RHEINISCH-WESTFÄLISCHE TECHNISCHE
HOCHSCHULE AACHEN

MASTERS THESIS

Automatic Multiple Choice Question Generation for Slides using the Semantic Web

---

Author: Ainuddin Faizan
Matriculation Number: 351035

First Examiner: Prof. Dr. Sören Auer
Second Examiner: Prof. Dr. Jens Lehmann
Supervisor: Dr. Steffen Lohmann

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science (Media Informatics)

in the

Smart Data Analytics Group
Institute of Computer Science

December 11, 2017

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties. I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Abstract

Multiple Choice Questions are questions which consist of question text along with two or more choices. Some of these choices are correct answers to the question whereas the others are wrong choices, also called distractors. There has been significant research done in the area of automatic question generation where the source content was in the form of either textual information or data extracted from knowledge bases. This thesis aims to create an automatic multiple choice question and distractor generation system for slide content. A generic approach to produce questions and distractors of different varieties is discussed and is used to implement a service to generate certain varieties of multiple choice questions and distractors. Apart from working independently, this service is designed to work as a part of the SlideWiki architecture. Questions generated by the service were evaluated by fifty users based on several aspects including difficulty and relevance. Observed results suggest that the questions generated are sutiable for evaluating knowledge gained by slide content.

Acknowledgements

This thesis could not have been possible without the help and guidance of some people. Firstly, I would like to thank Prof. Dr. Sören Auer, Prof. Dr. Jens Lehmann and Dr. Steffen Lohmann for letting me pursue my master thesis at the Smart Data Analytics (SDA) group at the Computer Science Department in the University of Bonn.

I am grateful to my thesis supervisor Dr. Steffen Lohmann for his continued guidance and support from the very beginning right up to the finish of this thesis.

Apart from my supervisor, I sincerely thank Vinay Modi for sharing his immense knowledge and experience to help me achieve many milestones during the course of this work.

Thanks to Antje Schlaf for educating me about natural language processing and other SlideWiki team members for being a big help so that this thesis could be included in the SlideWiki platform.

I also want to acknowledge the help provided by many members of the Smart Data Analytics group. I gained a lot of insight and expertise from them.

I thank my friends for providing me with motivation and being there during the more difficult parts.

Finally, I thank my family for their unconditional support, without whom, this thesis would not have been possible.

Contents

List of Figures

List of Listings

List of Tables

List of Abbreviations

FOAF	**F**riend **O**f **A** **F**riend
IRI	**I**nternational **R**esource **I**dentifier
MCQ	**M**ultiple **C**hoice **Q**uestion
NER	**N**amed **E**ntity **R**ecognition
NLP	**N**atural **L**anguage **P**rocessing
RDF	**R**esource **D**escription **F**ramework
RDFS	**R**esource **D**escription **F**ramework **S**chema
SPARQL	**S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage
XML	**E**xtensible **M**arkup **L**anguage
YAGO	**Y**et **A**nother **G**reat **O**ntology

Dedicated to my parents

Chapter 1

Introduction

1.1 Motivation

E-learning is an ever-growing noble field, which aims to assist and expand traditional education using the various abilities of electronic devices and digital media. It has evolved into a big industry, with services ranging from sample quizzes for self assessment, to video lectures and interactive web sessions for emulating classroom environments. Our focus here is self assessment using questions. They help an individual judge his/her proficiency of the concerned field. A question can be of numerous types, each serving a separate purpose, testing a different aspect of the knowledge. Difficulty of the question also helps the user in determining his/her competence in the field.

1.2 Problem

Examiners have to take many factors into account while manually creating questions:

- Important topics to be covered
- Level of difficulty
- Relevance of a question to the topic
- Collection of question types

Keeping all of this in mind and coming up with questions can be a challenge, even to experts of the field. One might even need to do some sort of study on the topic to give variety to the questions. What if we use the vast reserves of knowledge that are available to us, to somehow generate such questions automatically? Automatic question generation is a growing area of interest in the field of semantic web and natural language processing. It has the objective of solving the aforementioned problem. Questions generated automatically would help take the burden off teachers, who would otherwise have to spend much more time and effort to manually create them. Some teachers might not usually consider having multiple choice questions as part of their teaching method. These questions could possibly provide a

different perspective to the teachers or some external knowledge that would aid in the creation of more questions.

1.3 Aim

This thesis aims to generate Multiple Choice Questions (MCQs) for aiding students and alike in learning, using the techniques and research done in the question generation field. In this thesis, we present our generic approach for generating multiple choice questions and distractors, and apply it to generate three varieties of MCQs: 1) gap-fill questions, generated by taking a sentence where an entity is mentioned, replacing it with a blank and providing other similar entities as distractors; 2) choose-the-type, where the question asks about the type of the entity, and the distractors are types similar to the type of the entity; 3) *Jeopardy-style*[1] questions, which include some hints about the answer. These hints make the question more interesting and also help to generate appropriate distractors.

Examples of these three MCQ varieties are shown in Listing 1.

1. *The Eiffel Tower is located in _____.*
   a) Madrid b) Amsterdam ***c) Paris*** *d) Lyon*

2. *Lionel Messi is a:*
   a) Golf player b) Cricketer
   c) Gridiron football player ***d) Soccer player***

3. *I am a scientist. I starred in Atomic Power (film) and Julius Sumner Miller was influenced by me. Who am I?*
   a) Albert Einstein *b) Enrico Fermi*
   c) J. Robert Oppenheimer d) Vannevar Bush

LISTING 1: Examples of questions generated using our approach

1.4 Thesis Outline

In the upcoming chapter, namely Chapter 2, we will firstly look at a brief background of the concepts that are relevant to this field of study. We also go through some of the specific technologies that are used for the completion of this thesis.

We examine the literature of related work in Chapter 3. Contributions in the field of automatic question generation are discussed, some of which follow a very similar approach to ours.

[1] Inspired by the T.V. show "Jeopardy!"

Our approach is described in detail in Chapter 4. The contents include the individual steps involved, followed by the general workflow of the system. We also discuss how a variety of questions is generated and what is done differently to achieve that.

We throw light on the implementation in Chapter 5. The design and architecture of the system are explained at different levels of granularity followed by the technology stack used to build it.

Parts of this thesis were presented at CSCUBS 2017 [1].

Chapter 2

Background

2.1 MCQ Structure

There are a variety of MCQs that we aim to generate but they all share the same structure. Each MCQ has one correct choice which is the answer, the question text and a set of wrong choices, called *distractors*.

Answer

The answer for an MCQ can either be a named entity or some sort of information extracted from the entity. We try to ensure that, in case there is more than one correct answer to a question, only one of the correct answers is provided.

Question Text

The phrasing and type of the question text can vary greatly. For example, in the case of *gap-fill* questions, the question text is a sentence from the source content with the answer entity removed. One can also generate questions based on properties of an entity or based on its relations with other entities. The information about an entity can be presented as-is in the form of a statement and the user can be asked whether this statement is true or false.

Distractors

The essence of an MCQ is the choices it has. We decided on a four-choice format for the questions. Hence, each question will have one correct choice and three *distractors*. Depending on what the question variety is, the distractors can be entities that are related to each other and the answer in some way, or can be types that an entity can belong to.

2.2 The Semantic Web

The Semantic Web can be described as "an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [2]. The semantic web is a web of data and is standardized to

help the exchange of data in a format set by the World Wide Web Consortium (W3C). It aims to connect data from different sources which necessarily would not be share data with each other e.g. applications which store data privately. The standardized format helps achieve integrating and combining data collected from various sources. It also aims to make computers understand hyperlinked information.

A use case of the semantic web can be seen in Hypertext Markup Language (HTML) – the language used to design web pages. It consists of text and multimedia elements in the form of tags which specify details about the layout of the web page. These tags do not provide any semantic information like the nature of the data being displayed or relations between different data entities. The semantic web proposes a solution to this problem by making use of languages like the Resource Description Framework (RDF) and the Web Ontology Language (OWL) which were designed keeping data in mind. The most important exchange format is the Resource Description Format (RDF) which we later discuss in Section 2.3.

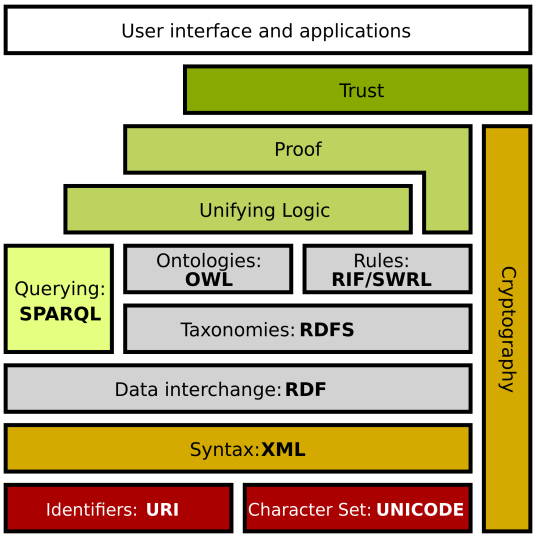


FIGURE 2.1: The layers of the Semantic Web[1]

2.2.1 Linked Data

Linked data is a set of guidelines[2] for publishing structured data on the web. They are:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards like RDF, RDFS, OWL ontologies, SPARQL (refer Section 2.3.3), etc.

[1] https://en.wikipedia.org/wiki/Semantic_Web
[2] https://www.w3.org/DesignIssues/LinkedData

4. Include links to other URI's so that a person or machine can discover more things.

These guidelines help not only to have standardized data but also to link data from different sources using typed relations just like the traditional web being a set of HTML documents linked using hyperlinks. All linked data eventually becomes part of a global data graph.

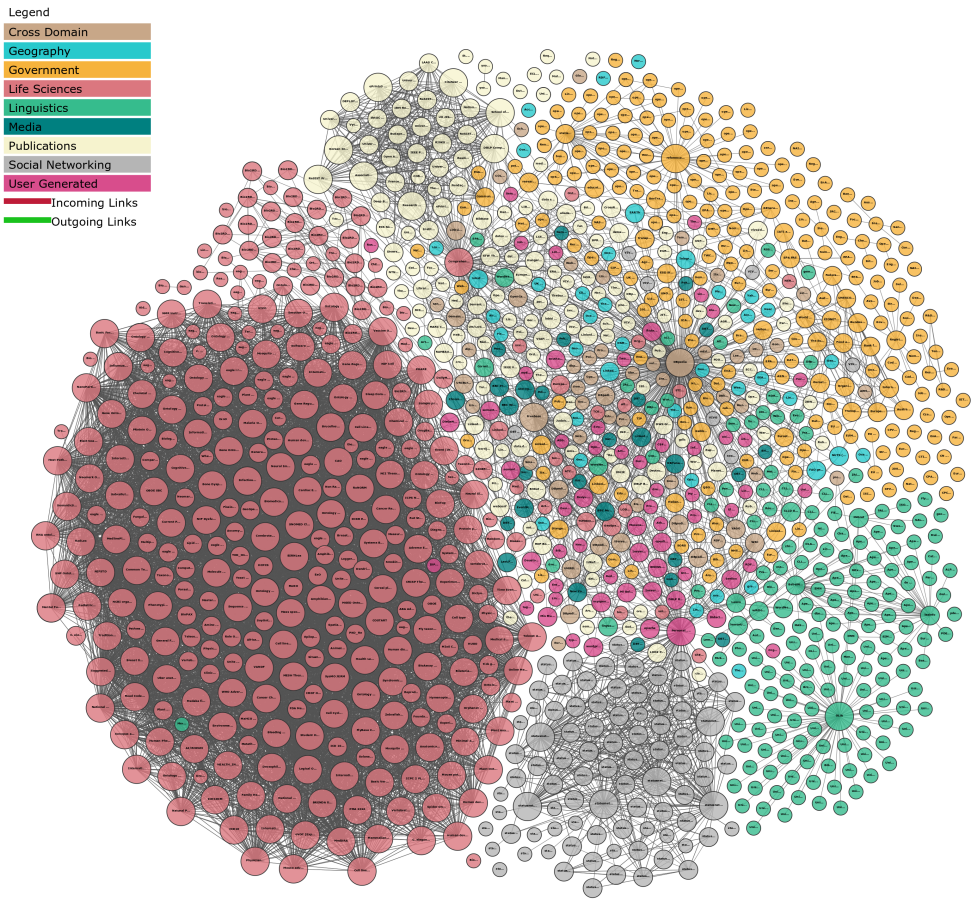


FIGURE 2.2: The Linked Open Data Cloud as per August 2017[3]

Linked Open Data (LOD) is linked data that has been released under an open license which means it can be reused freely.[4] DBpedia[3] is an example of a large linked open dataset. The LOD cloud diagram(shown in Figure 2.2) is an illustration of all the datasets that were published as linked data under the linked open data community project and otherwise.

2.2.2 Ontology

An ontology is a specification of a conceptualization[4]. It describes concepts and relations between them [5]. A concept can be an object, e.g. dog, school, bus or

[3]Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak.`http://lod-cloud.net/`

[4]`https://www.w3.org/DesignIssues/LinkedData`

it could even be an activity like running, sitting, etc. It all depends on what the ontology wants to exhibit.

On the other hand, relations represent how two concepts are connected to each other. Connections can show relationships – cars run on roads, planets revolve around stars, they can describe properties of an object – dogs are loyal etc. Ontologies are similar to RDF (refer Section 2.3) but they describe rules at the class level whereas RDF describes relations between entities. So the statement planets revolve around stars is part of an ontology because it describes a generic fact. On the other hand, the Earth revolves around the Sun can be an RDF triple that is compatible with this ontology.

