b
UNIVERSITÄT
BERN

Modelling the Acquisition of Natural Language

Bachelor Thesis

Karan Sethi
from
3032 Hinterkappelen, Switzerland

Faculty of Science
University of Bern

October 31, 2015

Prof. Dr. Oscar Nierstrasz
Dr. Mircea Lungu

Software Composition Group
Institute of Computer Science
University of Bern, Switzerland

Imagination is more important than knowledge.
Knowledge is limited.
Imagination encircles the world.

Albert Einstein

Acknowledgements

First of all my sincere thanks to *Prof. Dr. Oscar Nierstrasz* for giving me the opportunity to do this bachelor project in the Software Composition Group of the University of Berne.

Special thanks to *Dr. Mircea Lungu* for the continuous guidance, his invaluable suggestions and his belief in me and the project that gave me lot of confidence and was a source of inspiration.

I am also sincerely grateful to *Pascal Giehl*, *Linus Schwab* and *Ada Lungu* for helping me to bring this project to a higher level with their work and support in the Zeeguu Platform.

Finally, a big thanks to all the people behind the scenes who have taken their time in assisting me and giving me the help and support required to be able to finish this project successfully.

Abstract

The Zeeguu Platform is a language learning platform for people who want to expand their language skills by reading articles or stories they love and practise the words which they have not understood by doing exercises.

The aim of this thesis is to investigate and find a way to quantify the user's knowledge about the language which is currently being learned and give them a metric about their progress. In the first part, we will describe how the probability of a user knowing a particular word is computed and how this data is then aggregated and used to give the user feedback on his progress. In the second part, we present a meta-model that enables us to keep track of the user interactions with the language and compute the desired metrics for the user. In the final part, we study the results of the implemented metrics for a particular user and discover that they represent a good approximation of the current state of the users knowledge of the language.

Contents

1 Introduction

Traditionally, people have been joining language classes or having private lessons to learn a new language. Those who prefer individual study can use other resources like textbooks, vocabulary lists, and exercises. These resources are nowadays available on a computer and also on a smart phone for those who want to study while on the move.

In this project, we are extending the Zeeguu Platform[1] which is targeted at people who already have a basic understanding of a language and want to learn and improve it. Zeeguu offers the users the possibility to look up the words in the article they are reading. These words, their translations, and the context can be "bookmarked" in their profile. The list of the bookmarks can later be verified. Tailored exercises are created for them to practice these bookmarked words.

The goal of this project is to give the users of the Zeeguu Platform, feedback about their progress in learning the vocabulary of the desired language. This means, we need to quantify their knowledge about the vocabulary. To be able to have an accurate quantification, it is necessary to collect as much data as possible about the user's interaction with foreign texts. However, because our collection of the information cannot be perfect or complete, (users also encounter the language besides interacting with devices) it is likely that such a quantification can only be approximative, and thus it will have to be done by heuristic methods.

One desirable property of the quantified feedback is that the users not only get information about their learning progress, but it should also motivate and inspire them to study the language more.

[1] `https://www.zeeguu.unibe.ch/`

2 Related Work

This chapter gives an overview of the related work and is divided into two parts. The first part is about the work related to Zeeguu and the second part is about external related products which are available on the market.

2.1 The Zeeguu Platform

In 2013 *Simon Marti* implemented a first version of the Zeeguu Platform as a bachelor's project[1]. At that time, the platform allowed learners to read content in the internet and translate the words they did not understand. Users also had the possibility to bookmark words and do exercises to practice those words.

2.1.1 The Zeeguu Translate Application

In 2015, for his bachelor's project, *Pascal Giehl* created an Android Application which extends the Zeeguu Platform to mobile devices[2]. It gives the users the possibility to read articles even when they are on the move, translate the words they do not understand and bookmark them. Furthermore, Zeeguu Translate can also let the user do exercises to practice those words.

2.2 Other Language Learning Platforms

In today's world there is always competition in everything, from children wanting to get the best grades to companies capturing the most market shares. This is also the case for the Zeeguu Platform. Several other applications can help people learn and improve a language[1]. Following are two similar projects:

2.2.1 Duolingo

Duolingo[2] offers written lessons and dictations in many different languages. It has a skill tree in a game form. Through this the user can progress and can practice the learned words. Users can gain experience points whenever they finish a lesson. Skills are considered learned when all the lessons related to the skill are done. For every correct answer the user gains a point and for every wrong answer loses a point. The lesson is validated when ten points are reached. Each skill consisting of maximum ten lessons has a strength bar that estimates how well the user knows the words. After some time the strength bar reduces, indicating the user to do certain lessons again.

Duolingo gamifies the learning process, and this is great, but the experience points it gives are just coins. We consider this to be a problem, since the points that the user collects have no actual significance for him. It would have been better if the user would have a more direct metric: e.g. words learned in the target language.

Another limitation of the Duolingo platform is that the lessons have to be prepared by the app developers. It would be nice to let the user read whatever he wants[3].

2.2.2 Babbel

Babbel[3] offers many courses for speaking and listening. There are up to 3'000 new words per language one can learn with this tool. It is even possible to get a certificate for every course that has been completed. Every word or phrase in the users personal vocabulary can be assigned to one of the six knowledge levels depending how they do the course. As a result the Review Manager tells us which word or phrase needs to be reviewed again.

The disadvantage of this is that users need to pay to use this platform which makes them think twice before they use it. Also, you cannot read whatever you like with Babbel. It has predefined courses. It would be better if this would not be the case, so that the user is not dependent on the things this platform offers.

[1] However, alternatives are good, since first it means there is a market and second, the users have a choice.

[2] `https://www.duolingo.com/`

[3] `https://uk.babbel.com/`

3 Quantifying Progress in Second Language Acquisition

This chapter introduces the *Ranked Words List* and gives an overview of the problems quantifying the user's knowledge in the language he is currently learning with the Zeeguu Platform. Furthermore, it explains the heuristic method which is a possible solution about how the quantification problem can be solved. Last but not the least, it also gives possibilities of how the quantification can be improved in the future.

The biggest problem here was to find out an approach to approximate how well the user knows a particular language, because we can never be sure, how much the user is using the application and how serious he is to learn a new language.

3.1 Introduction to the Ranked Words List

To be able to estimate the percentage of the vocabulary of a language that a user knows, we need a reference point. Such a reference point should be a list of the words that the user desires to learn in the target language. Multiple approaches are possible for establishing such a list. In our case, we use the top 10'000 most frequently used words in movie subtitles in the target language[1]. We call this 10'000 words list the Ranked Words List.

[1]Zeeguu uses the word frequency lists that are published at: `https://invokeit.wordpress.com/frequency-word-lists/`

Studies show that movie subtitle based frequency lists are a very accurate approximation of the words used in daily life[4]. Ranking the words helps us give the user a feedback of the relative importance of each bookmarked word. Furthermore, it also helps to quantify how well the user knows the language he is currently learning.

The probabilities that are described in this chapter are only computed if the words exist in the Ranked Words List, because we want to give the user a metric on the basis of words that are being used in daily life.

3.2 Exercise Based Probability

One component of the Zeeguu Platform are the exercises which help a learner consolidate the words he is learning (the words he has bookmarked). Figure 3.1 shows actions the user can take in such an exercise:

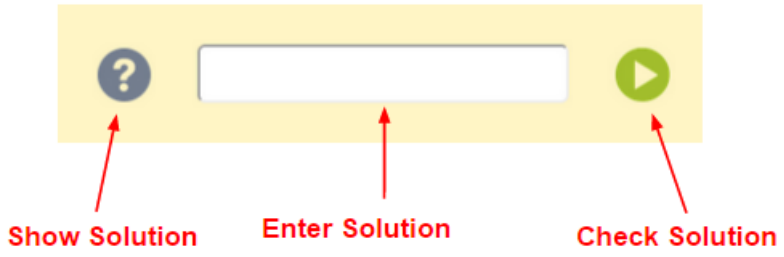

Figure 3.1: Actions before answering an Exercise

The user is shown a word in the native language and a text in the learned language. He has to find the word in the text which translates the word shown in the native language and type it in the field shown in 3.1. In figure 3.2 we show the learner if the answer was correct and a possibility that the user can go to the next exercise:

Figure 3.2: Actions after answering an Exercise

In an exercise the learner has a possibility to give us a feedback about his knowledge of that particular exercise and save it as an outcome. Each outcome can either be one of the four options:

1. *correct* - when the learner types the correct answer.
2. *wrong* - when the learner types the wrong answer.
3. *show solution* - when the learner does not know the answer, he can look at the solution.
4. *too easy* - when the learner thinks he has learned that particular exercise.

These outcomes help us quantify if the learner knows that particular word and this is where the Exercise Based Probability comes into the picture.

3.2.1 Computing the Exercise Based Probability

Whenever a user bookmarks a word, an initial Exercise Based Probability needs to be saved for that word. The probability cannot be zero, as the user has already seen the meaning of the word he does not understand. Therefore, there is a possibility that the user has memorized the meaning of that word. As a result the initial and minimum probability is set to 0.1.

In addition, whenever the user is doing an exercise, the probability of knowing that particular word increases or decreases depending on the outcome of the exercise. The pseudo code of Algorithm 1 explains how the estimate of the probability that the user knows a given word is to be computed[2]:

This code iterates through all the outcomes of an exercise of a particular bookmark. The probability is computed depending on the outcome:

Correct - we use the data *count_correct_after_another* to boost the probability as described in the code lines 17 to 20.

Wrong - we use the data *count_wrong_after_another* to reduce the probability as described in the code lines 24 to 27.

Too easy - we boost up the probability to 1.0 as described in the code line 7.

Show solution - we halve the probability as described in the code line 10[3]

[2]The Python implementation of this algorithm is to be found at: `https://github.com/karans13/Zeeguu-Web/blob/master/zeeguu/api/model.py`

[3]An alternative would have been to reduce the probability to 0.1. But it is possible that the user knows the word and made a typing mistake and therefore, we only halve the probability.

