UNIVERSITÄT LEIPZIG

Institut für Medizinische Informatik, Statistik und Epidemiologie (IMISE)

Automatische Erstellung von Quizfragen aus einer Ontologie von Krankenhausinformationssystemen

Besondere Lernleistung

Leipzig, Dezember 2020

vorgelegt von:

Max Niclas Wächtler
geb. am: 17.05.2003

Betreuer:
Konrad Höffner

Zusammenfassung

Krankenhausinformationssysteme sind Informationssysteme, die sich in medizinischen Umgebungen mit der Verarbeitung von Daten, Informationen und Wissen beschäftigen. Informationssysteme bestehen aus Hardware, Software als auch Anwendern. Kompetenz in der Nutzung von Krankenhausinformationssystemen ist wichtig, um in Krankenhäusern beispielsweise die Zusammenarbeit mit Patienten effektiv und zeitsparend zu gestalten. Jedoch müssen Anwender diese Kompetenz erst einmal durch Quellen wie Lehrbücher erwerben, welche jedoch verschiedene Sichten auf das Themengebiet aufweisen und keinen schnellen Überblick über bestimmte Themen bieten können. Um dieses Problem anzugehen, habe ich mir als Ziel dieser besonderen Lernleistung gesetzt, ein Programm zu entwickeln, welches diese Informationen aus einer Ontologie, also eine Menge von Begrifflichkeiten mit Beziehungen untereinander, über eine Schnittstelle entnimmt, diese nutzerfreundlich in Form von zufällig generierten Fragen aufarbeitet und dem Nutzer ausgibt. Diese sollen eine grammatikalisch korrekte Form aufweisen und nebst der richtigen Antwort auch falsche Antworten enthalten, die trotzdem logisch wirken sollen, anstatt ein Zufallsverfahren zur Auswahl zu nutzen.

Inhaltsverzeichnis

Begriffs- und Abkürzungsverzeichnis

SPARQL SPARQL Protocol and RDF Query Language

W3C World Wide Web Consortium

KI Künstliche Intelligenz

RDF Resource Description Framework

HTML Hypertext Markup Language

XML Extensible Markup Language

URIs Universal Resource Identifier

SNIK Semantisches Netz des Informationsmanagements im Krankenhaus

WWW World Wide Web

W3C World Wide Web Consortium

HTTP Hypertext Transfer Protocol

MIG Management von Informationssystemen im Gesundheitswesen

Kapitel 1

Einleitung

1.1 Gegenstand und Motivation

1.1.1 Gegenstand

Das Semantisches Netz des Informationsmanagements im Krankenhaus (SNIK) ist ein abgeschlossenes Projekt, welches Begriffe des Informationsmanagements in dem Teilgebiet der medizinischen Informatik sowie deren Beziehungen untereinander beschreibt. Es ist in der Lage, dieses Wissen in einer Ontologie darzustellen und diese sowohl maschinen- als auch menschenlesbar auszugeben. Dabei nutzt es ein dem Semantic Web Stack ähnliches Modell (Abb. 2.2), welches Anwendungen wie dem SNIK Graph ermöglicht, Informationen abzufragen und diese Endnutzern in aufbereiteter Form zur Verfügung zu stellen. Das Projekt konzentriert sich hauptsächlich auf Wissen der medizinischen Lehre. Dieses wird für Krankenhausinformationssysteme zur Verfügung gestellt, um die Arbeit für medizinisches Personal zu erleichtern. Speziell hat der durch Prof. Dr. Alfred Winter geleitete Teilbereich Management von Informationssystemen im Gesundheitswesen (MIG) die Zielsetzung, durch Entwicklung von Methoden und Werkzeugen für das Informationsmanagement zu einer besseren Gesundheitsversorgung beizutragen.
Um nicht mit Ontologien versierten Nutzern eine Anwendung, die die Daten des Projektes nutzt, darzustellen, wurde unter anderem auch ein einfaches Multiple-Choice-Quiz auf Basis eines öffentlich verfügbaren Templates programmiert. Diese nutzt die Ontologie hinter SNIK, um Anwendern einfache, automatisch generierte Fragen zu stellen. Diese Fragen werden mithilfe der typisierten, gerichteten Verbindungen zwischen einzelnen Begriffen erzeugt, wobei das Quiz

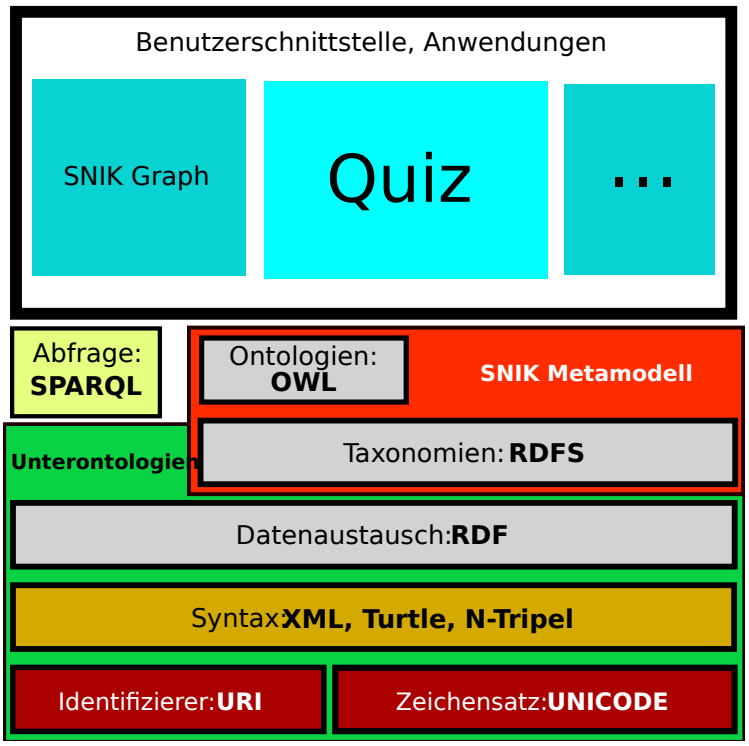

Abbildung 1.1: Das semantische Modell von SNIK.

aus der SNIK-Ontologie zwei Gruppen von Verbindungen nutzt:

- 1. Objekt A definiert Objekt B (A defines B),
- 2. Objekt A beschreibt die Verbindung zweier Objekte B und C (B is related to C in way A)

Dadurch werden für jeden Verbindungstyp unterschiedliche Fragen generiert. Für Definitionen generiert sich der Fragetitel hierbei zum Beispiel nach dem folgenden Muster:
„What is defined by A?“
Unabhängig vom Fragetyp werden jeder Frage noch verschiedene Antworten zugeteilt. Dazu zählt einmal die richtige Antwort, neben der aber auch drei andere, falsche Antworten durch einfache SPARQL-Queries generiert werden.

1.1.2 Problematik

Die Nutzer sollen möglichst mit logischen, aber nicht zu einfachen Fragen herausgefordert werden. Um mithilfe des Multiple-Choice-Quiz ihren Wissensstand zu verbessern, sollte das Quizprogramm bestimmte Vorgaben erfüllen und diese

auch in der Qualität der Fragen widerspiegeln. Das stellt sich als relativ schwer heraus, da die Implementierung von natürlichsprachlicher Grammatik sich seit jeher als ein Problem in der Informatik herausstellt.

Korrektheit Für den Anwender soll die Frage gut lesbar und in seiner Grammatik verständlich sein. Da die Quizfragen in einem bestimmten Zeitfenster bearbeitet werden sollen, ist es wichtig, dass sie schnell vom Nutzer verstanden werden. Da der Hauptzweck die Selbstüberprüfung ist, werden dadurch fehlende Korrektheit gegenüber dem User die Ergebnisse verfälscht , was zu einer falschen Selbsteinschätzung führt.

Übersichtlichkeit Eine Frage muss für den Nutzer übersichtlich gestaltet sein. Sie soll schnell durchgelesen sein, damit er den metaphorischen „Faden“ nicht verliert und Fragen nicht erneut lesen muss. Um dies zu garantieren, muss die Anzahl der Zeichen im Fragetitel eine bestimmte Zahl nicht übersteigen. Dies kann zum Beispiel bei langen Definitionen (Fragetyp 1) ein Problem sein.

Qualität Neben den anderen beiden Bedingungen muss bei Fragen auch die Qualität und das Niveau der Frage sowie des Quiz allgemein hoch gehalten werden. Doch das ist im bereits existenten Quiz an vielen Stellen nur teilweise vorhanden, was sich in verschiedenen Aspekten widerspiegelt. Darunter fallen zum Beispiel die Erwähnung der richtigen Antwort im Fragetitel bei z.B. Definitionen oder in einfacheren Punken wie z.B. dem Doppeln von Fragen. Dadurch wird der Nutzer nicht genug herausgefordert, was dazu führt, dass er die Fragen nicht mithilfe seines Fachwissens, sondern einfach durch logisches Denken beantworten kann.

Heterogenität Das Quiz sollte für den Nutzer möglichst einen großen Teil der Möglichkeiten der SNIK-Ontologie anschaulich machen. Da jedoch das existierende Quiz nur zwei der 18 Typen von Verbindungen zwischen Objekten nutzt, die in der SNIK-Ontologie vorkommen, variieren die Fragetypen nur wenig, was auf Dauer für Nutzer langweilig oder anstrengend wird. Außerdem wird dadurch ein großer Teil des verfügbaren Wissens nicht abgefragt, was dem eigentlichen Sinn der riesigen Datenmenge einer Ontologie nicht gerecht wird.