A popular example of an ontology is the FOAF (friend-of-a-friend)[5] ontology. It is a basic ontology which describes the format for storing records of people (like an address book) as RDF data. Listing 2 shows a basic document stored according to the FOAF ontology.

```
<foaf:Person rdf:about="#danbri"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:name>Dan Brickley</foaf:name>
  <foaf:homepage rdf:resource="http://danbri.org/" />
  <foaf:openid rdf:resource="http://danbri.org/" />
  <foaf:img rdf:resource="/images/me.jpg" />
</foaf:Person>
```

LISTING 2: A basic document describing a person in the FOAF (friend-of-a-friend) ontology

2.2.3 Knowledge Base

A knowledge base (KB) is a centralized repository of information that is machine readable. Usually KBs contain information about a certain domain or subject but many have been designed to serve as a central store of data from all across the internet. DBpedia [3] and YAGO [6] are two knowledge bases that we will use as a source of external data in our approach. KBs comprise of entities that are related to each other and are classified using a number of classes. These classes can also be related to each other.

DBpedia

DBpedia [3] is a knowledge base of structured data extracted from Wikipedia. The extraction framework can be seen in Figure 2.3. The structured information includes infobox templates, images, geo-coordinates, external links, etc. The RDF dataset of

[5] `http://xmlns.com/foaf/spec/`

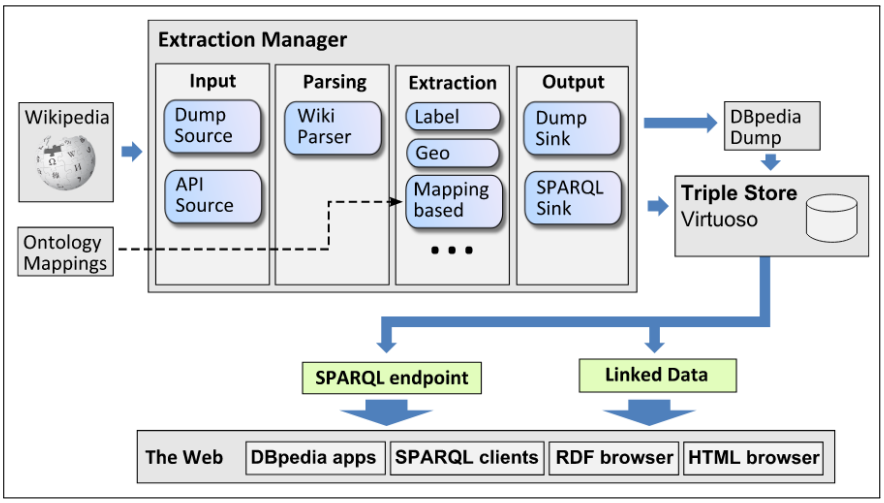


FIGURE 2.3: The DBpedia Extraction Framework[3]

DBpedia is hosted using Openlink Virtuoso[6] which enables the data to be accessed by either SPARQL queries (refer Section 2.3.3) or HTTP requests. DBpedia also consists of the DBpedia ontology which is created using the info-boxes in Wikipedia. As per 2014, DBpedia consisted of 3 billion RDF triples out of which 580 million were extracted from the English edition of Wikipedia, 2.46 billion were extracted from other language editions[7]. Due to this sheer amount of information stored in it, DBpedia is our prime source of information.

YAGO

YAGO (Yet Another Great Ontology) [6] is another knowledge base. It consists of data extracted from Wikipedia, Wordnet [7] and GeoNames [8]. Currently, YAGO has more than 10 million entities and contains more than 120 million facts about these entities[8]. Since it uses the taxonomy of WordNet along with Wikipedia categorization, it contains more than 350,000 classes[8]. Due to this high number of classes, YAGO provides us with a lot of granular information when it comes to classes and categories of entities.

2.3 Resource Description Framework

The Resource Description Framework (RDF), as introduced earlier in Section 2.2, is a popular exchange format for linked data. It is used to describe resources which can be documents, people, objects, concepts etc. This helps applications to understand the semantics of the data on the web. These applications also include those for which

[6] https://virtuoso.openlinksw.com/
[7] http://wiki.dbpedia.org/about
[8] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

the data was not initially meant for but since it was designed keeping the framework in mind, it can be processed by a foreign application as well[9].

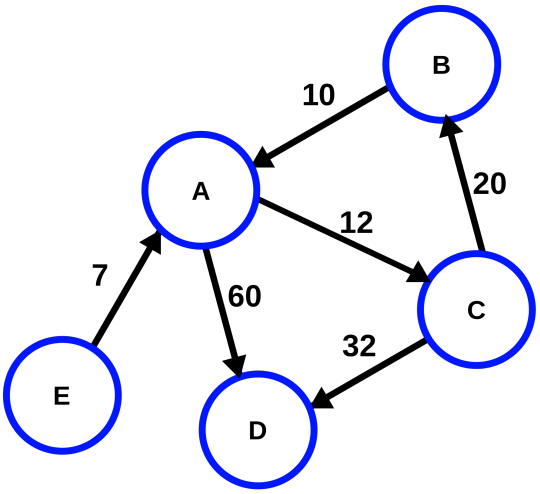


FIGURE 2.4: An example of a graph[10]

Data stored in RDF format is inspired from graph theory, which is the study of a set of vertices (which represent entities) connected by a set of edges (which represent connections between entities). Figure 2.4 shows an example of a graph.

RDF Statements are used to convey information about resources. These statements consist of three parts: the `subject`, the `predicate` and the `object`. These three-part statements are called RDF *triples*.

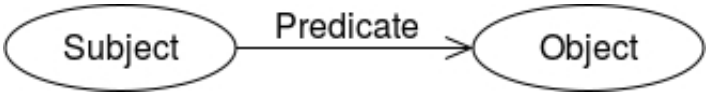


FIGURE 2.5: An RDF Graph for a Triple

Figure 2.5 shows us an RDF triple in the form of graph data. This is the core data model of RDF[11].

The subject and object are two resources that are related via the predicate. Predicates or properties e.g. `fullName`, `country` tell us about the relation between two resources. They are directional i.e. from subject to object. Examples of RDF triples are shown in Listing 3. As can be seen in Listing 3, a resource (in this case Lionel Messi) can be part of multiple triples.

```
<Lionel_Messi> <birthPlace> <Rosario>
<Lionel_Messi> <type> <SoccerPlayer>
```

LISTING 3: RDF Triples about Lionel Messi

[9] `https://www.w3.org/TR/rdf11-primer/`
[10] `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)`
[11] `https://www.w3.org/TR/rdf11-concepts/`

Each RDF resource is identified by an International Resource Identifier (IRI). RDF does not impose rules on IRI's but a particular ontology may use certain IRI formats to identify a particular kind of resource. For example, DBpedia uses IRI's of the form `http://dbpedia.org/resource/Name` to denote the thing described by the corresponding Wikipedia article.

2.3.1 Resource Description Framework Schema

The RDF Schema (RDFS) language provides a data modeling vocabulary for RDF data[12] and hence can be used to define semantic characteristics. It is an extension of RDF in the sense that it allows grouping related resources and describe relationships between them. These groups are called *classes* and each resource is an *instance* of a class. An instance can have the `rdf:type` property to point to a class of which it is a member. Classes themselves are also resources, often identified by IRI's and may be described using RDF properties. A class can be an instance of itself. The group of resources that are RDF Schema classes is itself a class called `rdfs:Class` and all things described by RDF are called `rdfs:Resource`s.

Class name	**Comment**
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:langString	The class of language-tagged string literal values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.

TABLE 2.1: Some RDF and RDFS classes

Property name	**Comment**	**Domain**	**Range**
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

TABLE 2.2: Some RDF and RDFS Properties

Each `rdfs:Resource` may have many `rdf:type`s like `Person`, `President` etc. The property `rdfs:subClassOf` helps show the relation between two classes. So if class C is a *subclass* of C', then all instances of C are also instances C'. Inversely, C' is the super-class of C. This forms a hierarchy as seen in Figure 2.6.

Taking the example of Lionel Messi, he can have types (belong to classes) like `Person`, `Athlete`, `SoccerPlayer`, `BarcelonaPlayer` amongst others. Each of

[12] `https://www.w3.org/TR/rdf-schema/`

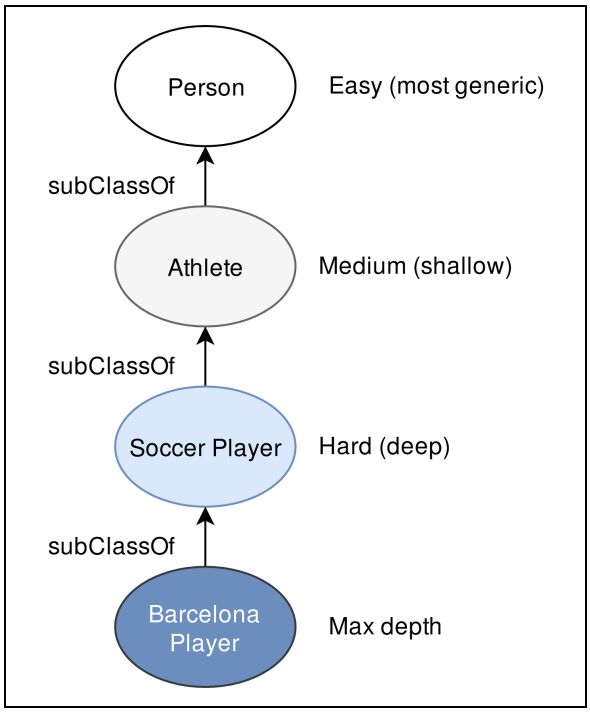


FIGURE 2.6: An example of class hierarchy in RDFS

these classes is a *subclass* of the previous. As we go from `Person` to `Barcelona-Player`, deeper into the hierarchy, the granularity of the class increases. A deeper `rdf:type` gives us more specific information about the resource compared to a shallow type. This is crux of the difficulty calculation algorithm that we discuss later.

2.3.2 OWL Web Ontology Language

The OWL Web Ontology Language[13] is a language designed by the World Wide Web Consortium to express ontologies. OWL was derived from the DAML+OIL[14] web ontology language. The OWL language shares certain concepts with RDFS such as classes and class extensions (instances of a class). Below is a list of common concepts from OWL[5]:

- *Class*: Group of objects with shared characteristics
- *Class extension*: All instances of a class
- *Class axioms*: Basic statements about classes. `rdfs:subClassOf` (inherited from RDFS) is an example of this. Other examples include `owl:equivalentClass` and `owl:disjointWith`.
- *Property*: Properties to describe a class (Same as RDFS)
- *Property extension*: All pairs of subjects and objects that can be connected with the property

[13] `https://www.w3.org/TR/owl2-primer/`
[14] `https://www.w3.org/TR/daml+oil-reference`

- *Property axioms*: Basic statements about properties e.g. `rdfs:subPropertyOf`.

2.3.3 SPARQL

SPARQL Protocol and RDF Query Language (SPARQL) [9] is an RDF query language designed to query and manipulate RDF data. SPARQL and its constructs can be used to filter out data to discover relations between resources and their properties. It allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.[15] Queries mostly contain a basic graph pattern which is a set of triple patterns like RDF triples except that the triple can consist of variables. The results of the query is an RDF subgraph which matches the pattern in the query when RDF terms from the subgraph are substituted for the variable [16].

```
1 PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
2 SELECT ?name
3 WHERE {
4         ?person a foaf:Person .
5         ?person foaf:name ?name .
6 }
```

LISTING 4: SPARQL query to fetch names of all persons in the foaf ontology

Listing 4 shows a very basic SPARQL query. This query looks for all persons in the FOAF ontology (refer Section 2.2.2) and returns a list of their names. The query in Listing 4 is a SELECT query which returns tabular results of values from a SPARQL endpoint.

2.4 Natural Language Processing

Natural Language Processing (NLP) is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things[10].

There are several applications of NLP including machine translation, text classification (finding the topic of a of text), sentiment analysis, language modeling (finding relationships between words and predicting upcoming words), speech recognition, text summarization, question answering amongst others.

NLP is an umbrella term for a variety of subtasks that help to achieve the aforementioned applications. We describe some of these tasks below[17]:

[15] `http://www.xml.org/xml/news/archives/archive.10062006.shtml`
[16] `https://www.w3.org/TR/rdf-sparql-query/`
[17] `https://en.wikipedia.org/wiki/Natural_language_processing`

- *Part-of-speech tagging*: Also called POS tagging, this is used to identify the part of speech (noun, verb, adjective etc.) of each word in a text.
- *Parsing* Determining the parse tree of a sentence. A parse tree is a syntactic breakdown of a sentence to find relations between words and their properties.
- *Sentence breaking*: Splitting a text into sentences using sentence boundaries like periods or exclamations.
- *Stemming*: Getting the root form of a word
- *Named Entity Recognition*: Named Entity Recognition (NER) is the process of finding actual entities (such as people, places etc.) in a text.
- *Natural language generation*: Generate human readable information from machine data.
- *Question Answering*: The process of understanding a human readable question and generating the answer for it.
- *Word sense disambiguation*: Deriving the correct meaning of a word based on the context it is used in. For example, the word "Germany" can mean the country or a sports team depending on the topic of the text.

2.5 Semantic Annotation

Semantic annotation is an amalgamation of NLP and the semantic web. A semantic annotator finds named entities in text, disambiguates them and links each entity to a resource in the linked open data cloud[18]. So unlike traditional named entity recognition (NER), semantic annotation can help find additional information about the text.