Algorithm 1 Know Bookmark Algorithm

```
1:  procedure KNOW_BOOKMARK_PROBABILITY(bookmark)
2:      count_correct_after_another ← 0
3:      count_wrong_after_another ← 0
4:      for Each exercise in bookmark.sorted_exercise_log_after_date do
5:          outcome ← exercise.outcome
6:          if outcome = too_easy then
7:              probability ← 1.0
8:              count_wrong_after_another ← 0
9:          else if outcome = show_solution then
10:             probability ← probability ÷ 2
11:             if probability < 0.1 then
12:                 probability ← 0.1
13:             count_correct_after_another ← 0
14:         else if outcome = correct then
15:             count_correct_after_another ← count_correct_after_another + 1
16:             count_wrong_after_another ← 0
17:             new_probability ← probability + 0.1 × count_correct_after_another
18:             probability ← min(1.0, new_probability)
19:         else if outcome = wrong then
20:             count_wrong_after_another ← count_wrong_after_another + 1
21:             count_correct_after_another ← 0
22:             new_probability ← probability − 0.1 × count_wrong_after_another
23:             probability ← max(0.1, new_probability)
```

It is possible that the user has bookmarked the same word multiple times and therefore, the word comes up in many exercises. When this is the case, the algorithm iterates through all the bookmarks with that particular word and saves the average[4] of their probabilities.

Computing the average for multiple bookmarks can be seen in the pseudo code of Algorithm 2[5]. It is executed, whenever a new outcome of a particular exercise is saved in the database.

Algorithm 2 Exercise Based Probability Algorithm

```
    procedure EXP
2:      total_probability ← 0
        count_b_by_user_and_word ← bookmarks_by_user_and_word.count
4:      for Each bookmark in bookmarks_by_user_and_word do
            probability ← know_bookmark_probability(bookmark)
6:          total_probability ← total_probability + probability
        final_probability ← total_probability ÷ count_b_by_user_and_word
```

3.3 Encounter Based Probability

One way in which the Zeeguu Quantifier can approximate the user's knowledge of the vocabulary is to keep track of all the words not looked up in the context in which a bookmark is saved. The list of these words is named *not-looked-up-words*. The assumption is that when a user looks up a word, he knows all other words in the context where the unknown word appears.

However, we cannot be sure that the user knows the words in the *not-looked-up-words* list. For example, it might be that the user did not read all the context with attention. This is where the Encounter Based Probability comes into picture.

3.3.1 Computing the Encounter Based Probability

To increase the confidence that the user knows the words in the previously mentioned list, we count how many times a particular word was encountered but was not looked up. We decide to track only those words that are used most frequently in daily life and therefore are in the Ranked Words List (See Section 3.1). Whenever the user bookmarks a word

[4] An alternative would have been to collect all the outcomes of all the bookmarks and then iterate over all of them at once in Algorithm 1. It was not chosen, because each outcome belongs to the bookmark and not the word.

[5] The Python implementation of this algorithm is to be found at: `https://github.com/karans13/Zeeguu-Web/blob/master/zeeguu/gym/views.py`

we iterate through the *not_looked_up_words* in the context to compute their probabilities and do the following:

1. We check if an Encounter Based Probability exists for that particular word. If that is the case, the number for not looking up a particular word goes up by 1 and 0.1 is added to the probability, as shown in code lines 4, 5 in Algorithm 3[6].

2. Otherwise, the initial probability of 0.5 and 1 is saved as the number of times for not looking up that particular word, as shown in code line 7 in Algorithm 3. The reason behind this probability is that there are doubts if the user understood the gist of the sentence only and not the *not-looked-up-words*.

Algorithm 3 Encounter Based Probability Algorithm

```
    procedure ENC
        for Each ranked_word in context_not_looked_up_words_with_rank() do
3:          if EncounterBasedProbability.exists(user, ranked_word) then
                not_looked_up_counter ← not_looked_up_counter + 1
                probability ← min(1.0, (probability + 0.1))
6:          else
                create_default_encounter_based_probability(0.5, 1)
```

It is also possible that a user bookmarks a word and the same appeared multiple of times in different context, but he had not looked it up at that time. Whenever this occurs, the Encounter Based Probability is set back to 0.5.

3.4 Known Word Probability

It is possible that a word can have both an Exercise Based Probability and an Encounter Based Probability. In such a case, we must merge the two. This is done by giving them certain weights and adding them together.

3.4.1 Computing the Known Word Probability

Firstly, the Exercise Based Probability is computed through the information that the user gives us. On the other hand, the Encounter Based Probability is based on assumptions made about the user. Therefore, the Exercise Based Probability receives a higher weight.

$$P_{\text{known}}(word) = 0.8 \times P_{\text{EX}}(word) + 0.2 \times P_{\text{ENC}}(word) \tag{3.1}$$

[6]The Python implementation of this algorithm is to be found at: `https://github.com/karans13/Zeeguu-Web/blob/master/zeeguu/api/model.py`

There is also a possibility that the Exercise Based Probability exists for a word, but the Encounter Based Probability does not and vice versa. In this case, the Known Word Probability is equal to the probability that exists. If neither of the two probabilities exist, then it means there will be no Known Word Probability.

In the end, all the words that have a probability level of minimum 0.9 belong to the list of *probably-known-words*.

3.5 Computing User Progress Metrics

As explained, the Known Word Probability is saved for every word that the user encounters. This probability can help compute different metrics for the user:

- We compute the percentage from the total bookmarked words of those looked up words in the bookmarks that have a Known Word Probability greater or equal to 0.9.
- According to Sam Gendreau[5], if a person knows *3'000* words and they are used quite often, it can be deduced that he knows the basics of the language. The confidence interval of the percentage about how well the user knows the basic vocabulary of the language is computed with the help of the top *3'000* words[7] in the Ranked Words List. We calculate the lower and upper bound by the following way:
  - lower bound: It is calculated by taking the ratio of the number of the *probably-known-words* in the *BASIC_VOCABULARY_SIZE* to the *BASIC_VOCABULARY_SIZE*.
  - upper bound: It is calculated by taking the ratio of the number of the *not-looked-up-words* in the *BASIC_VOCABULARY_SIZE* to the *BASIC_VOCABULARY_SIZE*.
- We can also compute the percentage of how many words the user knows in the top *10'000* words[8] in the Ranked Words List. This helps the user see if he knows the vocabulary which is now more complex. Here again we proceed as before and make a confidence interval, but instead of the *BASIC_VOCABULARY_SIZE*, we use the *EXTENDED_VOCABULARY_SIZE*.

[7]This number of top *3'000* words in the Ranked Words List we will from now on refer as *BASIC_VOCABULARY_SIZE*.

[8]This number of top *10'000* words in the Ranked Words List we will from now on refer as *EXTENDED_VOCABULARY_SIZE*.

3.6 Future work

3.6.1 Influence of Retries on the Exercise Based Probability

In the current exercises the user has the possibility to try to answer the exercise until either of the two scenarios occur: Either he answers correctly, or he tries until he gives up on the exercise and decides to select *show solution*. How this influences the Exercise Based Probability and to find a solution could be the scope for future work. Counting every retry as an exercise outcome *wrong* would imply that the Exercise Based Probability can decrease rapidly just after doing the exercise once. This also implies that it would take the user more time to reach a higher probability. However, on the other hand, a user who guesses multiple times until he gets the right answer cannot be considered to really be *correct*. Therefore, there is scope for improvement.

A possible solution would be to count the exercise outcome *wrong* just once, independent of the fact that the user retried multiple times. Furthermore, if the exercise is answered correctly, the outcome should get a weight depending on how many times the user retried.

3.6.2 Multiple Types of Exercises

At present we have only one type of exercise in which the user can recognize the translation of the word asked in a sentence. In the future, it is planned to also make different levels of exercises, where we can also test if the user can write the words correctly in the language they are learning. Depending on how well the user answers the exercise of that particular word, it will go up to a higher level. These exercises will also help us to make a better quantification of the user's knowledge. We would either have to create another Exercise Based Probability and somehow, include that in our algorithm or try to boost the current Exercise Based Probability with the data collected through the newly added exercises. Also, different exercises might have different weights since some might be harder than others.

3.6.3 Multiple Translations for a Word

It is possible that the user has answered the exercises correctly, but the exercise system considers the answer as wrong. This can happen, because there can exist synonyms of a word which result in multiple answers, but there is only one translation in the application, saved for a particular bookmark.

In the future, we plan to make sure that the user will have the possibility to add a list of translations to a particular bookmark. This would help us to get a better metric about how well the user is currently answering the exercises and make the quantification of the user's knowledge more accurately.

4 A Meta-Model To Quantify Language Learning

This chapter gives a picture of the original meta-model of Zeeguu and the changes made to it, in order to enable collecting user data and quantifying their progress in learning the new language. In addition, it discusses various alternatives that were taken into consideration in defining the new meta-model.

4.1 Python and SQLAlchemy

The Zeeguu backend infrastructure originally uses Python[1] for implementation. Python is an object oriented programming language which incorporates modules, exceptions, dynamic typing, very high level dynamic data types and classes.