1.1.3 Motivation

Das Ziel des SNIK-Projektes ist es, ein theoretisch und empirisch begründetes, sowie erprobtes Semantisches Netz des Informationsmanagements im Krankenhaus (SNIK), das die Konzepte des Informationsmanagements beschreibt und verbindet, zu kreieren. Um Medizinstudenten und anderen Mitarbeitern aus dem medizinischen Umfeld eine beispielhafte Nutzungsmöglichkeit für diese Ontologie zu bieten und ihnen einen einfachen Zugriff neben dem SNIK-Graph auf das Themengebiet zu ermöglichen, kann ein Multiple-Choice-Quiz einen einfachen Einblick auf die Einsatzmöglichkeiten von einer Ontologie aus Krankenhaussystemen bereitstellen und Interessenten aus nichtinformatischen Teilgebieten eine Vorstellung ermöglichen.

1.2 Zielsetzung

Das Ziel meiner besonderen Lernleistung ist es, automatisch Multiple-Choice-Quizfragen auf Basis der SNIK-Ontologie zu generieren. Dazu wird ein existierender Prototyp analysiert und hinsichtlich der oben genannten Qualitätsmaße optimiert.

- Ziel Z1: Analysieren des vorhandenen Prototyps
- Ziel Z2: Optimieren des vorhandenen Prototyps

1.3 Vorgehensweise

- Aufgabe A1.1 zu Ziel Z1: Analyse der Strategien zur Generierung der Stämme und Distraktoren
- Aufgabe A2.1 zu Ziel Z2: Entwickeln und Verbessern der Strategien zur Generierung von Fragen und Antworten
- Aufgabe A2.2 zum Ziel Z2: Aufstellen von Regeln zur Nutzerfreundlichkeit der Fragen
- Aufgabe A2.3 zum Ziel Z2: Entwickeln einer plattformunabhängigen Lösung zur Anzeige der Fragen

Kapitel 2

Grundlagen

2.1 Semantic Web

Das Semantic Web (oder auch: semantisches Netz) stellt eine Methode dar, um maschinenlesbares Wissen über das World Wide Web (WWW) in Form von HTML-Dokumenten zu verbreiten. Anders als bei der klassischen Website aber wird hier nicht nur primär Wert auf die Lesbarkeit durch einen Menschen, sondern auch auf die Maschinenlesbarkeit Wert gelegt, indem die Seite nebst dem normalen HTML-Code auch Informationen, die durch Computer gelesen werden können, hinterlegt. Diese sind durch Technologien des semantischen Webs (RDF, SPARQL, ...) einles- und verarbeitbar.

2.1.1 WWW

Das WWW bildet ein verknüpftes Kommunikationsmodell zwischen allen Ressourcen und Nutzern des Internets, die das Hypertext Transfer Protocol (HTTP) nutzen. Es wurde vom Direktor der World Wide Web Consortium (W3C) Tim Berners-Lee 1991 entwickelt und ermöglicht den Zugriff auf sowie den Datenaustausch mit HTML-Dokumenten. Es definiert also das, was die Allgemeinheit als „das Web“ bezeichnet.

2.1.2 Linked Data

Der Begriff „Linked Data“ bezieht sich auf eine Reihe bewährter Methoden zum Veröffentlichen und Verbinden strukturierter Daten im Web. Es verknüpft Dateien aus dem semantischen Web untereinander, sodass sie durch semantische

Abfragen benutzbar werden. Linked Data basiert auf standardisierten Modellen für den Datenaustausch im Web, besipielsweise HTML, um auch wie beim semantischen Prinzip maschinenlesbare Daten zu generieren. Zielsetzung von Linked Data ist es, das Internet zu einer globalen, computerverbarbeitbaren Datenbank weiterzuentwickeln und so den Zugriff für technische Endgeräte weiter zu vereinfachen.

2.1.3 RDF

Resource Description Framework (RDF) ist ein weiteres Standardmodell für den Datenaustausch im Web, welches in den 90er Jahren des letzten Jahrtausends seine Anfänge findet. Im Gegensatz zu Hypertext Markup Language (HTML) und Extensible Markup Language (XML) ist es in der Lage, enthaltene Informationen zu kombinieren und weiter zu verarbeiten, anstatt diese nur korrekt anzuzeigen. Daher ist RDF auch oft als grundlegendes Repräsentationsformat für die Entwicklung im Semantic Web angesehen. Es erweitert die Verknüpfungsstruktur von diesem, um Universal Resource Identifier (URIs) einzubinden. Solche URIs werden im semantischen Netz dazu genutzt, Beziehungen zwischen Dingen beziehungsweise zwischen den beiden Enden der Verknüpfung herzustellen, um Dateien zu verknüpfen, verfügbar zu machen und für Anwendungen freizugeben. Dadurch können Daten zusammengeführt werden, die sich eigentlich im Schema grundlegend unterscheiden, ohne dass alle Datenkonsumenten geändert werden müssen. Diese Zusammenführung geschieht in einem Schichtenmodell: Aussagen werden als Tripel modelliert und bilden eine Aussage, die aus Subjekt, Prädikat und Objekt zusammengefügt werden kann. Man kann also einen Tripel mit einem einfachen Satz vergleichen: als Beispiel hier einmal der Teilsatz *Google produziert Prozessoren*. Hier stellt *Google* das Subjekt dar, *produziert* ist das Prädikat und *Prozessoren* sind das Objekt. Hat man eine Menge von Tripeln, bilden diese ein semantisches Netz, was tabellarisch übersichtlich dargestellt werden kann, siehe Tabelle 2.1.

2.1.4 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) ist ein Standard für die Abfrage von in RDF spezialisierten Informationen sowie für die Darstellung der zugehörigen Resultate, siehe Abb. 2.1. Es basiert auf einfachen RDF-Anfragen in Form von Graphmustern, also einer Menge von Tripeln,

Subjekt	Prädikat	Objekt
New York	liegt in	USA
USA	liegt in	Nordamerika
Nordamerika	liegt in	Amerika
Südamerika	liegt in	Amerika

Tabelle 2.1: Beispiel für RDF-Tripel.

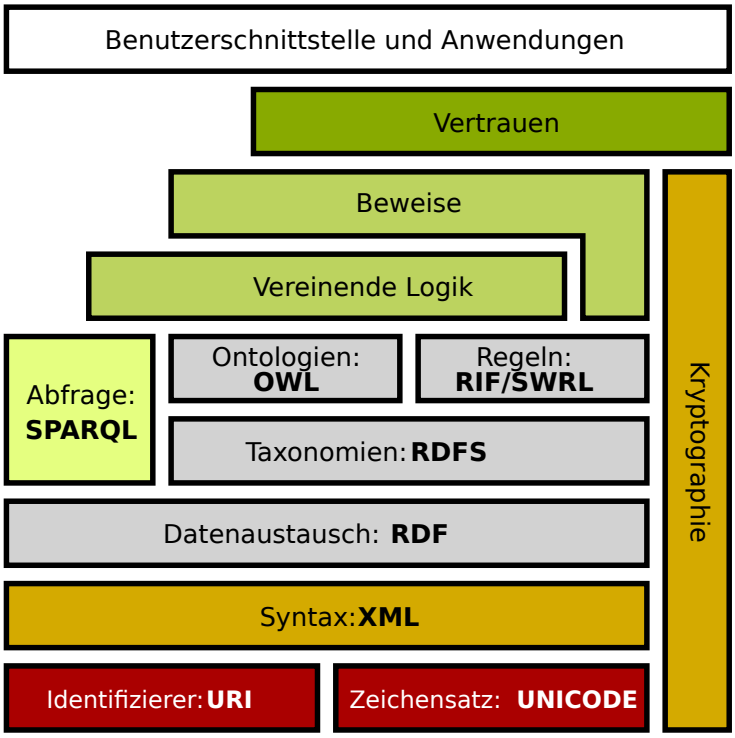

Abbildung 2.1: Der Semantic Web Stack.

enthält jedoch auch erweiterte Funktionen für die Konstruktion komplexerer Anfragemuster, die Verwendung zusätzlich hinzufügbarer Filterbedingungen und für die Formatierung der schlussendlichen Ausgabe.

Man kann also SPARQL als eine faktische Abfragesprache des semantischen Webs bezeichnen.

2.1.5 Ontologien

An ontology is an explicit specification of a conceptualization.

—Thomas R. Gruber

Ontologien sind (im informatischen Kontext) meist sprachlich gefasste und geordnete Darstellungen einer Menge von Begrifflichkeiten mit festen Beziehungen untereinander. Diese Beziehungen befassen sich stets auf einen bestimmten Gegenstandsbereich und werden dazu genutzt, Wissen in digitalisierter Form zwischen Anwendungen und Diensten auszutauschen. Dabei müssen bestimmte Interferenz- und Integritätsregeln eingehalten werden, also Regeln zu Schlussfolgerungen sowie zu der Gewährleistung ihrer Gültigkeit. Ontologien erfreuen sich seit der Idee des semantischen Webs einem stets wachsenden Bekanntheitsgrad, was dazu führt, dass sie als Teil der Wissensrepräsentation im Teilgebiet KI einen großen Einfluss haben. Dabei beschreibt die Idee des Semantic Web eine Erweiterung des vorhandenen Netzes, um Daten zwischen Rechnern einfacher austausch- und verwertbar zu machen und sie somit besser zu explizieren, anstatt sie unkonstruiert stehen zu lassen. Zum Sinne dieser Realisierung dienen Standards zur Veröffentlichung und Nutzung maschinenlesbarer Daten – insbesondere RDF.

