und die Schnittstelle zum triple store. Der Serverprozess agiert als Webserver, der auf Anfragen (http/GET) wartet und eine entsprechende Antwort (als HTML) generiert. Das NLP-modul wandelt eine natürliche Frage in eine SPARQL-query um und führt diese über die Schnittstelle zum triple store aus.

Der Vorteil dieser Architektur ist, dass die nötigen Schnittstellen für das Hauptprojekt im Vorprojekt bereits implementiert wurden, sodass lediglich das NLP-modul neu entwickelt werden muss.



4. Struktur- und Entwurfsprinzipien



Die Klasse ServerProcess verfügt über die main-Methode, in welcher der http-Server aus der Java-Standardbibliothek unter Verwendung der Klasse RequestHandler gestartet wird. Der RequestHandler extrahiert in der Methode handle aus dem übergebenen Objekt vom Typ HttpExchange die angefragte URI und aus dieser anschließend die gestellte Frage als einfache Zeichenkette.

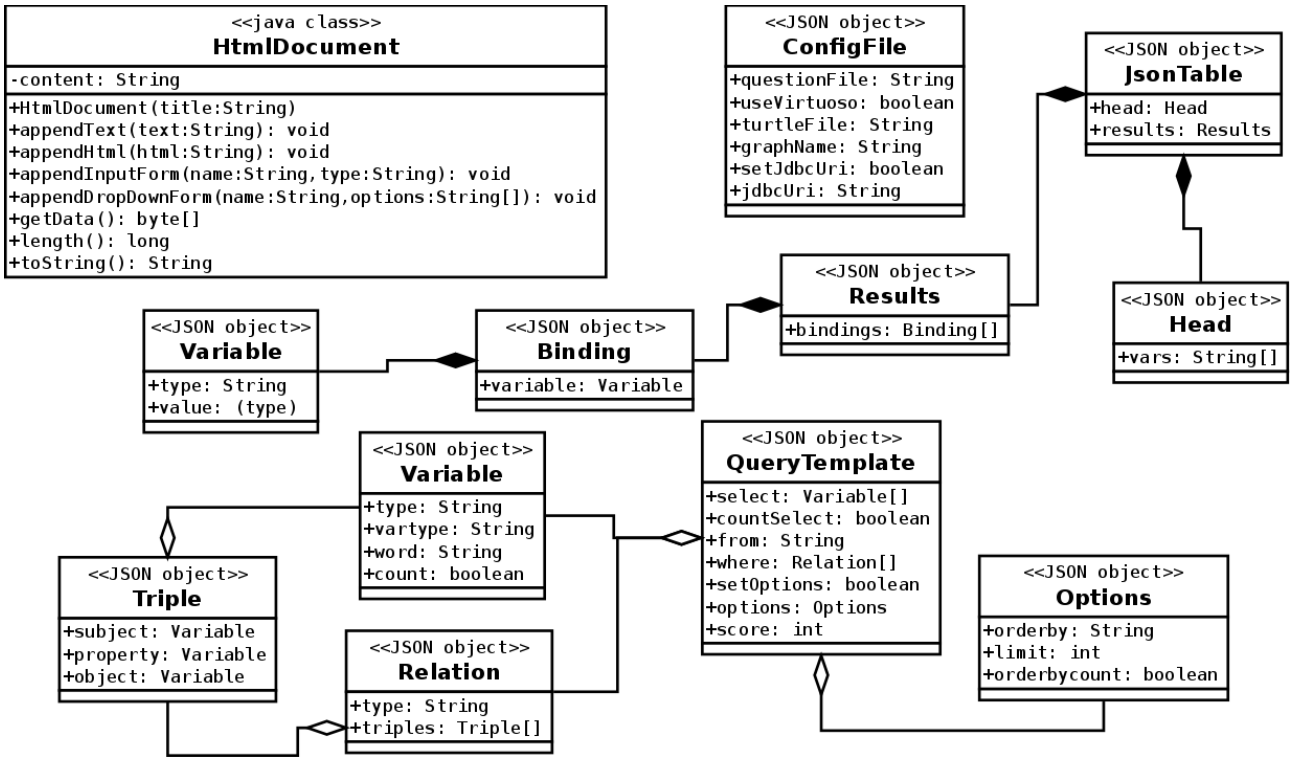
Dieser String wird durch Aufruf der Methode process an die Klasse NLPModule übermittelt. Dort wird der String zunächst an eine Instanz der Klasse QuestionTagger übergeben, welche unter benutzung des Stanford Parsers ein Array aus TaggedWord-Objekten erstellt. Dieses wird im nächsten Verarbeitungsschritt von der QuestionParser-Instanz unter benutzung eines Lexikons in mehrere mögliche query-templates überführt. Im letzten Schritt werden die Slots der Templates durch entsprechende URIs (mit der geringsten Editierdistanz) ersetzt, wobei mittels eines Rankingverfahrens das beste Template ausgewählt wird. Das Resultat ist eine SPARQL-query in Form eines Strings.