To provide the REST API used by the various clients, Zeeguu uses two well-known Python libraries:

- SQLAlchemy[2] – is an ORM implementation which maps Python classes to relational tables, and provides an API which transparently synchronizes the DB and the application objects.[3]

[1] `https://www.python.org/`
[2] `http://www.sqlalchemy.org/`
[3] Details about this are shown in the Appendix: *Anleitung zu Wissenschaftlichen Arbeiten*

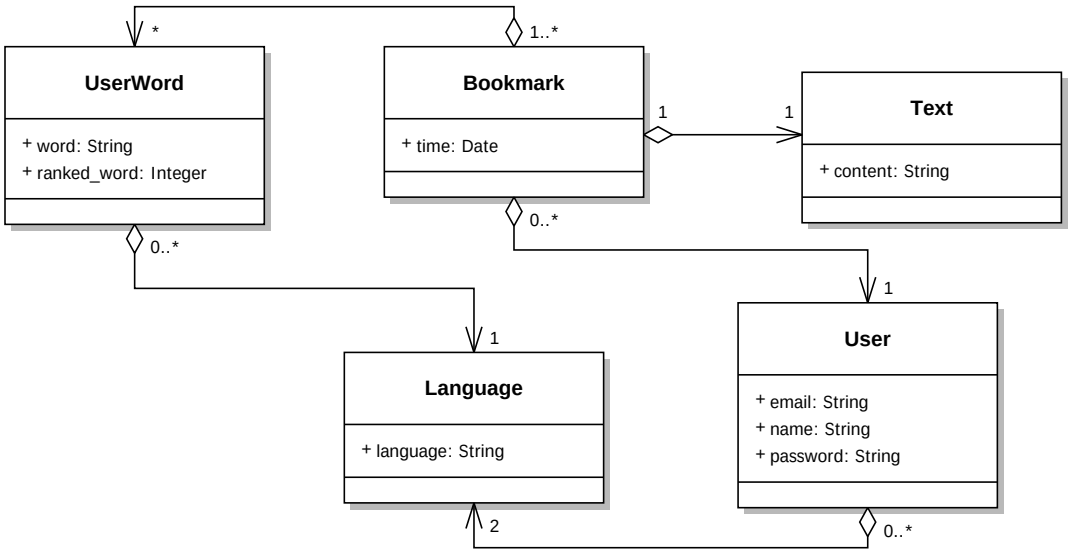

Figure 4.1: Initial Meta-Model of Zeeguu

- Flask[4] – is a lightweight web framework that routes the REST endpoints that the server exposes

In the following sections we describe the meta-model before our contributions and after. We will be referring to classes and their attributes.

4.2 The Original Meta-Model

Figure 4.1 introduces the initial meta-model that was defined before we introduced support for quantifying user knowledge:

- `Bookmark`: As shown in figure 4.1, this class has many different relationships to different classes. One such class is the `User`. It describes who created the bookmark. The other class is `UserWord`. It indicates which word the user bookmarked. Last but not the least, is the class `Text` which represents the content in which the unknown word was looked up[5].

- `User`: This class models the personal data of the user. It has a relationship with the `Bookmark` class. It also references the `Language` class, as it needs to know the native language and the language the user currently is learning[6].

[4] `http://flask.pocoo.org/`

[5] Usually this is the entire paragraph in which the word was looked up.

[6] The model at the moment does not support a user learning multiple languages simultaneously.

- `UserWord`: All the words that were looked up are saved in this class independent of the user. As mentioned, it is referenced by the `Bookmark` class. It also references the `Language` class, as it needs to save the language this word is from. In addition, it also has an attribute ranked_word which tells us if that particular word is frequently used. Details about this are shown in Section 3.1.

- `Language`: This class models all the languages the Zeeguu Platform supports. It is referenced by the classes `User` and `UserWord`.

- `Text`: This class models the content in which the unknown word was looked up. It is referenced by the `Bookmark` class. The reason why the `Text` class exists is because our goal was to normalize the table that has been mapped by the `Bookmark` class.

4.3 Collecting Exercise Data of a User

As explained in the previous section, the words the user translates and bookmarks, are saved in their profile. For these words, the platform offers exercises to help them practice.

The exercise model is designed to express how well the user is performing for each exercise. At a minimum, it should be possible to express the information of whether the answer is correct or wrong in the meta-model. However, other types of information can also be saved with regard to one exercise i.e:

- The user can have the possibility to see the solution without answering the exercise.

- The user can report that the exercise is too easy.

- The time the user needed to solve the exercise.

- The type of exercise the user is currently doing. The model supports multiple types of exercises.[7]

The above mentioned information is also modelled to increase the accuracy of the user knowledge quantification. The updated version of the meta-model can be seen in figure 4.2:

In the figure the `Exercise` class is presented. This class models three important aspects of an exercise:

- time – a timestamp of when the exercise has been created.

- outcome – a relationship with the `ExerciseOutcome` class. It can be either *correct, wrong, too easy or show solution* depending on the user's knowledge.

[7] At the time of writing only one type of exercise exists.

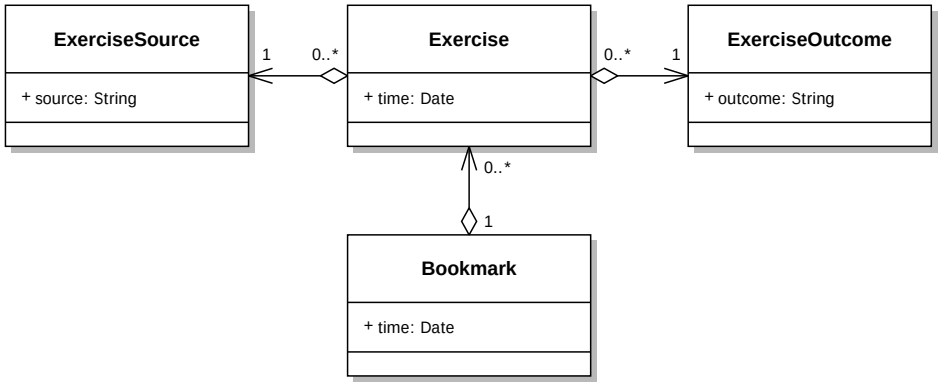

Figure 4.2: Exercise Data Model

- source – a relationship with the `ExerciseSource` class. It tells the type of the exercise the user is currently doing.

We created the two classes `ExerciseOutcome` and `ExerciseSource`, because we intended to normalize the table that had been mapped by the `Exercise` class. Whenever the user tries to solve an exercise, a new instance of the `Exercise` class is created. In the end, we want to be able to obtain a list of exercise outcomes for a particular bookmark.

4.4 Modelling the Ranked Words List

Prior to this project, the Ranked Words List was saved in a plain text file. To be able to use the information the file was parsed. However, this resulted in performance problems. Therefore, we decided to migrate the Ranked Words List to the database.

The newly added `RankedWord` class supports modelling the information about the Ranked Words List. We evaluated two approaches to integrate the Ranked Words List into the meta-model. Below, we describe the two alternatives which explain why we favoured the second solution:

4.4.1 Alternative 1: One Class With All the Words

In the first alternative we normalized the `UserWord` class. This was split into `RankedWord`, `UserWord` and `Word` class. All the words that a user had bookmarked and also the words that were in the Ranked Words List were saved in the `Word` class, independent of the language and the user.

As shown in figure 4.3, the `RankedWord` and the `UserWord` classes have a relationship with the `Word` and the `Language` class. This means, whenever a word from the `UserWord` or the `RankedWord` class is taken, a *join* must be done with the table that has been mapped by the `Word` class.

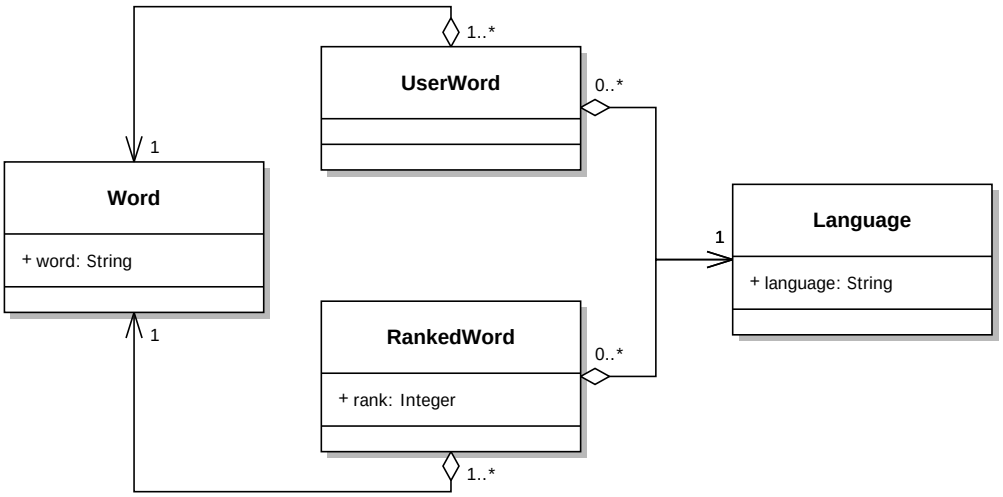

Figure 4.3: Ranked Word Model Alternative I

- Advantages: The normalization of the data means that we are saving memory space. The reason is that a word appears only once, either because it was in the Ranked Words List or because it was in some of the users data.
- Disadvantages: The table which has been mapped by the `Word` class is huge and is queried very often[6]. It has words from `UserWord` and `RankedWord` class independent of the language (and thus it includes all the words of all the languages which are supported by the platform). Performing a *join* with this table can lead to performance degradation. As such, it was taking more than 3 seconds to prepare a new exercise for the user.

4.4.2 Alternative 2: Separate Ranked Word and User Word

This alternative is a denormalized version of the first one. It has no `Word` class, and as such the `UserWord` and the `RankedWord` class have their own list of words. As shown in figure 4.4, the `UserWord` has a relationship with the `RankedWord` class. This implies that the rank for every User Word should also be saved there. If there is no rank for that word, it is saved as *null*.

- Advantages: When preparing the exercise of the user, a *join* is required only with the table that has been mapped by the class `UserWord`. This table is quite small in comparison with the table that had been mapped by the `Word` class in Alternative 1.

  Another advantage is that there exists a clear separation between the Ranked Words List data and the user data. Should there be new word frequency lists available, it will be easy to update the database.
- Disadvantage: This solution is space inefficient. However, the difference is minor as compared to the first alternative.

