Parlira

An Interactive Phrasebook for Android Devices

Bachelor of Science Thesis in Computer Science and Engineering

BJÖRN HEDSTRÖM
MATILDA HORPPU
DAVID MICHAËLSSON

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden, June 2016

Parlira
An Interactive Phrasebook for Android Devices
BJÖRN HEDSTRÖM
MATILDA HORPPU
DAVID MICHAËLSSON

Examiner: Niklas Broberg, Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The logotype for Parlira designed by Björn Hedström.

Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Parlira
An Interactive Phrasebook for Android Devices

BJÖRN HEDSTRÖM
MATILDA HORPPU
DAVID MICHAËLSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Bachelor of Science Thesis

Abstract

This bachelor thesis documents the development of Parlira. Parlira is an offline phrasebook application for Android devices with a foundation built on the functional programming language the Grammatical Framework(GF). The goal of the project was to develop an application where the user can alter different phrases to fit specific situations. The main problems were to create an intuitive user interface with dynamic updating of the translations, as well as making the application configurable.

During the project, several commonly used practices in user interface design and software development were used and reviewed for example Scrum and user stories. They were used in order to ensure the fulfillment of the requirements and the smooth development of the application.

The project has resulted in an application that supports dynamic updating by representing the abstract syntaxes of GF with a tree structure in Java. As a result of this, a user can choose a phrase and alter different parts of it according to his or her needs. All configurations of the available phrases are stored in an extensible XML file. This means that new phrases can be added by just changing the configuration file without altering the code of the application.

Keywords:Grammatical Framework, Android, translation

Sammandrag

Denna kandidatrapport dokumenterar utvecklingen av Parlira. Parlira är en frasbokssapplikation för Android-enheter som baseras på det funktionella programmeringsspråket Grammatical Framework(GF) och fungerar utan internetuppkoppling. Målet med projektet var att utveckla en applikation där användaren kan ändra i olika fraser för att dem ska passa specifika situationer. De främsta problemen att lösa var att skapa ett intuitivt användargränssnitt med dynamiskt uppdaterande översättningar och att göra det lätt att konfigurera applikationen.

Under projektets gång användes flera beprövade metoder inom design av användargränssnitt och mjukvaruutveckling, exempelvis Scrum och användarberättelser. Detta för att försäkra att de specifierade kraven skulle uppnås samt att utvecklingen skulle fortgå på ett effektivt sätt.

Projektet resulterade i en applikation som tillhandahåller dynamisk uppdatering genom att representera GF:s abstrakta syntaxer med hjälp av en trädstruktur i Java. Till följd av detta kan en användare välja en fras och ändra vissa delar av den baserat på hens behov. Alla konfigureringar av de tillgängliga fraserna lagras i en utökningsbar XML-fil. Detta betyder att nya fraser kan läggas till i XML-filen utan att behöva ändra i koden för applikationen.

Nyckelord:Grammatical Framework, Android, översättning

Acknowledgements

We would like to especially thank our supervisor Krasimir Angelov. Without his support, guidance and expertise within the field this project would not have been possible to complete. Everyone who helped us with the testing of the application also deserves to be thanked. Without their help many bugs would not have been discovered and the application would probably not be as user friendly as it is.

BJÖRN HEDSTRÖM, Gothenburg, JUNE 2016
MATILDA HORPPU, Gothenburg, JUNE 2016
DAVID MICHAËLSSON, Gothenburg, JUNE 2016

Glossary

API: Application Programming Interface, specifies how software can communicate with another given software
Origin language: The language to be translated from
Target language: The language to be translated to
Abstract syntax: An abstract representation of a natural language
Syntax tree: A tree data structure used to represent phrasebook phrases
Scrum: A working method used in software development
Interlingua: Intermediary language

Contents

1 Introduction 3
1.1 Background . . . 4
1.2 Purpose . . . 5
1.3 Problem description . . . 5
1.3.1 Dynamic User Interface . . . 5
1.3.2 Back end configurability . . . 5
1.4 Scope . . . 6

2 Methods 7
2.1 Research and Project Planning . . . 7
2.1.1 Research . . . 7
2.1.2 Work Procedures . . . 7
2.2 Practices . . . 8
2.2.1 Personas and User Stories . . . 8
2.2.2 User Tests . . . 9

3 Technical Background 11
3.1 Android Development . . . 11
3.1.1 Application Structure . . . 11
3.1.2 Material Design . . . 12
3.2 XML . . . 12
3.3 Grammatical Framework . . . 13

4 Design Process 15
4.1 Identifying User Needs . . . 15
4.1.1 Personas . . . 15
4.1.2 User Stories . . . 16
4.2 Designs at Different Stages and User Tests . . . 17
4.2.1 Initial designs . . . 17
4.2.2 Paper Prototype User Test . . . 19
4.2.3 Software Prototype User Test . . . 21