Diese Anfrage wird dann über das NLPModule an das TripleStorInterface übergeben. Dieses setzt die Kommunikation mit dem Triplestore über die zur Verfügung stehende API Jena um. Das Ergebnis der Anfrage wird als JSON-Objekt zurück gegeben.

Die eben einzeln beschriebenen Klassen lassen sich zu Modulen im Sinne der Schichtenarchitektur zusammenfassen. Die Klassen ServerProcess und RequestHandler bilden das Paket Server, die gelb hinterlegten Klassen das Paket NLPModule und die Klassen TrurtleQueryInterface und VirtuosoQueryInterface mitsamt des von ihnen

implementierten Interfaces TripleStoreInterface das Paket TripleStoreInterface. Die Pakete entsprechen den in 3. beschriebenen Schichten der Anwendungsschicht.

5.Datenmodell



Die Java-klasse HtmlDocument dient dem Zweck, serverseitig Html-dokumente zu generieren. Neben der Klasse HtmlDocument und JSON-objekten für die Darstellung von Tabellen und Konfigurationsdaten gibt es noch JSON-objekte, welche Query Templates bzw. ihre einzelnen Bestandteile beschreiben. Bei den Variablen werden 4 Variablentypen unterschieden: var für normale Variablen, const für eine Variable mit einem festen Wert (z.B. einem Namen), slot für später durch URIs zu ersetzende Platzhalter und blank für einen leeren platzhalter (nur in der TemplateFactory relevant), der durch eine noch zu erzeugende Variable der anderen 3 Variablentypen ersetzt werden muss. Count-variablen werden durch den boolean wert count gekennzeichnet. Der wert type kann die werte class, class/property, property oder ressource annehmen. Ein Triple wird aus je einer class-, property-, und ressource- variablen erzeugt. Mehrere Triples können in einer Relation zusammengefasst werden, die vom Typ AND oder OR (UNION) sein können. Ein QueryTemplate kann wiederum mehrere solcher Relations enthalten.

6. Testkonzept

Wie in unserem Konzept zur Qualitätssicherung festgelegt, werden Komponententests für einzelne Klassen mit Hilfe von JUnit durchgeführt. Zum Integrationstest einzelner Module werden speielle Testklassen geschrieben, welche typische Testanfragen auf den entsprechenden Methoden der im Modul enthaltenen Klassen ausführen.

Funktionale Systemtests werden manuell durchgeführt, indem manuell geprüft wird, ob die Antworten auf alle möglichen Testanfragen fehlerfrei die gewünschten Informationen enthalten.

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. Eine auch für dieses Softwareprojekt sinnvolle gängige Unterteilung ist es, die GUI und alle mit Eingaben verbundenen Programmteile auf dem Client laufen zu lassen, jedoch die Datenbank und die Suchalgorithmen auf einem Server zu implementieren.

Appache Jena

Appache Jena ist ein Open Source-Framework in Java das zur Konstruierung von Semantic Web Anwendungen entwickelt wurde. Es verfügt über eine Programmierschnittstelle zum Speichern und Laden von RDF-Daten. Des Weiteren verfügt die Programmierumgebung über SPARQL, OWL(Web Ontology Language), GRDDL Frameworks. Die generelle Funtkionsweise umfasst Repräsentation der RDF-Graphen als abstrakte Modelle im Speicher oder Datenbanken die zum Teil auf OWL beruhen, die Abfrage via SPARQL und die Manipulation durch das SPARUL Modul. Dabei zählt Appache Jena zu den beliebtesten RDF-Frameworks.