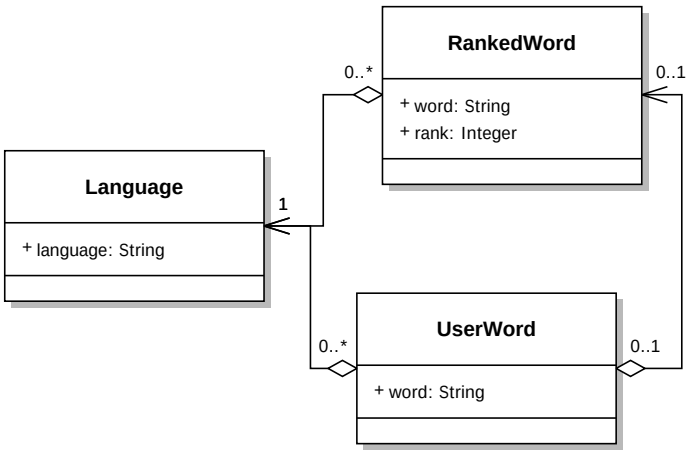

Figure 4.4: Ranked Word Model Alternative II

4.5 Modelling the Probability of Knowing a Word

This section describes how the probabilities are implemented in the meta-model. Details regarding the computation of the probabilities and their usage can be seen in the previous chapter. The following figures describe the design of the probability model.

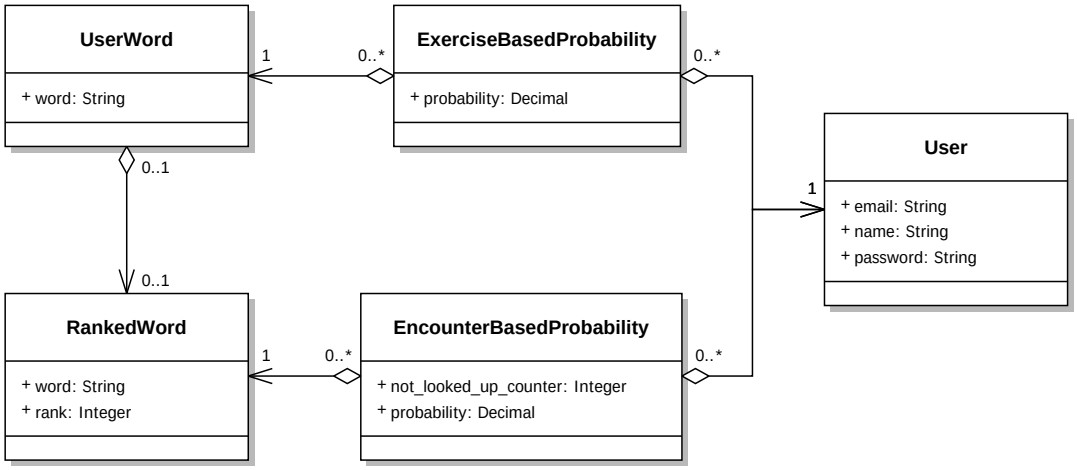

Figure 4.5: Probability Model

4.5.1 ExerciseBasedProbability Class

This probability approximates the user's knowledge of the words that he has bookmarked. Depending on how well the user has done the exercises concerning a given word, the probability of him knowing it will increase or decrease.

The class of this probability as shown in figure 4.5 has a relationship to the `User` and `UserWord` class. Along with this, there is also a probability attribute. Whenever a new word is bookmarked by a user, the Exercise Based Probability is calculated and saved.

4.5.2 EncounterBasedProbability Class

As mentioned in Section 4.2, whenever a word is bookmarked, the context in which the word appears is also saved. The Encounter Based Probability approximates whether the user knows the words that were not looked up by him in the context: The more times a word has been encountered and not looked up, the higher the likelihood that the user already understands it. The probability is only calculated for words that exist in the `RankedWord` class, because we want to quantify the user's knowledge on the basis of words that are used in daily life.

As shown in figure 4.5 the `EncounterBasedProbability` class also has a relationship to the `User` and the `RankedWord` class. The reason for making a reference to the `RankedWord` and not the `UserWord` class is that only the words that are looked up are considered as User Words. Along with this, there is also a probability attribute and not_looked_up_counter attribute. In the last mentioned attribute, the number is saved of how many times that particular word was not looked up in different contexts.

4.5.3 KnownWordProbability Class

The two probabilities which have already been introduced can both exist for one single word. Since both the probabilities are important, it is necessary to merge them together in order to compute the Known Word Probability. This is the final probability of a user knowing a particular word.

There are two ways to put the `KnownWordProbability` class into the model: The first alternative as shown in figure 4.6 was to consider a relationship with the `ExerciseBasedProbability` and `EncounterBasedProbability` class. This way it would be very easy to compute the Known Word Probability.

The second alternative, as shown in figure 4.7 was to have a relationship with the `RankedWord` and `UserWord` class.

In the end, the second alternative was selected. This was because the `ExerciseBasedProbability` and the `EncounterBasedProbability` classes both increase the performance in computing the Known Word Probability, but otherwise do not have much importance in the model. This way it would be easier to compute the percentage of the words the user knows both in the Ranked Words List and User Words for a particular language.

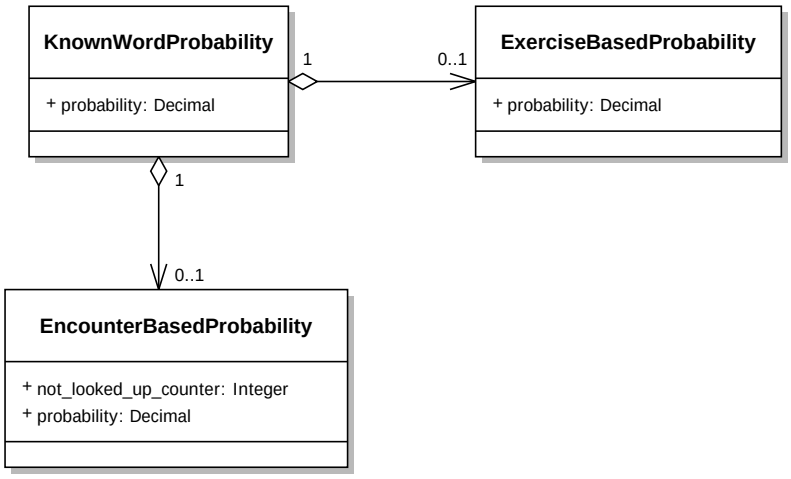

Figure 4.6: Known Word Probability Model Alternative I

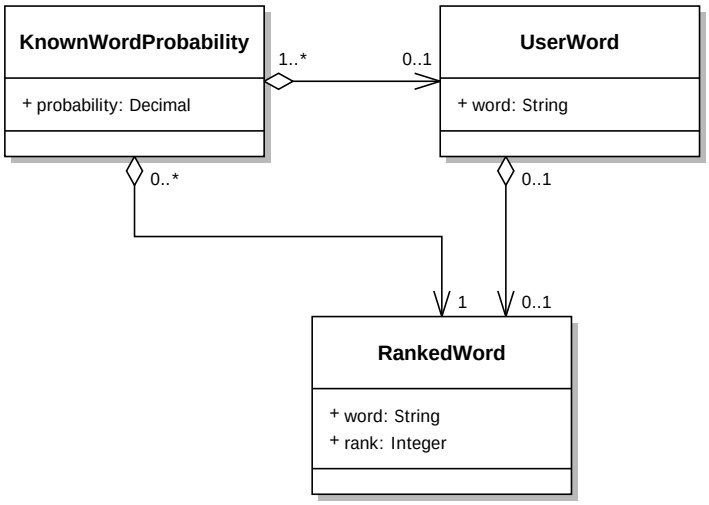

Figure 4.7: Known Word Probability Model Alternative II

5 Case Study

In this chapter we present a Case Study of a user who has been using the Zeeguu Platform for multiple months. We will describe how the user was using the application and analyse the feedback he is getting.

5.1 About the User

The Case Study will be of the user: *Dr. Mircea Lungu*[1]. He has been learning German for a few years and has been using Zeeguu for the past 14 months[2]. During this period, new features have constantly been added to the platform. In particular, the quantifier meta-model has been deployed since 20th August 2015.

5.2 Evaluating the Quantifier Metrics

Below we would like to present *DRML*'s results of different metrics which quantify the current progress of the learner in acquiring the German language vocabulary, analyse them and make interpretations about how well they work for him. The analysis has been executed on 2nd September 2015. The details about the implementation of the APIs which compute these quantifier metrics can be seen in the Appendix: *The Quantifier API*.

[1]From now on we will refer Dr. Mircea Lungu as DRML.
[2]June 2014 - August 2015

Words Being Learned

This metric tells us how many bookmarks are saved for a particular language, or how many words are currently[3] being learned. *DRML* currently has **918 words**.

Words Already Learned

These are the words that have been marked as *too easy* in the exercise section. There are currently **54 words** in his profile. It is approximately 5% of all the bookmarks, which seems quite low. There are three alternative explanations for this low number:

1. We started collecting the exercise data from May 2015, but the bookmarks from June 2014.
2. It is possible that he uses the platform only to look up words. But this seems unlikely, as he has saved 165 bookmarks in his profile and has done 865 exercises between May and August 2015.
3. He is not very sure himself if he has memorised a particular word and therefore, does not mark many exercises as *too easy*.

Micro-experiment #1. To find out whether the user has learned more words than he actually is aware of, he was asked to translate ten German words into English. The words selected were ones he had answered many times correctly in the exercise section but did not mark them as *too easy*. *DRML* was able to answer 4 out of 10 correctly.

Note that this exercise is qualitatively different than the exercises that the platform supports at the moment. The current exercises ask the user to recognize a given word, while the micro-experiment asked the user to recall the meaning of a word – arguably a task with a higher difficulty, which was still correctly solved in 4 out of 10 cases. Also, remember that originally the user had not the confidence that he knows the words, even in the context of the easier exercise.

We conclude that probably the third hypothesis is correct – the user is not confident that he knows words that he actually knows. Perhaps introducing other types of exercises can help the user become more certain about the words he has already learned.

Not Looked Up Words

The number is computed by counting all the *not-looked-up-words*. The reason behind this metric is to give the user an overview of how many words he understands or can interpret while reading something in that language. *DRML* currently has **1845 words**. This is clear when we look at the number of bookmarks saved in his profile. We can now interpret that he has the ability to read simple articles and understand their gist, and therefore has a basic knowledge about the language.

