Masarykova univerzita
Fakulta informatiky

Automatic Question Generation and Adaptive Practice

Bachelor thesis

Tomáš Effenberger

Brno, spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Tomáš Effenberger

Advisor: RNDr. Jan Rygl

Acknowledgement

I would like to thank RNDr. Jan Rygl for his guidance, support and also for his wisdom and infinite kindness. I would also like to thank members of the Natural Language Processing Centre and Adaptive Learning group for fruitful discussions, and all my friends who helped me to test the developed application and came up with a lot of great suggestions for future development.

Abstract

The thesis focuses on automatic question generation from text. Various approaches based on methods of artificial intelligence and natural language processing are described and the possibility of adaptive practice in the settings of automatically generated questions is also discussed. The thesis suggests a design of a modular framework for practicing knowledge from articles on Wikipedia. The framework has been implemented and is publicly accessible through a web interface.

Keywords

question generation, adaptive practice, knowledge representation, knowledge extraction, learning, multiple choice questions

Contents

1 Introduction

Repeated reading of the text is an inefficient method of learning, while answering to the related questions leads to better understanding and remembering [RP12]. Creating questions for a given topic is a time-consuming task and it is not possible to create a set of questions for each topic manually. However, we can generate questions automatically, using techniques of artificial intelligence and natural language processing. This thesis explores the state-of-the-art approaches to question generation and describes their advantages and disadvantages.

In spite of an active research in the field of question generation (boosted by the question generation shared task and evaluation challenge in 2009 [VG09]), there is still no publicly available web application to solve this task (as of April 2015). An online application for question generation would be invaluable. Typical example of usage is to make self-studying more efficient by answering a few generated questions after reading an article to verify and consolidate the knowledge acquired from the text.

That is why I have implemented the *Smartoo, Smart Artificially Intelligent Tutor*, modular and extensible framework for question generation and adaptive practice of knowledge from the *Wikipedia*[1] articles. The Smartoo framework is licensed under the GNU General Public License, version 2. The source code is available from its *GitHub repository*[2]. The prototype behavior has been deployed and is publicly available[3].

The thesis text is structured as follows. In chapter 2, I discuss the knowledge extraction from the text, which might precede the actual question generation. The question generation, together with the issue of ranking and filtering of the generated questions are the subject of chapter 3. In chapter 4, I talk about some common strategies for the adaptive practice, which is essential for a good learning experience. In chapter 5, I describe the implemented Smartoo framework and chapter 6 contains details about its deployment and evaluation. Finally, in chapter 7, I present planned future development of the Smartoo framework.

1. http://en.wikipedia.org/
2. http://github.com/effa/smartoo
3. http://smartoo.thran.cz

2 Knowledge Extraction and Representation

Instead of creating exercises directly from the text in natural language (unstructured data), one can at first attempt to extract the meaning from the text, i.e. convert the text into the structured data. This process is called *information extraction* [BKL09, p. 262]. Information extraction has several subtasks, the two most relevant for us are:

- *named entity recognition* – extraction of terms (e.g. *Abraham Lincoln*) and numeric expressions (e.g. *12 February 1809*) (more detailed definition is in section 2.3),
- *relation extraction* – extraction of relations between the discovered named entities, for example *birhDate(Abraham Lincoln, 12 February 1809)* (section 2.4).

One way to create exercises is to first perform the relation extraction and then create the exercises from built structured knowledge representation. Another approach is to generate exercises from text with discovered named entities. We will take a closer look at both of these approaches in chapter 3.

Before the discussion of the named entity recognition and relation extraction, I will talk about how to represent the extracted knowledge. Knowledge can be represented in many ways, such as using first order logic, semantic nets or frames [BL04]. In the following section, I will describe *RDF graphs*, since we will use this knowledge representation in other chapters.

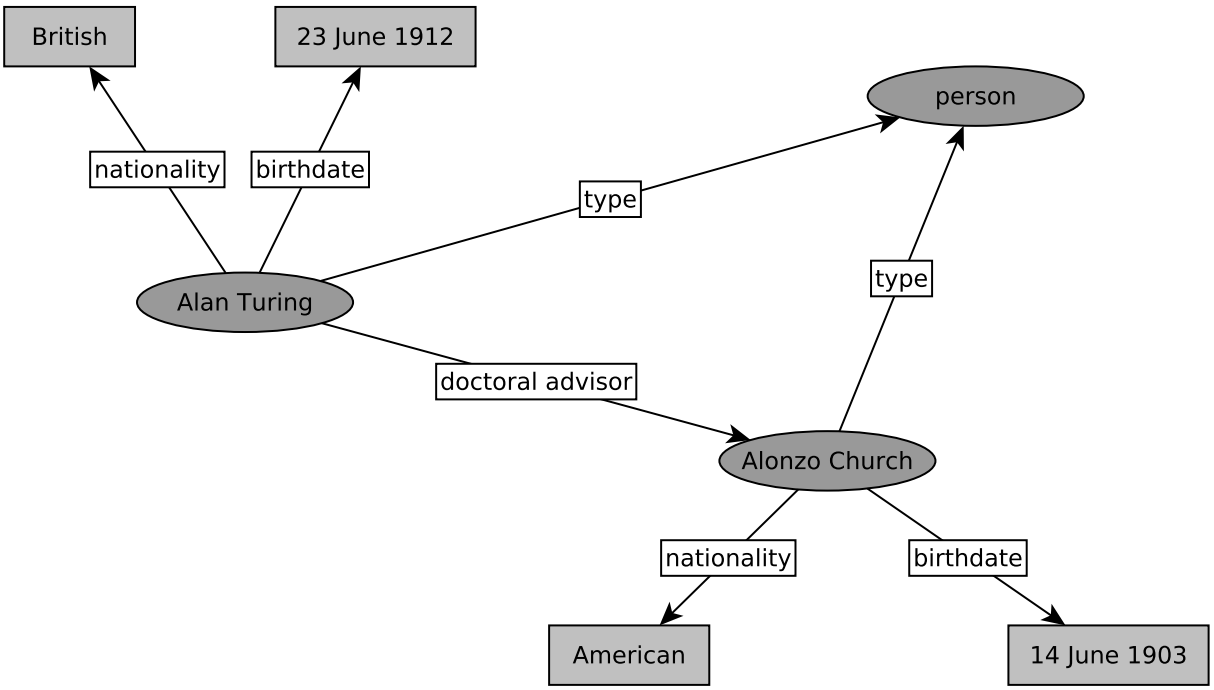

Figure 2.1: Simple RDF graph

2.1 RDF Graph

RDF[1] graph is a collection of subject-predicate-object triples [SET09, p. 63]. Each triple is called a *fact*. Simple example of a few facts about Alan Turing and Alonzo Church follows:

```
Alan Turing - type - person
Alan Turing - birthdate - 23 June 1912
Alan Turing - nationality - British
Alan Turing - doctoral advisor - Alonzo Church
Alonzo Church - type - person
Alonzo Church - nationality - American
Alonzo Church - birthdate - 14 June 1903
```

Figure 2.1 shows how can be this collection of facts thought of as a directed graph. Vertices correspond to subjects and objects. Each fact maps to one edge, with direction from the subject to the object and with edge value given by the predicate.

To avoid possible ambiguity (e.g. between two people who have both name *Alan Turing*) unique URIs[2] are used to represent subject, objects and predicates. URI consists of a prefix and a name:[3]

Objects can be either URIs or literals, such as numbers, strings or dates (e.g. *1912-06-23*). Simple example with prefixes (part of the graph above) follows:

```
<http://dbpedia.org/resource/Alan_Turing>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://xmlns.com/foaf/0.1/Person> .
<http://dbpedia.org/resource/Alan_Turing>
    <http://dbpedia.org/ontology/birthDate>
    1912-06-23+02:00
<http://dbpedia.org/resource/Alan_Turing>
   <http://dbpedia.org/property/nationality>
   "British"@en
```

The used format is called *N-Triples* [SET09, p. 68]. There are several others serializations of RDF graph, such as *RDF/XML* [SET09, p. 73] or *Turtle (Terse RDF Triple Language)*, which makes the serialization shorter by grouping facts by subjects (possibly also by predicates) and by using shortcuts for prefixes. The same graph as above would look like this in the Turtle syntax:

1. *Resource Description Framework*. See http://www.w3.org/standards/techs/rdf for RDF standards.
2. *Uniform Resource Identifier*. See RFC 3986 (https://tools.ietf.org/html/rfc3986) for details.
3. Different entities can have same name, for example `http://dbpedia.org/resource/Apple` and `http://smartoo.com/term/Apple`.

```
@prefix rsrc: <http://dbpedia.org/resource/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>
@prefix dbprop: <http://dbpedia.org/property/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>

rsrc:Alan_Turing
  rdf:type foaf:person ;
  dbpedia-owl:birthDate 1912-06-23+02:00 ;
  dbprop:nationality "British"@en .
```

RDF graph is great for a representation of simple facts (*declarative knowledge*), but it is useful to know about its limitations. In particular, RDF graphs are not designed to capture *procedural knowledge*, such as mathematical problem solving.

2.2 SPARQL

SPARQL[4] is a query language for convenient querying of RDF graphs [SET09, p. 84]. The most common query is the *select query* which resembles SQL select query and has the following main parts:

- **prefix** – definition of prefixes and associated shortcuts,
- **from** – dataset (RDF graph) on which to run the query,
- **select** – which information to return,
- **where** – condition on returned data,
- **order by** – ordering of results.

Variables are prefixed by a "?". As an example, have a look at the following SPARQL query, which would return the birthdate and birthplace of Alan Turing:

```
PREFIX rsrc: <http://dbpedia.org/resource/>
PREFIX owl: <http://dbpedia.org/ontology/>
SELECT ?date ?place
WHERE {
  rsrc:Alan_Turing owl:birthDate ?date ;
                   owl:birthPlace ?place .
}
```

More examples of SPARQL queries can be found in the section about creating exercises from knowledge graphs (section 3.3).

4. The name of this query language is a recursive acronym of *SPARQL Protocol and RDF Query Language*.

2.3 Named Entity Recognition

Named entities are "information units like names, including person, organization and location names, and numeric expressions including time, date, money and percent expressions" [NS07]. Sometimes, I will specifically refer to *terms*, which are named entities other than numeric expressions. As an example, *Abraham Lincoln* is a term, while *21 March 2015* is a numeric expression. The aim of the *named entity recognition* is to discover all named entities in a given text and label them with their types. Common types of named entities are organization, person, location, date, time, money, percent, facility and geo-political entity [BKL09, p. 281]. There are several approaches to the task of named entity recognition:

- **Regular expressions**
  Regular expressions are especially good for numeric expressions as they typically follow reasonably simple rules. As an example, consider the following regular expression (in Python dialect[5]) to capture dates such as *21 March 2015*:

```
(?P<date>
    # day (one or two digits followed by a space)
    \d{1,2} \s
    # month (followed by a space)
    (?: January | February | March | April | May | June | July |
        August | September | October | November | December) \s
    # year (four digits)
    \d{4}
)
```

- **List of terms**
  A simple way to find if some word or phrase is a named entity is to try to look it up in a list of named entities. There are for example publicly available *gazetteers* – lists of well-known geographical locations. However, this method is quite limited as it is impossible to have complete lists of all named entities. (In particular there is infinitely many numeric expressions.) Another problem is ambiguity of words, e.g. *Reading* is a town in the United Kingdom, but not in the sentence *Reading is important for everyone's personal development.* There is one exception when this method works great – if there is a known list of terms which can appear in the text. For example, for each article on Wikipedia the list of terms can be built from internal links which appear in the article. I use this fact in the Smartoo (section 5.2).
- **Frequency counting**
  After identifying nouns and noun phrases in the text, their relative frequencies are calculated and these which exceed a given threshold are considered as terms. This method is used by Mitkov [MAK06]. It works reasonable well for terms recognition, but it can't discover numeric expressions. This method can unfortunately also label as terms common names which are frequently used. A solution to this problem is to apply *tf-idf weighting* which normalizes term frequency in the document by its frequency in all documents (or in common texts) [MRS09, p. 118].

5. https://docs.python.org/2/library/re.html

- **Machine learning**
  The most difficult, but also the most powerful method is to use a machine learning-based classifier. Traditionally, supervised learning dominated in this field, but as it requires a lot of annotated sentences, novel approaches using semi-supervised or even unsupervised learning are developed [NS07]. Semi-supervised learning is often based on the method of *bootstrapping* – a few named entities of each type are given to the system, the system finds them in a large unannotated corpus and uses their context to find other similar named entities, and the whole process is repeated many times with increasing number of discovered named entities and contexts.