DBpedia Spotlight

We chose DBpedia Spotlight [11] as the semantic annotator for our approach. It uses DBpedia as its knowledge base. DBpedia Spotlight uses TF*ICF (Term Frequency Inverse Candidate Frequency) [12] which is like contextual scores used in text mining. This helps to find the context and relevance of the word, and hence to disambiguate candidates of a given surface form so as to link to the correct DBpedia resource. It provides decent F1 scores without over-annotating [12]. The confidence parameter of DBpedia Spotlight is set to 0.6 for best results [12]. The DBpedia Spotlight web service[19] is used to send text and get back resources that are further used for creating questions. As can be seen in Listing 5, the response from DBpedia Spotlight contains basic `rdf:types` that can be directly used to generate distractors. A limitation of

[18] `http://lod-cloud.net/`
[19] `http://www.dbpedia-spotlight.org/api`

```
{
    "@URI": "http://dbpedia.org/resource/Lionel_Messi",
    "@support": "736",
    "@types": "DBpedia:Agent,Schema:Person,
     Http://xmlns.com/foaf/0.1/Person,
     DBpedia:Person,DBpedia:Athlete,DBpedia:SoccerPlayer",
    "@surfaceForm": "Lionel Messi",
    "@offset": "13",
    "@similarityScore": "1.0",
    "@percentageOfSecondRank": "0.0"
}
```


LISTING 5: The DBpedia Spotlight Resource for the entity Lionel Messi

Spotlight is that it is designed to work for long, formatted texts, which is not the case for slide content.

Gerbil

Gerbil [13] is an open source tool used to compare semantic annotators. It gives the option to tweak various parameters including the experiment type, annotation match strength and the dataset. Table 2.3 shows the top four annotators according to F1 score and time performance. The annotators were put under the A2KB[20] experiment defined by the creators of Gerbil. The experiment consists of annotators required to perform entity recognition and linking to the knowledge base. The annotators were made to run on the OKE 2015 Task 1 gold standard sample[21] dataset. Also, the entity matching scheme was a strong match – the position of the entity has to overlap the position inside the gold standard. As one can see, DBpedia Spotlight performs well, even with a confidence value of 0 (not ideal). The entire results of the experiment can be found online[22].

Annotator	Micro F1	Macro F1	Avg millis/doc	Confidence
DBpedia Spotlight	0.5253	0.4721	83.8737	0
WAT	0.525	0.5087	64.0632	0.1862
FOX	0.4706	0.4683	971.4316	0
xLisa-NGRAM	0.5355	0.5215	1126.1579	0.514

TABLE 2.3: Comparison of semantic annotators using Gerbil

[20] https://github.com/dice-group/gerbil/wiki/Experiment-types

[21] https://raw.githubusercontent.com/anuzzolese/oke-challenge-2016/master/GoldStandard_sampleData/task1/dataset_task_1.ttl

[22] http://gerbil.aksw.org/gerbil/experiment?id=201703130002

Chapter 3

Related Work

Professionals from an array of diverse fields are working on the current research topic of question generation. Natural Language Processing (NLP) is one such field where researchers are highly intrigued by the topic. NLP enables us to gain information about the structure of texts and parts of speech. It also helps us generate natural language questions, which are, grammatically and semantically, as close to manually created questions, as possible. On the other hand, the enormous amounts of data available via the semantic web provides researchers with a plethora of opportunities to use this data in numerous ways. Of course, automatic MCQ generation is one application of the data.

Question generation has its applications in education and e-learning. The aforementioned abilities of NLP have given rise to a number of approaches for question generation in the field of language learning [14]–[20]. Using text books [21], [22] and other education material [23] as a source for generating questions is also a prevalent topic due to their use in the classroom environment.

Some works focused on using information at the sentence level [24], [25] whereas others followed a popular topic of generating gap-fill or *cloze* questions [26], [27].

The semantic web being a rich source of data has opened new doors for the question generation field. Works by Narendra, Agarwal, and shah [27], Papasalouros, Kanaris, and Kotis [28], Al-Yahya [29], Bühmann, Usbeck, and Ngomo [30], and Seyler, Yahya, and Berberich [31] are some examples that inspired us and hence we take a closer look at them in the upcoming sections.

3.1 Automatic Generation of Multiple Choice Questions from Domain Ontologies

This is one of the earlier approaches for automatically generating questions. Papasalouros, Kanaris, and Kotis [28] use domain ontologies to generate a variety of questions. As an example, they use the Ontology Web Language (OWL) to generate questions. Eleven different strategies are proposed which are divided into three categories: *class-based*, *property-based* and *terminology-based*. The first category consists of strategies that use the class and instance relationship. For example, the question would inquire the user about whether a resource is an instance of a class or not.

Property-based strategies are based on relations in the ontology. The relations could be between two instances of a class in the ontology or between an instance and a basic type (e.g. *string* or *integer*). Lastly, they present two *terminology-based* strategies. This strategy is based on the class-subclass relationship. So a question list a true class-subclass relationship, along with distractors which would be false class-subclass relationships.

This approach generates questions using domain ontologies directly whereas our approach finds entities in slide content and uses ontologies to help generate distractors. We use a similar approach to one of the discussed strategies to generate the type select variety of questions but we also take content extracted from slides and phrase questions as well as constructing natural language questions from RDF data. Also, from an implementation standpoint, the authors used OWL to generate questions which consists of mostly shallow types whereas our implementation uses DBpedia [3] and YAGO [6] which contain deeper types and hence help to generate more difficult questions, focusing on specific fields.

3.2 Automatic Cloze-Questions Generation

Narendra, Agarwal, and shah [27] designed a system to automatically generate gap-fill, also known as "cloze" questions. The students from IIIT Hyderabad make use of news reports on the 2011 Cricket World Cup to generate questions and distractors about the tournament itself. Their system is an end-to-end solution, which generates the gap-fill questions, with four options. One option is, of course, the answer to the question, whereas the other three are distractors. The distractors are generated using a knowledge base created by crawling info pages available at a cricket website.

Their approach is three-fold: sentence selection, keyword selection and distractor selection. For the first stage, they use a publicly available extractive summarizer to select important sentences. In the second stage, a list of potential keywords is made for each selected sentence. Each candidate is then tested against certain criteria like the presence of relevant tokens and prepositions. The ones not pertaining to the criteria are discarded. Named entities are given preference over other parts of speech. Finally, in stage three, relevant distractors are generated using the knowledge base previously mentioned. Only named entities are considered as keywords for the field of cricket. If the keyword is a cricketer, properties such as name, playing style, team are considered during distractor generation. Otherwise, distractors are selected at random.

Compared to our approach, this approach only generates a single variety of questions. Also, the knowledge base used for generating distractors is domain dependent since it is created by crawling news articles. Our approach on the other hand is domain independent.

3.3 OntoQue

Al-Yahya [29] used domain ontologies to create three different types of questions, namely, MCQ, true/false and gap-fill questions. The system generates questions by iterating over all triples in a domain ontology. An 8th grade history ontology on the Umayyad dynasty is used as an example for the prototype. Before generating questions, irrelevant triples like `rdf:comment`, `rdf:label`, `owl:domain` and `owl:range` are removed. Gap-fill questions are generated by removing either the subject or the object of a triple. For the true/false type, false statements are created by replacing the subject or the object with an alternative. This alternative is an entity of the same class as the correct answer. MCQs consist of two parts - the *stem* and the options. The *stem* is generated in a similar way to the true/false question type and the options consist of the key (correct answer) and the distractors. There are three strategies followed to generate MCQ items, namely, class-membership based, individual based and property based. In the first strategy, all classes and their instances are collected. The question *stem* enquires about the type of the instance. The answer or the *key* is the object of the RDF statement and distractors are chosen at random from the collection, leaving out the object class. In the individual based strategy, all individuals are listed and for each individual, all triples in which the individual is either the subject or the object are collected. These triples are used to generate the gap-fill questions. Lastly, for the property based strategy, all properties from the ontology are extracted and for each, subjects and objects are collected and questions are generated.

Similar to Papasalouros, Kanaris, and Kotis [28], this approach uses ontologies for question generation and hence does not involve any textual content. Gap-fill and true/false varieties generated in this approach use RDF triples for resources and are not multiple choice questions. For multiple choice questions that are generated, the approach makes use of more strategies apart from class-membership than we do but does not consider different levels of difficulty for distractors or any natural language questions.

3.4 ASSESS

Automatic Self-Assessment Using Linked Data (ASSESS) is a system designed by Bühmann, Usbeck, and Ngomo [30]. The authors aim to provide solutions to three problems: Automatic Verbalisation of RDF graphs, Entity Summarization and RDF Fragment Extraction. They provide a generic scheme of verbalisation that is independent of the knowledge base used. Entity summarization provides key properties to identify resources with. This enables forming terse questions. Lastly, Concise Bound Descriptions[1] (CBDs) are used for fragmenting input knowledge to provide domain specific questions (domain chosen by user).

[1] `https://www.w3.org/Submission/CBD/`

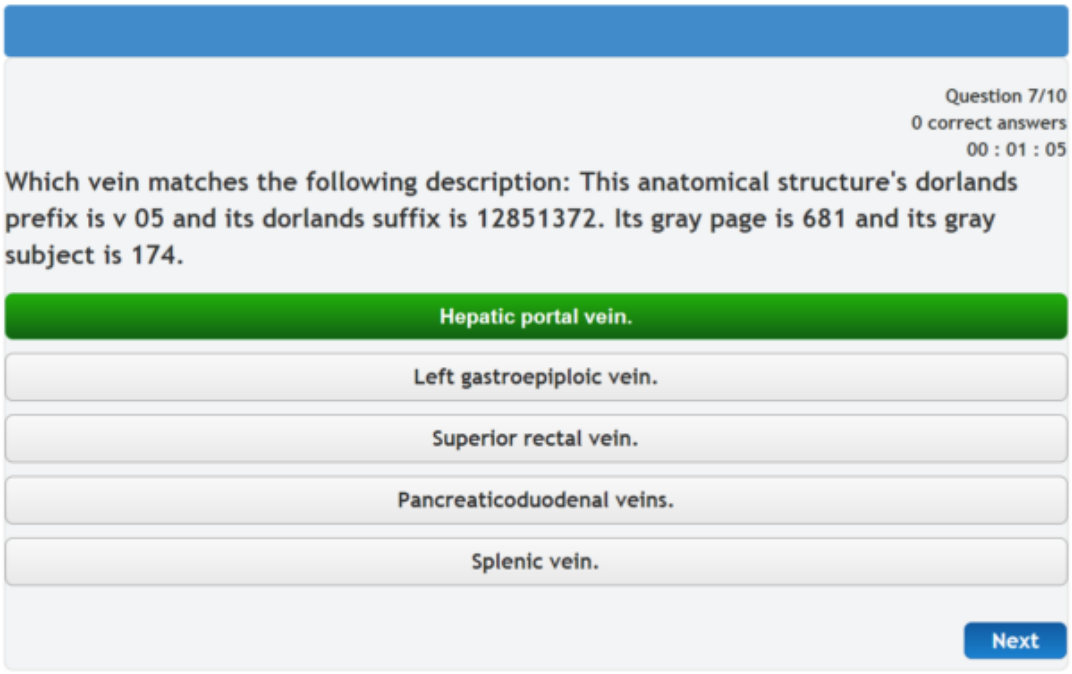


FIGURE 3.1: A Jeopardy Question on the ASSESS Client

The system is divided into four layers: 1) Data layer, 2) Natural-language generating layer, 3) Entity summarization layer and 4) Question and answer generation layer. The *data layer* is responsible for communicating with one or more knowledge bases, CBD extraction and selection of fragments based on the user's interest. It also implements in-memory and hard-drive driven caching solutions to provide scalability. As the name suggests, the natural-language generation layer converts RDF triples, SPARQL basic graph patterns (BGPs) and SPARQL queries to natural language. The layer is an extension of the SPARQL2NL framework [32]. It can be divided into two parts: 1) the verbalizer, which creates a sentence parse tree from RDF or SPARQL data and 2) the realizer, to which this data is fed as input to generate natural language. In the entity summarization layer, the authors try to find the predicates that best describe a given resource. The most specific class of a given resource is found and the predicates that are most frequently used in association with the instances of that class. The top k predicates are chosen to make it easier to work with larger CBDs.

Lastly, the question and answer generation layer generates yes/no, *jeopardy* and multiple choice questions (MCQs). Yes/no questions are generated by using an RDF triple of a given resource (where the resource is the subject) and either keeping or replacing the object of the triple. *Jeopardy* questions (as shown in Figure 3.1 are created using the class of the resource and a series of predicate-object (p,o) pairs. The correct answers share all the (p,o) pairs whereas the wrong answers share only some of them. The last variety i.e. MCQs are produced by verbalising the subject s and predicate p. The options comprise of objects o for which (s,p,o) holds true (correct choices) and (s,p,o') (wrong choices).

ASSESS generates *jeopardy* questions similar to our approach using a similar strategy but does not consider resource popularity while generating distractors.

3.5 Knowledge Questions from Knowledge Graphs

The premise of this work is to generate *Jeopardy* style questions from a knowledge graph having varying difficulties. Seyler, Yahya, and Berberich [31] use the YAGO2s [33] knowledge base as a source of data. The system takes the name of a personality as an input and outputs a question with one possible answer.

The authors find types that best describe a resource by using an entity-annotated corpus and converting the textual types to semantic types. The semantic types, being Wordnet synsets [34], can be looked up using the Wordnet lexicon. RDF triples for the entity are also generated. A SPARQL query is constructed using the types and triples found. The query should not have any textual overlap with the resource's surface form and should not contain redundant facts. Difficulty is estimated using a classifier trained on *Jeopardy!* quiz game data. Popularity of an entity is considered to be a measure of difficulty for a question about the entity. Apart from this, the likelihood of two entities occurring together also affects difficulty. For example, the pair `(BarackObama,WhiteHouse)` is more likely to occur together compared to `(BarackObama,GrammyAward)`. Hence, a question about the latter pair is more difficult even though `BarackObama` is a popular entity. The SPARQL query constructed previously is verbalised to a natural language question. Entities, since coming from Wikipedia, are verbalised using surface forms of links to each entity's Wikipedia entries. The option for multiple choice questions is also provided. Distractors generated should be of the same type as the answer resource and distractor confusability (similarity to each other and the answer) is also considered a governing fact for the difficulty of the question.