[3] At the date of running this analysis: August 2015

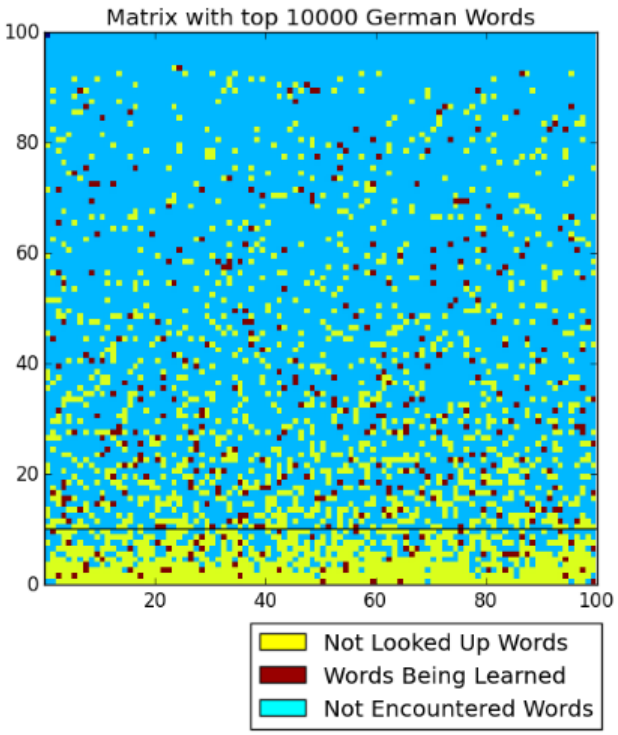

Figure 5.1: A matrix with the top 10'000 words in the German language organized from the most frequently used at the bottom. The colors highlight the words that the user is learning (red), did not look up (yellow) and did not encounter (blue).

$$N = Rank\ of\ a\ Word \tag{5.1}$$

$$x = N\ mod\ 100 \tag{5.2}$$

$$y = floor(N/100) \tag{5.3}$$

The above equations describe how the rank of a word is mapped to each coloured dot at coordinate (x,y) in figure 5.1. The ranks in the figure go up to 10'000. The aim of this chart is to tell us if he has encountered words that are used daily and were looked up by him or not. Below the horizontal line where y = 10, we can see that most of the top 1'000 words have been encountered by *DRML* and are *Not Looked Up Words*. As seen in the chart, he has encountered words that have a higher rank more than the words which have a lower rank. This implies that the Ranked Words List based on movie subtitles has a good ranking.

Not Encountered Words

This is a metric of the words that the user has not yet encountered from our Ranked Words List. This means it gives the user information about the words that have not been encountered, but are used in daily life. Currently, *DRML* has **7155 words** which he has not encountered. There are many, even though he has been learning the language with Zeeguu for 14 months. The reason behind this could be that the words with a higher rank are encountered many times and on the other hand the words with a lower rank are not encountered much or not at all.

Probably Known Words

These words are computed by the heuristic method explained in Chapter 3. The words are added to this list if the Known Word Probability is 0.9 or more.

The current result in our Case Study is **420 words**. This number is much higher than *Words Already Learned*. Below we list the possible reasons for this:

- This number not only consists of bookmarked words which have been learned by the user, but also words which were not looked up, minimum four times.
- The heuristic method is giving too high results.
- *DRML* is not using the feature *too easy* very often. This seems likely, because of the facts mentioned in the *Words Already Learned* metric analysis.

Percentage of Basic Vocabulary Known

This metric is not a value but a confidence interval, because we cannot be sure of the actual percentage. The goal of this metric is that the user can approximate how well he knows the basic vocabulary of the language. The confidence interval is calculated by the computations listed below:

- The lower bound is calculated by taking the ratio of *Probably Known Words* and the top *BASIC_VOCABULARY_SIZE* words in the Ranked Words List.
- The upper bound is calculated by taking the ratio of *Not Looked Up Words* and the top *BASIC_VOCABULARY_SIZE* words in the Ranked Words List.

The interval currently is **11.83% - 40.53%**. This can be problematic since *DRML* might not be able to approximate how well he knows the basic vocabulary, because the interval is quite big. The possible reasons for this could be:

- The lower bound is calculated with the help of the heuristic method. It is possible that the probability level of 0.9 should be lower. This will be discussed in Section 5.3.
- The upper bound is set too high, since it is unlikely that *DRML* knows all the words he encountered with Zeeguu.

We could ask ourselves a question for future research work, if the confidence interval narrows the more the user uses the platform.

Percentage of Extended Vocabulary Known

This metric is also a confidence interval. The goal of this metric is that the user can approximate how well he knows not only the basic, but also the complex vocabulary of the language. The confidence interval is calculated by the computations listed below:

- The lower bound is calculated by taking the ratio of *Probably Known Words* and the top *EXTENDED_VOCABULARY_SIZE* words in the Ranked Words List.
- The upper bound is calculated by taking the ratio of *Not Looked Up Words* and the top *EXTENDED_VOCABULARY_SIZE* words in the Ranked Words List.

The interval currently is **3.93% - 18.45%**. This makes sense looking at the results of the metric *Percentage Of Basic Vocabulary Known*. This interval, in comparison with the previous one, is more narrow. *DRML* can make a much better approximation about how well he knows the vocabulary of the language, which goes beyond the basics.

Percentage of Probably Known Bookmarks

The percentage is calculated by the ratio of the *probably-known-words* which were bookmarked and the total amount of bookmarks saved in the profile. The result is **6.75%**. This seems quite low, especially in the light of the fact that *DRML* actually solves many exercises. We thus organized a micro-experiment to verify whether this number is realistic:

Micro-experiment #2. To be able to better judge this, we asked *DRML* to translate 24 words having the Exercise Based Probability greater or equal to 0.6. (There were no words with probability of 0.8) The result were the following:

- Out of 12 words which had a probability between 0.6 and 0.7, he was able to answer 3 out of 12 correctly which is only 25%.
- Out of 12 words which had a probability between 0.9 and 1.0, he was able to answer 9 out of 12 correctly which is 75%.

According to this analysis the probabilities seem to be appropriate, and the probability level of 0.9 is a very good approximation. Therefore, we can assume that the result of **6.75%** is not far away from being correct.

5.3 Probability Level of Knowing a Word

In Section 5.2 we expressed doubts about the probability level.

Micro-experiment #3. To analyse this possible problem we took ten words from each of the Known Word Probability 0.6, 0.7, 0.8, 0.9 and 1.0 and asked *DRML* to translate them in his native language English. The result came out the following way:

- For 0.6, 0.7 and 0.9 *DRML* answered eight out of ten correctly.
- For 0.8 and 1.0 *DRML* answered ten out of ten correctly.

DRML was able to translate almost all the words independent of the Known Word Probabilities they have.

Therefore, we could come to the conclusion that the probability level is too high and should be somewhere in the middle of 0.6 and 0.9. The results are different than in Section 5.2 in the metric *Percentage of Probably Known Bookmarks*. The reason behind the difference is that this metric can also consist of words of the metric *Not Looked Up Words* that have been encountered by the user many times in a context. We cannot guarantee that these results give us the right picture, as we took ten random words of the above mentioned probabilities.

5.4 User Involvement

In this section we will try to analyse the user's involvement with the platform. To do this, we will be looking at two charts representing (1) the bookmarks saved and (2) the exercises solved per month.

Figure 5.2 shows that *DRML* has been saving the bookmarks in his profile every month, but not very consistently. On average, he has been saving 61.2 bookmarks a month.

As shown in figure 5.3 he has also been doing the exercises every month, but not very consistently. E.g in the month of June he did approximately four times more exercises than any other month. The exercises in the chart can also contain duplicate exercises, if the user has come across them many times. On an average, he has solved 216.25 exercises a month.

Analysing the figure 5.4, we can see that this chart almost looks the same as the previous one. This makes sense, because depending on the number of exercises he solves, his knowledge should increase and also the exercises marked as *too easy*.

Looking at the figures 5.2 and 5.3, we can see that in June 2015 *DRML* has been investing a lot of time learning German with Zeeguu. But lately in the last two months, we see that he has been doing more exercises than saving bookmarks into his profile. One of the possible explanations for this is a new feature in the Zeeguu Chrome Extension, which redirects a user to the Zeeguu exercise page any time he is going to one of the web pages: Facebook, Twitter, and GMail.

From figures 5.3 and 5.4 we see that although *DRML* did fewer exercises per month, he marked proportionally more words as *too easy*. This might mean that he started getting bored by some of the exercises.