Often, several approaches are combined, e.g. *The Mentor* (a question generation tool) uses regular expressions, gazetteers and a machine learning recognizer [MCC11]).

2.4 Relation Extraction

Having discovered named entities, we can extract relations between them. For simplicity, we restrict our attention to binary relations, i.e. relations of the form *relation(a, b)*, for example *birhDate(Abraham Lincoln, 12 February 1809)*. Relations of this form can be directly stored in a RDF graph as a subject-predicate-object triple *(a, relation, b)*.

One way to tackle this task is to consider each pair of named entities which are in the same sentence and decide in which relation they are (or that they are not in any relation) according to words between them [BKL09, p. 284]. As an example, consider the following pattern for *birthDate* relation between a person and a date:

```
Pattern: PERSON was born .* (in | on) DATE
                          ↓
Relation: birthDate(PERSON, DATE)
```

This pattern covers all of the three following cases:

- *Abraham Lincoln was born in 1809.*
- *Abraham Lincoln was born on 12 February, 1809.*
- *Abraham Lincoln was born in Hodgenville, Kentucky, on 12 February, 1809.*

Relation extraction can be also formulated as a classification problem: given a sentence with two named entities a and b and a particular relation type r, decide, if a and b are in the relation r [BB07]. If there are enough positive and negative labeled examples for each relation of our interest, supervised learning can be used, with features such as types of the named entities, words between them, number of words between them, path between them in a syntactic tree of the sentence, etc. [Kam04]. As an alternative to the feature based methods, which take a vector of extracted features as an input, *kernel methods* are sometimes used, which take the full representation of the sentence, possibly with some structural information, as an input [ZAR03]. As creating a large labeled training sets is usually expensive, semi-supervised learning became popular as well, using the bootstrapping method (which was described in the previous section) [Car+10].

The mentioned approaches work only if the relation types are known in advance, so that patterns can be written or a classifier trained for each of the relation. For unrestricted domain, we can try to find the relation type in the sentence itself. For example, from a simple sentences with a subject, a verb and an object, relation *verb(subject, object)* may be extracted [AFF14]. However, sentences are often quite complex, so this method requires to preprocess and simplify the text in order to work.

2.5 Knowledge Bases

There are several huge publicly available knowledge bases, which try to capture as many facts about the world as possible. Examples of popular knowledge bases are *DBpedia* [Leh+14] and *Freebase* [Bol+08]. Knowledge bases are used for many task in natural language processing, such as named entity disambiguation [Men+11], semantic search and question answering [Fer+10]. And we can make use of the information in these knowledge bases to create better exercises as well, for example to select appropriate incorrect alternatives in multiple choice questions (see section 3.4).

DBpedia was built by extracting structured information from Wikipedia, such as titles, links between articles, external links, information from infoboxes and categories. Facts are stored in the form of an RDF graph. Whole knowledge base can be be downloaded[6], and it is also possible to use public *DBpedia SPARQL endpoint*.[7]

DBpedia Version 2014 contains approximately 580 million facts (RDF triples) retrieved from the English Wikipedia, describing 4.5 million of things [Leh+14]. Ontology[8] of DBpedia includes more than 650 classes and 2700 properties. In Figure 2.2, a fraction of DBpedia class hierarchy is shown.

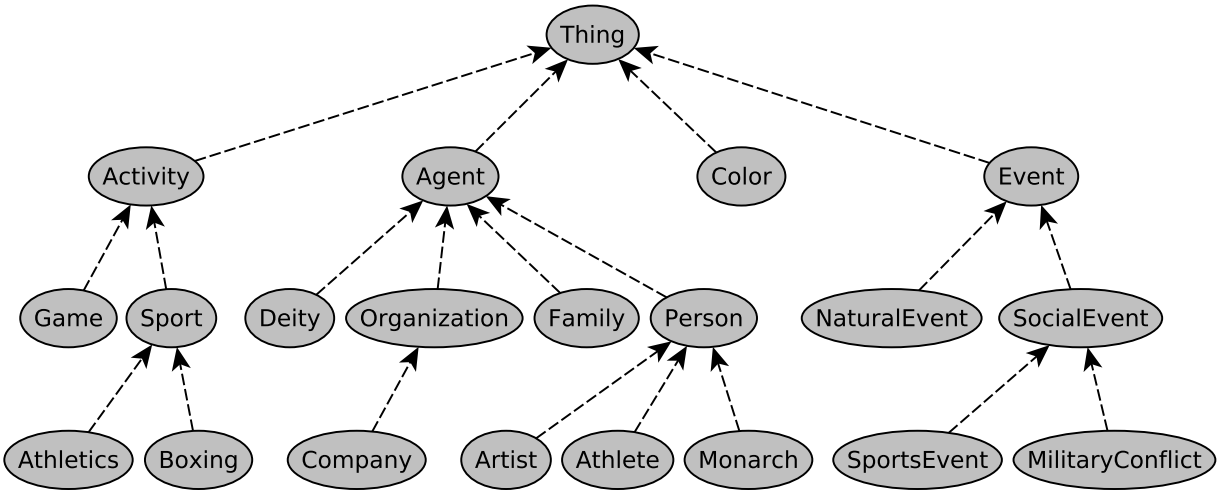

Figure 2.2: A fraction of DBpedia class hierarchy. Edges represent subclass relationship.

6. http://wiki.dbpedia.org/Downloads
7. http://dbpedia.org/sparql
8. Ontology is a description of properties, classes and their hierarchy.

3 Question Generation

The first question generation tool was introduced as early as in the 1970s [Wol76] and up to now (2015), several approaches to the task of question generation were proposed. Every system uses different methods and although usually the authors perform some kind of evaluation (e.g. by two people ranking generated questions as good or bad), it is not possible to say which approach is the best as the systems work in different settings, use different source texts from which the questions are generated, create different question types, have different target groups and use different evaluation metrics. For example, some systems try to assess vocabulary [BFE05] or reading comprehension [HS09], some focus on factual texts [Hei11] and sometimes authors use for evaluation only a single textbook or a topic, such as linguistics [MAK06] or biology [AM11].

Based on reported performances of the systems, it can be concluded that question generation is still an open problem in the natural language processing and there is plenty of room for improvements or even completely new ideas. Some systems proved to be usable in the offline mode, in which a teacher submits a text to the system, the system outputs a list of questions and it is assumed that the teacher will filter them and make some corrections in the generated questions. Ideally, there is only a few questions that need to be discarded or corrected and so this semi-automated process of creating questions is faster than creating them fully manually [MAK06].

Although the chapter is called *question generation*, it is possible to create more complex exercises as well. Broader range of exercises is especially useful if we want not only to assess one's knowledge, but rather to deepen his or her remembering and understanding of the text. Generally, people are more likely to remember things that entertain them. However, complex exercises are more difficult to implement and evaluate, so usually only questions are generated, most often multiple choice questions. *Multiple choice questions* consist of a question text (called *stem*) and a few (e.g. four) *choices*, from which one is the *correct answer* and the rest are incorrect alternatives (called *distractors*). Special type of multiple choice question is *fill-in-the-blank multiple choice question*, in which the stem is a sentence with a blank space which should be filled with the correct answer.

Although the approaches to the question generation are diverse, some common tasks can be identified throughout the systems. The general workflow has the following 5 steps, although some of them may be missing in some systems:

1. First, text of the article is pre-processed. Pre-processing can range from mere sentence filtering, to text rephrasing, and even to knowledge extraction from the text and building a knowledge graph (section 3.1).
2. The text is then scanned for the patterns which define transformations from sentences to question-answer pairs. Every time a pattern is matched, a new question and the corresponding answer are generated (section 3.2). If a knowledge graph is used instead of text, the procedure is similar, except that now we have transformations from facts in the knowledge graph. The main advantage is that we can make use of relationships between subjects in the graph and thus create questions which span more then one sentence in the original article (section 3.3).

3. In case of multiple choice questions, suitable distractors (incorrect alternatives) are selected (section 3.4).
4. In the previous steps, questions are usually overgenerated, so it is important to filter them at the end (section 3.5). Instead of direct filtering, we can rank the questions and either select some predefined number of questions with the highest rank or employ adaptive practice. In adaptive practice, decisions about which question is next presented to the student are not made in advance, but during the practice itself, based on already answered questions and success of the student. (Adaptive practice is discussed in detail in chapter 4.)

3.1 Text Pre-processing

Usually, the text of the document is pre-processed before it is used for question generation. Pre-processing may add useful information to the text (e.g. part-of-speech tags, semantic role labels), remove useless sentences (e.g. sentences without information about the topic), modify (rephrase) sentences or even create different (more machine readable) representation of the knowledge contained in the article, such as a knowledge graph.

- **Parsing and labeling**
  The text is split into sentences and each sentence is tokenized (*tokens* are words and punctuation). Afterwards, each sentence might be *syntactically parsed* (its syntax tree is built) or at least *shallow-parsed* (each token is assigned a part-of-speech tag). Named entities recognition or even semantic role labeling might be performed (see section 3.2 for details).
- **Filtering**
  The goal of the filtering is to discard sentences which are not relevant to the article or has not enough context to generate a question from them. Typical example of a sentence without sufficient context are *anaphoric references*, i.e. referring back to something mentioned earlier in the text. For example, in the sentence "He was the 16th President of the United States.", the word "he" refers to Abraham Lincoln. Some possible heuristics to detect which sentences cannot be used to generate good questions are [Wol76]:
    - sentence is too long (e.g. more than 35 words),
    - contains demonstrative word (such as *this* and *that*),
    - starts with a verb.

  Instead of heuristics, we can use a more sophisticated machine learning-based classifier to select only the sentences which are informative, relevant to the topic and have enough context so that questions can generated from them. This approach was tested by Agarwal [AM11]. Some of the features used in his classifier were:
    - the number of words in the sentence,
    - the number of nouns,
    - the number of words from the headline,

  - contains abbreviation? (binary feature),
  - contains superlativum? (binary feature).

- **Text rephrasing**
  Ideal text for question generation would contain exactly one fact in one sentence, but in reality most of the sentences are either too complex (consisting of many clauses) or they lack important context (e.g. due to using pronouns). We might want to transform (rephrase) the original text into simple factual sentences [Hei11; Fos09], for example split complex sentences into simple ones, remove unimportant information (such as content of brackets) and resolve anaphoras.

- **Knowledge extraction**
  Another strategy is not to create questions directly from the text, but rather from a knowledge graph built from the article by the process of knowledge extraction (as described in section 2.4).

3.2 Sentence Transformations

Most question generation tools create questions directly from the text without the knowledge extraction step. Usually, the system considers every sentence one by one and tries to transform it to a question. Unless the filtering in the pre-processing step is extensive, considering every sentence may lead to questions which are not much relevant, but this issue can be solved later by questions ranking and post-filtering (section 3.5).

There might be more possible transformations from a sentence to a question-answer pair. Instead of generating all possible questions from the sentence (or picking one of them by random), we might select a word from the sentence which would be the correct answer as a first step and then generate a question from the rest of the sentence as a second step. Agarwal [AM11] employed a machine learning-based classifier to select the most important word in the sentence. For the classification he used the following three features:

- number of occurrences of the word in the document,
- is contained in the headline? (binary feature),
- depth of the word in the parse tree of the sentence.

Transformation rules describe how to create a question-answer pair from a sentence which satisfies a given pattern, such as *The sentence starts with a term followed by a verb and an object*. These transformation rules may be either hard-coded into the program (*imperative rules*) [Wol76; HS09; ASM11] or stored in the external templates (*declarative rules*) [WP09; MN14]. The latter case is more difficult to implement, but it allows for a simple modification of transformation rules and addition of new ones.

Patterns may be defined on syntactic or semantic level. Many approaches are somewhere in between these two extremes and use syntactic and a little bit of semantic information (such as which words denote terms), but do not rely on complete semantic roles labeling. Syntactic information used in patterns include:

- Words in the sentence, such as *Pablo*, *Picasso*, *was*, *born*, *in*, *Spain* or their canonical forms (*lemmas*) – *Pablo*, *Picasso*, *be*, *bear*, *in*, *Spain*.
- *Part-of-speech (POS) tags* – lexical categories (also known as word classes), such as noun, verb, adjective or determiner. Using *The University of Pennsylvania (Penn) Treebank Tag-set* [San90], the sentence *Pablo Picasso was born in Spain.* would be POS-tagged as follows.

```
[NNP Pablo] [NNP Picasso] [VBD was] [VBN born] [IN in] [NNP Spain]
```

  where NNP stands for proper noun, VBD for past tense verb, VBN for past participle verb and IN for preposition.
- *Syntax tree* (also called *parse tree*) – tree representation of the deep structure of a sentence. On the Figure 3.1, syntax tree for the sentence from previous paragraph is shown. Besides POS-tags, the tree contains tags for sentence (S), noun phrase (NP), verb phrase (VP) and prepositional phrase (PP).