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.

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.

Hashmap

Eine Hashmap dient zum Speichern von beliebigen Java Objekten. Dabei wird zu den Objekten (Schlüsseln) jeweils ein Wert berechnet, dank dem es in konstanter Zeit eingefügt und entfernt werden kann. Dabei wird die Hashmap bzw. Hashtabelle auf eine bestimmte Größe ausgerichtet und muss bei überschreiten eines festgelegten Füllgrades neu berechnet werden.

JSON

JSON (JavaScript Object Notation) ist ein standardisiertes Datenaustauschformat mittels dessen sich Anwendungen unabhängig von ihrer Programmiersprache austauschen können. Dabei werde sowohl Schlüssel/Wert Paare (siehe Hashmap) als auch Listenstrukturen wie Arrays oder Vektoren verwendet. Dabei ist jedes JSON-Dokument ein gültiges Java-Script.

Bytestream

Als Bytestream (deutsch: Bytestrom) wird eine Sequenz aus 8 Bit großen Bytes bezeichnet. Diese Datenströme werden allgemein von einem Medium ins andere übertragen. In der Regel wird ein Bitstrom zu einem Bytestrom gegliedert und diese gliedern sich weiter in Blöcke und Datenpakete inunteraschiedlicher Protkolle und Formate.

RDF / Turtle

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. Turtle ist eine spezielle Formatierung von RDF-Daten.

HTTP

HTTP ist die Abkürzung für Hyper Text Transfer Protocol ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. In den meisten Fällen wird dieses Protokoll dazu verwendet, um Webseiten auf einen Webbrowser zu laden.

HTTP-Request/Response

Im Http wird zwischen Request (Anfrage) und Response (Antwort) unterschieden, wobei der Client eine Request an den Server sendet und dieser mit einer Response antwortet. Diese Nachrichten müssen dabei dem Http Standard mit Header und Body genügen.