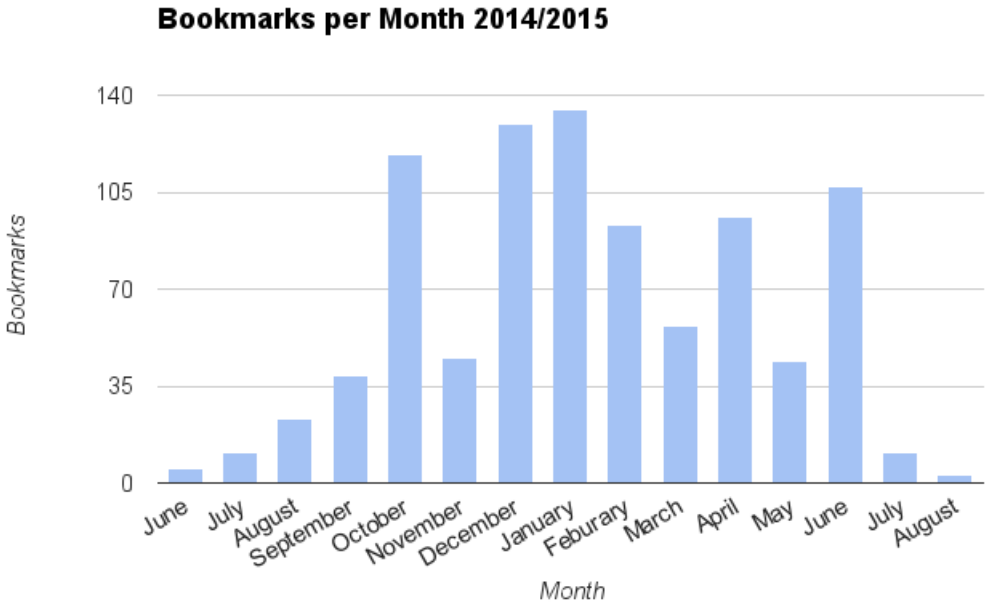

Figure 5.2: Bookmarks per Month Chart

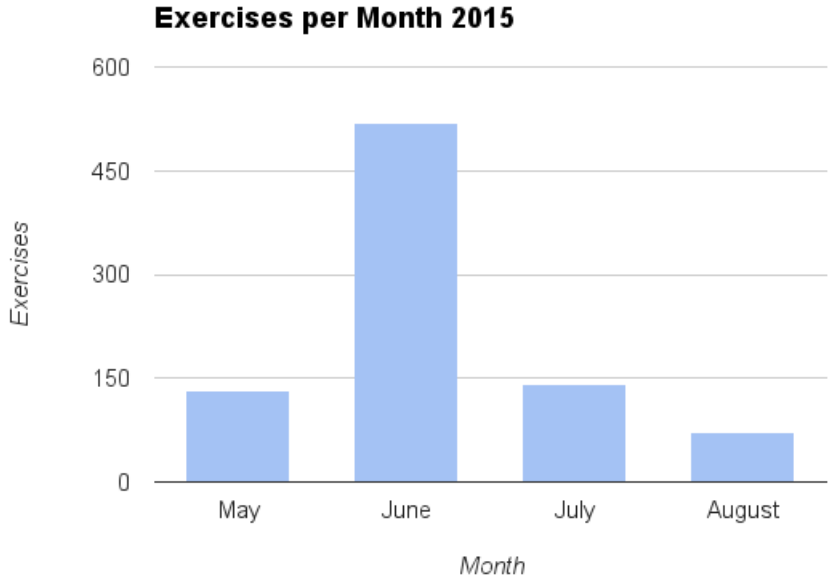

Figure 5.3: Exercises per Month Chart

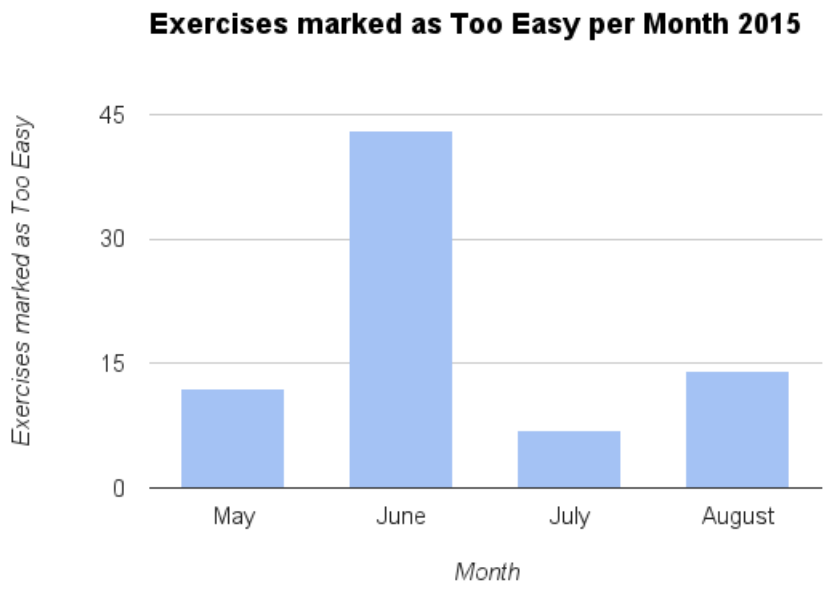

Figure 5.4: Too Easy Exercises per Month Chart

5.5 Implications of Independent Study

We have also asked an expert native German speaker to answer the following questions, by analysing the user data of *DRML* to investigate the threats of using the Zeeguu Platform. The questions and answers in percentage are listed below:

Q1: Are the translations that the user bookmarks correct? What is the percentage of incorrect translations?

A1: The percentage of incorrect translations was approximately 2% given by the expert. This is roughly 18 out of 918 words which were bookmarked. This is a lot considering that the user is learning 1 wrong translation out of 51.

Q2: Is the user learning unusual words, words that a native speaker would not actually use? What is the percentage of such words?

A2: After going through all the bookmarks, the expert reported that approximately 5.5% of the words *DRML* has bookmarked are not used by a native speaker on a daily basis. This is roughly 50 out of 918 words. Most of them are not in the top 10'000 ranked words. This implies that the Ranked Words List is a good approximation of words used in daily life.

5.6 Threats to Validity

As explained in Chapter 3, the heuristic method is based on the Ranked Words List. Below we list the problems that are or can be created if the list is not appropriate for the platform:

- It is possible that the Ranked Words List does not match the words encountered in daily life, and therefore can create uncertainty if the quantification gives the right results. However, as we have seen in this section, the fact that the user encounters many more frequently used words which are ranked higher, is a validation that the list is a promising foundation.
- There are certain words that have almost the same meaning and are spelled similarly, but have a different rank, e.g a singular noun and its plural form. There is a possibility that the user has encountered the singular noun many times, but never the plural form. Unfortunately, the platform does not boost up the probability of its plural form even though the singular noun has been encountered many times.

The problems related to the Ranked Words List imply that there is a possibility that the quantification of the user's knowledge is not very accurate.

6 Conclusion and Future Work

In this chapter we are looking at the goals we wanted to achieve in the beginning of the project, and also analyse if they were fulfilled. We also want to give an overview about the experiences gained during this project. At the end of this chapter we will describe future works related to this project.

6.1 The Contributions of this Thesis

The contributions of this thesis are the following:

1. A meta-model that can support the monitoring and quantification of a user's progress in learning a new language.
2. An algorithm for computing how well the user is progressing in learning a given word based on exercises.
3. An algorithm that helps estimate whether a user knows a given word.
4. Definition of multiple metrics that can be used to provide feedback to a learner.
5. A Case Study in which we analyse how accurate the feedback metrics are of a long term user of the platform.
6. An implementation of the meta-model, and algorithms which are integrated in the Zeeguu Platform together with a set of 11 test cases.[1]

[1]The Python test cases are to be found at: `https://github.com/karans13/Zeeguu-Web/blob/master/zeeguu/tests/api_tests.py`

6.2 Personal Lessons Learned in this Project

Before I started, I had imagined the project to be only about quantifying the user's knowledge of a particular language, since I did not have much experience in data analysis. As it turned out the project did not only consist of that, but it had a lot more to offer.

1. A major preparatory step was required. We needed to upgrade the existing meta-model to be able to collect and quantify user data. To do this, I needed to learn about Python, SQLAlchemy and Flask. Some of the lessons are reported in the Appendix: *Anleitung zu Wissenschaftlichen Arbeiten*.

2. It was necessary to get users to use the Zeeguu Platform. I made two spike efforts towards this goal:

   - I presented Zeeguu to a group of people who converse with each other on a weekly basis to improve their German.
   - Moreover, I visited the *Migros Klubschule* together with the Zeeguu Team to present the Zeeguu Platform to the students and convince them to use it.

   In the beginning all of them were fascinated by Zeeguu. Everybody wanted to install it and use it as soon as possible. Unfortunately, this did not last long. Probably, because it is still under development or it is still quite complicated to install the different components of the application, which are:

   - Chrome Extension
   - Android Application - where it is necessary to join the *Google Plus Community* to be able to download the alpha-test version.

   The goal was to use the user data of these people and check if my algorithm for computing the metrics about their knowledge was of reasonable use.

6.3 Future Work

The work presented here can and must be continued in several directions:

Creating a Ranked Words List

At present we are using a Ranked Words List based on movie subtitles. In the future when we have enough users using Zeeguu, we plan to analyse the encountered words used by most of them and create a new Ranked Words List. This way the list can be changed dynamically and we can quantify the user's knowledge more accurately.

Creating a User Interface for the Metrics

Now that we have computed different kinds of metrics, our goal in the future is to show them to the user in a very nice way. Below we try to describe how we plan to do this. We categorize the metrics in two groups. In the charts the time interval can be set by the user:

- Metric about Progress: We introduce three types of metrics of this category. Our goal is to create a bar chart for each one of them:
  - Percentage of Known Words per Day, Month or Year: We want to show the user how much progress he is making daily, monthly, or yearly.
  - Bookmarks per Day, Month or Year: This metric tells the user how many words he has bookmarked daily, monthly or yearly. The goal of this chart is to motivate and inspire the user to do more.
  - Exercises per Day, Month or Year: This metric is similar to the previous one. The goal is that the user can see how many exercises he does daily, monthly or yearly.
- Metric about Speed: We introduce two similar metrics of this category which can be displayed differently:
  - Words are learned per Day, Month or Year: This metric should be displayed as a bar chart. It should tell the user how many new words he has learned in the past few days, months or years.
  - Words learned as a Message: This metric is displayed as a message to the user. It should give the user the feeling that he is communicating with Zeeguu. We visualize this as shown in figure 6.1:

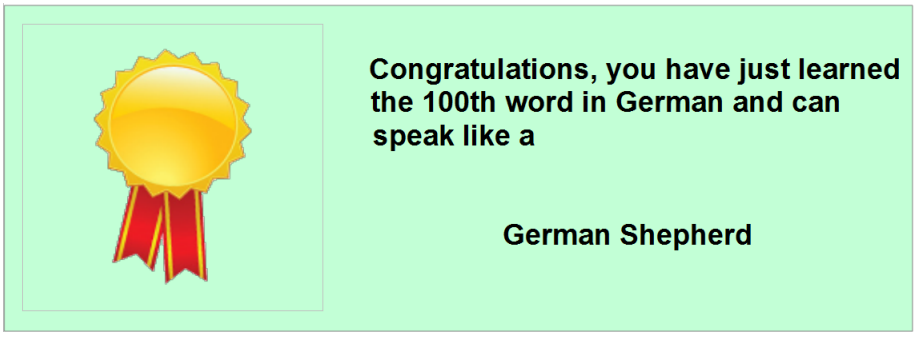

Figure 6.1: Potential Encouraging Metric Message

Creating Exercises for the User

There are users who are too lazy to read something on the internet and look up words they do not understand, but they would like to do some exercises. Therefore, we are thinking of creating

exercises independent of the words the user bookmarks. Depending on how well the user answers these exercises, we would give them more or less difficult ones to solve. There are two ways how we can proceed:

- We can create them by going through the Ranked Words List and create exercises for words that are used most often. This way we would have more information which words the users know in the Ranked Words List and therefore, we could make quantification of a user's knowledge more accurate.
- We can follow a Language Learning Book and make exercises based on it. This way we would be sure that we are asking a person who does not know the language very well, the right questions.