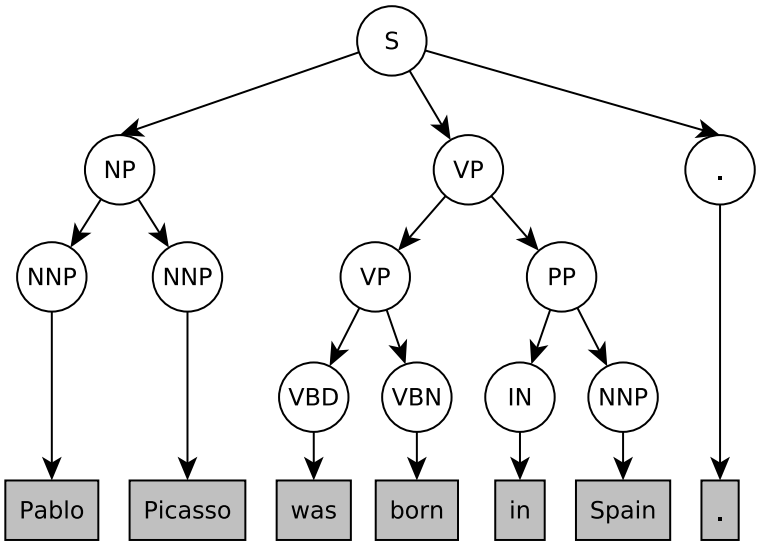

Figure 3.1: Syntax tree

Semantic information which can be used to define patterns is:

- Named entities and their types, discovered by named entity recognition described in section 2.3. Example for the same sentence as in the paragraph above:

```
[Person| Pablo Picasso] was born in [Location| Spain]
```

- *Semantic role labels* which mark predicate of the sentence and its arguments – agent (subject), patient (object), cause, purpose, manner, location, time etc. Semantic roles can be found using a statistical classifier [GJ02].

```
[agent| Pablo Picasso] [predicate| was born] [location| in Spain]
```

- Semantic relations between words, such as *hypernymy*, i.e. being a super-class (e.g. *animal* as a hypernym for *dog*). These relations between words can be found in *WordNet* [Mil95].

Examples of transformation rules

The simplest pattern is for the fill-in-the-blank questions (used for example by Agarwal and Mannem [AM11]):

Pattern: beforeTerm term afterTerm.

↓

Question: beforeTerm _____ afterTerm
Answer: term

Sentence: *The Southern Chinese martial art Hung Ga is based on the movements of the tiger and the crane.*
Question: *The Southern Chinese martial art _____ is based on the movements of the tiger and the crane.*
Answer: *Hung Ga*

Typical example of a mostly syntactic approach, with a little bit help of the semantics (using WordNet) to create the final question [MAK06]:

Pattern: [NP subject] [VP verb] [NP object].

↓

Question: *Which* hypernym(subject) verb object?
Answer: subject

Sentence: *Logic studies reasoning.*
Question: *Which discipline studies reasoning?*
Answer: *logic*

The following example shows that sometimes, questions might be generated just from a phrase, instead of the full sentence and it also makes use of named entities [MCC11].

Pattern: [NP object] [VBN predicate] *by* [Human subject]

↓

Question: *Who* pastTense(predicate) object?
Answer: subject

Sentence: *... Hamlet, written by Shakespeare ...*
Question: *Who wrote Hamlet?*
Answer: *Shakespeare*

Some connectives (because, so that, since, when, for example, ...) can be used to generate questions [ASM11; Wol76].

Pattern: A, *because* B.

↓

Question: *Why* questionForm(A)?
Answer: Because B.

Sentence: *Photosynthesis is important, because it produces oxygen.*
Question: *Why is photosynthesis important?*
Answer: *Because it produces oxygen.*

Function `questionForm(A)` searches for an auxiliary verb in `A` and if it finds one, it moves the auxiliary verb to the front. If there is no auxiliary verb in the sentence, it adds appropriate auxiliary verb to the beginning of the sentence based on the tense of the main verb and the main verb is substituted for its base form (bare infinitive). If the main verb is *to be* (as in our example), it is moved to the front without adding any auxiliary verb.

As mentioned above, semantic role labels might be used to describe patterns [MPJ10; MN14].

Pattern: *By* manner, agent predicate patient.
Filter: manner *contains gerundium*

↓

Question: *How* questionForm(agent, predicate) patient*?*
Answer: *by* manner

Sentence:	*By reducing halting problem to problem X, one can show undecidability of the problem X.*
Question:	*How can one show undecidability of the problem X?*
Answer:	*by reducing halting problem to problem X*

Even more sophisticated patterns can be created using syntax trees [WP09; HS09].

In most tools, transformation rules were created manually. However, machine learning can be put into use to discover them automatically. *The-Mentor* system [MCC11] uses a machine learning technique of *bootstrapping* [RH02], together with the *QuestionBank*, a corpus of 4000 parse-annotated questions [JCG06], and the large amount of the information on the Web via *Google* search to find syntactic patterns automatically from a few question-answer seed pairs. They have discovered more than 1000 unique patterns, which is many more than tools with manually created patterns use. This allows for more specific patterns, such as the following:

Pattern: [NP subject] [VBD aux] [VBN verb] *in* [Location:Country| X].

↓

Question: *In which country* aux subject verb*?*
Answer: X

Sentence:	*Pablo Picasso was born in Spain.*
Question:	*In which country was Pablo Picasso born?*
Answer:	*Spain*

3.3 Creating Exercises from Knowledge Graph

In this section, we will consider the situation when we have built a knowledge graph and want to create exercises from it, instead of from the original text. Similarly as with the text, the question generation tool can take a fact one by one and transform it to the question if possible. For example, fact `(Alan Turing, birthDate, 23 June 1912)` leads to the question *When was Alan Turing born?* with the correct answer *1912*. However,

there are some predicates which are not suitable for creating questions from them, such as `rdfs:label` or `rdf:comment` and these need to be filtered.

This approach of transforming a single fact at a time was used in *OntoQue* [AlY11]. OntoQue supports 3 types of exercises: multiple choice question, true/false and fill-in-the-blank. Fill-in-the-blanks are created simply by removing subject or object from the fact. Multiple-choice-questions are fill-the-blanks with 4 provided choices. As distractors, subjects or objects of the same type as the removed one are used. Finally, true statements in true/false questions are just unmodified facts from the knowledge graph, while false statements are created by replacing subject or object for a different one (suitable alternative is found in the same way as the distractors for multiple choice questions).

OntoQue uses `rdf:label` property to transform facts into sentences, so the quality of the generated questions depends on the quality of the labels. Consider two situations shown in Figure 3.2. From the left graph, OntoQue could generate fill-in-the-blank question "Alan Turing was born in _____ ", while the right graph would be transformed into the question "Alan Turing birth date _____ ". Another possible problem could be with the answer, sometimes it needs to be modified (in our example, *1912-06-23* could be simplified to *1912*).

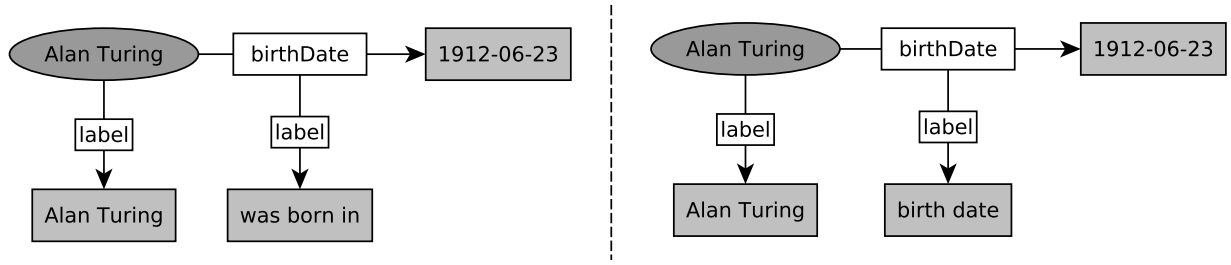

Figure 3.2: Labels

For this reason we might want to use more specific patterns to influence transformation from facts to sentences and simplify answers. For convenient retrieval of information from a knowledge graph, we can use SPARQL which was described in section 2.2. As a simple example, rule for "When was X born?" question would look like this:

```
PREFIX ont: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?subjectName ?birthDate
WHERE {
  ?subject ont:birthDate ?birthDate ;
      rdfs:label ?subjectName .
}
```

↓

Question: *When was* subjectName *born?*
Answer: parseYear(birthDate)

The following transformation would generate questions about start dates of all events in the knowledge graph.

```
PREFIX ont: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?eventName ?date
WHERE {
  ?event a ont:Event ;
    rdfs:label eventName ;
    ont:startDate ?date .
}
```

↓

Question: *When did* eventName *happen?*
Answer: date

But the great thing about knowledge graphs is that we can select more facts and then ask a question which spans more than one sentence, such as "Could hypothetically Leonhard Euler meet Carl Friedrich Gauss?" This transformation is described in the following example.

```
PREFIX ont: <http://dbpedia.org/ontology/>
SELECT ?name1 ?birth1 ?death1 ?name2 ?birth1 ?death2
WHERE {
  ?person1 a ont:Person ;
      rdfs:label name1 ;
      ont:birthDate ?birth1 ;
      ont:deathDate ?death1 .
  ?subject2 a ont:Person ;
      rdfs:label name2 ;
      ont:birthDate ?birth2 ;
      ont:deathDate ?death2 .
}
```

↓

Question: *Could hypothetically* name1 *meet* name2 *?*
Answer: bool(death2 $\geq$ birth1 $\wedge$ death1 $\geq$ birth2)

3.4 Distractor Selection

It is important to present a multiple-choice question with competitive *distractors*. As shown by experiments performed by Little and Bjork [LB15], multiple choice questions with competitive incorrect alternatives not only help to remember the correct answers to the original questions, but they also improve later performance on the previously non-tested questions for which these alternatives may be the correct answers. If the alternatives are not competitive, a student can recognize the correct answer without retrieval, just by pattern matching. As an example, consider exercise 3.1.

Clearly, the student does not need to remember who assassinated Lincoln to answer this question correctly and they are also not forced to retrieve any knowledge about alternatives.

Lincoln was assassinated by _____ , a Confederate sympathizer.

A. Emancipation Proclamation

B. John Wilkes Booth

C. Illinois

D. Department of Agriculture

Exercise 3.1: Question with noncompetitive alternatives

Of course, this was an extreme example to illustrate the point – you definitely would not come across such poor distractors in manually created multiple-choice questions. Compare the previous question to the exercise 3.2.

Lincoln was assassinated by _____ , a Confederate sympathizer.

A. Thomas N. Conrad

B. Robert E. Lee

C. John Wilkes Booth

D. Ward Hill Lamon

Exercise 3.2: Question with competitive alternatives

This questions is much more likely to force student to retrieve knowledge about John Wilkes Booth as well as about other alternatives. Little and Bjork also showed that competitive distractors are not confused with correct answers in later questions more often than noncompetitive distractors.

The consequences are clear – when creating a multiple-choice question, we should try to find distractors which are as competitive as possible, which is what most of the current question generation tools do [MAK06; AM11; MCC11]. Common strategies for distractor selection are:

- **Named entities from the text**
  One possibility is to use named entities from the text discovered by named entity recognition (section 2.3). As distractors, named entities of the same type are chosen (e.g. if the correct answer is person, choose people as distractors). This approach was used for example in the Mentor [MCC11]. As an alternative to common types, common context can be used to select appropriate distractors. Agarwal [AM11] uses features such as comparison of words and their POS tags before and after the correct answer and the distractor or frequency of shared words in the question sentence and a sentence with the distractor.

- **Subjects in the knowledge graph**
  In the knowledge graph, subjects play similar role as the named entities in the text. The difference is that usually one subject has many types. (For example, the record about *Abraham Lincoln* in DBpedia contains more than 50 types, including *agent*, *person*, *office holder*, *president of the United States* and *people murdered in Washington, D.C.*) Several strategies were proposed to select the most competitive distractors in the knowledge graph [AlY11; PKK08]. One of them is to select a subject of the same type as the correct answer. If there are many of them, we can prefer those which have more shared types.

- **Named entities from external sources**
  Using *WordNet* [Mil95] or a knowledge base, we can find similar terms, even if they are not in the text. For example, Mitkov et al. [MAK06] use *coordination relation* in WordNet (coordinates are words sharing at least one hypernym) to find appropriate distractors for given term. If more coordinates were found in WordNet, the ones which occur in the text are preferred.