2.1.6 Künstliche Intelligenz

Künstliche Intelligenz (KI) ist ein Teilgebiet der Informatik, welches sich mit der Automatisierung intelligenten Verhaltens befasst. Es spiegelt eine Möglichkeit wieder, bestimmte Entscheidungsstrukturen des Menschen nachzubilden und es somit Computern zu ermöglichen, komplexe Probleme selbstständig zu bearbeiten und zu lösen. Um diese Informationen weiter zu nutzen, können Methoden des Semantic Web genutzt werden, um sie als Tripel einfach maschinenlesbar darzustellen und für andere Computer verfügbar zu machen.

2.2 Informationssysteme

Ein Informationssystem ist ein System, welches durch die Bildung logischer Zusammenhänge eine Deckung der Informationsnachfrage zur Aufgabe hat. Es produziert, beschafft, verteilt und verarbeitet Daten durch eine Zusammenarbeit zwischen Mensch und Technik durch Aufgaben.

2.2.1 Krankenhausinformationssystem

Krankenhausinformationssysteme beschreiben die Gesamtheit aller Informationssysteme zur Produktion, Beschaffung, Verteilung und Verarbeitung von medizinischen und administrativen Daten im Krankenhaus. Dazu gehören verschiedene Formen der Datenbereitstellung sowie auch konventionelle Methoden der papierbasierten Dokumentation und der sprachlichen Kommunikation. Das grundsätzliche Ziel eines Krankenhausinformationssystems ist, die Kommunikation zwischen Mitarbeitern zu verbessern und den Ablauf in Krankenhäusern zu steuern, indem Mitarbeitern gezielt Zugriff auf für ihn relevantes Wissen aus der ihm zugeteilten Benutzerrolle gegeben wird, zum Beispiel über den gerade zu behandelnden Patienten.

2.2.2 Grundbegriffe

Information Information ist die Kenntnis über bestimmte Sachverhalte oder Vorgänge, im engeren Sinne über einzelne, konkrete, benennbare Objekte der realen und der virtuellen Welt.

Wissen Wissen ist die Kenntnis über den in einem Fachgebiet zu gegebener Zeit gegebenen Konsens, vor allem bezogen auf eine gültige Terminologie, erlaubte Interpretationen, bestehender Zusammenhänge und Gesetzmäßigkeiten sowie empfehlenswerter Methoden und Handlungen. Wissen ist also auch Information, aber in dem Fall nicht auf einzelne Objekte, sondern eine Menge von Objekten und deren Beziehungen untereinander bezogen, vgl. **?**.

Daten Daten sind Gebilde aus Zeichen oder kontinuierliche Funktionen, die durch bekannte oder unterstellte Beziehungen Informationen darstellen können. Sie stellen die Grundlage oder das Ergebnis eines Verarbeitungsschrittes dar.

2.3 SNIK-Projekt

Das SNIK ist ein vollendetes Projekt, welches Begriffe des Informationsmanagements sowie deren Beziehungen untereinander beschreibt. Es ist in der Lage, diese Menge an Informationen in einer Ontologie darzustellen und diese sowie maschinen- als auch menschenlesbar auszugeben. Dabei nutzt es ein dem Semantic Web Stack ähniches Modell(Abb. 2.2), welches Anwendungen wie

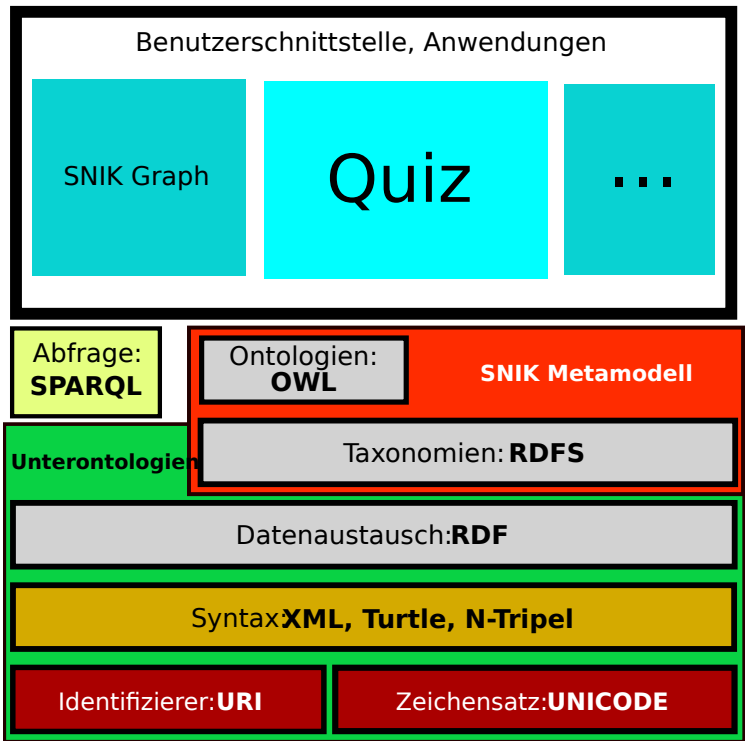

Abbildung 2.2: Das semantische Modell von SNIK.

dem SNIK Graph oder einem Multiple-Choice-Quiz ermöglicht, Informationen abzufragen und diese Endnutzern in aufbereiteter Form zur Verfügung zu stellen. Dabei konzentriert sich das SNIK-Projekt hauptsächlich auf die Daten der medizinischen Lehre, um diese für Krankenhausinformationssysteme zur Verfügung zu stellen und die Arbeit für medizinisches Personal zu erleichtern.

2.3.1 JavaScript

JavaScript ist eine leistungsstarke und flexible Skriptsprache, die von Entwicklern zunehmend zur Erstellung interaktiver Webanwendungen eingesetzt wird. Die Sprache ist dynamisch, schwach typisiert und verfügt über eine Vielzahl an für Webanwendungen essentiellen Funktionen. Sie lässt sich außerdem hervorragend mit anderen Websprachen wie CSS und HTML implementieren und besitzt die Fähigkeit, mit diesen zu interagieren.
JavaScript wird bei der Quiz-Webanwendung des SNIK-Projekts genutzt, um Fragen und Antworten anzuzeigen und diese dem Nutzer interaktiv zum Beantworten bereitzustellen.

2.4 Multiple-Choice-Quiz

Ein Multiple-Choice-Quiz ist eine Ansammlung an Fragen, die allesamt aus einem Stamm (der Frage), einem Schlüssel (der richtigen Antwort) und mehreren Distraktoren (eine Menge von inkorrekten, aber plausibel klingenden Antwortmöglichkeiten), vgl.
Ein Beispiel für eine solche Frage wäre hier aufgeführt:
Q: In einer Hand halte ich ein rotes, ein grünes und ein blaues Blatt Papier. Ich lege nun das rote und das blaue Blatt aus meiner Hand. Welche Farbe hat das Blatt, welches sich immer noch in meiner Hand befindet?

- Rot
- Grün
- Blau

Hierbei ist die Antwort „Grün“ der Schlüssel. „Rot“ und „Blau“ sind *Distraktoren*, die von dem Schlüssel ablenken sollen.
Der Sinn bei einem Multiple-Choice-Quiz besteht darin, den Befragten für richtige Antworten mit Punkten zu belohnen. Antwortet er falsch, bleibt seine Punktzahl entweder gleich oder Punkte werden abgezogen. Die Anzahl der vergebenen Punkte kann dabei von unterschiedlichen Faktoren abhängig gemacht werden, beispielsweise an der Zeit, die zum Lösen der Frage benötigt wurde.

2.4.1 Distraktoren

Distraktoren beschreiben im Kontext eines Quiz falsche Antwortmöglichkeiten, die den Befragten von der richtigen Antwort ablenken sollen. Durch sie soll vermieden werden, dass falschen Antworten durch Logik einfach von der richtigen Antwort unterschieden werden können. Der Sinn von Distraktoren ist es also, für den Befragten plausibel zu erscheinen. Daraus resultierend kann eine schlechte Auswahl von Distraktoren Fragen um einiges einfacher gestalten. Nehmen wir dazu einmal als Beispiel unsere Frage im Abschnitt 2.4: die Distraktoren klingen plausibel, da sie im Stamm der Frage erwähnt werden. Würde man hier allerdings andere Farben zur Auswahl stellen, die nicht im Kontext der Frage stehen und somit direkt ausgeschlossen werden können, zum Beispiel „Gelb“, könnte der Befragte diese logisch ausschließen.

Kapitel 3

Lösungsansatz

Dieses Kapitel soll dazu dienen, den Lösungsweg der genannten Probleme genauer zu beschreiben. Eine Erläuterung der Ausführung dieser erfolgt im nächsten Kapitel.