Promoting Zeeguu

Now that we have the Zeeguu Platform in a state where the user can look up words, practice them and even get different metrics, it is time to make the next big step which is to promote it. This would consist of getting the product to the customer, advertising it, speaking to the customers and getting feedbacks about the product. This would make Zeeguu known to the people, we would get to know what people think about it and improve the metrics even better. It would also give us a lot of user data to work with and improve the quantification of the user's knowledge about the language.

Bibliography

[1] Simon Marti. *A Platform for Second Language Acquisition Through Free Reading and Repetition.* University of Berne, 2013.

[2] Pascal Giehl. *The Zeeguu Translate Application.* University of Berne, 2015.

[3] Stephen Krashen. *False Claims About Phonemic Awareness, Phonics, Skills vs. Whole Language, and Recreational Reading.* http://www.nochildleft.com/2003/may03reading.html, 2003.

[4] Maria Dimitropoulou, Jon Andoni Duabeitia, Alberto Avils, Jos Corral and Manuel Carreiras. *Subtitle-Based Word Frequencies as the Best Estimate of Reading Behavior: The Case of Greek.* http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3153823/, 2010.

[5] Sam Gendreau. *How many words do I need to know? The 95/5 rule in language learning.* http://www.lingholic.com/how-many-words-do-i-need-to-know-the-955-rule-in-language-learning-part-2/.

[6] Arshad Ali. *Partitioning in SQL Server.* http://www.databasejournal.com/features/mssql/partitioning-in-sql-server-part-1.html, 2012.

A

Anleitung zu Wissenschaftlichen Arbeiten

This chapter provides an overview of ORMs. As an example of ORM, we will explain the SQLAlchemy in detail. It is a tool kit of the programming language Python which is used in the Zeeguu Platform.

ORM stands for Object Relational Mapping[1]. The goal of an ORM system is to automatically convert data between the object-oriented world and the relational databases world. It creates the perception of a virtual database within the language.

A.1 SQLAlchemy

Any non trivial application has a database behind it, and since Python is no different, a series of solutions for using URML in Python exist. SQLAlchemy is such an ORM solution for Python. It gives full control and flexibility of SQL to the developers and is designed for efficient and high performing database access. SQLAlchemy communicates with an underlying database, e.g in the case of Zeeguu MySQL is used.

A.1.1 Constraints in SQL

Before discussing about SQLAlchemy, we need to introduce the SQL Constraints which are used in different columns of SQL tables. We will mention these many times in the future sections of this chapter:

[1] `https://en.wikipedia.org/wiki/Object-relational_mapping`

- Not Null: This tells the DB engine who enforces this constraint later that the column cannot be *null*.
- Unique: This makes sure that one or many columns have unique values.
- Primary Key: It is a combination of *unique and not null*. This helps us to find data in a table easily.
- Foreign Key: It makes sure that the data matches to another table if there is a relationship between them.

A.1.2 Mapping Classes to Tables

Normally, we model the tables directly in the database in which there are different columns to be considered. For example, the `user_word` table as shown in figure A.1:

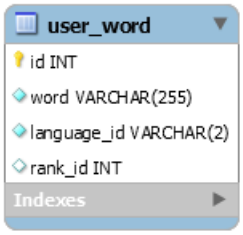

Figure A.1: `user_word` table

- id - a unique identifier for a column also called the primary key.
- word - a column which has a textual data type *VARCHAR*.
- language_id - a column that is a foreign key and references to the `language` table.
- rank_id - a column that is a foreign key and references to the `ranked_word` table.

The tables can also be modelled when we utilize SQLAlchemy, the toolkit of Python. We need to define classes which are then mapped to the tables[2] as shown in figure A.2. The code of the figure A.2 is explained here below:

- __tablename__: It defines which table this class maps to. Here it is `user_word`.
- __table_args__: It defines which encoding this table uses. In this case, it is *UTF8_bin*. UTF8 can represent any Unicode character, but still allows representation of ASCII. It supports various non-English alphabets such as Arabic or Chinese. This encoding helps us save words from different languages. *UTF8_bin* performs case sensitive comparisons, because it compares binary values. This we cannot ignore, since the database can have words with different meaning depending if they are written lower case or not.

[2] `http://docs.sqlalchemy.org/en/rel_1_0/orm/tutorial.html`

```
class UserWord(db.Model, util.JSONSerializable):
    __tablename__ = 'user_word'
    __table_args__ = {'mysql_collate': 'utf8_bin'}

    id = db.Column(db.Integer, primary_key=True)
    word = db.Column(db.String(255), nullable =False, unique = True)
    language_id = db.Column(db.String(2), db.ForeignKey("language.id"))
    language = db.relationship("Language")
    rank_id = db.Column(db.Integer, db.ForeignKey("ranked_word.id"), nullable=True)
    rank = db.relationship("RankedWord")
    db.UniqueConstraint(word, language_id)
```

Figure A.2: `UserWord` Model in SQLAlchemy

- id - is an identifier of a column in a table and is known as the primary key.
- word - is an attribute of type *String* which is automatically mapped to a *VARCHAR* column in SQL. We can define many properties for any column in SQLAlchemy, e.g if it is *null* or *unique*.
- language_id - it is a foreign key referencing the `language` table.
- language - it is an attribute of the `Language` class. It is created with the help of the function *relationship()*.
- rank_id - it is a foreign key referencing the `ranked_word` table. We can also define this key as *null*.
- rank - it is created similar to the language attribute.
- UniqueConstraint - this can be declared for many or one column as unique. This helps us to find the data much easily by using a query, e.g wanting to know if that particular data already exists in a table or if we need to add it.

Then we can make a constructor with the attributes defined in figure A.3:

```
def __init__(self, word, language, rank = None):
    self.word = word
    self.language = language
    self.rank = rank
```

Figure A.3: `UserWord` Constructor

This helps us to create instances of the classes that are mapped to the tables in the database. Whenever we want to add new data into the database, we create new instances of that class and add them to our session. At this point, the instances added are still pending, which means no row has been created in SQL. Moreover, it is also possible to delete instances by creating queries to find that particular data. In the end we need to commit the data, so that all remaining changes are flushed to the database and the transaction committed.

A.1.3 Association Tables

Whenever there is a many to many relationship between two tables, an association table is added between the two. In Chapter 4 we had the same situation, when we created the exercise data model.

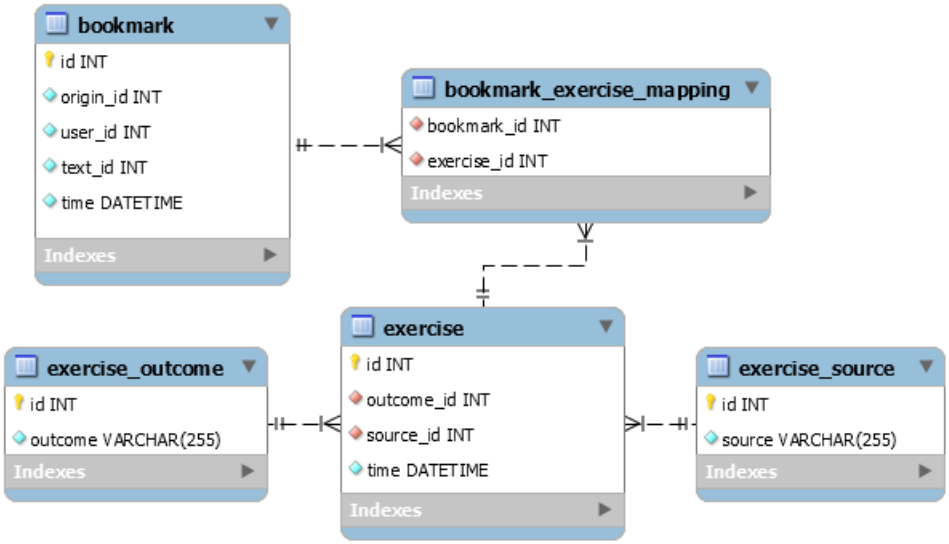

Figure A.4: Exercise Model

As shown in figure A.4 we have an `exercise` table which has a relationship to the tables `exercise_outcome` and `exercise_source`. Moreover, we have a `bookmark_exercise_mapping` table which references the `bookmark` and `exercise` tables. This is an association table, which connects a list of exercises with a particular bookmark.

```
bookmark_exercise_mapping = Table('bookmark_exercise_mapping', db.Model.metadata,
    Column('bookmark_id', Integer, ForeignKey('bookmark.id')),
    Column('exercise_id', Integer, ForeignKey('exercise.id'))
)
```

Figure A.5: Association Table in SQLAlchemy

As shown in figure A.5 an association table can also be defined in SQLAlchemy. It is not defined as a normal class but as a table. The association table is indicated in the corresponding classes by the *secondary* argument of the function *relationship()*. This way it is possible to compute a list of exercise instances of a particular bookmark instance.