- **Correct answer modification**
  If the correct answer is a numerical expression, we can easily obtain distractors by changing the date or value by some partially random relative shift or multiplication [PKK08]. For instance, if the correct answer is 1616, the distractors generated by multiplying the correct answer by 0.92, 0.95 and 1.04 would be 1487, 1535 and 1680.

All these strategies work only if the correct answer is a named entity or a single word. If the correct answer is a full clause (e.g. "Why is photosynthesis important?" – "Because it produces oxygen."), generating meaningful and competitive distractors is much harder. Simple (but not very good) way to find at least some distractors would be to use the correct answers of the generated questions of the same question type.

3.5 Question Ranking and Filtering

Common strategy in the question generation tools is to overgenerate questions and then filter them [MCC11; MPJ10; HS09; Wol76]. The goal of the filtering is to discard questions which are:

- invalid (e.g. because they contain anaphoric references, such as "Where is there?"),
- irrelevant to the topic,
- too easy (e.g. because of noncompetitive distractors).

As an alternative to direct filtering, we can rank questions first and then select N best questions (e.g. Mannem [MPJ10] selects 6 best questions per a paragraph). Ranking allows to separate the task of estimating parameters and the task of selecting questions for the student based on the already answered questions, i.e. it allows for adaptive practice (chapter 4). We can even separately rank various parameters of questions, such as probability of correctness (validity), relevance and difficulty.

Both filtering and ranking can be implemented either via heuristic decisions or via a machine-learning based classifier. Some possible heuristic criteria to discard generated questions are [Wol76]:

- Question or answer is too long.
- Question contains word "and", colon or quotes.
- The first word in the answer is a relative pronoun.

Mannem [MPJ10] uses heuristic ranking, with depth of the predicate from which is the question generated as a primary criterion (to penalize questions from subordinate clauses) and the number of pronouns as a secondary criterion (to penalize possible anaphoric references).

Heilman [Hei11] trained a machine learning-based classifier for question ranking using features such as:

- question length,
- which wh-word (who, where, when, ...) was used (if any),
- includes negation (no, not, never)? (binary feature),
- likelihoods of a question would appear in a text according to an unigram and a trigram language models,
- numbers of proper noun, pronouns, adjectives, etc.

The presented rankers are primarily concerned with correctness (validity) of generated questions. However, for the adaptive practice, difficulty ranking is necessary as well. Smartoo uses a simple heuristic for difficulty ranking, which is described in section 5.5.

4 Adaptive Practice

The goal of *computerized adaptive practice* [KSvM11] is to make the practice as efficient as possible. This is related to, but different from the *computerized adaptive testing* [vLG00], which aims at precise and reliable measuring of student's knowledge (or skill). There are many systems providing practice customized to the learner, these systems are called *intelligent tutoring systems* [Van06]. Some of them focus on the procedural knowledge (such as solving math problems) [KSvM11], other on adaptive practice of facts [PPS14].

For practice to be efficient, it is important to balance difficulty of the task with respect to the learner. If the practice is too easy, the learner doesn't learn anything new and quickly becomes bored. On the other hand, if it is too difficult, the learner becomes anxious and frustrated. Hence, the key of efficient adaptive practice is to find the optimal difficulty – Thomas Butt talks about "the sweet spot of learning" [But12]. Generally, the ideal state of absolute immersion, focus and enjoyment is called *flow* [Csi90].

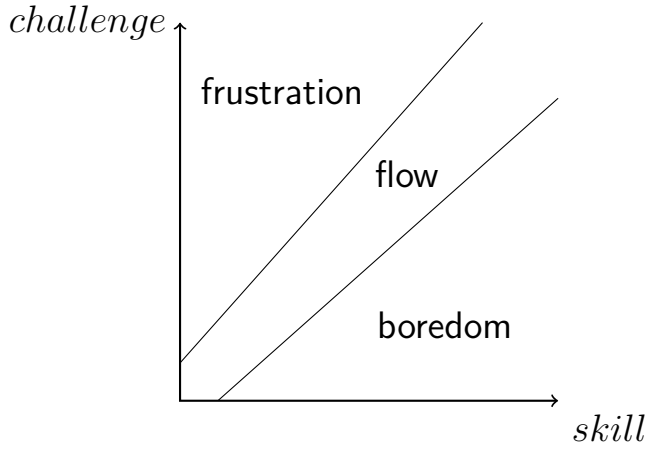

Figure 4.1: Relationship between challenge and skill.

Intelligent tutoring systems use answers (attempts) of learners to estimate both knowledge (skill) of each learner and the difficulty of all facts (tasks). As most people use these tutors repeatedly and not only once, the process of learning can be also captured and the systems may have special estimate parameter for each learner-fact pair [PPS14].

Our situation is, however, different, as there is no limited set of facts. In each practice session, tens (or even hundreds) of new questions are generated and we don't know much neither about the questions, nor about the learner's knowledge of the topic. As there is usually so many generated questions, we can't ask learner all of them, but carefully select just a few questions, which are the most relevant to the learner and are not too easy nor too difficult. The generated questions are brand-new, so we necessarily need to rank them for relevance and difficulty estimation (and possibly other attributes as well), otherwise there is no way to distinguish between generated questions.

Having these limitations in mind, we can now formulate the goal of the adaptive practice in the settings of the automatically generated questions. The system shall ask questions

which are relevant to the topic, ideally covering the whole article and which are not answerable without thinking. Furthermore, the system shall attempt to estimate the global knowledge of the student about the topic as quickly as possible and using this estimate, prefer adequately difficult questions, so that the student is neither bored by too easy questions, nor frustrated from too difficult questions.

In section 4.1, I describe several common models of a student used to estimate one's knowledge, then I briefly mention dynamic adjustment of target difficulty (section 4.2), and finally I discuss the selection of the best question (section 4.3).

4.1 Knowledge Estimation

The collection of persistent information about the student which is used to decide which skill to practice next is called the *student model* [Van06]. It may include information about which skills have been mastered by the student (and to which extent), or even enjoyment and motivation of the student [DB12].

In the settings of facts practicing, student model is used to estimate the probability that given student knows a given fact [PPS14]. The most common student models for factual knowledge are described in the following paragraphs.

- **Logistic model**
  One-parameter logistic model is the basic model of student in the *item response theory* [Par04]. This model uses logistic function with a parameter d, which denotes the difficulty of the fact. The probability, that the student with knowledge k knows a fact with difficulty d is given by the formula:

$$P(k,d) = \frac{1}{1+e^{-(k-d)}} \tag{4.1}$$

  The logistic function (Figure 4.2) is suitable for probability modeling, because it ranges from 0 to 1 for any possible input. In case of multiple choice questions, we need to adjust the formula a little bit to account for the possibility of guessing. With n options, the probability of correct guess is $\frac{1}{n}$.

$$P(k,d,n) = \frac{1}{n} + \left(1 - \frac{1}{n}\right) \cdot \frac{1}{1+e^{-(k-d)}} \tag{4.2}$$

  Given enough data, model parameters can be estimated using one of *maximum likelihood estimation* algorithm [dAya13]. However, since we use generated questions, we do not have any data about students' answers to them. That is why we need to have some heuristic estimates of difficulty. Student knowledge is then estimated using questions he or she already answered and their difficulties, according to *maximum likelihood principle*, i.e. the knowledge level is selected which maximizes the likelihood of the student history. Although this cannot be solved analytically, fast numeric methods can be used, e.g. *Newton-Raphson method* [Tho09].

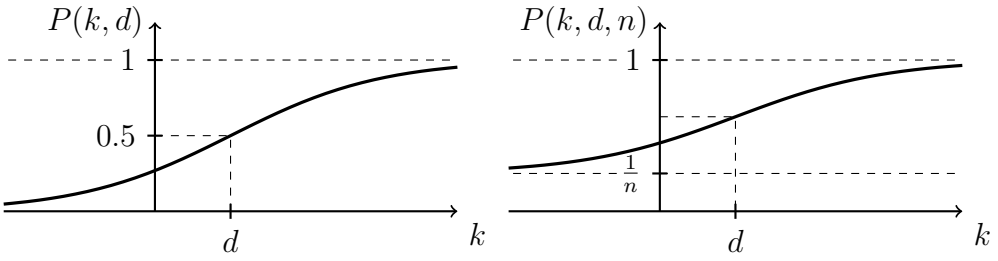

Figure 4.2: Logistic model without guessing (left) and with guessing (right)

- **Elo model**
  Elo model extends the logistic model to capture changing knowledge. The model is inspired by the rating of chess players [Elo78] and interprets each attempt of a student to answer an question as a "match" between the student and the question. According to the result R of the match (1 for correct and 0 for incorrect answer) and the expected result $P(k, d)$ (computed by the logistic function 4.1, or 4.2 for multiple choice questions) the knowledge k and difficulty d are updated as follows:

$$\begin{aligned} k &\leftarrow k + K(R - P(k, d)) \\ d &\leftarrow d - K(R - P(k, d)) \end{aligned} \tag{4.3}$$

  The difference $(R - P(k, d))$ expresses how much the result was unexpected. The K parameter influences the weight of the last match to the current estimate. Although the primary use of Elo model was to describe the process of learning, it can be also used for the gradual improvements of estimates for prior knowledge of students and difficulty of facts [PPS14]. However, the constant K is problematic – if it's too large, estimation is unstable; if it's too small, estimation converges too slowly. For this reason, instead of a constant, an attenuation function

$$K(m) = \frac{a}{1 + bm} \tag{4.4}$$

  (m is the order of the match, a and b are fixed parameters) can be used. The main advantage of using the Elo model to estimate the prior knowledge and difficulties in comparison with the maximum likelihood estimation algorithms is its simplicity and intrinsic online nature. Experiments suggest that it leads to similar results as the maximum likelihood estimation [Pel14].

- **Performance factors analysis**
  Performance factors analysis [PCK09] is also similar to the logistic model and enables to capture learning. According to this model, the probability that a student answers correctly to a question depends on the difficulty d of the question and number of correct (a) and incorrect (b) answers of the student to this question in the past. The exact formula is:

$$P(a, b, d) = \frac{1}{1 + e^{-k}} \quad \text{where} \quad k = \alpha \cdot a + \beta \cdot b - d \tag{4.5}$$

  The main drawback of this model is that it ignores the order of answers, so answering firstly 10 times incorrectly and then 10 time correctly leads to the same prediction

as answering alternately correctly and incorrectly (and while the first case suggests that the student has learned the item, in the second case it's clear that the student hasn't learned it already). To take the order of answers into account, performance factors analysis can be combined with the Elo model [PPS14] – after each answer, estimate of knowledge parameter k is updated according to the rule 4.3.

4.2 Target Difficulty Adjustment

As we mentioned in the introduction to this chapter, the key to the efficient practice is to provide the student with adequately difficult questions. Although the ideal success rate may vary for different people (and even for one person in different conditions), for simplicity it is usually treated as a constant and set e.g. on 75 % [PPS14; KSvM11].

Experiments performed in the online adaptive system for learning geography facts suggest that the optimal success rate may be 65–70 % [PP15]. This result was obtained by self-reports of students (after some number of answered questions, students were asked if the practice is "too easy", "appropriate", or "too difficult"); each student was assigned to one of 10 groups with different target probability of correct answer (from 50 to 95 %).

It was also noticed that the actual success rate is usually higher than the target probability of correct answer. To approach the desired target success rate more closely, an algorithm was proposed for dynamic adjustment of target difficulty [PP15]. It is based on the idea of *proportional controller*. If the actual success rate is lower than the target probability of correct answer, the target probability is increased (proportionally to the "error"). If the actual success rate is higher, the target probability is proportionally decreased. The exact adjustment function is shown in Figure 4.3.

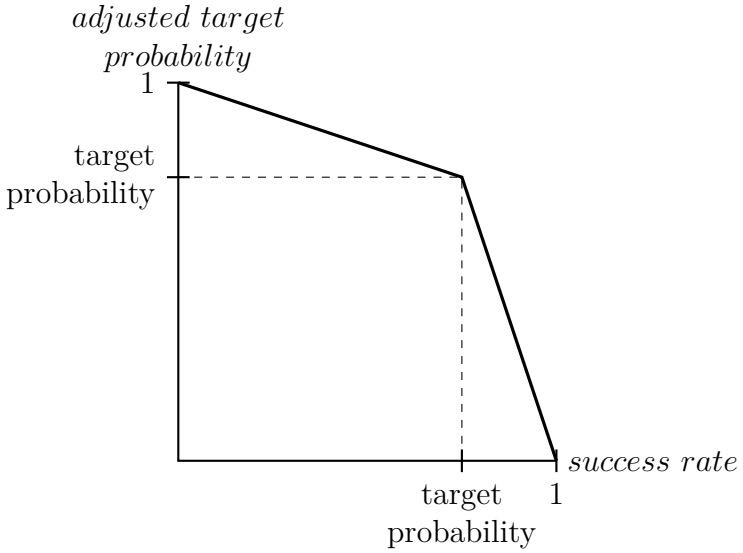

Figure 4.3: Function describing adjustment of target probability

This dynamic adjustment of target probability proved to be useful – it increased by 5 % the set of people who reported that the difficulty is "appropriate" [PP15].