Just like ASSESS [30], distractors generated for the questions in this approach are not checked for popularity. Hence the approach lacks an extra parameter to generate more closely related distractors.

Chapter 4

Approach

The current approach to generate questions relies on a semantic annotator using a knowledge base to find named entities in slide content. These entities are then used as building blocks to generate different varieties of MCQs. In this chapter, we discuss and describe the steps involved in our approach to generate these questions.

We first take a look at some specifics regarding semantic annotation and what the output of this step is. We then discuss the different strategies we use to generate questions followed by strategies to generate distractors along with the type of distractors generated. Finally, we take a look at our verbalisation approach.

4.1 Semantic Annotation

We can call this step 0 of our approach. The contents of the deck of slides is fed as input to DBpedia Spotlight and a list of DBpedia resources is retrieved. As we have already discussed in Chapter 1, the semantic annotator links the entities to a knowledge base which uses RDF to store its data. These entities are therefore `rdfs:Resource`s. Semantic annotation is not always accurate and some entities can be incorrectly recognized. An entity can occur multiple times in a slide or deck and hence we sort these entities by the frequency of occurrence. The top n most frequent resources are chosen to avoid questions based on entities that might not be entirely relevant to the slides.

4.2 Question Generation

For each of the retrieved, short-listed resources provided by the previous step, we first query for information that is either based on the RDF type of the resource and/or the RDF triples it is a part of. Essentially, our approach is to first derive an answer and then reverse engineer to generate a question for it.

4.2.1 Generation using RDF Type

As discussed previously in Section 2.3.1, a resource can potentially have several `rdf:type`s. Out of these types, one is selected as a reference to identify the resource. Let us call this the *base type*. The base type is the focal point from where

questions are generated and distractors are found.

The selection of the base type depends on the required level of difficulty. As discussed in Subsection 2.3, a deeper type is more specific. Hence, we correlate difficulty to specificity.

$$\text{Difficulty} \propto \text{Type Depth} \tag{4.1}$$

We use the * operator provided in SPARQL for a transitive query along with the `rdfs:subClassOf` property. This acts as a recursive method, and gets the path from the concerned class till the most generic class. The count of the number of classes till the top is the *depth* of a type.

```
select distinct ?t (count(*) as ?count) where {
{
        select distinct ?t where {
                <Lionel_Messi> rdf:type ?t .
        }
}
        ?t rdfs:subClassOf* ?path .
} order by desc (?count) limit 1
```

LISTING 6: SPARQL query to fetch the most specific type of a resource

The depth helps us control how much we want to delve into the subject (amount of difficulty). A very shallow type will lead to a very generic or maybe even a vague question whereas a very deep type would result in asking about a very peculiar property of the given resource.

To put this into perspective, we discuss an example based on types seen in Figure 2.6 for the resource *Lionel Messi*. If we pick the base type as `Person`, we would be formulating a question regarding *Lionel Messi* being a person. This question does not really serve a purpose since this is common knowledge. On the other hand, if we pick a deeper type like `SoccerPlayer`, people not familiar with soccer might find this difficult. Extrapolating this approach, using the type `BarcelonaPlayer` as the base type would be further challenging. This is in parallel with Equation 4.1.

Deeper types also lead to more closely related distractors. For example, *Lionel Messi* is a soccer player. He does not have as much in common with *Barack Obama* (both of type `Person`) as he does with *Cristiano Ronaldo* (another soccer player) and hence would prove to be a more fitting distractor in most cases. This is discussed further in Section 4.3.

4.2.2 Generation using RDF Triples

RDF triples that hold true for the concerned resource can also be used to frame questions and look for distractors. As seen in the work by Papasalouros, Kanaris, and

Kotis [28], changing one part of a triple can help generate a question. For example, if we consider the first triple in Listing 3, removing the object would create a question about the birthplace of *Lionel Messi*.

In the case of triples, we consider the difficulty to be inversely proportional to the popularity of the triple (popularity of the subject and the object).

$$\text{Difficulty} \propto \frac{1}{\text{Popularity of Triple}} \tag{4.2}$$

We draw this correlation from the fact that a question about a popular triple would consist of information that is well-known.

To achieve different levels of difficulty, we get all RDF triples where the given resource is either the subject or the object, and sort them in decreasing order of popularity using the LinkSUM project query [35]. This query makes use of DBpedia Pagerank value [36] as a popularity measure. Pagerank (here) for a resource is the number of links present in Wikipedia leading to the resource. Since the DBpedia Extraction Framework [1] is used to calculate the pagerank and DBpedia URIs are used to identify resources, it is named the DBpedia pagerank. For a given triple, the pagerank value of the subject is considered if the resource is the object in the triple or vice-versa.

4.3 Distractor Generation

Distractors hold a significant role in a multiple choice question. Unlike a regular question (without choices), the quality of an MCQ is greatly affected by the quality of the distractors. The type of the answer to a question governs what kind of distractors need to be generated. For instance, if the answer to a question is an `rdf:type`, then the distractors also need to be `rdf:type`s. On the other hand, if the answer is an `rdfs:Resource`, then the distractors also need to be `rdfs:Resource`s (of the same or derived `rdf:type`).

Distractors are divided into two categories based on their source: *in-text* and *external*. In-text distractors are found from within the slide content whereas external distractors are fetched from a knowledge base.

4.3.1 In-Text Distractors

The resources shortlisted after semantic annotation are grouped together by `rdf:type`. Hence, for a question to which the answer is an `rdfs:Resource` in a group, all the other members in the group can be distractors. These distractors are usually found if the semantic annotator being used returns some or all the `rdf:type`s that apply to an annotated resource.

[1] `https://github.com/dbpedia/extraction-framework`

4.3.2 External Distractors

In the case of an absence of in-text distractors or shallow in-text distractors, external distractors are queried for. These are fetched from the knowledge base using the base type of the answer resource. Once again, we either look for `rdfs:Resources` belonging to the base type or `rdf:types` somehow related to the base type.

Answer is a Resource

```
1 select distinct ?distractor where {
2         ?distractor rdf:type <baseType>
3         filter (?distractor != <answer>)
4 }
```

LISTING 7: SPARQL query to fetch distractors of type `baseType`

```
1 select distinct ?distractor where {
2         <answer> <p1> <o1>
3         <answer> <p2> <o2>
4         ?distractor rdf:type <baseType>
5         <distractor> <p1> <o1>
6         Not exists {<distractor> <p2> <o2>}
7 }
```

LISTING 8: SPARQL query to fetch distractors using RDF triples of the answer

Listing 7 shows a very simple SPARQL query to get distractors. In it, we look for `rdfs:Resources` that have the base type (selected previously in Subsection 4.2.1) as one of its `rdf:types`.

If RDF triples of a resource were used in forming a question, they are also considered while querying for distractors. We see an example in Listing 8. Lines 2 and 3 show two triples that hold true for the answer resource. These two triples were used to formulate the question (this kind of question is further discussed in Subsection 5.2.3). On lines 4 and 5, we see statements used for fetching a distractor – it should be of the base type, and the first triple holds true for it. This would help find more relevant distractors since the . On line 6, we make sure that the second triple holds false for the resource. This ensures that a distractor would not be the correct answer. This case is used for more difficult questions since one of the triples holds true for it and hence produces ambiguity between choices. On the other hand, if both triples are false for the distractor, then not only are the distractors not as relevant (making

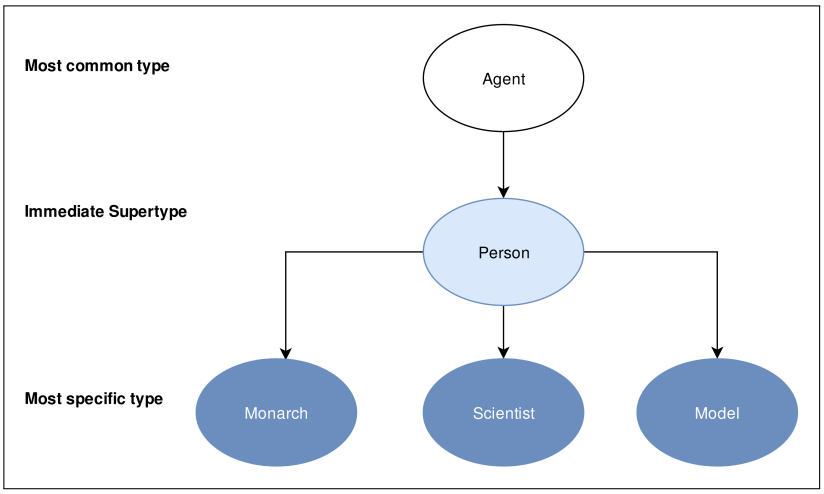


FIGURE 4.1: Sibling Types

it easier to choose), but also the entire question itself would hold true only for the answer and not the other choices.

For all cases, the results with the closest popularity to the answer are chosen. To do so, we calculate the difference d between the DBpedia pagerank value of the answer and each query result. We sort the results in increasing order of d and choose the top three (number of distractors). A check is always made to ensure that a resource has a pagerank or not. In case the pagerank is not available for the answer resource, the top three resources based on pagerank are used.

Answer is a Type

In case the question revolves around the type of a resource, we apply a different strategy: as shown in Figure 4.1, we find *sibling types*[2] that do not apply to the answer resource. These types are meant to be the distractors. So, as an example, if the answer is type `Scientist`, we go to the super-type, `Person` and then look for other sub-types which do not apply to the resource. A simple query to fetch sibling types is shown in Listing 9.

```
1  select distinct ?distractor where {
2          ?distractor rdfs:subclassof <superType>
3          filter (?distractor != <baseType>)
4  }
```

LISTING 9: SPARQL query to fetch Sibling types

[2]Types that are children of the same super-type, at the same level of depth

4.4 Verbalisation

The verbalisation step depends on the variety of question we want to produce. As shown in Listing 1, the gap-fill variety of questions do not require any verbalisation. For the choose-the-type variety, we use a simple phrase containing the name of the answer resource.

Template-based verbalisation is extensively performed in case the question is generated using RDF triples (like in the case of *Jeopardy-style* MCQs). The position of the answer resource in the triple (subject or object) is noted. Manually created templates for commonly occurring predicates are used. So for resources of type `Person`, the predicate `birthPlace` is common and would be verbalised as *"born in"*. Hence, the first triple shown in Listing 3 would be verbalised as *"Lionel Messi was born in Rosario"*. These templates can be applied irrespective of the position of the answer resource in the triple.

In case no template is found for a predicate, some patterns are searched for. As an example, predicates like `author` and `creator` have the suffix *-or* and the predicate `influenced by` has the keyword *by*. These can be used to generalise the verbalisation strategy. More templates as well as patterns can be added to extend functionality. Verbalisation is discussed in more detail in Section 5.2.3.

Chapter 5

Implementation

This chapter talks about the realization of the approach discussed in the previous chapter. We created a service, namely *QGen*, which generates MCQs for both textual data as well as slide content. We discuss the architecture and components of the QGen service, followed by its workflow. We shed light on tools, data sources and technologies we employed. Finally, we show how our approach has been used to create a microservice [1] that is part of the SlideWiki [2] backend.

5.1 Architecture

The QGen service comprises of several smaller components that have different responsibilities. Some or all of these components are used depending on which variety of questions are to be generated. The service exposes RESTFul API endpoints (refer Section 5.4) that are described in the `Application` class.

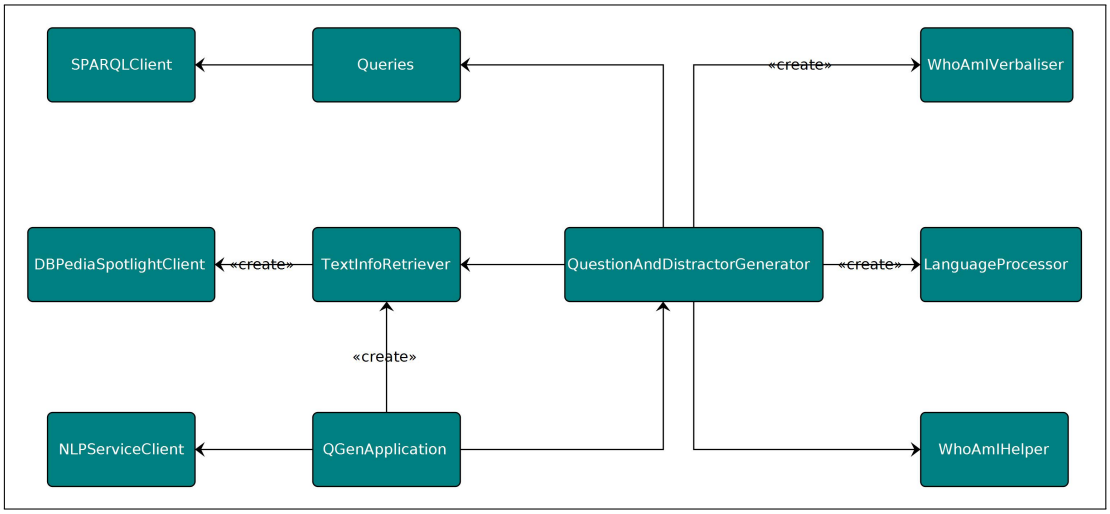


FIGURE 5.1: Components in the QGen System

[1] An architectural pattern for breaking down an application into smaller loosely coupled services `http://microservices.io/`

[2] SlideWiki is an open source platform for the collaborative editing and sharing of slides, with the main purpose of supporting teaching and training `http://slidewiki.org`

QGenApplication

The `Application` class is the single point of entry to the service. Once the class receives an HTTP request, it extracts the desired variety of MCQ and difficulty level from the request parameters. If questions for a particular deck are requested, it asks the `NLPServiceClient` to fetch the NLP results for the required deck. Otherwise, the `DBpediaSpotlightClient` is asked to provide results of semantic annotation for the text. Finally, it delegates the generation of questions and distractors to the `QuestionAndDistractorGenerator` class.