A.1.4 SQLAlchemy Queries

In SQL we perform queries to get data from one or many tables of a database. This is also possible with SQLAlchemy. Here is an example of a simple query in SQL and SQLAlchemy:

```
SELECT *
FROM bookmark
WHERE id = 1
```

Figure A.6: SQL Query

```
bookmark = Bookmark.query.filter_by(
    id=1
).first()
```

Figure A.7: SQLAlchemy Query

As shown in figure A.6 we want to get data with *id* = 1 from the `bookmark` table. This can also be executed with SQLAlchemy as shown in figure A.7.

The SQLAlchemy code uses the function *filter_by()*. By this, we are able to filter by the attributes defined in that particular class. As we know the *id* is unique and so there will be only one result in this query. This is the reason we can use the function *first()* and get the first result in this query. In the end, we get an instance which contains the wanted data.

A.1.5 Evolving the Domain Model

As the application progresses many changes are made, as well in the database. This means, new tables with data are migrated. Adding new tables to an existing database is done via SQL scripts. After this, it is necessary to add the classes that are mapping to these tables into the application. When it came to compute complex data for these tables, a new Python script had to be created.

These scripts are independent from the application and need to be executed just once usually, when an upgrade of the database is performed, or when data needs to be initialized in a given table. SQLAlchemy is used to make changes to the database. The script that makes changes in the database has usually the following five steps:

1. Get the required data from the database and also the external data that is used to make computations.
2. If required make computations with this data to get new data.
3. Make new instances of the class that is mapped to the particular table where the data needs to be added.
4. Add the instances to the session.
5. Commit the changes in the session.

In this project the Python scripts were used many times. These scripts are introduced below:

- populate: In the application there is also a database for testing purpose. This makes sense as we do not want to change anything in the main database while doing the tests. For this purpose, a script was created. This has the following steps:
  1. Delete all the tables
  2. Create them depending on the model
  3. Add the test data into the tables.

The script is executed whenever a test needs to run. It is also extended with new test data depending if a newly added API needs to be tested.

- populate_probability: This script has the responsibility to add a default Exercise Based Probability and Encounter Based Probability into the newly created `exercise_based_probability` and `encounter_based_probability` table.
- populate_boost_prob: After the populate_probability script is executed, the script is responsible to boost both the probabilities, depending on the conditions explained in Chapter 3. In addition to this, it also needs to compute the Known Word Probability and add it to its table in the database.
- add_ranked_words: This script has two responsibilities: (1) to migrate the vocabulary words which are ranked based on their frequency of occurrence in the target language, from plain text to the database; and (2) to modify the information of the Ranked Words List, so that we can give the users the right metric about the words they are currently learning.

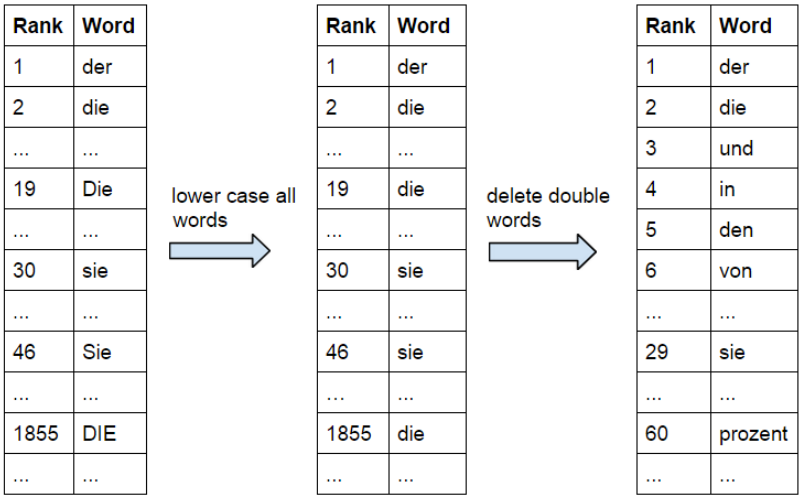

Figure A.8: Modification of the Ranked Words List based on Movie Subtitles

The initial Ranked Words List can have a different rank depending on whether it is upper case or lower case. As shown in figure A.8 the words are converted to lower case. After that, all the words that come up more than once are deleted. It is very important that the order remains as it is and that we iterate through the list starting with rank one. This way one is sure not to delete the word that comes up in the list more than once with a higher rank. The only disadvantage of this approach is that in some languages there are some words that mean differently, depending on their cases. e.g as shown in figure A.8 the word *sie* in German has a different meaning, depending on if its written in upper or lower case. Luckily, this is just very seldom and can be ignored.

B
The Quantifier API

As shown in figure B.1 the architecture of the Zeeguu ecosystem consists of a series of components that can be grouped in Front-End and a Back-End. Here we elaborate on the backend since this is the concern of this thesis.

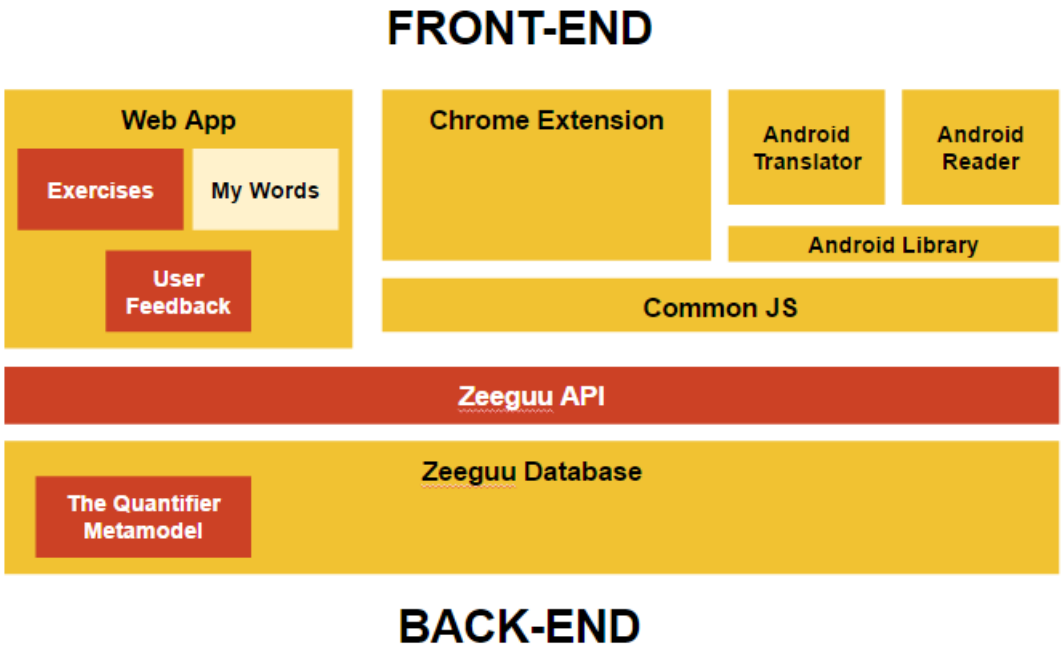

Figure B.1: Zeeguu Architecture Diagram

- The Quantifier Metamodel: The meta-model (presented in Chapter 4) describes the different properties of all the entities in the domain of the application. The meta-model is defined in Python model and the corresponding database and relationships are generated by SQLAlchemy. (see Appendix: *Anleitung zu Wissenschaftlichen Arbeiten*)
- Zeeguu API: To be able to make changes in the database, the components in the *Front-End* need to send a *get or post* request to the *Zeeguu API* and get a response depending on the request.

The points below describe all the quantifier APIs[1].

- get_known_bookmarks(lang_code): This API gives a list of all bookmarks that were marked as *too easy* during the exercises of a particular language given by the parameter *lang_code*. For a bookmark to be in the list, the latest exercise outcome of the bookmark should not be *show solution* or *wrong*, but can be *too easy*. If it is *correct* the procedure must be repeated with the next latest exercise outcome.
- get_not_looked_up_words(lang_code): This API gives a list of all the *not-looked-up-words* from the contexts where the user bookmarked a word of a particular language given by the parameter *lang_code*. This list approximates how many words the user already knows or has learned in the meantime. As a matter of fact, we cannot be sure if the user has understood all these words in the context, or if he only understood the gist of it and therefore, did not bookmark those words.
- get_probably_known_words(lang_code): This API gives a list of all the words that have an Known Word Probability of greater or equal to 0.9. In this way, we can be certain that the user knows the words this list contains. It includes an approximation of words the user already knew and has learned.
- get_percentage_of_known_bookmarked_words: Thanks to the previous API, it is possible to compute a percentage of how many words the user knows from the words bookmarked. This will tell the user how many User Words he knows and if more time has to be spent doing the exercises.
- get_lower_bound_percentage_of_basic_vocabulary: This API approximates the lower bound about how well the user knows the basic vocabulary. It is computed by calculating the ratio of *probably-known-words* and the top *BASIC_VOCABULARY_SIZE* words of the Ranked Words List.
- get_upper_bound_percentage_of_basic_vocabulary: This API approximates the upper bound about how well the user knows the basic vocabulary. It is computed by calculating the ratio of *not-looked-up-words* and the top *BASIC_VOCABULARY_SIZE* words of the Ranked Words List.

[1]The Python implementations of these APIs are to be found at: `https://github.com/karans13/Zeeguu-Web/blob/master/zeeguu/api/endpoints.py`

- get_lower_bound_percentage_of_extended_vocabulary: This API approximates the lower bound about how well the user knows the extended vocabulary which includes more complex words. It is computed by calculating the ratio of *probably-known-words* and the top *EXTENDED_VOCABULARY_SIZE* words of the Ranked Words List.
- get_upper_bound_percentage_of_basic_vocabulary: This API approximates the upper bound about how well the user knows the extended vocabulary which includes more complex words. It is computed by calculating the ratio of *not-looked-up-words* and the top *EXTENDED_VOCABULARY_SIZE* words of the Ranked Words List.