4.3 Question Selection

In the previous sections, we discussed how to estimate students' knowledge and how to dynamically adjust target difficulty of questions. Using these two components, we can find the question with the optimal difficulty. However, several criteria should be considered to select the question which is the most useful for the student and difficulty is just one of them. Let us review the most important criteria.

- **Relevance**
  The question should be relevant to the practiced topic. Besides this *objective relevance* which is independent of the student and the practice session, there is also what we could call *subjective relevance*, i.e. what concept is the most relevant to the student right now. The subjective relevance changes during the practice, as some parts of the article are already practiced and some are not. Generally, the most subjectively relevant for the student is *the least known concept* [Wiß+13] (provided it is also objectively relevant to the topic).
- **Difficulty**
  We already discussed that the optimal probability of correct answer is about 70 % and we also mentioned a method for dynamic adjustment of this target probability (section 4.2). Using a student model (section 4.1) we can estimate the probability that a student would answer given question correctly. The more closer is the estimated probability to the target probability, the more is this question preferred.
- **Variability**
  Especially important is the variability in the practiced concepts. If one concept is practiced in several questions in a row, it gets quickly boring, and it negatively influences both the relevance (as the student would like to practice other concepts from the article as well) and the difficulty (since it is easier to answer questions about the same concept which was already practiced). Besides this crucial variability of concepts, variability of question types may also help, because diversity increases enjoyment. The variability of question types is taken into account e.g. in *Slepé mapy* [PPS14] and *DynaLearn* [Wiß+13].

Common approach is to use an appropriate scoring function for each of the considered criteria, and then obtain the total score as a linear combination of individual scores [PPS14]. The question with the highest total score is selected and presented to the student.

5 Smartoo Framework

Smartoo is a framework for question generation and adaptive practice. It is written in a modular way and decomposes the process into the following components:

- **KnowledgeBuilder** – takes an article as the input and extracts knowledge, building a RDF graph of facts contained in the article.
- **ExercisesCreator** – creates a set of questions (or possibly other types of exercises) using the built knowledge graph.
- **ExercisesGrader** – ranks generated exercises with respect to attributes such as difficulty or relevance to the original article.
- **Practicer** – controls the practice session itself, i.e. gives the student one exercise at a time based on the previous exercises and answers.

Each component (e.g. *KnowledgeBuilder*) has variable behavior (e.g. *KnowledgeBuilderBehavior*), which is a parametrised code, i.e. an implementation of the behavior (the code) and a set of parameters which the code uses. Component itself is just a wrapper around the behavior which performs common tasks, most importantly persistence management. At the beginning of each new session, 4-tuple of components with behaviors is chosen.

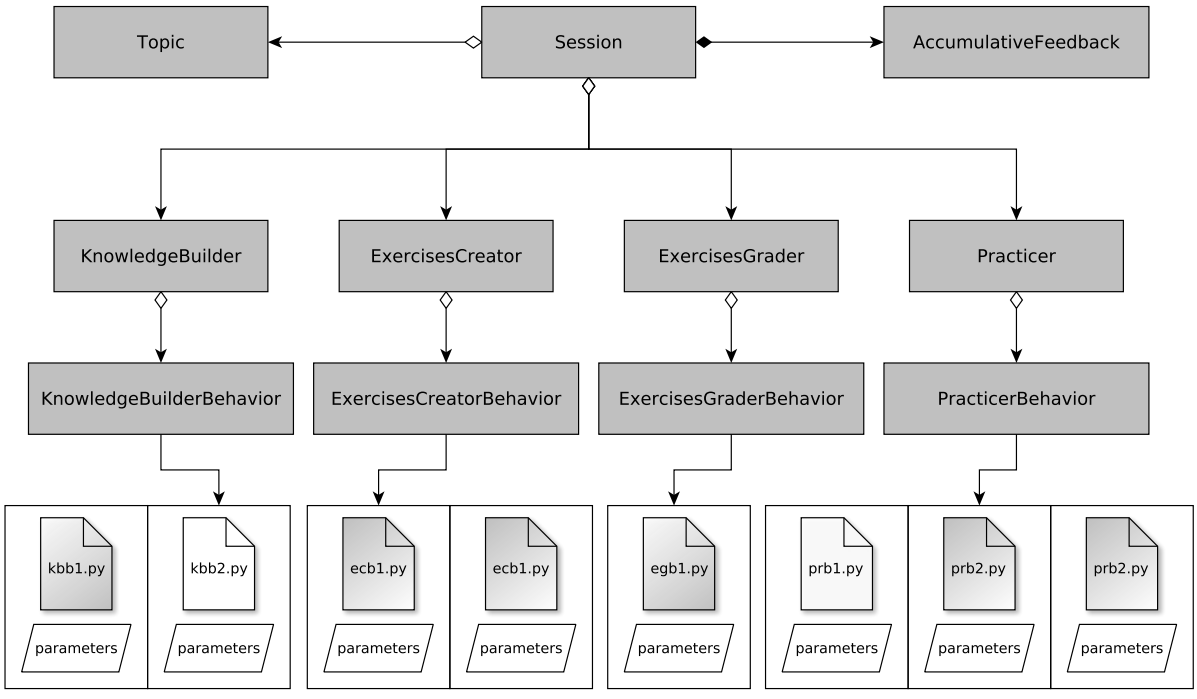

Figure 5.1: Components and behaviors

The proposed modular design allows for easy integration and testing of new behaviors. Simple prototype behaviors based on heuristic approaches were implemented. More sophisticated behaviors will be the subject of future development.

Except for *ExercisesCreator* and *ExercisesGrader*, the components does not communicate with each other directly, but via results (knowledge graphs and exercises) stored in the database. The communication and the flow of data is depicted in Figure 5.2. Individual processes are triggered by signals from the client. Time ordering of the signals and processes is shown in Figure 5.8 on page 38.

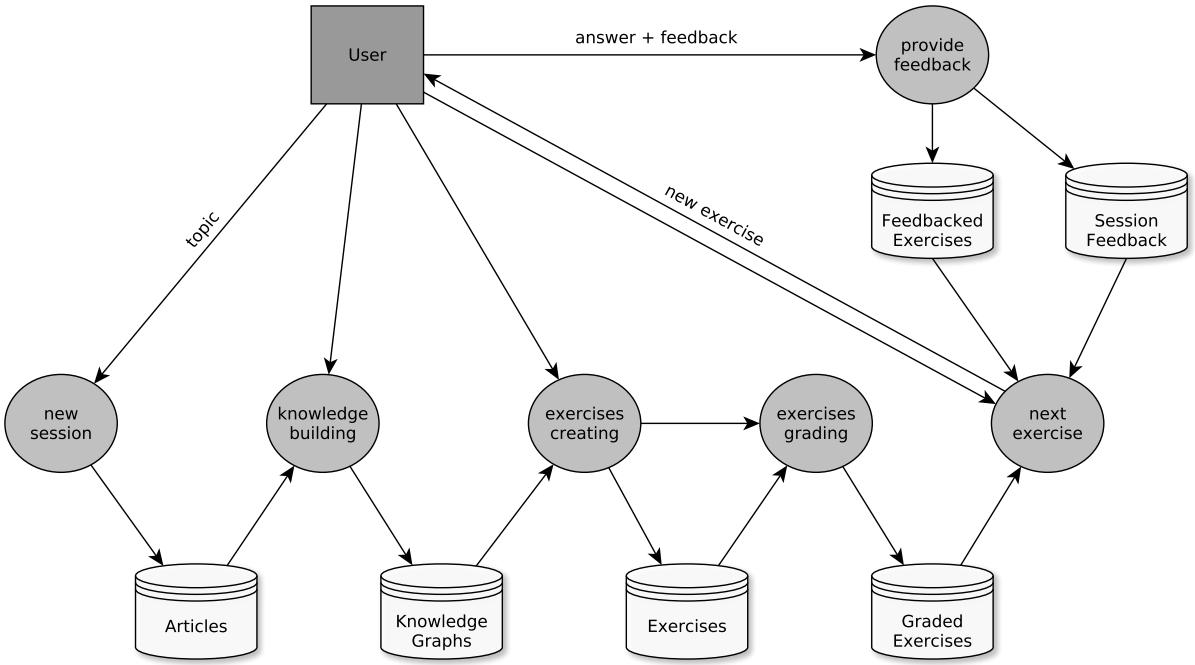

Figure 5.2: Data flow diagram

5.1 Behaviors Selection

Each session has assigned a *performance* which is computed as a weighted sum of the user final rating and the ratio of exercises marked as invalid or irrelevant. As we know which behaviors participated in which sessions, we can compute the average performance of a 4-tuple of behaviors $p(x_1, x_2, x_3, x_4)$.

At the beginning of each new session, 4-tuple of behaviors (k_1, k_2, k_3, k_4) (k_1 is knowledge builder, k_2 exercises creator, etc.) is chosen in a partially random way which prioritizes behaviors with better performance in the past. Ideally, the probability of a certain 4-tuple being selected would be proportional to its expected relative performance (relative to the expected performances of other 4-tuples). As we do not know the future, our best guess for the expected performance is the performance in the past. Using conditional probability we can easily derive formulae for probability P of components (x_1, x_2, x_3, x_4) being selected. I use asterisk $(*)$ to denote "any behavior", so e.g. $p((x_1, *, *, *))$ means average performance of all sessions in which knowledge builder was x_1.

$$P(k_1 = x_1) \propto p((x_1, *, *, *)),$$

$$P(k_2 = x_2 \mid k_1 = x_1) = \frac{P(k_1 = x_1 \wedge k_2 = x_2)}{P(k_1 = x_1)} \propto \frac{p((x_1, x_2, *, *))}{p((x_1, *, *, *))},$$

$$P(k_3 = x_3 \mid k_1 = x_1 \wedge k_2 = x_2) = \frac{P(k_1 = x_1 \wedge k_2 = x_2 \wedge k_3 = x_3)}{P(k_1 = x_1 \wedge k_2 = x_2)} \propto \frac{p((x_1, x_2, x_3, *))}{p((x_1, x_2, *, *))},$$

$$P(k_4 = x_4 \mid k_1 = x_1 \wedge k_2 = x_2 \wedge k_3 = x_3) = \ldots \propto \frac{p((x_1, x_2, x_3, x_4))}{p((x_1, x_2, x_3, *))}.$$

5.2 Articles Parsing

If the user requests topic which were never requested before, the corresponding article is looked up and retrieved from Wikipedia. Some text preprocessing is performed, e.g. sections without useful content, such as "See also", "External links" or "References", are discarded. Then the article is parsed into sentence trees with marked terms occurrences.

At first, text is split into sentences, each sentence is tokenized and each token is assigned a part-of-speech tag. For these natural language processing tasks I use the *NLTK* (Natural Language Toolkit) library [BKL09]. NLTK uses *The University of Pennsylvania (Penn) Treebank Tag-set* [San90].

Tag	Description	Example
NN	common noun, singular	house
NNP	proper noun, singular	Lincoln
DT	determiner	the
VB	verb, base form	eat
JJ	adjective	popular

Table 5.1: Part-of-speech tags examples [San90]

As an example, consider sentence "Six days after the surrender of Confederate commanding general Robert E. Lee, Lincoln was assassinated by John Wilkes Booth, a Confederate sympathizer." This sentence would be tokenized and tagged as follows:

```
[['Six', 'CD'], ['days', 'NNS'], ['after', 'IN'], ['the', 'DT'],
['surrender', 'NN'], ['of', 'IN'], ['Confederate', 'NNP'],
['commanding', 'NN'], ['general', 'NN'], ['Robert', 'NNP'],
['E.', 'NNP'], ['Lee', 'NNP'], [',', ','], ['Lincoln', 'NNP'],
['was', 'VBD'], ['assassinated', 'VBN'], ['by', 'IN'],
['John', 'NNP'], ['Wilkes', 'NNP'], ['Booth', 'NNP'],
[',', ','], ['a', 'DT'], ['Confederate', 'NNP'],
['sympathizer', 'NN'], ['.', '.']],
```

In addition to the text of the article, I also retrieve a list of titles of *internal links*, i.e. links to another pages on the Wikipedia. These link titles can be thought of as terms (as they have their page in an encyclopedia). The terms are shallowed-parsed (tokenized and

tagged) and their occurrences in the text are inferred. As there are often hundreds of terms in an article, I first made a trie from all terms, which then enables to search for all terms simultaneously. To represent shallowed parsed sentences with marked terms occurrences, I use shallow sentence trees. You can see an example of a sentence tree in Figure 5.3.