NLPServiceClient

The `NLPServiceClient` is responsible for interacting with the SlideWiki `NLP-StoreService`. The store service calculates the NLP results (DBpedia Spotlight results as well as word frequencies. Refer to Appendix A for more details) for contents of a deck.

QuestionAndDistractorGenerator

This is the main class that is responsible for generating the MCQs. It comprises of functions for each variety of MCQ, which in turn contain decision trees for different levels. These functions compile the questions and distractors fetched into neatly formed sets.

TextInfoRetriever

This class is responsible for parsing the text content and returning information like the frequency of words. The retriever is used when the `NLPStoreClient` is either not working or if a piece of text was passed to the `QGen` Service.

LanguageProcessor

The `LanguageProcessor` handles tasks related to processing text like splitting the text into sentences, fetching synonyms and singularize plural words. It makes use of the OpenNLP library as well as the RiTa library for Wordnet.

DBpediaSpotlightClient

This client class interacts with the DBpedia Spotlight API. It fetches the HTTP response and returns it in the form of POJOs (Plain Old Java Object).

SPARQLClient

This class uses the SPARQL query service available from the *Apache Jena* library to execute queries.

Queries

This is a constants class which stores all the queries used in the service.

WhoAmIVerbaliser

The `WhoAmIVerbaliser` is used to formulate readable *WhoAmI* questions using the template based approach discussed in 4.4.

WhoAmIHelper

Helper class containing functions which help in the generation of *WhoAmI* questions.

5.2 Workflow

To generate the three different varieties of MCQs, we use some or all of parts of the approach discussed in Chapter 4.

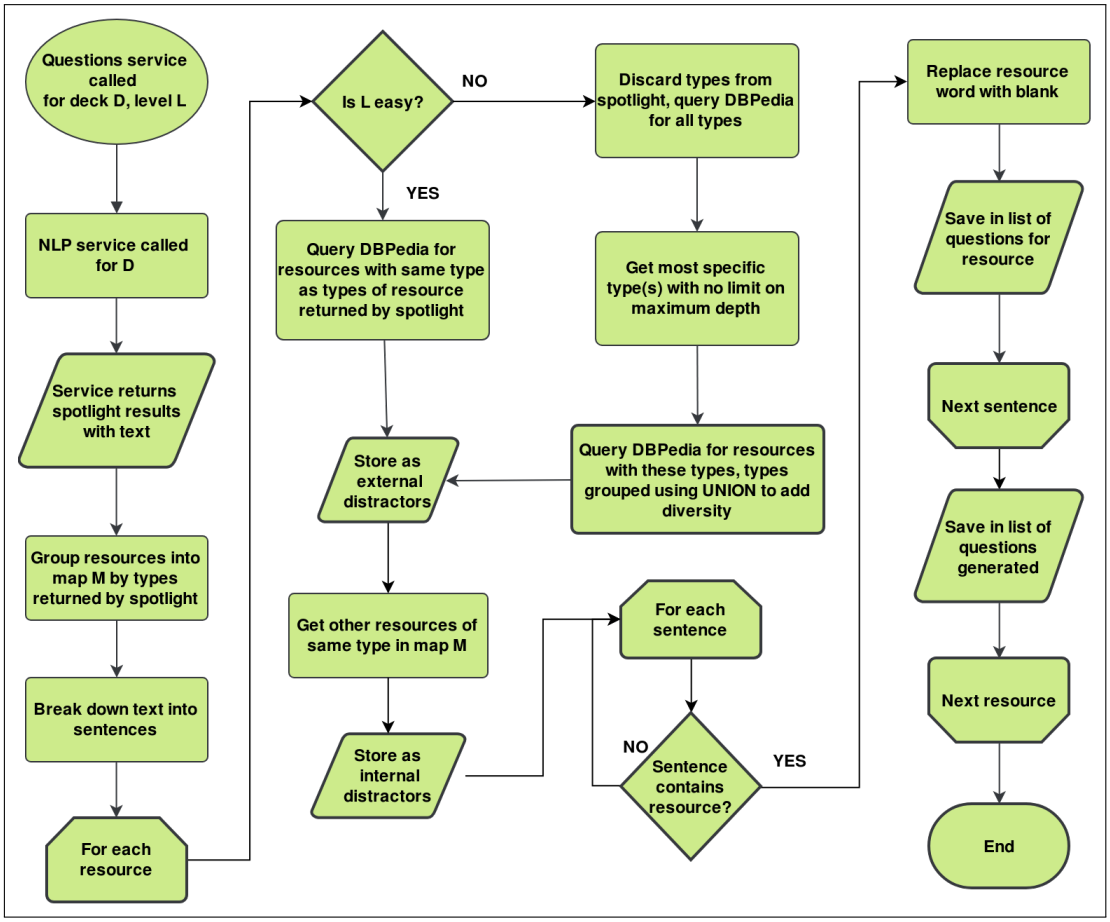


FIGURE 5.2: Workflow for generating Gap-Fill Questions

5.2.1 Gap-Fill Questions

For generating questions for the Gap-Fill variety, we make use of the approach discussed in Section 4.2.1. The `rdf:type` is considered as the basis for generating the MCQ. Both internal and external distractors are considered for this variety.

What is unique to this variety of questions is that the actual text from the slides is taken and broken down into sentences. The list of sentences is iterated through and if the resource is encountered in a sentence, the surface form (the entity in text) of the resource is replaced with a blank. Hence, the number of questions generated per resource is equal to the frequency of occurrence of the resource in the entire set of slides. A detailed description of the workflow is shown in Figure 5.2.

5.2.2 Type Description Questions

This variety of questions are generated with relative ease. The MCQ generation revolves around sibling types (as discussed in Section 4.3.2 which serve as distractors. The question text is a simple statement which asks about the type of the resource. One question is generated per resource for this variety. An example is shown in Listing 1.

5.2.3 Who Am I Questions

To generate the *Who Am I* or *Jeopardy* query, we designed a certain algorithm. For easy difficulty, two random triples are chosen from the top ten triples (ten triples are considered to limit time complexity). Top triples have large pagerank values, and based on Equation 4.2 (difficulty), they lead to easy questions. In case of hard difficulty, the algorithm shown in Figure 10 is used.

```
1  float lower_bound = 0.2
2  float upper_bound = 0.5
3  Triple triple = NULL
4  while((triple == NULL AND upper_bound LTE maxVRank)
5          triple = getTriple(lower_bound, upper_bound)
6          lower_bound = upper_bound
7          upper_bound = upper_bound * 2
```

LISTING 10: Algorithm to get triples for hard Jeopardy question

The initial bound values are chosen based on observations of pagerank values. In either difficulty, it is ensured that both triples are not duplicates and neither are their properties. Hence, two different aspects about the resource are asked for in the question. Just like type description questions, we generate one question per resource for this variety also. The verbalisation approach discussed in Section 4.4 is applied

here. Since these questions are for `Person` types only, predicate variation is limited. The patterns mentioned in Section 4.4 are used in this case. Since the identity of the answer resource should remain hidden in the generated query, first person singular pronouns *I* and *me* are used. In case the predicate is such that the position of the answer does not matter (e.g. *birthPlace*), the pronoun *I* is used. Otherwise, the position of the answer is mapped directly to the pronoun used for the respective part-of-speech i.e. *I* for the subject and *me* for the object. As shown in Figure 5.3, if the answer is David Hume (Scottish philosopher), there can be two types of triples found for the same predicate.

```
<David_Hume> <influenced> <Albert_Einstein>
```
"I influenced Albert Einstein"

```
<Gottfried_Wilhelm_Leibniz> <influenced> <David_Hume>
```
"Gottfried Wilhelm Leibniz influenced me"

FIGURE 5.3: Verbalisation of RDF Triples with the same predicate

In case none of the strategies work, a generic verbalisation *"I was the **predicate** of **subject**"* or *"My **predicate** was **object**"* is applied. This is kept as a last resort since using it would generate grammatically monotonous queries which might not be very engaging to the user.

5.2.4 Distractors

The base types used in our approach can be either DBpedia Ontology (DBO) classes or YAGO [6] types. DBO classes are observed to be not as deep as YAGO types. For example, the most specific DBO class for Lionel Messi is `SoccerPlayer` whereas the most specific YAGO type is `LaLigaFootballer`.

Based on this observation, the most specific DBO class is considered as the base type of a resource for easy difficulty. In case even that is too generic (e.g. `Person`), we get a random type between the depths of 11 and 14 (both exclusive). These depths usually contain YAGO [6] types that are similar in specificity to the most specific DBO types. For hard difficulty, we choose the most specific type a resource has, as the base type (as shown in Listing 6).

For the gap-fill questions, the distractors are just other resources of the base type. Hence, easy distractors are resources having an `rdf:type` as either the most specific DBO class or a YAGO class (depth between 11 and 14).

For the type description variety, distractors are sibling types (see Section 4.3.2) of the base type. The same procedure for the base type is followed for easy difficulty as gap-fill questions. For hard difficulty, a slightly different approach is used. To add a bit of variety to the distractors, the *n* most specific `rdf:type`s of the answer are taken and sibling types for any of these are found. The `UNION` operator is used in the SPARQL query for this purpose.

For *Jeopardy style* questions, along with the base type, the two triples that the question is made of, are also considered. For easy difficulty distractors, both should hold false and for hard, either one should hold true (shown in Listing 8). If no triples for the hard difficulty are found, the easy triples are used.

```
1  PREFIX v:<http://purl.org/voc/vrank#>
2  SELECT ?e ?r
3  FROM <http://dbpedia.org>
4  FROM <http://people.aifb.kit.edu/ath/#DBpedia_PageRank>
5    WHERE
6    {
7      ?e rdf:type dbo:Scientist;
8      v:hasRank/v:rankValue ?r.
9    }
10   ORDER BY DESC (?r) LIMIT 10
```

LISTING 11: A simple query to get the Top Ten Scientists by Pagerank [36]

As discussed in Section 4.3.2, apart from having the same `rdf:type`, distractors are narrowed down by popularity (in case the answer is a resource and not a type). Listing 11 shows a basic query to get the top ten resources having the type `dbo:Scientist` by pagerank.

Listing 12 shows how the query is used to get distractors of similar popularity. The query shown comprises of three nested `SELECT` queries. The inner-most query (lines 20 to 27) is a combination of the query shown in Listing 11 and Listing 7. We use it to find all resources of the base type (`dbo:SoccerPlayer` in this case), i.e. distractors, having a pagerank value, except for the answer resource *Lionel Messi*.

The immediate parent query (starting on line 20) runs for each of the resources found in the previous query. For each resource, it calculates the difference between the pagerank value of the resource and the answer resource (30.758 is the pagerank value). The absolute value is taken and stored in the variable *diff*.

Finally, the outermost query gets the `rdfs:label` or the `foaf:name` in English (whichever is available) of the resources found. The resources are sorted in increasing order of pagerank difference between the answer and distractors.

The result of running this query is shown in Table 5.1. The variable `d` shown in the first column is the distractor, `dev` is the deviation of pagerank and label is the name of the resource in English (as requested in the query on line 31 of Listing 12).

```
PREFIX vrank:<http://purl.org/voc/vrank#>
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX :<http://dbpedia.org/resource/>
PREFIX pagerank:
<http://people.aifb.kit.edu/ath/#DBpedia_PageRank>
PREFIX dbp:<http://dbpedia.org>

select distinct ?d
  (SAMPLE(?diff) AS ?dev)
  (SAMPLE (?dlabel) AS ?label) where {
    {?d rdfs:label ?dlabel}
    UNION
    {?d foaf:name ?dlabel}
    {
      select distinct ?d
      (ABS(30.758 - ?v) AS ?diff) where {
      {
        SELECT distinct ?d ?v
        FROM dbp:
        FROM pagerank:
        WHERE {
          ?d a dbo:SoccerPlayer .
          ?d vrank:hasRank/vrank:rankValue ?v.
          filter (?d != :Lionel_Messi)
        }
      }
    }
  }
  filter (langMatches(lang(?dlabel), "EN"))
} group by ?d ?diff order by (?diff) limit 3
```

LISTING 12: SPARQL query to fetch top 3 distractors by Pagerank

d	dev	label
http://dbpedia.org/resource/Diego_Maradona	3.0451818542	Diego Maradona@en
http://dbpedia.org/resource/Zinedine_Zidane	4.4163404541	Zinedine Zidane@en
http://dbpedia.org/resource/Pelé	4.5886026459	Pelé@en

TABLE 5.1: Result of query shown in Listing12

5.3 Technology Stack

Server and API Logic

The service is written in Java 8, with an API designed according to JAX-RS 2.0. JAX-RS, short for Java API for RESTful Web Services, is a specification designed by Oracle for Java to comply with the Representational State Transfer (REST) standards. JAX-RS provides annotations for HTTP request methods, method parameters and content types. Jersey is a reference implementation of JAX-RS 2.0 that is used as a library in the service to design the API. Apache Tomcat 7 is used to serve the application. Apache Tomcat is an open source implementation of the Java Servlet API. Maven is used as a build tool, bundling the project into a Web Application Resource (WAR) file. This WAR file is then deployed on to Tomcat.