3.1 Lösungsansatz zu Aufgabe A1.1

Um den bestehenden Prototyp zu verstehen und diesen zu erweitern bzw. zu verbessern, müssen wir dazu die Methodik zur Generierung der Fragen analysieren. Diese Generierung hat die Zielsetzung, menschenlesbare Quizfragen zu erzeugen. Dazu wird als Erstes im existierenden Prototyp eine Abfrage an die Ontologie gesendet, um die nötigen Daten für die Generierung der Frage zu erhalten. Solche Abfragen werden in der SNIK-Ontologie durch SPARQL-Queries (Abschnitt 2.1.4) gehandhabt. Gesendete Queries werden von einem Endpunkt auf dem SNIK-Server angenommen, welcher diese dann als Abfrage an die Ontologie weiterleitet und diese umsetzt. Diese Rückgabewerte werden dann in dem Quiz genutzt und mithilfe von einfachem JavaScript-Code im

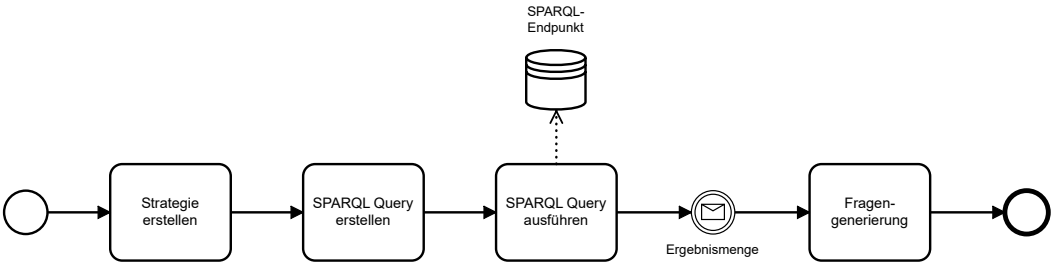

Abbildung 3.1: Die SPARQL-Abfrage im Ablauf.

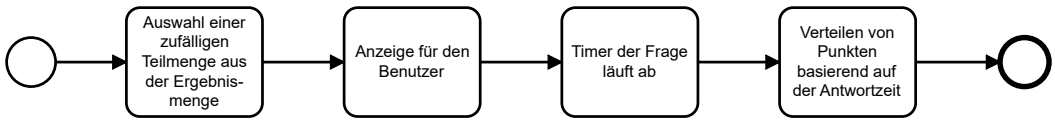

Abbildung 3.2: Ausgabe der Fragen an den Nutzer.

Browser des Benutzers in einfacher Wortform angezeigt. Um Ressourcen zu schonen, wird jedoch nicht bei jedem Aufruf des Quizzes eine einzelne Frage generiert - das Quiz greift auf eine große Liste aus vorher generierten Fragen zu. Diese sind im JavaScript-Code, welcher durch den Browser angezeigt wird, fest implementiert.
Um den Inhalt dieser Fragen zu erhalten, muss eine Abfrage an den SPARQL-Endpunkt gesendet werden, der auf die Ontologie zugreift und mithilfe der Abfrage bestimmte Informationen an das Programm ausgibt. Die erhaltenen Daten werden nun von dem Programm spezifisch für den Verbindungstyp verarbeitet, wie in Abschnitt 1.1.1 schon erkärt wurde. Bei der spezifischen Verarbeitung gibt es eine wichtige Unterscheidung: die Methodik der Generierung des Stammes und die Methodik der Generierung der Distraktoren. Während ersteres für jeden Fragetyp unterschiedlich ist, werden Distraktoren stets gleich generiert. Hierzu müssen also im nächsten Abschnitt die Generierung der verschiedenen Fragen und Antworten analysiert werden.

3.2 Lösungsansatz zu Aufgabe A2.1

Um nun diesen vorhandenen Prototypen zu verbessern, müssen die Strategien der Queries überarbeitet werden, die die generellen Fragen und die richtige Antwort generieren. Hierzu müssen in Abschnitt 1.1.2 beschriebene Probleme gelöst werden. Weiterhin sollen weitere Typen von Abfragen entwickelt werden, die neben den bestehenden Abfragen genutzt werden können. Die SNIK-Ontologie bietet eine Vielzahl von Verbindungstypen, welches sich das Quiz hierbei zunutze machen kann.

3.3 Lösungsansatz zu Aufgabe A2.2

Um die einzelnen Fragen im Sinne der Nutzerfreundlichkeit zu verbessern, müssen die neuen Query-Strategien, welche den eben genannten neuen Konzepten

zur Fragenbildung folgen, mit Regeln belegt werden. Diese sollten Aspekte wie beispielsweise die maximale Anzahl an Zeichen für eine Frage bestimmen, um sie so verständlich wie nur möglich zu gestalten. Die Fragen sollten in normaler Textform stehen, sodass ein Nutzer sie ohne jegliche Kenntnisse einer Ontologie einfach verstehen und beantworten kann. Auch in Erwägung zu ziehen währen weitere Faktoren wie die Sprache der Fragen, um Menschen ohne Englischkenntnissen das Verstehen der Fragen zu vereinfachen.

3.4 Lösungsansatz zu Aufgabe A2.3

Zur letztendlichen Verarbeitung der Queries und zum Anzeigen der Fragen sollte eine Lösung entwickelt werden, die die Fragen aus Queries generiert und anzeigt. Sie sollte in der Lage sein, dem Nutzer nur durch Auswahl des Fragetyps Fragen auszugeben. Darüber hinaus sollte die Lösung weitere Funktionen wie das Öffnen der Fragen in externen Programmen wie QuizMaster berücksichtigen. Auch müssen weiterhin die Regeln der Nutzerfreundlichkeit aus Abschnitt 2.2 umgesetzt werden.

Kapitel 4

Ausführung der Lösung

4.1 Lösung für Aufgabe A1.1

4.1.1 Untersuchen der bestehenden Queries

Wie in Abschnitt 1.1.1 bereits beschrieben wurde, gibt es bei dem existierenden Prototypen 2 Typen von Strategien, die zur Generierung von Fragen genutzt werden:

- die DEFINITION-Query (A defines B),
- und die SUBJECT-Query (A is related to B in way C)

Die zwei vorhandenen Strategien generieren auf Basis dieser Queries Fragen. Um die Verständlichkeit zu erhöhen, werde ich die erste Strategie umfangreicher erklären und umfangreicher auf bestimmte Operationen und Methoden eingehen.

DEFINITION-Query

Query:

```
SELECT SAMPLE(replace(str(?def),str(?cl),"X","i") as ?def)
SAMPLE(str(?cl) as ?cl)
SAMPLE(str(?a1l) as ?a1l)
SAMPLE(str(?a2l) as ?a2l)
SAMPLE(str(?a3l) as ?a3l)
# Gebe ?def, ?cl, ?a1l, ?a2l, ?a3l aus
(
```

```
?class a owl:Class.
# Sei ?class eine Klasse
?class rdfs:label ?cl.
# Sei ?cl der woertliche Bezeichner von ?class
FILTER(LANGMATCHES(LANG(?cl),"en"))
# Sei ?cl in englischer Sprache

?class skos:definition ?def.
# Sei ?def die Definition von ?class
FILTER(STRLEN(?def)>10&&STRLEN(?def)<600).
FILTER(LANGMATCHES(LANG(?def),"en"))
# Sei ?def in englischer Sprache und zwischen 10 und 600
# Zeichen lang

?class (!meta:subTopClass){1,2} ?a1,?a2,?a3.
# Seien ?a1, ?a2, ?a3 in direkter oder indirekter
# Nachbarschaft von ?class mit Pfadlaenge 1-2

# Pfad darf hierbei nicht ueber die meta:subTopClass fuehren

owl:Class ^a ?a1,?a2,?a3.FILTER(
?class!=?a1&&?class!=?a2&&?class!=?a3&&?a1<?a2&&?a2<?a3)
# Seien ?a1, ?a2, ?a3 Klassen

# Seien ?a1, ?a2, ?a3 paarweise verschieden und
# ungleich ?class

?a1 rdfs:label ?a1l.FILTER(LANGMATCHES(LANG(?a1l),"en"))
?a2 rdfs:label ?a2l.FILTER(LANGMATCHES(LANG(?a2l),"en"))
?a3 rdfs:label ?a3l.FILTER(LANGMATCHES(LANG(?a3l),"en"))
# Seien ?a1l, ?a2l, ?a3l die woertlichen Bezeichnungen von
# ?a1, ?a2, ?a3 in englischer Sprache
) GROUP BY ?class limit 3
```

Die DEFINITION-Query dient als Ausführung der gleichnamigen Strategie der Generierung für Fragen, bei denen ein Parameter gegeben ist und ein anderer