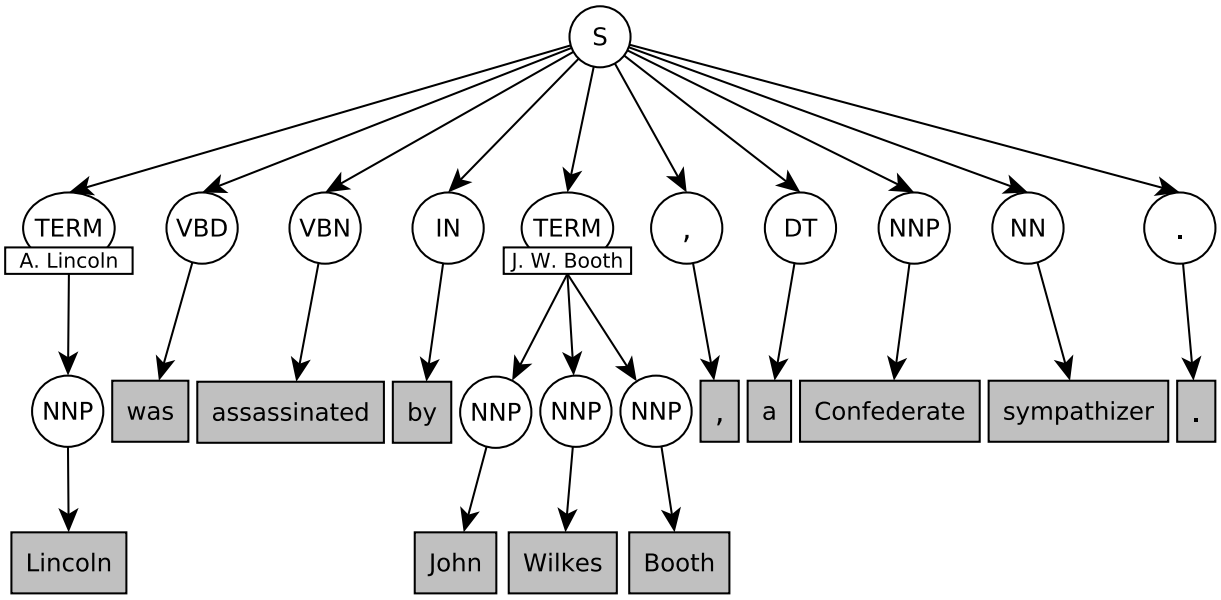

Figure 5.3: Sentence tree

5.3 Knowledge Building

KnowledgeBuilder component is responsible for building knowledge graph from a given article. Knowledge graph is a RDF graph of facts contained in the article.

Prototype behavior creates graph of *quasi facts* about terms in sentences. By the term *quasi fact* I mean some information usable for creating exercises, but not really an elementary unit of knowledge, which ordinary facts should be. The algorithm is simple:

1. Firstly, an empty graph is created (let us call it G).
2. If it was not already in the past, knowledge graph of the topic term is retrieved from DBpedia (section 2.5). All terms in this retrieved knowledge graph and also all terms in the article are added into the graph G together with their types and labels (which are also found using DBpedia).
3. Then we take each sentence in the article and check if it contains a term and is understandable without a context. The used heuristic is: it contains between 5 and 35 tokens (or 50, depending on the selected parameters), does not contain pronouns, parentheses or quotations or some typical anaphoric words (referring to the context) such as “this”.

4. For each sentence which satisfies these conditions, we create an quasi fact about the term in this sentence (and add it to G). Each quasi fact consists of several pieces of information: text before the term, text after the term and the term itself (and its label and types). An example of a created quasi fact can be seen in Figure 5.4.
5. Graph G is returned.

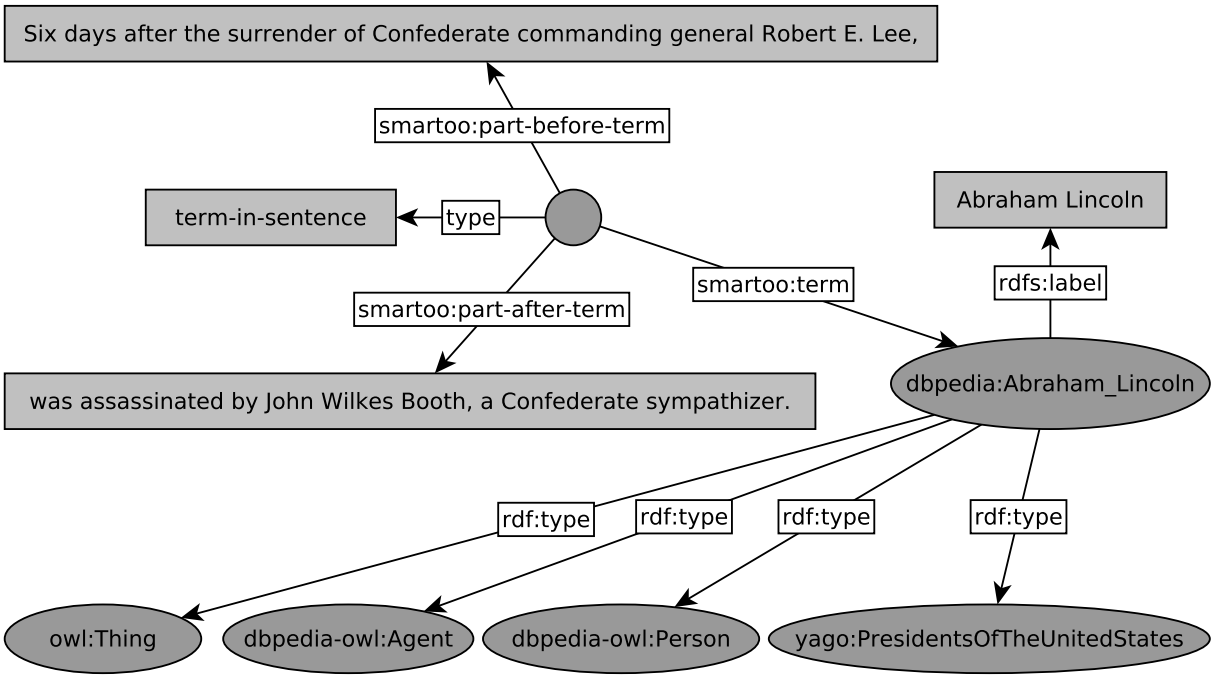

Figure 5.4: Example of a quasi fact

5.4 Exercises Creating

ExerciseCreator component uses built knowledge graph to generate a set of exercises. Exercises consist of presentation data and semantic information.

- **presentation data**
  Currently the only supported type of exercise is a multiple choice question[1]. Presentation data for multiple choice question consists of a question string, a list of choice labels and a correct answer label (which must be one of the choice labels).
- **semantic information**
  Semantic information is used for exercises grading. For this purpose, I decided to use a list of term pairs for which holds, that if the user confuses them, it can lead to an incorrect answer. For a multiple choice question with correct answer A and distractors B, C and D, the term pairs are (A, B), (A, C) and (A, D).

1. Other types of exercises, e.g. matching pairs or questions requiring the user to type the answer, instead of just selecting from a list of choices, bring nontrivial challenges and are planned for the future development.

Prototype behavior uses all the quasi facts about terms in a sentence and creates *fill in the gap* questions with multiple options. All quasi facts about terms in sentence can be retrieved from the knowledge graph with the following simple SPARQL query:

```
SELECT ?before ?term ?after
WHERE {
    ?quasifact a "term-in-sentence" ;
        smartoo:part-before-term ?before ;
        smartoo:part-after-term ?after ;
        smartoo:term ?term .
}
```

As we deal with multiple choice question, we also need to select appropriate distractors. Guided by the results about distractors presented in section 3.4, the most similar terms to the correct answer are chosen. Terms similarity heuristics is based on the number of shared types $C(t_1, t_2) = |types(t_1) \cap types(t_2)|$[2]:

$$sim(t_1, t_2) = 2\left(\frac{1}{1 + \exp(-|C(t_1, t_2)|/\kappa)}\right) - 1 \tag{5.1}$$

The constants κ influences the speed of convergence to 1. Terms without any shared type would have 0 similarity and the more common types they have, the more close to 1 the similarity is.

As an example, let us look at the exercise created from the quasi fact in Figure 5.4. The presentation data looks like this:

```
{'question': 'Six days after the surrender of Confederate commanding
             general Robert E. Lee, _______ was assassinated by
             John Wilkes Booth, a Confederate sympathizer.',
'choices': ['Andrew Johnson', 'James Buchanan',
           'Abraham Lincoln', 'Franklin D. Roosevelt'],
'correct-answer': 'Abraham Lincoln'}
```

And the respective semantic information is following:

```
{'term-pairs': [
    ['Abraham Lincoln', 'Franklin D. Roosevelt'],
    ['Abraham Lincoln', 'James Buchanan'],
    ['Abraham Lincoln', 'Andrew Johnson']]}
```

5.5 Exercises Grading

ExerciseGrader takes an exercise (in particular the semantic part) and assigns it grades for difficulty, relevance to the article and possibly any other attributes, e.g. grammatical correctness probability.

2. The number is not normalized by $|types(t_1) \cup types(t_2)|$, because it would made terms with a lot of types not similar to anything (e.g. Abraham Lincoln has over 50 types and most of them are very specific).

In the prototype behavior, difficulty of an exercise is estimated as the average similarity between all term pairs which can be mistaken. Similarity of two terms is measured in the same way as during exercise creating (section 5.4). The value is than normalized to range from -2 to 2 (-2 means extremely simple, 0 average, 2 extremely difficult), because it is later used by the practicer in the logistic model.

Relevance of the exercise is estimated as the *cosine similarity* measure between the exercise and the article, i.e. cosine of an angle between the article and the exercise represented as sparse vectors mapping terms to their frequency in the article or the exercise [MRS09, p. 121]. Cosine similarity between two documents (in our case one document is the article A and another is the exercise E) is given by the following formula:

$$similarity(A, E) = \frac{A \cdot E}{|A||E|} = \frac{\sum_{i=1}^{n} A_i E_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} E_i^2}} \tag{5.2}$$

where A_i (E_i) denotes numbers of occurrences of the i-th term in article (exercise). This measure is commonly used in the information retrieval for estimating similarity between two documents of arbitrary lengths. The cosine similarity is a real number between 0 and 1, the higher it is, the smaller is the angle between documents in the vector space and the more similar the documents are. In other words, the closer the grade is to 1, the more relevant the exercise is.

5.6 Adaptive Practice

The task of the *Practicer* is to control the practice session itself, i.e. to give the student one exercise at a time based on the previous exercises and answers. In the prototype behavior, three attributes are considered to select the most suitable exercise:

- relevance to the article,
- difficulty (compared to the so far success of the user),
- repetitiveness (to not ask about one term over and over again).

Weights of these three attributes in the total score, as well as the target success are given by the behavior parameters (Table 5.2).

	parameter	used values					
τ	target success	0.65	0.65	0.65	0.65	0.65	0.65
α	relevance weight	1.00	1.25	1.00	1.00	1.25	1.25
β	difficulty weight	1.00	1.00	1.25	1.00	1.00	1.25
γ	repetitiveness weight	1.00	1.00	1.00	1.25	1.25	1.25

Table 5.2: Parameters of prototype adaptive practice behavior

Relevance of the exercise to the article was computed by *ExercisesGrader* (section 5.5). Repetitiveness score is set to the opposite of cosine similarity between the exercise and the

exercises already used in the current practice session (see section 5.5 for the explanation and formula of the cosine similarity).

The most interesting part is the difficulty score. To estimate the probability that the student correctly answers a question with given difficulty (taking into account the possibility of guessing), I use the logistic model of student as described in section 4.1. I have also implemented the dynamic adjusting of the target probability (section 4.2). The difficulty penalization is then set to a difference between the adjusted target probability and the expected probability as calculated by the student model.

Difficulty adjustment is especially useful for students with extremely high knowledge – the more time a student answers correctly, the lower is the adjusted target probability, the more difficult exercises are preferred. In Figure 5.5, an example of knowledge estimation process during one session is shown. Table 5.3 contains detailed information about the same session.