SPARQL Querying

The Apache Jena[3] library is used for working with SPARQL queries. The queries are sent to the official DBpedia SPARQL endpoint[4] for execution. Apache Jena's `QueryExecutionFactory` class is used to send queries to the endpoint whereas the `ResultSetFactory` class is used to store the results of the queries. The queries are always executed on the DBpedia graph.

Deployment

A Tomcat docker image is used to deploy the service. Docker containerizes applications and provides isolation from the host environment. A docker container consists of all dependencies, packages, system settings and code required to run a particular software. The application does not need to be installed on the host and runs the same way, regardless of the environment. Hence, if the service runs on a local machine in a docker container, it will run anywhere.

Language and Semantics

Natural Language Processing has already been explained in Section 2.4. Apache OpenNLP[5] is an open source library that is meant for applying various NLP techniques to input data. We use this library to split the content of the slides into sentences to be used for certain varieties of questions. We also use Wordnet[6], which is a lexical database of the English language, to generate synonyms and also singularize words.

[3] Apache Jena is an open source Java framework for building Semantic Web and Linked Data applications `https://jena.apache.org/index.html`

[4] `http://dbpedia.org/sparql`

[5] `https://opennlp.apache.org/`

[6] `http://wordnet.princeton.edu/wordnet/`

5.4 RESTFul API

The service exposes REST endpoints that allow the service to be called via HTTP. This enables it to function independently as well as part of a bigger system (see Section 5.5).

The service exposes two endpoints – one for generating questions from text and the other to generate questions for a particular deck. Path parameters are used to specify the desired variety and difficulty of the questions along with the source of the content. One example for each of the aforementioned endpoints is described below.

1.
```
Route : /qgen/:type/:level/:deckID
Method : GET
Description: Generate questions for a deck
Path Parameters: Type of question, difficulty level and
                 deck ID
Query Parameters :  None
Headers : None
Body : Not Applicable
Response Codes: 200, 204, 404, 500
```

```
curl -X GET \
  http://localhost:8080/qgen/gap-fill/hard/1640
```

LISTING 13: GET Request to fetch questions for a deck

```
[{
    "slideId": "12597-1",
    "questionSet": [{
        "questionText": "_________:  neutrally charged
        subatomic particles",
        "answer": "neutrons",
        "inTextDistractors": [
          "electron cloud",
          "interact",
          "protons"],
        "externalDistractors": [
          "Neutronium",
          "Tetraneutron",
          "Radionuclide"]
     }]
 }]
```


LISTING 14: Response for GET at /qgen/gap-fill/hard/1640

2. Route : /qgen/:type/:level/text
   Method : POST
   Description: Generate questions for a given text
   Path Parameters: Type of question and difficulty level
   Query Parameters : None
   Headers : Content-Type: text/plain
   Body : Text to generate questions for
   Response Codes: 200, 204, 404, 500

```
curl -X POST \
  http://localhost:8080/qgen/gap-fill/easy/text \
  -H 'content-type: text/plain' \
  -d 'Germany won the 2014 FIFA World Cup. \
  Mario Goetze scored the winning goal. \
  They beat Argentina in the final.'
```

LISTING 15: GET Request to fetch questions for text

```
[{
    "slideId": null,
    "questionSet": [{
        "questionText": "_________ won the 2014
        FIFA World Cup.",
        "answer": "Germany",
        "inTextDistractors": [
            "Argentina"
        ],
        "externalDistractors": [
            "France national football team",
            "Scotland national football team",
            "Italy national football team"
        ]
    }]
}]
```

LISTING 16: Response for GET at /qgen/gap-fill/easy/text

Both Listing 14 and Listing 16 show responses for gap-fill questions. In this case, since the actual content of the slides becomes part of the questions, they are grouped by slides. Hence, when used in SlideWiki (refer Section 5.5), questions can be presented according to slide content even though they were generated for the entire deck at once.

5.5 Integration into SlideWiki

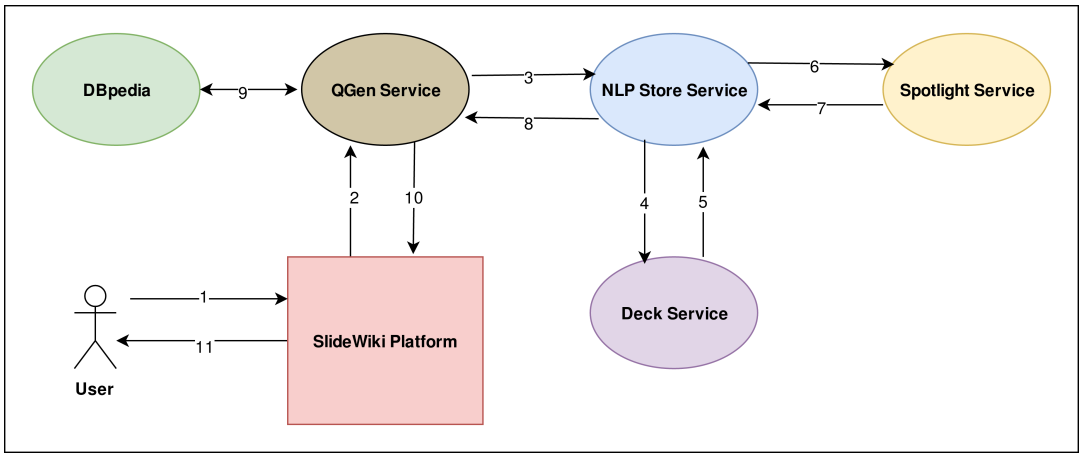


FIGURE 5.4: High level architecture and work flow of our system with SlideWiki

The service is planned to be used as the *Automatic Questions Service* in the new version of SlideWiki. It will be deployed as a microservice in the SlideWiki backend architecture. A microservice architecture ensures autonomous parts i.e. each microservice can be changed internally or new ones can be added without affecting the rest of the system.

SlideWiki consists of collections of slides, categorised by topic. These collections are called *decks* and are identified by a unique `deckID`. Our microservice is designed to communicate with another SlideWiki microservice called the *NLP Store* microservice. This service precalculates (listens to changes in deck contents) DBpedia Spotlight results, using their web service[7], for the entire deck contents. The output is clean text extracted from slides, along with spotlight results.

To communicate with our system, a RESTful API is provided with the URLs comprising of path parameters like question type and difficulty level. The `deckID` can be sent to the service to generate questions for the entire deck. Plain text can also be selected from a slide and sent to request questions for it.

Figure 5.4 shows how our microservice communicates with the other microservices and the SlideWiki platform. Here are the steps of the work flow:

1. User requests questions for a particular deck
2. The SlideWiki platform makes a call via our RESTful API
3. If questions for this deck are cached, skip (10). Else, ask the NLP Store service for annotation results and text of the concerned deck
4. The NLP Store service asks for deck contents from the Deck service
5. The Deck service returns deck contents
6. The contents are sent to DBpedia Spotlight for annotation
7. DBpedia Spotlight returns the results
8. Annotation results along with additional NLP results (out of scope) are returned
9. The questions are generated by querying DBpedia
10. Questions are returned as JSON
11. The questions are presented to the user as shown in Figure 7.1

As mentioned earlier, the *NLP Store* service performs steps (4) through (7) in advance. Hence, the above steps show the flow of data and may not be the same chronologically.

After our service returns a set of questions to the SlideWiki platform, the questions can be further changed as per the liking of the user framing the questions.

[7] `http://www.dbpedia-spotlight.org/api`

Many distractors (both in-text and external) are provided by our service and hence the user can choose from among these and frame a final multiple choice question depending on the required number of choices. The service is designed to return a fixed number (three) of external distractors by default. This number can be changed to provide a wider range for the user to choose from.

Chapter 6

Evaluation

To validate the quality of the results we got, we chose to perform a user based evaluation. We discuss the factors evaluated, the approach taken to evaluate the same and the results of the evaluation.

6.1 Factors

There are a number of factors that we need to consider to evaluate our approach properly.

Question Quality

This factor tells us how much the question text reveals about the answer. It only applies to gap-fill and jeopardy questions since the question text for the choose-the-type questions is not automatically generated. As discussed in Section X, we try to ensure that all forms of the answer are removed from the gap-fill sentence. Similarly, for *jeopardy*, if the question has a direct hint. The question quality should remain high irrespective of the difficulty level.

Distractor Quality

Distractor quality means the measure of homogeneity amongst choices. So, if the answer is a country, the other choices should also be countries. The distractor quality should be low for easy questions as compared to hard questions.

Difficulty

Since we promise two levels of difficulties, we want to ensure that an easy question is in fact easy for the user and a hard question is hard. Difficulty is a very subjective measure as it depends on the user's prior knowledge of the subject and his/her background. Hence, even if certain users don't agree to the level of difficulty, overall, more users should consider the easy questions to be easier than the hard ones.

Relevance

The relevance of a question to the slides tells us how much the subject matter of the question matches that of the slides. So, if the question is generated based on some external knowledge that isn't very related to the slide content, it has low relevance.

Grammatical Correctness

This applies only to jeopardy questions since we verbalise data to form natural language questions. If the basetype and triples are verbalised properly, and the grammatical structure of the overall question is correct, the grammatical correctness would be high.

6.2 Procedure

We generated questions for both difficulties and all varieties, for three decks of slides with the following topics: 1) World War I (33 slides), 2) The Atom (14 slides) and 3) Geography/General knowledge (37 slides). For each deck, we tried to select one easy and one hard question, generated for the same resource, to make it simple to compare the two difficulties. We did this for the three varieties, bringing the total to six MCQs per deck (four for decks 2 and 3 since there were no people mentioned in the slides and hence no *Jeopardy* questions were generated). In the case of gap-fill questions, this results in MCQs that have the same question text and answer, but different distractors according to difficulty. We showed the slides from each deck where the resources (for which the questions were generated) were mentioned. We also showed some other slides from the deck to give the reader more context.

The number of distractors generated can be more than four since we generate both in-text and external distractors. We chose the distractors we thought were a good fit. This is just in line with the real use case where an examiner or slide creator would choose the appropriate choices. We used in-text distractors for hard questions where possible and external ones for easy questions. In-text ones cause greater confusion since they belong to the slide content. For example, in the World War I deck, one of the questions is about Germany. The in-text distractors are other countries mentioned in the slides, that fought in the war. The external distractors are also countries that fought in the war but might not be mentioned in the slides. We think that choosing from the countries mentioned in the slides would probably be more difficult.

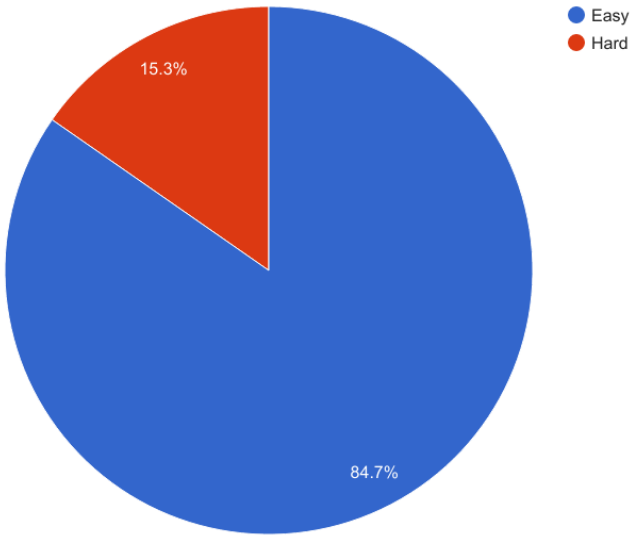
6.3 Results

50 users recruited via social media participated in the evaluation. The users were asked to rate the factors discussed in Section 6.1. The measures for question quality, distractor homogeneity and grammatical correctness were inspired by the work by

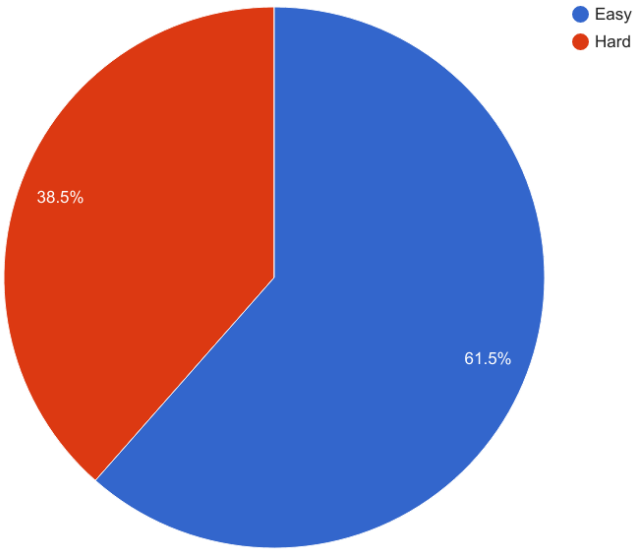
Kurdi, Parsia, and Sattler [37]. The actual evaluation[1] and results[2] can be found online.

6.3.1 Difficulty

84.7% users agreed that the easy questions were in fact easy whereas only **38.5%** users agreed that hard questions were difficult. Difficulty is of course subjective and also depends on the user's prior knowledge.



(A) Easy Questions



(B) Hard Questions

FIGURE 6.1: Percentage of users that agree with the difficulty level

[1]Copy of the Evaluation PDF
[2]Results for the evaluation on Google Forms

6.3.2 Relevance

As Table 6.1 shows, over **40%** users thought that the gap-fill questions had the highest relevance. This was expected since they are extracted from the slide text itself. Jeopardy questions came in second with most users rating the relevance between 3 and 4. The choose-the-type MCQs were not highly rated with the majority (**31%**) giving it the lowest relevance. This is likely since most of those questions inquired about tangential knowledge and lacked context.