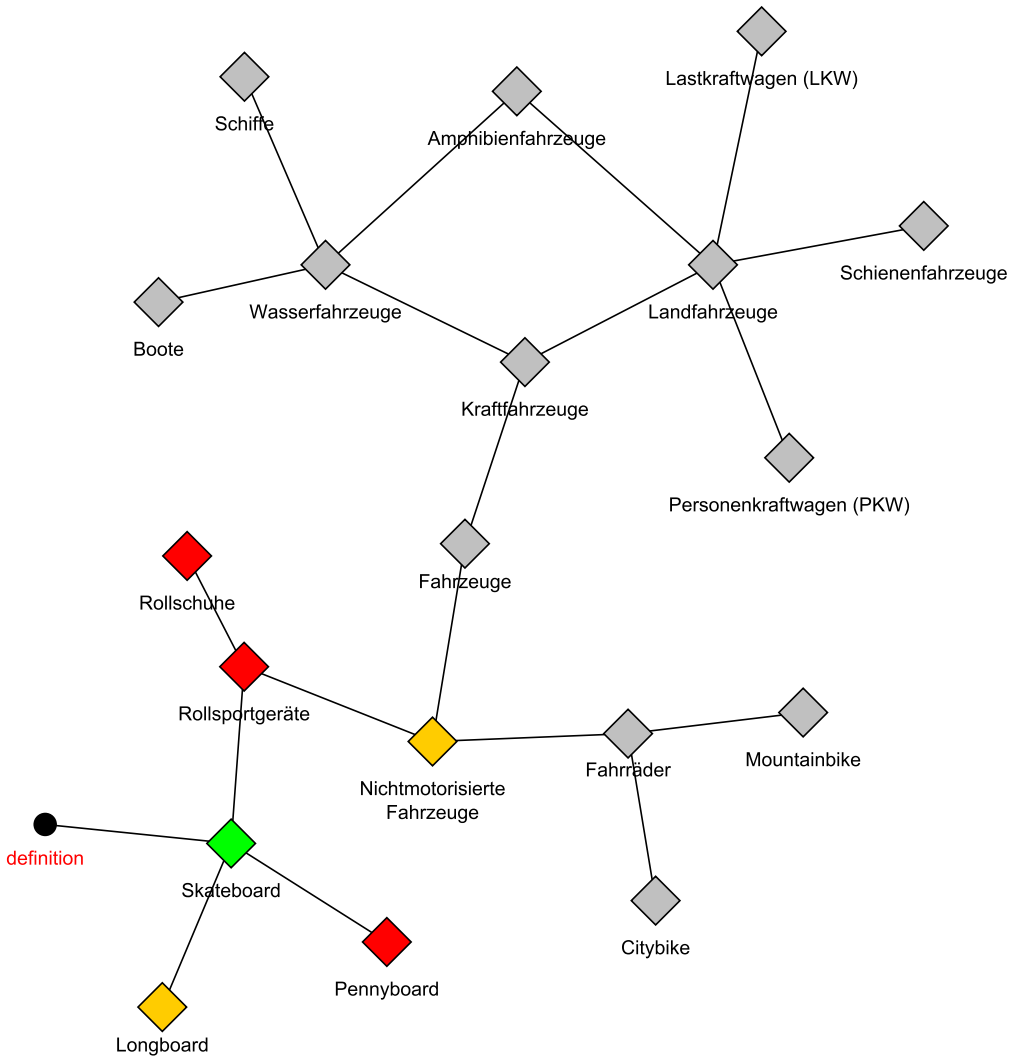

Abbildung 4.1: Fragengenerierung mithilfe der DEFINITION-Query.

durch Antwortmöglichkeiten ausgewählt werden muss. Dabei ist im Speziellen hier eine Definition einer Klasse x vorgegeben, zu der diese Klasse x gesucht wird. Die Ergebnismenge von ?class wird nun auf Klassen beschränkt. Dadurch werden alle Attribute der Ontologie, die ein Element von owl:class sind, als potentielle Werte für ?class angesehen. Einer zweiten Variable, ?cl, wird nun der Klarname der Klasse ?class in der gewünschten Sprache zugewiesen.
Hierbei ist ?class ist eine Kategorisierung, die eine Menge an Tripeln verbindet. Dabei sieht sich ?class einem Primärschlüssel in SQL ähnlich: ein Identifier, der eindeutig und einzigartig ist. Während die Klasse selbst neben einem Identifikationsschlüssel keine Informationen beinhaltet, finden sich diese Informationen in den Unterelementen wieder. Dazu können beispielsweise Informationen wie Definitionen oder Ober- bzw. Unterklassen stecken.

Um das Beispiel „Skateboard“ verständlich zu halten, bezieht es sich anstelle der SNIK-Ontologie auf Wikidata. Wikidata ist eine Ontologie, die einen Großteil der Informationen der Wissenssammlung Wikipedia zusammenfasst.

„Skateboard“ ist hier als eigene Klasse mit dem Identifier „Q15783“ gelistet. Sie besitzt unter anderem die Eigenschaft „label“, die mehrsprachig vorliegt. Die Eigenschaft „label“ beinhaltet den Klarnamen der ?class, also „Skateboard“, in verschiedenen Sprachen. Um nun die gewünschte Sprache auszuwählen, wird ein FILTER genutzt. Dieser schließt hier alle Attribute der Eigenschaft label für ?class aus, die nicht in englischer Sprache sind. Dieses Label wird in der Variable ?cl gespeichert.
Im nächsten Schritt wird auf ein weiteres Unterelement von ?class zugegriffen: der Definition. Diese wird der Variable ?def zugewiesen und liegt erneut mehrsprachig vor. Hier wird erneut auf FILTER zugegriffen, um nur englischsprachige Elemente anzuzeigen. Weiterhin wird die Anzahl der Zeichen so beschränkt, dass nur Ergebnisse zwischen 10 und 600 Zeichen angezeigt werden.
Im nächsten Schritt werden nun die Distraktoren generiert. Diese heißen im Beispiel ?a1,?a2 und ?a3 und sind ebenfalls Klassen, die jedoch nicht mit ?class („Skateboard“) zusammenhängen dürfen. Nun wird die Limitierung !subTopClass(1,2) gesetzt, die beschränkt, welche Entfernung die Elemente von ?class besitzen dürfen. Diese Entfernung darf nicht über Oberelemente gemessen werden, sondern ausschließlich über Unterklassen.

```
?def: "Ein␣X,␣gelegentlich␣verdeutscht␣auch␣Rollbrett␣genannt,
ist␣ein␣Brett␣(Deck)␣mit␣zwei␣Achsen␣(Trucks)␣[...]"
?cl:  "Skateboard"
?a1l: "Pennyboard"
?a2l: "Rollschuhe"
?a3l: "Rollsportgeraete"
```

Tabelle 4.1: Beispielausgabe der DEFINITION-Query in Datenform.

Was wird hiervon definiert? „Ein X, gelegentlich verdeutscht auch Rollbrett genannt, ist ein Brett (Deck) mit zwei Achsen (Trucks) [...]“	
Rollschuhe	Rollsportgeräte
Skateboard	Pennyboard

Tabelle 4.2: Fragestellung der DEFINITION-Query, basierend auf der generierten Fragestellung.

Im letzten Schritt werden die drei Variablen ?a1l, ?a2l und ?a3l mit den Labels von ?a1, ?a2, ?a3 versehen. Nun werden die Variablen ?cl, ?a1l, ?a2l, ?a3l sowie ?def ausgegeben, wobei bei ?def noch jedes Vorkommen von ?cl durch X ersetzt wird.
Hierbei ist „Skateboard“ die richtige Antwortmöglichkeit.

CONTAINS-Query

```
SELECT DISTINCT ?p
?o
(min(?c1) as ?c1)
(min(?c2) as ?c2)
(min(?c3) as ?c3)
?c4
# Gebe ?p, ?o, ?c1, ?c2, ?c3, ?c4 aus
from <http://www.snik.eu/ontology/bb>
from <http://www.snik.eu/ontology/derived>
# Nutzung mehrerer Ontologien
{
 ?c1 meta:closeNeighbour ?c2,?c3,?c4.
# Seien ?c2, ?c3, ?c4 direkte Nachbarn (Pfadlaenge 1) mit ?c1
```

```
  owl:Class ^a ?c1,?c2,?c3,?c4,?o.

  filter(?o!=?c1&&?o!=?c2&&?o!=?c3&&?o!=?c4).
  filter(?c1!=?c2&&?c1!=?c3&&?c1!=?c4&&?c1<?c2&&?c2<?c3&&?c3<?c4).
# Stelle sicher, dass die Antworten ungleich einander sind


  graph <http://www.snik.eu/ontology/bb>
  {?c1 ?p ?o.
# Sei ?c1 in Beziehung ?p mit ?o (richtige Antwort)
  FILTER(?p!=meta:isAssociatedWith).
# Entferne alle Beziehungen, die keine Beziehung verdeutlichen
  MINUS {?c2 ?p ?o}
  MINUS {?c3 ?p ?o}
  MINUS {?c4 ?p ?o}
# Entferne alle Antwortmoeglichkeiten, die auch richtig waeren,
# da sie die selbe Beziehung wie die richtige Antwort zum Praedikat
# besitzen wuerden
  }

}
```

Die CONTAINS-Query wird als Strategie für Fragen genutzt, bei denen zwei Parameter gegeben sind und ein dritter zur Auswahl steht. Sie besitzt ihren Namen, da hier nur Beziehungen betrachtet werden, die sich untereinander beinhalten (contain). Hier ist dabei im Besonderen eine Klasse X und die Beziehung Y dieser Klasse X zu einer anderen Klasse Z gegeben, wobei hier Klasse Z gesucht wird. Dabei verläuft die Fragengeneration ähnlich wie bei der DEFINITION-Query. Im ersten Schritt wird die direkte Nachbarschaft von ?c1, ?c2, ?c3 und ?c4 sichergestellt. Eine direkte Nachbarschaft im ontologischen Sinne wird dadurch sichergestellt, dass die Pfadlänge zwischen den einzelnen Elementen genau 1 beträgt. Es kann so keine Zwischenelemente geben, die zwischen den Klassen stehen.