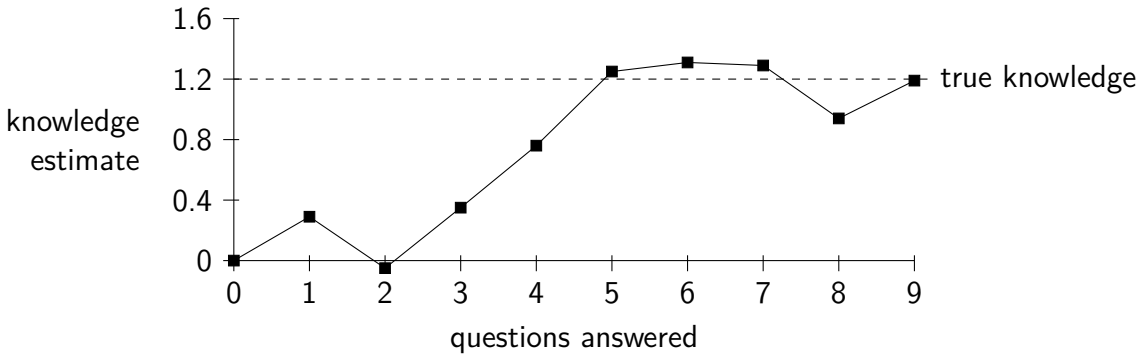

Figure 5.5: Knowledge estimation during a session

	knowledge est.	target	difficulty	probability est.	true probability	answer
0	+0.000	0.65	−0.2	0.662	0.852	correct
1	+0.290	0.52	+0.8	0.531	0.699	incorrect
2	−0.050	0.69	−0.4	0.690	0.874	correct
3	+0.346	0.63	+0.4	0.615	0.767	correct
4	+0.760	0.50	+1.4	0.509	0.588	correct
5	+1.250	0.41	+1.8	0.524	0.516	correct
6	+1.305	0.34	+1.8	0.534	0.516	incorrect
7	+1.294	0.54	+1.8	0.532	0.516	incorrect
8	+0.941	0.66	+0.8	0.651	0.699	correct
9	+1.191	0.62	+1.2	0.623	0.625	incorrect

Table 5.3: Adaptive practice for a student with knowledge 1.2

After the partial scores for relevance, difficulty and repetitiveness are calculated, they are summed (with weights given by behavior parameters) to obtain the total score.

$$\text{totalScore} = \alpha \cdot \text{relevanceScore} + \beta \cdot \text{difficultyScore} + \gamma \cdot \text{repetitivenessScore}$$

An exercise with the highest total score is chosen.

5.7 Web Interface

The client controls triggering of processes by requests to the server – start session, build knowledge, create exercises, next exercise (repeatedly) and session feedback. *Next exercise* requests (with the exception of the first one) have also a feedback from the previous exercise attached to them. Feedback includes whether the question was answered correctly or incorrectly, and whether the exercise was marked as invalid or irrelevant. After the last exercise, a session feedback form is displayed, prompting user to express his or her overall feeling from the practice session (with 3 choices: "Bad", "So so", and "Good"). A typical sequence of client requests and server responses is shown in Figure 5.8 (gray zones denotes computation, black zones waiting for user response).

Figure 5.6: Smartoo home page

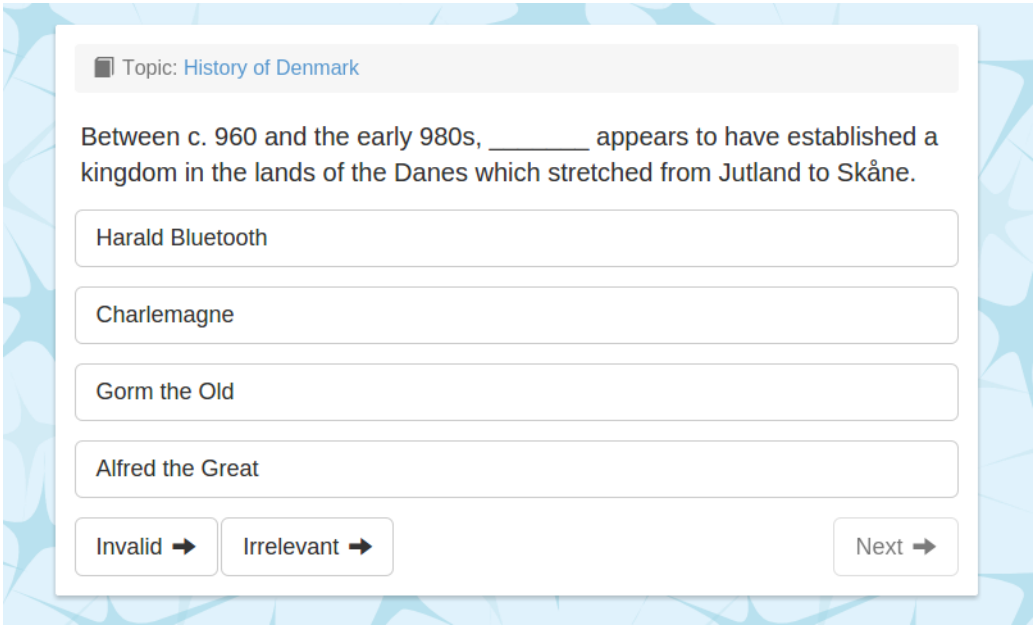

Figure 5.7: Question about the history of Denmark

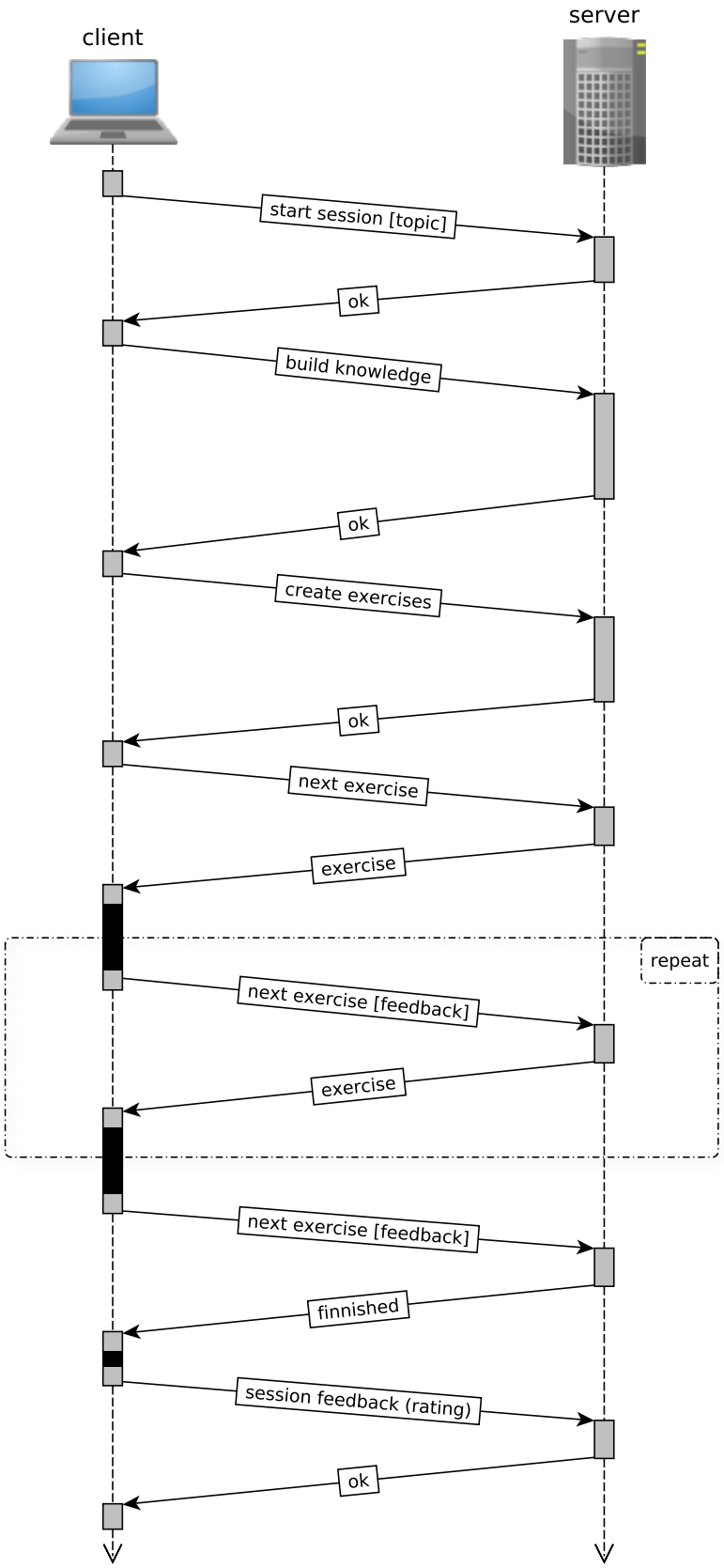

Figure 5.8: Client-server communication

6 Deployment and Evaluation

The project structure is described in Appendix A, the source code is available from its repository[1]. Smartoo has been deployed at `http://smartoo.thran.cz`. In this chapter, I mention a few deployment details and then I analyze the feedback collected from the users.

6.1 Used tools and libraries

The main tools and libraries used in the project are:

- `Python 2.7`
  The backend of the application is developed in Python 2.7[2]. To create an isolated Python environment for the project, `Virtualenv`[3] is used. After activating a new virtual environment, all necessary Python packages can be installed simply by command `pip install -r requirements.txt` executed in the project root directory.
- `Django 1.7`
  Django[4] is a Python web framework, using *Model–View–Template* architecture.
- `NLTK 3.0`
  NLTK[5] (Natural Language Toolkit) is a Python library for natural language processing tasks, such as tokenization and part-of-speech-tagging.
- `SPARQLWrapper 1.6`
  SPARQLWrapper[6] provides an interface to DBpedia public SPARQL endpoint.
- `RDFLib 4.1`
  RDFLib[7] is a Python library for working with RDF graphs.
- `Wikipedia 1.4`
  Wikipedia[8] is a simple Python wrapper for the Wikipedia API.
- `SQLite` or `MySQL`
  Database is accessed through Django, so PostgreSQL should probably work as well.
- `AngularJS`
  AngularJS[9] is a popular JavaScript framework for a simple development of the *Model–View–Controller (MVC)* architecture at the client side.
- `Bootstrap`
  Bootstrap[10] is a HTML and CSS framework for responsive design of the web.

1. http://github.com/effa/smartoo
2. https://www.python.org/download/releases/2.7/
3. https://virtualenv.pypa.io
4. https://www.djangoproject.com/
5. http://www.nltk.org/
6. http://rdflib.github.io/sparqlwrapper/
7. https://rdflib.readthedocs.org/
8. https://github.com/goldsmith/Wikipedia
9. https://angularjs.org/
10. http://getbootstrap.com/

6.2 Performance Evaluation

Besides the correct or incorrect answers, Smartoo collects feedback from users through the "Invalid" and "Irrelevant" buttons under each generated question, the practice session final rating form ("bad", "so so" or "good") and an optional "Write to us" button.

During testing, 71 practice sessions were taken, with total of 616 questions (counting only the questions with some response to them, i.e. which were either answered or marked as invalid/irrelevant). From 588 questions with answers, 424 were answered correctly and 164 incorrectly, resulting in overall 72 % success rate. Fifty-one questions (8 %) were marked as invalid or irrelevant. Concerning the practice session final ratings, 31 % were rated as "good", 65 % as "so so" and 4 % as "bad".

Tested version of Smartoo used 2 behaviors for knowledge building. Knowledge builder behaviors were nearly the same, they only differ in the "maximum sentence length" parameter. Their average performances are shown in Table 6.1. The behavior using shorter maximum length (30 tokens) performed slightly better.

	max length	performance
1	40	0.701
2	30	0.771

Table 6.1: Performances of knowledge builder behaviors

Six different behaviors were used for practicers. They differ in 3 parameters: relevance weight, difficulty weight and repetitiveness weight. The used values and their performances are shown in Table 6.2. One observation we can made is that setting relevance or difficulty weight higher than repetitiveness weight (behaviors 2 and 3) decreases the performance.

	relevance	difficulty	repetitiveness	performance
1	1.00	1.00	1.00	0.782
2	1.25	1.00	1.00	0.671
3	1.00	1.25	1.00	0.629
4	1.00	1.00	1.25	0.766
5	1.25	1.00	1.25	0.783
6	1.00	1.25	1.25	0.787

Table 6.2: Performances of practicer behaviors

In the written notes from users, some comments appeared several times. A lot of questions are either quite easy or not completely clear and out of context. Topic of the article is often used as a correct answer and this usually leads to very simple questions. Sometimes, the correct answer can be determined just from its type (e.g. the correct answer is a continent and all distractors are countries). The practice session progress (in particular, how many questions still remain) should be shown and a button for terminating the session and jumping back to the home page should be available.