RELEVANCE	GAP-FILL	CHOOSE-THE-TYPE	JEOPARDY
1	2.67%	31.00%	22.00%
2	7.33%	24.00%	20.00%
3	20.67%	21.67%	29.00%
4	29.00%	14.67%	19.00%
5	40.33%	8.67%	10.00%

TABLE 6.1: Relevance ratings for all 3 varieties

6.3.3 Question Quality

We recorded the quality for gap-fill (whether all forms of the answer resource were successfully removed or not) and *Jeopardy* (there were no mentions or obvious hints that revealed the answer) as shown in Table 6.2. Since we generate the phrase for the generic MCQs manually, we do not consider them in this case. The ratings for both varieties were mostly in the 3-4 range, especially for *Jeopardy* questions.

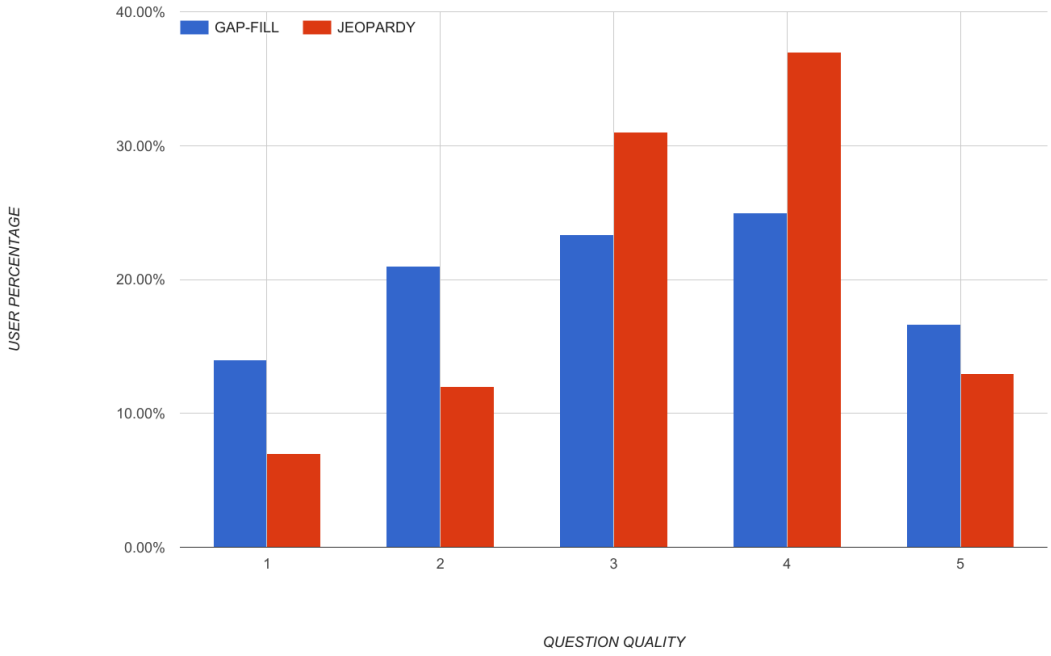


FIGURE 6.2: Distractor quality ratings of all 3 varieties

QUESTION QUALITY	GAP-FILL	JEOPARDY
1	14.00%	7.00%
2	21.00%	12.00%
3	23.33%	31.00%
4	25.00%	37.00%
5	16.67%	13.00%

TABLE 6.2: Question Quality ratings for Gap-Fill and Jeopardy

6.3.4 Distractor Quality

Distractor ratings were of utmost importance to us since distractors are the prime focus of our work. The consolidated results for the three varieties are shown in Figure 6.4. As expected, ratings for gap-fill and choose-the-type varieties are lower than *Jeopardy* questions. This can be attributed to the fact that the easy counterparts for these questions have low distractor quality on purpose to keep the question easy. As Table 6.3 shows, more than half of the users rated easy questions to have quality 2 or lower whereas most users rated hard distractors to be at rating 3 on average.

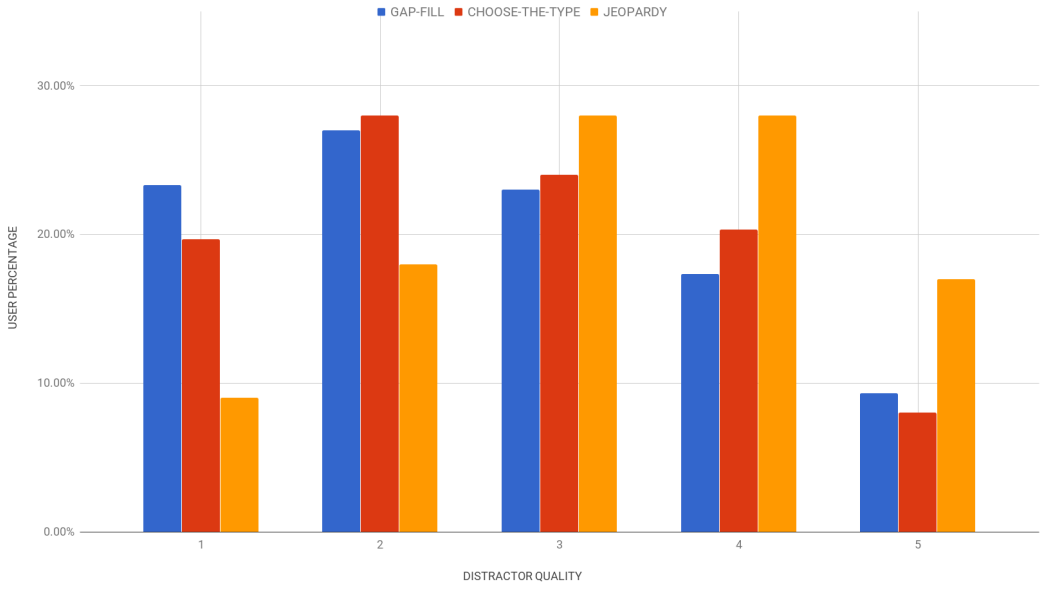


FIGURE 6.3: Distractor quality ratings of all 3 varieties

DISTRACTOR QUALITY	EASY	HARD
1	26.67%	14.50%
2	26.67%	25.75%
3	21.00%	26.50%
4	16.00%	23.25%
5	9.67%	10.00%

TABLE 6.3: Distractor Quality: Easy vs Hard

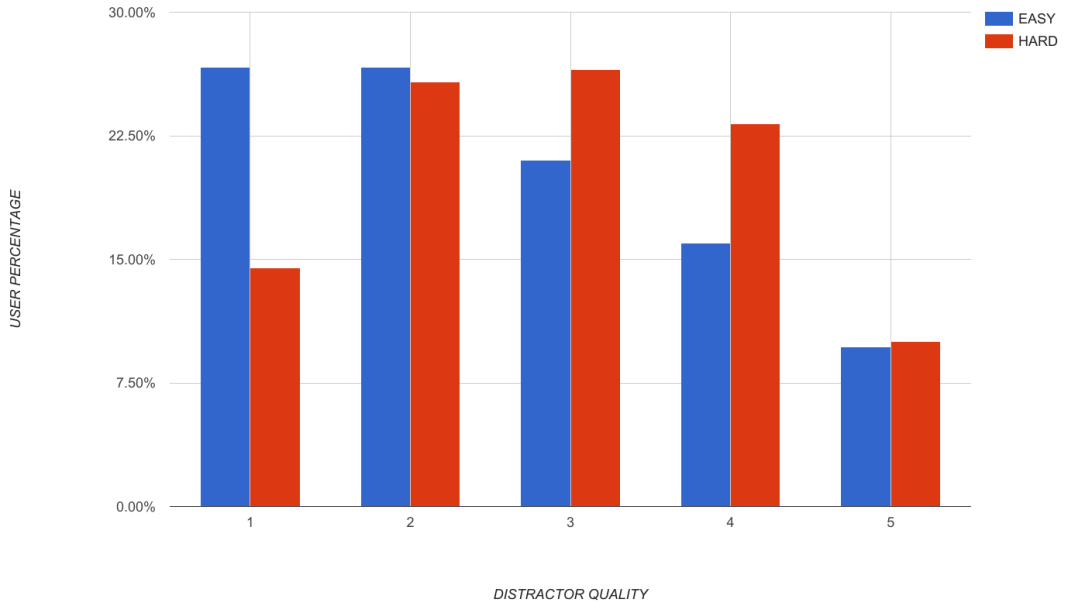


FIGURE 6.4: Distractor quality: Compared by Difficulty level

6.3.5 Grammatical Correctness

40% users rated the grammatical correctness of the *Jeopardy* questions at 4 and **28%** rated it at 5. This shows that the verbalisation scheme enabled us to form grammatically clear questions that were understandable for the users.

Chapter 7

Summary and Future Work

This chapter will summarize the thesis work. We discuss the contributions of the service apart from generating questions for slides, throw light on potential future extensions of this work and also discuss some limitations of the current version and how they can be overcome.

7.1 Summary and Contributions

We observe that the quality of distractors is directly proportional to the number of slides in a deck. This is due to the fact that the quality of semantic annotation is higher when more text is present. Better annotation leads to better distractors. The second and third variety prove to be a helpful addition since they do not use text from the slides and hence also work well for slides with short phrases or one entity bullet points. Incorrect types for a resource (second variety) reveal discrepancies in knowledge base data. Looking for other resources of a shallow type (for easy difficulty) requires more runtime: Shallow types, being generic, are present in a large number of resources.

Since our algorithm compares popularity, a larger deck requires more time. Distractors for the first variety of questions are generated simply by looking for similar resources irrespective of the content of the question. This could lead to distractors being correct answers for the question in some cases. A solution could be to use a natural language to RDF converter. This would provide the context of the sentence and help generate accurate distractors. While selecting a base type for easy difficulty, if the most specific DBpedia ontology class is too generic, sometimes a type between a given depth range is randomly selected. The type may or may not identify the resource well and hence may produce distractors that are not very accurate. A corpus could be used to mine data and learn about salient types [31]. Some triples included in a *Jeopardy style* question belong to only the given resource for the given base type. Hence, distractors are generated only considering the base type and not the triples. An alternative approach can be to look for resources that have the answer resource's triples hold true for them but are not of the base type. For example, for a resource with base type `Scientist`, who has acted in a film, the distractors could also be regular actors (not scientists) who acted in the same film.

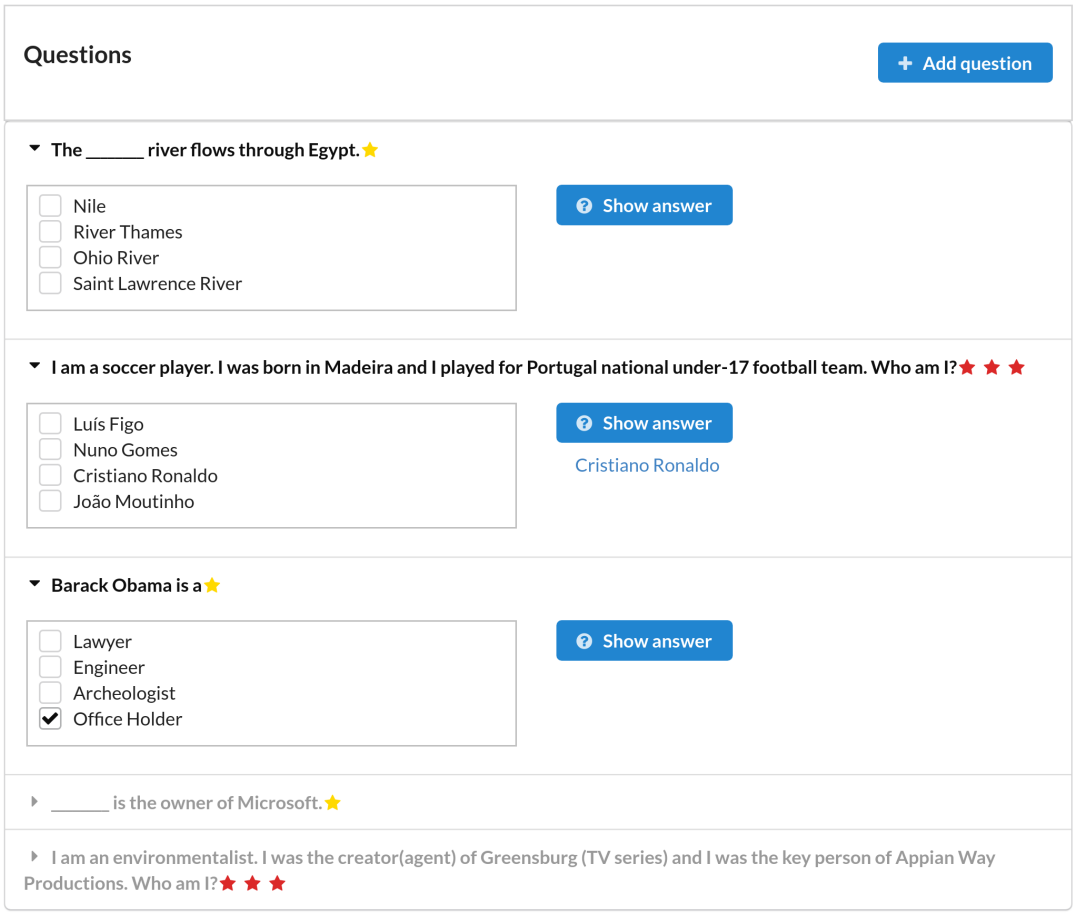


FIGURE 7.1: User interface of the questions section in SlideWiki

7.2 Future work

Our approach can be easily extended to generate many more question varieties like using other properties of an RDFS resource apart from `rdf:type` as done by Papasalouros, Kanaris, and Kotis [28]. Since we presented a generic approach by using types, triples and class hierarchies, more varieties of questions can be generated by simply extending the system. For instance, the RDF triples of the answer resource can be used to also generate gap-fill questions where the triple is verbalised as done in Section 5.2.3 and then replace the answer resource with a blank. This can be seen as the first example in Listing 17. One can also use triples and form *true/false* questions. Another possibility is the inverse of the type description questions where the type of the question is given and the resource is to be selected. These are examples 2 and 3 in Listing 17 respectively.