Da nun die Generierung von vier untereinander benachbarten Antworten beendet ist, wird nun bei der ersten Antwort eine Beziehung zu einem Objekt hergestellt. Diese Beziehung ?p zu Objekt ?o wird im nächsten Schritt auch

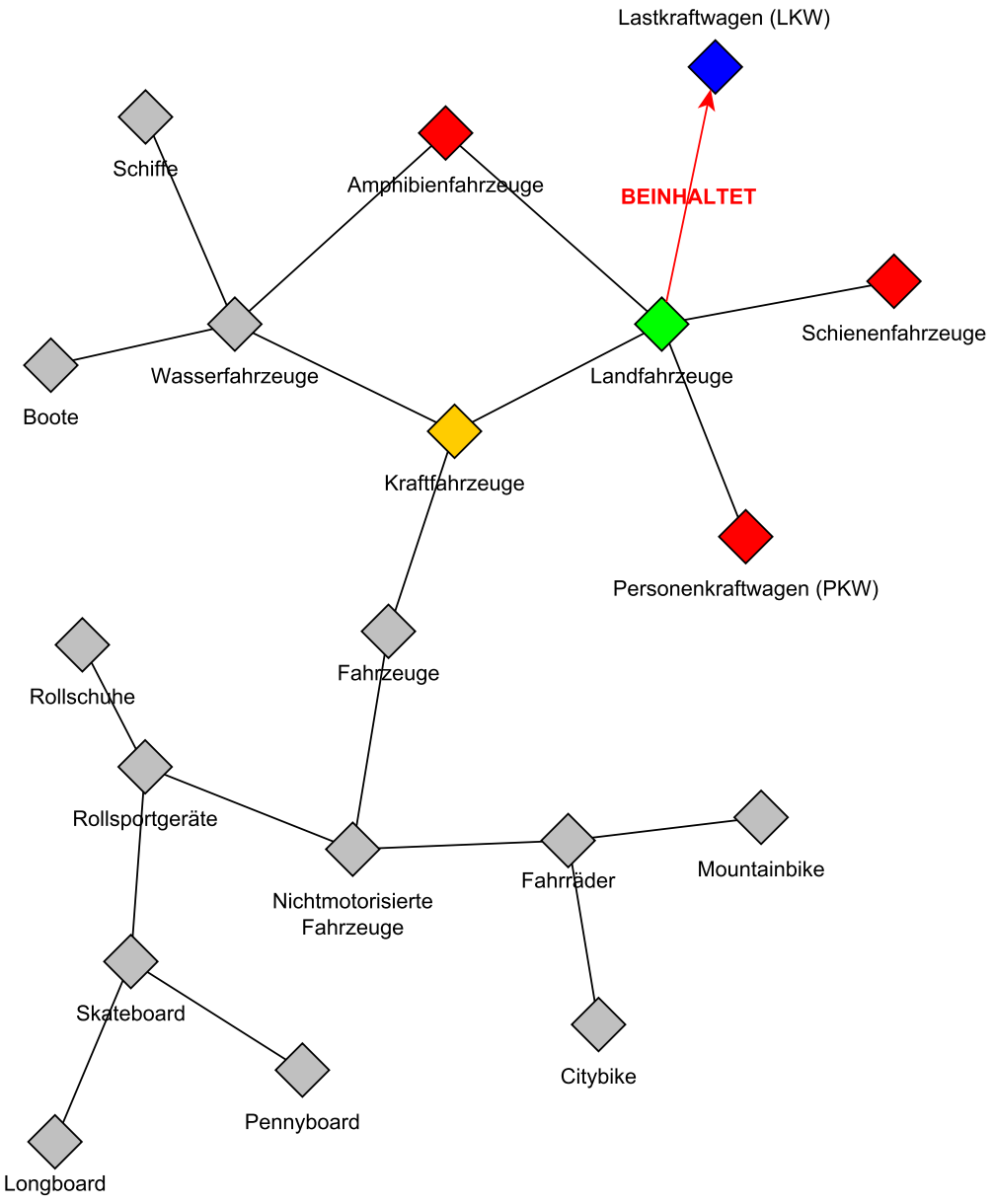

Abbildung 4.2: Fragengenerierung mithilfe der CONTAINS-Query.

Welche Klasse beinhaltet „Lastkraftwagen (LKW)“?	
Personenkraftwagen (PKW)	Landfahrzeuge
Amphibienfahrzeuge	Schienenfahrzeuge

Tabelle 4.3: Fragestellung der CONTAINS-Query.

zu den anderen 3 Antwortmöglichkeiten geprüft. Besteht diese Beziehung dort ebenfalls, wird die potentielle Antwortmöglichkeit entfernt. Stehen keine Antwortmöglichkeiten mehr zur Verfügung, wird das gesamte Anfangstripel übersprungen und die nächste Beziehung wird aufgebaut.
Am Ende werden die vier Antwortmöglichkeiten, das Objekt ?o und die Beziehung ?p ausgegeben.

4.2 Lösung für Aufgabe A2.1

4.2.1 Entwickeln neuer Strategien zur Generierung von Fragen

Da die Ontologie viele Arten von Eigenschaften bietet, die genutzt werden können, um daraus Fragen zu generieren, müssen zur Entwicklung neuer Strategien Eigenschaften gefunden werden, die sich zur sinnvollen Fragengenerierung anbieten. Hierzu eignet sich der Fakt, dass ein großer Teil der Elemente, welche in der Ontologie ein Label besitzen, auch eine Definition als Eigenschaft enthalten.

LABEL-DEFINITION-Query

```
SELECT
# Gebe korrektes Label (?correctlabel) und korrekte Definition
# (?correct) sowie 3 Distraktoren (?distractor1-3) aus
SAMPLE(str(?q)) as ?correctlabel
SAMPLE(str(?ans)) as ?correct
SAMPLE(str(?d1)) as ?distractor1
SAMPLE(str(?d2)) as ?distractor2
SAMPLE(str(?d3)) as ?distractor3
WHERE
{
```

```
  ?class a owl:Class.
# Sei ?class eine Klasse
  ?class rdfs:label ?q.
# Sei ?q die woertliche Umschreibung der Klasse ?class
  ?class skos:definition ?ans.
# Sei ?ans die Definition der woertlichen Umschreibung ?q
  ?class (!(meta:subTopClass|meta:chapter)){1,2}
         ?dist1,?dist2,?dist3.
# Seien ?dist1, ?dist2, ?dist3 Nachbarn der Klasse ?class
# mit Pfadlaenge 1-2 und keine Bestandteile der Bibliothek
# meta:chapter

# (meta:chapter beinhaltet nur ObjectProperty, welches
# fuer die Fragengenerierung ungeeignet ist und unwichtig ist)
  FILTER(?class!=?dist1&&?class!=?dist2&&?class!=?dist3
         &&?dist1<?dist2&&?dist2<?dist3).
# Sei die Klasse ?class ungleich der drei Distraktoren
  ?dist1 skos:definition ?d1.
  ?dist2 skos:definition ?d2.
  ?dist3 skos:definition ?d3.
# Seien ?d1, ?d2, ?d3 die woertlichen Definitionen der Klassen
# ?dist1-3
  FILTER (STRLEN(?ans)<225).
  FILTER (STRLEN(?d1)<225).
  FILTER (STRLEN(?d2)<225).
  FILTER (STRLEN(?d3)<225).
# Seien die Definitionen hoechstens 225 Zeichen lang
  FILTER(LANGMATCHES(LANG(?ans),"en")).
  FILTER(LANGMATCHES(LANG(?q),"en")).
  FILTER(LANGMATCHES(LANG(?d1),"en")).
  FILTER(LANGMATCHES(LANG(?d2),"en")).
  FILTER(LANGMATCHES(LANG(?d3),"en")).
# Seien die Definitionen in englischer Sprache
  FILTER ( 1 > <bif:rnd> (10, ?class) )
# Treffe beim Suchen eine zufaellige Auswahl der ?class
# beim Ausfuehren der Query -> sorgt manchmal fuer
```

Welche der Definitionen definiert den Begriff „Fahrrad“?	
Motorisiertes Zweirad	Nichtmotorisiertes Zweirad mit Pedalen
Dreirädriges Fahrzeug	Öffentliches Verkehrsmittel

Tabelle 4.4: Fragestellung der LABEL-DEFINITION-Query.

```
# wenige Antwortmoeglichkeiten
} GROUP BY ?q ORDER BY RAND()
```

Die LABEL-DEFINITION-Query ist ein weiterer Querytyp, bei dem ein Parameter gegeben ist und einer zur Auswahl steht. Bei dieser Query wird zuerst eine Klasse generiert, von der der wörtliche Name an eine Variable übertragen wird. Neben dem wörtlichen Namen wird noch die Definition abgezweigt. Nach der Abtretung der Werte werden nun Distraktoren generiert. Durch meta:subTopClass mit Startwert 1 und Endwert 2 werden im Umfeld zwischen 1 und 2 Schrittweiten entfernt von der Klasse nach geeigneten Klassen gesucht. Dabei besteht die Bedingung, dass diese Klassen eine Definition mit einer Gesamtzeichenanzahl unter 225 besitzen, welches sich als ein guter Kompromiss zwischen Lesbarkeit der Frage und generierter Anzahl der Fragen herausstellt. Filtert man hierbei nach einer zu kleinen Gesamtzeichenanzahl, so handelt es sich um einen corner-case und es werden wenige Klassen mit der Bedingung gefunden, während bei einer zu großen Anzahl von Zeichen die Frage unleserlich wird. Die Definition muss außerdem englischsprachig sein, um sie sprachlich homogen mit den restlichen Definitionen und der Frage zu gestalten. Schlussendlich wird die Auswahl der Klasse ?class randomisiert. Durch diesen Parameter durchläuft die Query die Ontologie nicht in einem vorgegebenen Muster, sondern sucht stattdessen an zufälligen Knotenpunkten. Das hat zur Folge, dass das Ergebnis beim Ausführen der Query im SPARQL-Endpunkt immer ein anderes ist. Hierdurch wird es möglich, neue Quizfragen im Hintergrund zu generieren, wodurch Live-Demonstrationen oder etwa Quizprogramme mit unendlich vielen Fragen möglich sind.