7 Conclusions and Future Plans

There is no doubt that despite the effort of the bachelor thesis, the question generation remained an open problem of the natural language processing. Evaluation showed clear limitations of used heuristic approaches. However, the system was developed with the importance of modularity and extensibility in mind. I plan to publish a research paper about the framework, so that other people could use it to test their own ideas for question generation. New, more sophisticated behaviors for knowledge building, questions creating, questions grading and adaptive practice may be implemented and easily integrated into the system.

Besides the implementation of advanced behaviors for knowledge extraction and questions creation using machine learning approaches, there are some other improvements I would like to do in the future. For example, extend the system for the possibility of accepting a plain text, instead of Wikipedia articles. This would require reasonably good detection of named entities in the text.

I would also like to generate different types of questions and exercises, for example asking for the cause (why something happens), questions with short answers without choices (e.g. who invented something), matching pairs (scientist – discovery, event – date, ...), what is on the picture, ordering time events, etc.

In case of long articles, it could be useful to interleave reading and practice. It would be also great to make the practice length variable based on the student performance and provide exercises until we are relatively confident that the student knows the topic. This would require to possibly present one question several times and to use a student model which can capture learning.

Another idea is to employ an evolutionary algorithm for incremental improvements of used population of behaviors. Average performance in a few practice sessions would serve as a fitness function, crossover and mutation operations can be straightforwardly done on parameter values, although this is only applicable for behaviors with same or similar code.

For me, automatic question generation and adaptive practice is an amazing field full of challenges and I would like to continue to devote my effort to create an online question generation tool which would make learning more efficient and fun.

Bibliography

[AFF14] T. Atapattu, K. Falkner, and N. Falkner, "Acquisition of triples of knowledge from lecture notes: a natural language processing approach", *Proceedings of the 7th International Conference on Educational Data Mining*, J. Stamper, Z. Pardos, M. Mavrikis, and B. McLaren, Eds., pp. 193–196, 2014.

[AlY11] M. Al-Yahya, "OntoQue: a question generation engine for educational assesment based on domain ontologies", in *Advanced Learning Technologies (ICALT), 11th IEEE International Conference*, Jul. 2011, pp. 393–395. DOI: 10.1109/ICALT.2011.124.

[AM11] 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*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 56–64, ISBN: 9781937284039. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043132.2043139.

[ASM11] M. Agarwal, R. Shah, and P. Mannem, "Automatic question generation using discourse cues", in *Proceedings of the 6th Workshop on Innovative Use of NLP for Building Educational Applications*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 1–9, ISBN: 9781937284039. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043132.2043133.

[BB07] N. Bach and S. Badaskar, "A review of relation extraction", *Literature review for Language and Statistics II*, 2007.

[BFE05] 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.

[BKL09] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[BL04] R. Brachman and H. Levesque, *Knowledge representation and reasoning*. Elsevier, 2004.

[Bol+08] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge", in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM, 2008, pp. 1247–1250.

[But12] T. Butt, "Adaptive practice: high-efficiency game-based learning", 2012. [Online]. Available: http://www.brainrush.com/pdf/AdaptivePracticeWhitepaper.pdf.

[Car+10] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka Jr, and T. M. Mitchell, "Coupled semi-supervised learning for information extraction", in *Proceedings of the third ACM international conference on Web search and data mining*, ACM, 2010, pp. 101–110.

[Csi90] M. Csikszentmihalyi, "Flow: The psychology of optimal experience", 1990.

[dAya13] R. J. de Ayala, *Theory and practice of item response theory*. Guilford Publications, 2013.

[DB12] M. C. Desmarais and R. S. J. d. Baker, "A review of recent advances in learner and skill modeling in intelligent learning environments", *User Modeling and User-Adapted Interaction*, vol. 22, no. 1-2, pp. 9–38, 2012.

[Elo78] A. E. Elo, *The rating of chessplayers, past and present*. Batsford London, 1978.

[Fer+10] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, *et al.*, "Building Watson: an overview of the DeepQA project", *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[Fos09] R. M. Foster, "Improving the output from software that generates multiple choice question (MCQ) test items automatically using controlled rhetorical structure theory", in *Proceedings of the International Conference RANLP*, I. Temnikova, I. Nikolova, and N. Konstantinova, Eds., 2009, pp. 29–34.

[GJ02] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles", *Computational linguistics*, vol. 28, no. 3, pp. 245–288, 2002.

[Hei11] M. Heilman, "Automatic factual question generation from text", PhD thesis, Carnegie Mellon University, 2011.

[HS09] M. Heilman and N. A. Smith, "Question generation via overgenerating transformations and ranking", Carnegie-Mellon University, Tech. Rep., 2009.

[JCG06] J. Judge, A. Cahill, and J. V. Genabith, "QuestionBank: creating a corpus of parse-annotated questions", in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL (COLING-ACL-06*, 2006, pp. 497–504.

[Kam04] N. Kambhatla, "Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations", in *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, Association for Computational Linguistics, 2004.

[KSvM11] S. Klinkenberg, M. Straatemeier, and H. L. J. van der Maas, "Computer adaptive practice of maths ability using a new item response model for on the fly ability and difficulty estimation", *Computers & Education*, vol. 57, no. 2, pp. 1813–1824, 2011.

[LB15] J. L. Little and E. L. Bjork, "Optimizing multiple-choice tests as tools for learning", *Memory & Cognition*, vol. 43, no. 1, pp. 14–26, 2015, ISSN: 0090-502X. DOI: 10.3758/s13421-014-0452-8.

[Leh+14] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, *et al.*, "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia", *Semantic Web Journal*, 2014.

[MAK06] R. Mitkov, L. An Ha, and N. Karamanis, "A computer-aided environment for generating multiple-choice test items", *Natural Language Engineering*, vol. 12, no. 02, pp. 177–194, 2006.

[MCC11] A. C. Mendes, S. Curto, and L. Coheur, "Bootstrapping multiple-choice tests with The-Mentor", in *Computational Linguistics and Intelligent Text Processing*, ser. Lecture Notes in Computer Science, A. F. Gelbukh, Ed., vol. 6608, Springer Berlin Heidelberg, 2011, pp. 451–462, ISBN: 978-3-642-19399-6. DOI: 10.1007/978-3-642-19400-9_36.

[Men+11] 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.

[Mil95] G. A. Miller, "Wordnet: a lexical database for English", *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[MN14] K. Mazidi and R. D. Nielsen, "Pedagogical evaluation of automatically generated questions", in *Intelligent Tutoring Systems*, ser. Lecture Notes in Computer Science, S. Trausan-Matu, K. E. Boyer, M. Crosby, and K. Panourgia, Eds., vol. 8474, Springer International Publishing, 2014, pp. 294–299, ISBN: 978-3-319-07220-3. DOI: 10.1007/978-3-319-07221-0_36.

[MPJ10] P. Mannem, R. Prasad, and A. Joshi, "Question generation from paragraphs at UPenn: QGSTEC system description", in *Proceedings of QG2010: The Third Workshop on Question Generation*, 2010, pp. 84–91.

[MRS09] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2009.

[NS07] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification", *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[Par04] I. Partchev, "A visual guide to item response theory", 2004, Friedrich Schiller Universität Jena.

[PCK09] P. I. Pavlik, H. Cen, and K. R. Koedinger, "Performance factors analysis–a new alternative to knowledge tracing", in *Proceedings of Artificial Intelligence in Education (AIED)*, IOS Press, 2009.

[Pel14] R. Pelánek, "Time decay functions and elo system in student modeling", *Proc. of Educational Data Mining (EDM)*, 2014.

[PKK08] A. Papasalouros, K. Kanaris, and K. Kotis, "Automatic generation of multiple choice questions from domain ontologies.", in *Proceedings of IADIS International Conference on e-Learning*, Citeseer, 2008, pp. 427–434.

[PP15] J. Papoušek and R. Pelánek, "Impact of adaptive educational system behaviour on student motivation", in *Proceedings of Artificial Intelligence in Education (AIED)*, 2015.

[PPS14] R. Pelánek, J. Papoušek, and V. Stanislav, "Adaptive practice of facts in domains with varied prior knowledge", in *Proceedings of the 7th International Conference on Educational Data Mining (EDM 2014)*, J. Stamper, Z. Pardos, M. Mavrikis, and B. M. McLaren, Eds., International Educational Data Mining Society, 2014, pp. 6–13, ISBN: 978-0-9839525-4-1.

[RH02] D. Ravichandran and E. Hovy, "Learning surface text patterns for a question answering system", in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics, 2002, pp. 41–47.

[RP12] H. L. Roediger and M. A. Pyc, "Inexpensive techniques to improve education: applying cognitive psychology to enhance educational practice", *Journal of Applied Research in Memory and Cognition*, vol. 1, no. 4, pp. 242–248, 2012.

[San90] B. Santorini, "Part-of-speech tagging guidelines for the Penn Treebank Project (3rd revision)", 1990.

[SET09] T. Segaran, C. Evans, and J. Taylor, *Programming the semantic web*. O'Reilly Media, Inc., 2009.

[Tho09] N. A. Thompson, "Ability estimation with item response theory", 2009, Assessment Systems Corporation. [Online]. Available: http://www.assess.com/docs/Thompson_(2009)_-_Ability_estimation_with_IRT.pdf.

[Van06] K. Vanlehn, "The behavior of tutoring systems", *International journal of artificial intelligence in education*, vol. 16, no. 3, pp. 227–265, 2006.

[VG09] R. Vasile and A. C. Graesser, "The question generation shared task and evaluation challenge", The University of Memphis, 2009.

[vLG00] W. J. van der Linden and C. A. W. Glas, *Computerized adaptive testing: Theory and practice*. Springer, 2000, ISBN: 978-0-7923-6425-2.

[Wiß+13] M. Wißner, F. Linnebank, J. Liem, B. Bredeweg, and E. André, "Question generation and adaptation using a bayesian network of the learner's achievements", in *Artificial Intelligence in Education*, ser. Lecture Notes in Computer Science, H. Lane, K. Yacef, J. Mostow, and P. Pavlik, Eds., vol. 7926, Springer Berlin Heidelberg, 2013, pp. 729–732, ISBN: 978-3-642-39111-8. DOI: 10.1007/978-3-642-39112-5_99.

[Wol76] J. H. Wolfe, "Automatic question generation from text - an aid to independent study", *SIGCUE Outlook*, vol. 10, no. SI, pp. 104–112, 1976. DOI: 10.1145/953026.803459.

[WP09] B. Wyse and P. Piwek, "Generating questions from OpenLearn study units", in *AIED 2009 Workshop Proceedings Volume 1: The 2nd Workshop on Question Generation*, 2009.

[ZAR03] D. Zelenko, C. Aone, and A. Richardella, "Kernel methods for relation extraction", *The Journal of Machine Learning Research*, vol. 3, pp. 1083–1106, 2003.

A Project Structure

Simplified project structure:

+ common	
- utils/	utilities, e.g. metrics, language processing
- fields.py	custom DB field for dictionary
- tests.py	
+ abstract_component	
- models.py	base class for all 4 components
- component_behavior.py	base class for all behaviors
+ knowledge	
- utils/	utilities, e.g. SPARQL and terms trie
- fields.py	custom fields for terms and graphs
- models.py	article, knowledge builder, knowledge graph
- knowledge-builder-behaviors/	implementation of behaviors
- tests/	
+ exercises	
- utils/	selecting distractors, difficulty normalization
- exercises-creator-behaviors/	behaviors for exercises creators
- exercises-grader-behaviors/	behaviors for exercises graders
- models.py	exercises creator and grader, (graded) exercise
- tests.py	
+ practice	
- utils/	logistic model, target probability adjustment
- exercises-grader-behaviors/	behaviors for practicers
- models.py	class for practicers
- tests.py	
+ smartoo	
- management/	commands for data dumping and statistics
- fixtures/	data for testing and DB initialization
- templates/	HTML templates
- static/	JavaScript and CSS files, images
- models.py	session and feedback
- components_selector.py	selecting components for new session
- views.py	functions for processing requests
- urls.py	mappings from URLs to corresponding views
- exceptions.py	
- tests.py	
+ production	
- settings.py	django settings (database, caching, logging, etc.)
+ data	
- sessions.csv	data about sessions
+ requirements.txt	list of dependencies
+ manage.py	script for django-related tasks