Another good addition to the approach could be to include speaker notes as content that can be annotated. This increases the amount of data available to the annotator and hence, the quality of annotation. Although the contents of the speaker notes would not be used, the named entities mentioned would prove to be useful.

The approach can be applied to other knowledge bases like YAGO [6]. Instead of using DBO classes that we did for easy difficulty as mentioned in Section 4.3.2, we

1. _____ *was influenced by Albert Einstein.*
   a) Julius Sumner Miller *b) Enrico Fermi*
   c) J. Robert Oppenheimer *d) Vannevar Bush*

2. *Isaac Newton was born in Lincolnshire.*
   a) False ***b) True***

3. *Which of these is a Soccer player:*
   a) Michael Jordan *b) Roger Federer*
   c) Sachin Tendulkar ***d) Lionel Messi***

LISTING 17: Examples of questions that could be generated using our approach

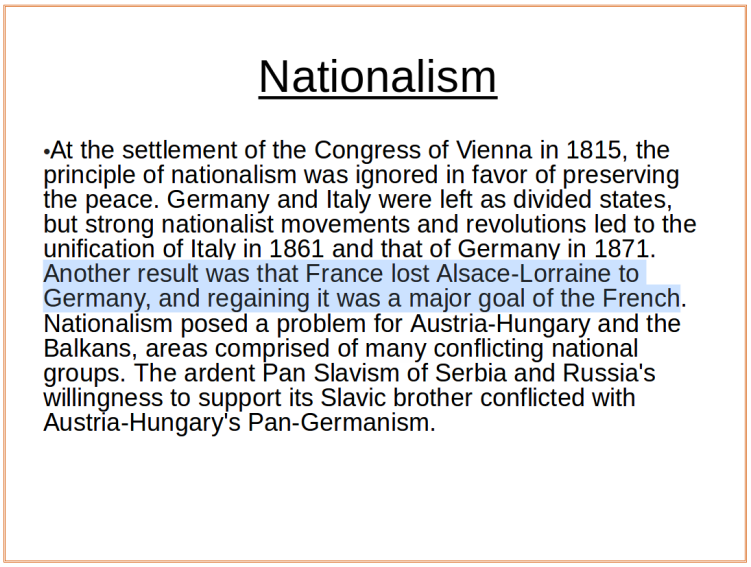


FIGURE 7.2: Potential Feature for Generating Questions for Selected Text

can use depth ranges directly which is a fallback in the current approach.

Figure 7.2 shows a potential use case in SlideWiki for our service. The text on a given slide can be highlighted and questions can be generated for it since our service has an endpoint to accept text via a POST request (refer Section 5.4). The challenge for this feature would be the quality of annotation since the content sent to the DBpedia Spotlight would be less.

Appendix A

Appendix

A.1 Sample Questions Generated

Here are few questions that were generated by our service for one of the decks[1] hosted on SlideWiki. We take the example of the slide (for gap-fill questions) shown in Figure 7.2 which belongs to the aforementioned deck. The type selection and jeopardy questions are the same for the entire deck.

Gap-Fill Easy

1. Question: _____ and Italy were left as divided states, but strong nationalist movements and revolutions led to the unification of Italy in 1861 and that of _____ in 1871.

   Answer: Germany
   In-text distractors: Austrian, France, Serbia
   External distractors: Tanzania, Kazakhstan, North Korea

LISTING 18: Gap-fill question generated for Easy level for a given slide

2. Question: Another result was that _____ lost Alsace-Lorraine to Germany, and regaining it was a major goal of the French.

   Answer: France
   In-text distractors: Austrian, Serbia
   External distractors: Phoenicia, Faroe Islands, Umayyad Caliphate

LISTING 19: Gap-fill question generated for Easy level for a given slide

[1] https://platform.experimental.slidewiki.org/deck/1033-1

Gap-Fill Hard

3. `Question: _____ and Italy were left as divided states, but strong nationalist movements and revolutions led to the unification of Italy in 1861 and that of _____ in 1871.`

   `Answer: Germany`
   `In-text distractors: Austrian, France, Serbia`
   `External distractors: Austria-Hungary, West Germany, Czechoslovakia`

LISTING 20: Gap-fill question generated for Hard level for a given slide

4. `Question: Another result was that _____ lost Alsace-Lorraine to Germany, and regaining it was a major goal of the French.`

   `Answer: France`
   `In-text distractors: Austrian, Serbia`
   `External distractors: Papal States, First French Empire, Second Polish Republic`

LISTING 21: Gap-fill question generated for Hard level for a given slide

Type Select Easy

5. `Question: Serbia is a :`

   `Answer: country`
   `In-text distractors: null`
   `External distractors: state, street, continent`

LISTING 22: Type selection question generated for Easy level for a given deck

6. `Question: Germany is a :`

```
Answer: country
In-text distractors: null
External distractors: settlement, territory, Intercommunality
```

LISTING 23: Type selection question generated for Easy level for a given deck

Type Select Hard

7. `Question: Serbia is a :`

```
Answer: Former Slavic Country
In-text distractors: null
External distractors: State And Territory Established In 519,
      State And Territory Established In 1335,
      State And Territory Disestablished In 1847
```

LISTING 24: Type selection question generated for Hard level for a given deck

8. `Question: Germany is a :`

```
Answer: State And Territory Established In 1871
In-text distractors: null
External distractors: Land District Of British Columbia,
      Shopping District And Street In South Africa,
      State And Territory Established In 1533
```

LISTING 25: Type selection question generated for Hard level for a given deck

Who Am I Easy

9. Question: I am a royalty. I was born in Graz and
I died in Sarajevo. Who am I?

Answer: Francis Ferdinand
In-text distractors: null
External distractors: Vladimir the Great,
Victor Emmanuel III of Italy,
Rudolf I of Germany

LISTING 26: Jeopardy question generated for Easy level for a given deck

Who Am I Hard

10. Question: I am a royalty. I was born in Graz and
my succession is Archduke of Austria-Este. Who am I?

Answer: Franz Ferdinand
In-text distractors: null
External distractors: Ferdinand II, Holy Roman Emperor,
Ferdinand III, Holy Roman Emperor,
Margaret of Austria, Queen of Spain

LISTING 27: Jeopardy question generated for Easy level for a given deck

A.2 Extraneous Tech Details

Dockerfile for Service

```
1 FROM tomcat:7.0-jre8
2 MAINTAINER "Ainuddin Faizan <andyfaizan@gmail.com>"
3
4 ADD settings.xml /usr/local/tomcat/conf/
5 ADD tomcat-users.xml /usr/local/tomcat/conf/
6 ADD wordnet.tar.gz /usr/local/tomcat/wordnet/
7 ADD en-sent.bin /usr/local/tomcat/
```

LISTING 28: Dockerfile for QGen Service

```
#!/usr/bin/env bash

cd 'mytomcatdocker'

docker pull andyfaizan/tomcat-dev
docker run --rm --name tomcat-dev \
        -p 8080:8080 andyfaizan/tomcat-dev &

cd '..'

mvn clean

if [ "$?" -ne 0 ]; then
    echo "Maven clean unsuccessful!"
    exit 1
else
  echo "Maven clean successful"
  mvn tomcat7:redeploy
  if [ "$?" -ne 0 ]; then
      echo "Maven deploy unsuccessful!"
      exit 1
  else
    echo "Maven deploy successful"
  fi
fi
```

LISTING 29: Script for Deployment

Bibliography

[1] A. Faizan, S. Lohmann, and V. Modi, "Multiple choice question generation for slides", in *The Computer Science Conference for the University of Bonn Students*, 2017.

[2] T. Berners-Lee, J. Hendler, O. Lassila, *et al.*, "The semantic web", *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data", *The semantic web*, pp. 722–735, 2007.

[4] T. R. Gruber, "A translation approach to portable ontology specifications", *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[5] M. Synak, M. Dabrowski, and S. R. Kruk, "Semantic web and ontologies", in *Semantic Digital Libraries*, Springer, 2009, pp. 41–54.

[6] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge", in *Proceedings of the 16th international conference on World Wide Web*, ACM, 2007, pp. 697–706.

[7] G. A. Miller, "Wordnet: A lexical database for english", *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[8] M. Wick, "Geonames", *GeoNames Geographical Database*, 2011.

[9] E. Prud, A. Seaborne, *et al.*, "Sparql query language for rdf", 2006.

[10] G. G. Chowdhury, "Natural language processing", *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[11] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction", in *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[12] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "Dbpedia spotlight: Shedding light on the web of documents", in *Proceedings of the 7th international conference on semantic systems*, ACM, 2011, pp. 1–8.

[13] R. Usbeck, M. Röder, A.-C. Ngonga Ngomo, C. Baron, A. Both, M. Brümmer, D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann, *et al.*, "Gerbil: General entity annotator benchmarking framework", in *Proceedings of the 24th International Conference on World Wide Web*, ACM, 2015, pp. 1133–1143.

[14] R. Mitkov and L. A. Ha, "Computer-aided generation of multiple-choice tests", in *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing-Volume 2*, Association for Computational Linguistics, 2003, pp. 17–22.

[15] A. Hoshino and H. Nakagawa, "A real-time multiple-choice question generation for language testing: A preliminary study", in *Proceedings of the second workshop on Building Educational Applications Using NLP*, Association for Computational Linguistics, 2005, pp. 17–20.

[16] J. C. Brown, G. A. Frishkoff, and M. Eskenazi, "Automatic question generation for vocabulary assessment", in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2005, pp. 819–826.

[17] H. Kunichika, T. Katayama, T. Hirashima, and A. Takeuchi, "Automated question generation methods for intelligent english learning systems and its evaluation", in *Proc. of ICCE*, 2004.

[18] Y.-C. Lin, L.-C. Sung, and M. C. Chen, "An automatic multiple-choice question generation scheme for english adjective understanding", in *Workshop on Modeling, Management and Generation of Problems/Questions in eLearning, the 15th International Conference on Computers in Education (ICCE 2007)*, 2007, pp. 137–142.

[19] M. Liu, R. Calvo, and V. Rus, "Automatic question generation for literature review writing support", in *Intelligent Tutoring Systems*, Springer, 2010, pp. 45–54.

[20] S. Smith, P. V. S. Avinesh, and A. Kilgarriff, "Gap-fill tests for language learners : corpus-driven item generation", *English*, 2010.

[21] V. K. Chaudhri, P. E. Clark, A. Overholtzer, and A. Spaulding, "Question generation from a knowledge base", in *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2014, pp. 54–65.

[22] M. Agarwal and P. Mannem, "Automatic gap-fill question generation from text books", in *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, Association for Computational Linguistics, 2011, pp. 56–64.

[23] G. Kumar, R. E. Banchs, and L. F. D'Haro Enriquez, "Revup: Automatic gap-fill question generation from educational texts", Association for Computational Linguistics, 2015.

[24] H. Ali, Y. Chali, and S. a. Hasan, "Automatic question generation from sentences", *Proceedings of TALN 2010*, pp. 19–23, 2010.

[25] D. L. Lindberg, "Automatic question generation from text for self-directed learning", PhD thesis, Applied Sciences: School of Computing Science, 2013.

[26] L. Becker, S. Basu, and L. Vanderwende, "Mind the gap: Learning to choose gaps for question generation", in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2012, pp. 742–751.

[27] A. Narendra, M. Agarwal, and R. shah, "Automatic cloze-questions generation.", in *RANLP*, 2013, pp. 511–515.

[28] A. Papasalouros, K. Kanaris, and K. Kotis, "Automatic generation of multiple choice questions from domain ontologies.", in *E-Learning*, Citeseer, 2008, pp. 427–434.

[29] M. M. Al-Yahya, "Ontoque: A question generation engine for educational assesment based on domain ontologies", in *ICALT*, 2011.

[30] L. Bühmann, R. Usbeck, and A.-C. N. Ngomo, "Assess—automatic self-assessment using linked data", in *International Semantic Web Conference*, Springer, 2015, pp. 76–89.

[31] D. Seyler, M. Yahya, and K. Berberich, "Knowledge questions from knowledge graphs", *ArXiv preprint arXiv:1610.09935*, 2016.

[32] A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann, and D. Gerber, "Sparql2nl: Verbalizing sparql queries", in *Proceedings of the 22nd International Conference on World Wide Web*, ACM, 2013, pp. 329–332.

[33] F. M. Suchanek, J. Hoffart, E. Kuzey, and E. Lewis-Kelham, "Yago2s: Modular high-quality information extraction with an application to flight planning.", in *BTW*, vol. 214, 2013, pp. 515–518.

[34] A. Kilgarriff and C. Fellbaum, *Wordnet: An electronic lexical database*, 2000.

[35] A. Thalhammer, N. Lasierra, and A. Rettinger, "Linksum: Using link analysis to summarize entity data", in *International Conference on Web Engineering*, Springer, 2016, pp. 244–261.

[36] A. Thalhammer and A. Rettinger, "Pagerank on wikipedia: towards general importance scores for entities", in *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*, Cham: Springer International Publishing, Oct. 2016, pp. 227–240, ISBN: 978-3-319-47602-5. DOI: 10.1007/978-3-319-47602-5_41. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47602-5_41.

[37] G. Kurdi, B. Parsia, and U. Sattler, "An experimental evaluation of automatically generated multiple choice questions from ontologies", in *International Experiences and Directions Workshop on OWL*, Springer, 2016, pp. 24–39.