4.2.2 Verbessern bestehender Strategien zur Generierung von Fragen

Verbesserte DEFINITION-Query

Hinweis: Zur Vermeidung von unnötigem Text im Hauptteil befinden sich die verbesserten SPARQL-Queries samt Kommentaren im Anhang, da sie im Gesamten trotzdem die alten Queries beinhalten. Die DEFINITION-Query wurde anfangs vom Prototyp übernommen und im Weitergang verbessert. Um Anzeigefehler wie ObjectProperty zu vermeiden, welche auch in der LABEL-DEFINITION-Query auftraten, wurde hier erneut ein Filter für meta:chapter hinzugefügt. Das begründet sich wie folgt, da das Element ObjectProperty erst später in die Ontologie eingeführt wurde, weshalb es in der originalen Query nicht gefiltert wurde. Weiterhin wurde für die Definition ein Zeichenlimit eingeführt, welches besagt, dass für die Anzahl der Charakter in der Definition $50<x<225$ gilt. Hierdurch soll unter anderem die Lesbarkeit der gesamten Frage verbessert werden, wie bereits in Abschnitt 4.2.1 beschrieben. Auch weiterhin eingefügt wurde der bif:rnd-Filter, der die Auswahl der Klasse bei jedem Durchlauf der Query randomisiert. Dadurch entstehen hier bei jedem Ausführen unterschiedliche Ergebnisse, was für Live-Queries mit Direktausführung wie in meinem Fall besonders geeignet ist. Will man eine größere Menge Fragen generieren, so ist der Filter ungeeignet, da durch den Erneutdurchlauf nach einer bestimmten Zeit alte Ergebnisse verworfen werden. Das Resultat ist hier eine Antwortmenge, die bei jeder Ausführung unabhängig von der Timeout-Zeit variiert, während sich ohne Randomisierung eine proportional zur Rechenzeit ansteigende Fragenmenge abzeichnet. Zuletzt wurde ein Fehler behoben, bei dem die korrekte Antwort nicht nach Sprache gefiltert wurde, was dazu führte, dass zu englischen Fragen beispielsweise deutsche Antworten herauskamen und außerdem die Fragenmenge deutlich reduziert wurde. Durch Verschachtelung der Queries konnte hier außerhalb gefiltert werden, was das Problem behob.

Verbesserte CONTAINS-Query

Da die CONTAINS-Query aus dem Prototyp im Original nicht funktional war, wurde sie hier vollständig überarbeitet. Hierbei wurde mit Verschachtelung gearbeitet, um die Mehrfachnennung von Klassen zu verhindern, d.h. um das mehrfache Auftreten einer einzelnen Klasse in mehreren Fragen mit gleicher

Beziehung, aber unterschiedlichen Distraktoren zu vermeiden. Zuerst wird hierbei eine Variable als Klasse A definiert, zu welcher ein weiteres Objekt O mit Beziehung B existiert. Nun sei die Auswahl der Klasse A erneut zufällig. Aus der Menge der Klassen A wird nun der niedrigste Wert für A gewählt. Im nächsten Schritt wird ein Distraktor definiert, der die Pfadlänge 1-2 zur Klasse A aufweist und ungleich A ist. Nun wird dieser Schritt verschachtelt zweifach mit den anderen beiden Distraktoren wiederholt, wobei bei diesen auch die Ungleichheit zu den bestehenden Distraktoren hergestellt wird. Im letzten Schritt werden die Distraktoren nach Sprache gefiltert, um englische Fragen und Antworten zu erhalten. Die wörtlichen Umschreibungen der Distraktoren werden nun auf Variablen beschrieben und als einzelne Samples (Beispiele) zur Verhinderung der Wiederholung ausgegeben. Die Fragen werden nun nach dem Tripel A,O,B gruppiert, um für jedes genannte Tripel nur eine einzige Frage zu erhalten. Dadurch wird ausgeschlossen, das mit gleichen Fragen und unterschiedlichen Distraktoren bei derselben Query immer gleiche Ergebnisse generiert werden.

Anders als bei der bisherigen Query können hier neben „A is responsible for (...)“ auch andere Beziehungen B generiert werden, die am Ende verwendet werden, beispielsweise „A is involved in (...)“. Eine mögliche Fragestellung hier wäre also:

What is a Doctor involved in?

Weiterhin wäre es möglich, andere Beziehungen wie *A approves B* oder *A uses B* zu nutzen.

4.3 Lösung für Aufgabe A2.2

4.3.1 Aufstellen von Regeln zur Nutzerfreundlichkeit

Begrenzung der Zeichenanzahl

Um die Lesbarkeit der Fragen zu verbessern, wurden in den oben beschriebenen Queries Filter eingebaut, um die Zeichenanzahl bei besonders charakterlastigen Elementen wie beispielsweise Definitionen zu minimieren. Diese wurde auf 225 Zeichen pro Definition begrenzt. Geht man nun von einer durchschnittlichen Zeichenzahl von 6 Zeichen pro Wort aus, und zieht man dabei in Betracht, dass nach jedem Wort meist genau ein Zeichen ist (Leerzeichen bzw. Punkt), so

kommt man auf 7 Zeichen pro Wort. Daraus ergibt sich eine durchschnittliche Wortzahl von 32 Wörtern pro Definition.

Handling der Fragenausgabe

Da die Fragengenerierung über einen Endpunkt bei IMISE geschieht, müssen die Fragen durch einen Zwischenweg zum Endnutzer gelangen, ohne dass dieser Queries eingeben muss. Dazu muss ein Programm entwickelt werden, welches durch einfache Eingabe der Querytypen Fragen ausgibt, die dann in einem Quizprogramm weiter genutzt werden können.

4.4 Lösung für Aufgabe A2.3

4.4.1 Entwickeln einer plattformunabhängigen Lösung zur Anzeige der Fragen

Einführung

Um eine plattformunabhängige Lösung zu erhalten, muss auf eine Programmiersprache zurückgegriffen werden, die logisch plattformunabhängig ist. Während Programmiersprachen wie Delphi hier ausscheiden, bietet sich hier Python an, da es sowohl als Web- als auch als normale Programmiersprache einfach nutzbar ist. Es besitzt eine Vielzahl frei verfügbarer Pakete und ist von jedem kostenlos nutzbar, weswegen ich mich für Python 3 zur Umsetzung eines Demoprogrammes entschieden habe.

SPARQLWrapper

SPARQLWrapper (https://github.com/RDFLib/sparqlwrapper) ist ein einfacher Python-Wrapper, der sich zur simplen Ausführung von SPARQL-Queries auf öffentlich verfügbaren Endpunkten eignet. Es wurde unter anderem von Iman Herman auf Basis eines ähnlichen JavaScript-Wrappers entwickelt. SPARQLWrapper bietet die Möglichkeit, mithilfe von HTTP-GET Queryergebnisse auszugeben und wandelt diese direkt in CSV-Format um, was wiederum einfach weiterverwendet werden kann. Hierbei können anders als bei ähnlichen Paketen direkt Argumente mitgesendet werden, die direkt vom Endpunkt mit beachtet werden, was in diesem Fall essentiell ist.

py2exe

py2exe (https://github.com/py2exe/py2exe) ist ein für Python verfügbares Paket, welches Python-Dateien in ausführbare Dateien für Windows konvertiert. Hierdurch müssen weder Python noch sonstige Pakete installiert werden, um Python-Code auf Windows-PCs auszuführen.

Aufbau

Das Demoprogramm selbst ist eine Kommandozeilenapplikation, die sich mit wenig Eigenaufwand nutzen lässt. Nach Eingabe des gewünschten Querytyps und der Anzeigeform zeigt das Programm nach kurzer Verarbeitungszeit die Fragen entweder auf der Kommandozeile oder in einem externen Quizprogramm an. Hierbei müssen dank der Nutzung von py2exe keine Python-Bibliotheken oder Python selbst installiert werden. Zu Demozwecken werden bei dem Programm von jedem Fragetyp 3 Fragen ausgegeben, wobei die in dieser Arbeit entwickelten Queries sowie zusätzlicher unterstützender Code genutzt werden. Zur Ausführung des Programms ist zwingend Internet erforderlich, um sich mit dem IMISE-Endpunkt zu verbinden.