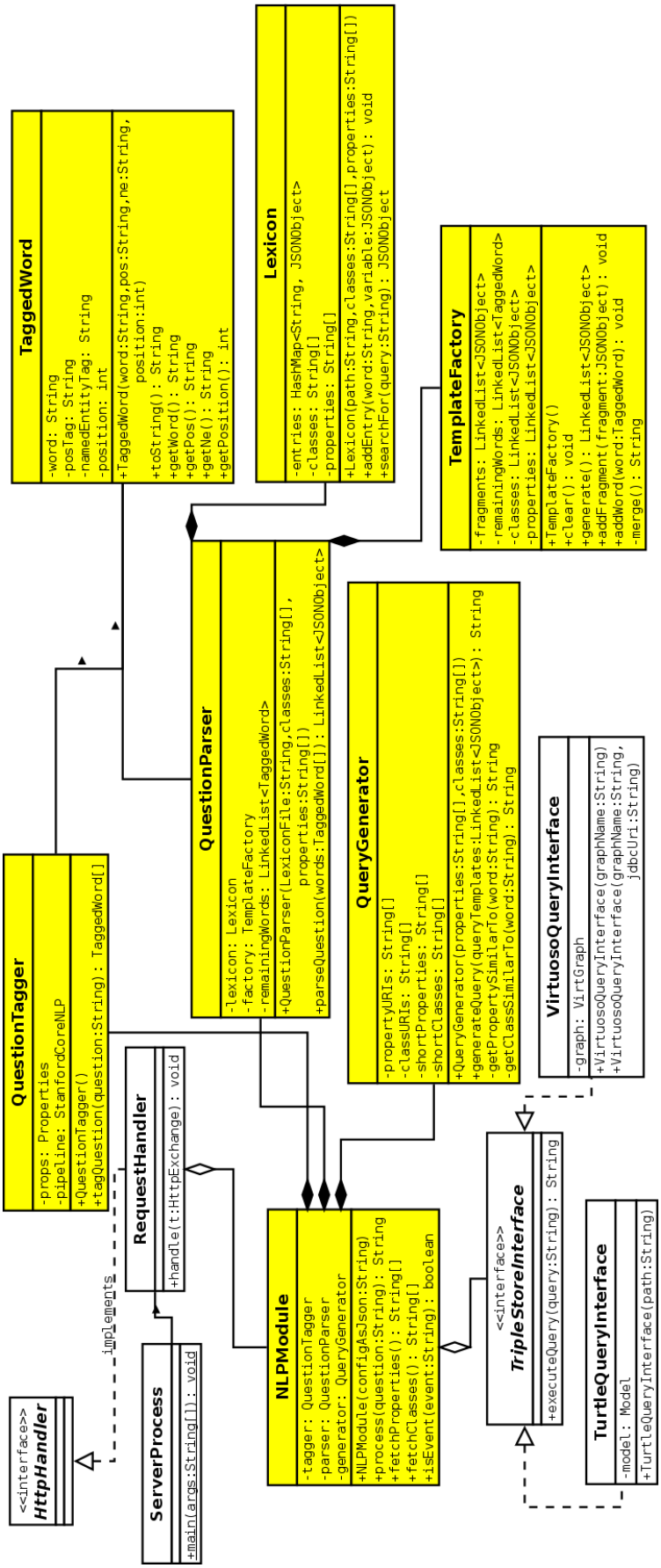
8. Quellen

http://schabby.de/hashmap/

https://jena.apache.org/

http://www.json.org/

https://www.w3.org/2001/sw/wiki/Apache_Jena


TaggedWord
-word: String
-posTag: String
-namedEntityTag: String
-position: int
+TaggedWord(word:String,pos:String,ne:String, position:int)
+toString(): String
+getWord(): String
+getPos(): String
+getNe(): String
+getPosition(): int
Lexicon
-entries: HashMap<String, JSONObject>
-classes: String[]
-properties: String[]
+Lexicon(path:String,classes:String[],properties:String[])
+addEntry(word:String,variable:JSONObject): void
+searchFor(query:String): JSONObject
TemplateFactory
-fragments: LinkedList<JSONObject>
-remainingWords: LinkedList<TaggedWord>
-classes: LinkedList<JSONObject>
-properties: LinkedList<JSONObject>
+TemplateFactory()
+clear(): void
+generate(): LinkedList<JSONObject>
+addFragment(fragment:JSONObject): void
+addWord(word:TaggedWord): void
-merge(): String
QuestionParser
-lexicon: Lexicon
-factory: TemplateFactory
-remainingWords: LinkedList<TaggedWord>
+QuestionParser(LexiconFile:String,classes:String[], properties:String[])
+parseQuestion(words:TaggedWord[]): LinkedList<JSONObject>
QuestionTagger
-props: Properties
-pipeline: StanfordCoreNLP
+QuestionTagger()
+tagQuestion(question:String): TaggedWord[]
QueryGenerator
-propertyURIs: String[]
-classURIs: String[]
-shortProperties: String[]
-shortClasses: String[]
+QueryGenerator(properties:String[],classes:String[])
+generateQuery(queryTemplates:LinkedList<JSONObject>): String
-getPropertySimilarTo(word:String): String
-getClassSimilarTo(word:String): String
VirtuosoQueryInterface
-graph: VirtGraph
+VirtuosoQueryInterface(graphName:String)
+VirtuosoQueryInterface(graphName:String, jdbcUri:String)
RequestHandler
+handle(t:HttpExchange): void
NLPModule
-tagger: QuestionTagger
-parser: QuestionParser
-generator: QueryGenerator
+NLPModule(configAsJson:String)
+process(question:String): String
+fetchProperties(): String[]
+fetchClasses(): String[]
+isEvent(event:String): boolean
<<interface>>
TripleStoreInterface
+executeQuery(query:String): String
TurtleQueryInterface
-model: Model
+TurtleQueryInterface(path:String)
<<interface>>
HttpHandler
implements
ServerProcess
+main(args:String[]): void