4.4.2 Funktionsweise

Da dies nicht Bestandteil dieser Arbeit ist, ist die Beschreibung der Funktionsweise des Programms absichtlich etwas gekürzt. Das Programm verbindet sich nach Initialisierung mit dem Endpunkt der IMISE und initialisiert genutzte Bibliotheken. Anschließend wird eine Nutzereingabe veranlasst, in der durch Eingabe von Kürzeln der Querytyp sowie die gewünschte Ausgabeform selektiert wird. Hierzu werden rekursive Funktionen mit der Abbruchbedingung einer korrekten Nutzereingabe verwendet. Ist die Eingabe vollständig erfasst, beginnt das Programm mit der Ausführung der Query. Hier wird je nach Querytyp ein verschiedenes Timeout gesetzt, nachdem die Ergebnisse der Query angefordert werden. In Folge dessen werden nach Erhalt der Daten die Daten formatiert und die Werte auf einen Array übertragen. Weiterhin werden die Werte des Array genutzt, um daraus die Frage und die 4 Antworten zu generieren. Hier werden beispielsweise für die CONTAINS-Query erst einmal alle Beziehungen verworfen, die nicht mit ïs"beginnen. Nun wird der String für den Fragenkopf beispielhaft wie folgt zusammengesetzt:

Wofür ist ein Doktor verantwortlich? Der String wird an der Stelle des 1. (0.) Element des Array eingesetzt. Die weiteren 4 Elemente des Array bilden nun die Antwortmöglichkeiten, wobei die richtige Antwort im Demoprogramm stets an oberster Stelle steht.
Als letzter Schritt wird aufgrund der besseren Lesbarkeit eine leere Zeile im Array eingefügt. Nun wird in der Schleife die nächste Frage behandelt. Am Ende werden die einzelnen Zeilen des Array ausgegeben und das Programm beendet sich.

Kapitel 5

Ergebnis

In der Besonderen Lernleistung wurde die automatische Erstellung von Quizfragen mithilfe einer Ontologie aufgeschlüsselt und dargestellt. Hierbei wurde mit der Querysprache SPARQL und der SNIK-Ontologie gearbeitet, um Fragen unter Nutzung der Daten der Ontologie zu generieren. Genutzt wurden hierbei unter anderem Eigenschaften wie etwa die Definition oder die Bezeichung einzelner Objekte sowie die Beziehung von Objekten untereinander. Die Rohdaten wurden mithilfe eines Python-Demoprogrammes verarbeitet, sodass am Ende reguläre Quizfragen als Ergebnis erreicht wurden. Hierbei wurde die öffentlich verfügbare Python-Bibliothek SPARQLWrapper mit Python 3.7 zur Verbindung zum Endpunkt genutzt.
Die am Anfang der Arbeit gestellten Ziele wurden bei der Ausführung zum größten Teil erfüllt. Als Ausblick hätte man beispielsweise eine GUI zu dem Python-Programm entwerfen können, welche Nutzern die Nutzung erleichtert. Auch eine Implementierung einer Übersetzungsfunktion wäre in Betracht zu ziehen, da ich bei der Implementierung dieser leider gescheitert bin, da sowohl Google Übersetzer als auch DeepL ausschließlich kostenpflichtige APIs für Übersetzer bieten.

Die entwickelten Queries und das Programm stehen unter
https://github.com/cubexy/sparql-converter/ zum Download bereit und können durch Klick auf den grünen Download-Button und "Download ZIP"heruntergeladen werden.

Kapitel 6

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegeben Hilfsmittel verwendet habe. Sämtliche wissenschaftlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Leipzig, den 19.01.2021

Anhang

Verbesserte DEFINITION-Query

```
SELECT
?corrAns ?cDef ?d1 ?d2 ?d3 ?class
{
?class rdfs:label ?corrAns.
FILTER(LANGMATCHES(LANG(?corrAns),"en"))
# Subquery, da innerer Filter nicht funktional
# Sei ?corrAns die woertliche Umschreibung von ?class
{
SELECT
?class
SAMPLE(str(?AnswDefinition)) as ?cDef
SAMPLE(str(?dist1l)) as ?d1
SAMPLE(str(?dist2l)) as ?d2
SAMPLE(str(?dist3l)) as ?d3
# Waehle ein SAMPLE(Beispiel) fuer jede Definition, um Repetition
# zu vermeiden
WHERE {
# Sei ?class eine Klasse
 ?class a owl:Class.
# Sei ?AnswDefinition die woertliche Definition von ?class
 ?class skos:definition ?AnswDefinition.
# Filter for english definitions only.
 FILTER(LANGMATCHES(LANG(?AnswDefinition),"en"))

 # Seien ?dist1-2 Bestandteile von meta:chapter und im nahen
 # Umfeld von ?class
 ?class (!(meta:subTopClass|meta:chapter)){1,2}
 ?dist1,?dist2,?dist3.
 # Seien ?dist1-3 Distraktoren
 owl:Class ^a ?dist1,?dist2,?dist3.
 # Filtern, um Aehnlichkeiten zwischen den Klassen zu verhindern
 FILTER(?class!=?dist1&&?class!=?dist2&&?class!=?dist3&&
        ?dist1<?dist2&&?dist2<?dist3)

 # Woertliche Umschreibungen, pruefe englische Sprache
 ?dist1 rdfs:label ?dist1l.
 FILTER(LANGMATCHES(LANG(?dist1l),"en"))

 ?dist2 rdfs:label ?dist2l.
 FILTER(LANGMATCHES(LANG(?dist2l),"en"))

 ?dist3 rdfs:label ?dist3l.
 FILTER(LANGMATCHES(LANG(?dist3l),"en"))

# Begrenze Laenge der Definition, um
# Lesbarkeit zu verbessern
 FILTER (STRLEN(?AnswDefinition)<225
         && STRLEN(?AnswDefinition)>50).
   # Randomisiere Auswahl
 FILTER ( 1 > <bif:rnd> (10, ?class) )
 FILTER (UCASE(SUBSTR(str(?AnswDefinition), 1, 1)))
}ORDER BY RAND() LIMIT 100
# Limitiere mit Anzahl der benoetigten Fragen
}
}GROUP BY ?class ORDER BY RAND()
```

Verbesserte CONTAINS-Query

```
SELECT DISTINCT
SAMPLE(str(?corrAns)) as ?corrAns
SAMPLE(str(?responsibleFor)) as ?responsibleFor
SAMPLE(str(?responsibility)) as ?responsibility
SAMPLE(str(?distractor1)) as ?distractor1
SAMPLE(str(?distractor2)) as ?distractor2
SAMPLE(str(?distractor3)) as ?distractor3
# Ausgabe der Antworten als Literale, um ungewollte
# Nebenbezeichnungen wie Sprachinformationen zu vermeiden
{
```

```
  ?c1 rdfs:label ?corrAns.
  FILTER(LANGMATCHES(LANG(?corrAns),"en"))
  ?p rdfs:label ?responsibleFor.
  FILTER(LANGMATCHES(LANG(?responsibleFor),"en"))
  ?y rdfs:label ?responsibility.
  FILTER(LANGMATCHES(LANG(?responsibility),"en"))
  ?c2 rdfs:label ?distractor1.
  FILTER(LANGMATCHES(LANG(?distractor1),"en"))
  ?c3 rdfs:label ?distractor2.
  FILTER(LANGMATCHES(LANG(?distractor2),"en"))
  ?c4 rdfs:label ?distractor3.
  FILTER(LANGMATCHES(LANG(?distractor3),"en"))
  # Seien ?c1,?p,?y,?c2,?c3,?c4 die woertlichen Bezeichungen
  # der zugehoerigen Klassen
  {
    SELECT DISTINCT ?c1 ?p ?y ?c2 ?c3 (SAMPLE(?c4) as ?c4)
    {
      meta:Role ^meta:subTopClass ?c4.
      ?c1 (!meta:subTopClass){1,2} ?c4.
      filter(?c1!=?c4&&?c2!=?c4&&?c3!=?c4&&?c1!=?y&&?c2!=?y&&?c3!=?y&&?c4!=?y)
      # Sei ?c4 eine Klasse mit Pfadlaenge 1-2 von ?c1, ungleich ?c1, ?c2, ?c3
      {
        SELECT DISTINCT ?c1 ?p ?y ?c2 (SAMPLE(?c3) as ?c3)
        {
          meta:Role ^meta:subTopClass ?c3.
          ?c1 (!meta:subTopClass){1,2} ?c3.
          filter(?c1!=?c3&&?c2!=?c3)
          # Sei ?c3 eine Klasse mit Pfadlaenge 1-2 von ?c1, ungleich ?c1, ?c2
          {
            SELECT DISTINCT
            ?c1
            ?p
            ?y
            (SAMPLE(?c2) as ?c2)
            {
              meta:Role ^meta:subTopClass ?c2.
              ?c1 (!meta:subTopClass){1,2}
              ?c2. filter(?c1!=?c2)
              # Sei ?c2 eine Klasse mit Pfadlaenge 1-2 von ?c1, ungleich ?c1
              {
                SELECT DISTINCT
                MIN(?c1) as ?c1 ?p ?y
                # Waehle einen minimalen Wert fuer ?c1 aus einer Gruppe von Variablen
                {
                  ?y a owl:Class.
                  # Sei ?y eine Klasse
                  ?c1 ?p ?y.
                  # Sei ?c1 ein Element, welches die Beziehung ?p zur Klasse ?y aufweist
                  ?c1 meta:subTopClass meta:Role.
                  # Sei ?c1 ein Element mit Nebenelement Rolle -> benoetigt, um Informationen wie
                  # "A ist ueber X zu B verwandt" zu erhalten
                  FILTER ( 1 > <bif:rnd> (10, ?y) )
                  # Randomisiere die Auswahl der Klassen
                }
              }
            } GROUP BY ?c1 ?p ?y
          }
        }
      }
    }
  }
}
GROUP BY ?corrAns ?responsibleFor ?responsibility ORDER BY RAND()
```