5 Implementation 23
5.1 Representing Sentences . . . 23
5.1.1 XML Configuration File . . . 23
5.1.2 Syntax Tree Structure . . . 26
5.2 Dynamic Updating . . . 27

6 The Final Product 29
6.1 Application . . . 29
6.2 Features of Parlira . . . 29

7 Discussion 33
7.1 Method Discussion . . . 33
7.1.1 Design Process . . . 33
7.1.2 Implementation Process . . . 34
7.2 Result Discussion . . . 35
7.2.1 The Application . . . 35
7.2.2 Further Development . . . 36
7.2.3 Parlira in Society . . . 37

8 Conclusion 39

A User stories I

B Sketches V

C Paper Prototypes IX
C.0.1 Paper Prototype Test Specification . . . IX
C.0.2 Summary of Observations . . . XII

D Usability Test Plan XIII
D.1 Test Specification . . . XIII
D.2 Product Under Test . . . XIII
D.3 Business Case . . . XIV
D.4 Participants . . . XIV
D.5 Equipment . . . XIV
D.6 Responsibilities . . . XIV
D.7 Location and Dates . . . XIV
D.8 Follow-up Questionnaire . . . XIV
D.9 Summary of Results . . . XV

1 Introduction

When travelling abroad one of the first obstacles encountered, which can often be proven to be the most significant, is the language barrier. The ability to communicate with others is key to not only making travelling enjoyable, but can also be of utmost importance in emergency situations. Even if many regions are proficient in English, some territories still require travellers to either have basic knowledge of the native language, or some sort of means for translation.

Most travelers in the past have made use of a phrasebook. Although a physical phrasebook is a powerful tool, it does have several limitations. The phrases are often limited and hard to alter to suit specific situations since the grammatical structure of the sentence can change quite drastically with small alterations. It can also be quite cumbersome to travel with a phrasebook, especially if the journey takes the traveler to several regions with different languages where separate phrasebooks will be needed for each one.

In later years, solutions to these problems have emerged with the introduction of smartphones. A phrasebook for a specific language can be downloaded to the phone, thus removing the problem of cumbersome transportation. Online translation tools have also gained popularity and can provide fairly accurate translations based on statistics. One example of such a tool is Google Translate [1]. One disadvantage with Google Translate is that it is based on statistical machine translation, which means that the translations are based on data gathered from a large amount of text [2]. Thus, the credibility of translations of this kind can be low since the data gathered may not be grammatically or semantically correct at all. In contrast, a phrasebooks do not suffer from these constraints, but is in turn limited by its smaller domain.

The fact that many translators require Internet connection raises the problem that connecting to the internet can be difficult or even impossible in some regions, especially in developing countries. Those translators that have offline capabilities on the other hand often have a quite restrictive domain. For example, they might have a limited number of words and languages available to them. As such, the coverage of the translation may not be adequate for the user's current situation.

1.1 Background

To resolve the problems of the traditional mobile translators there have been attempts to make a more complete translation tool. One programming language that can be used for developing translation programs is the Grammatical Framework (GF) [3]. GF is a functional language designed to write grammars which defines the rules of one or several languages. Using a grammar, it is possible to analyze a sentence in one language and then generate the same sentence in another language. This is possible through the separation between abstract syntax and concrete syntaxes. Every grammar contains one abstract syntax and one or more concrete syntaxes. The abstract syntax describes the abstract theory of the particular language domain while the concrete syntaxes hold the rules for the particular languages [4]. By utilizing both an abstract syntax representation in the form of tree structures, and a concrete representation which represents a specific language, GF allows for precise translation by parsing from one concrete language to an abstract tree and then generating a translation in another concrete language. As GF uses the abstract syntax as interlingua, the overall size for a wide covering translator will be small, with 15 languages fitting in 30 MBytes. For a limited domain with a more specific coverage, the size would be even smaller.

Examples of applications for mobile devices that use GF today are GF Offline Translator [5] and PhraseDroid [6]. GF Offline Translator allows the user to translate both text and speech input. One feature that GF Offline Translator has is the ability to show the quality of the translation to the user. This is done by colour coding the outputted translation, allowing the user to know whether to trust the translation or not. However, one thing that this application lacks is the ability to show the user what phrases the application is able to translate entirely correctly. In PhraseDroid, on the other hand, the user builds phrases from words that are shown in the user interface. Only the words that can follow the previously chosen words are shown, and thus it is ensured that it is only possible to enter input that can be correctly translated. Although the translations are always correct, the incremental approach to building a sentence can be inconvenient for the users. One issue is that it is difficult for the user to see if the desired phrase is actually covered by the language domain before the phrase is constructed.

The problem of not showing the user what phrases are available for translation is something that the web application Phrasomatic [7] solves. In Phrasomatic, a user can choose a type of sentence from a list and then customize it with different options in order to receive the desired phrase with perfect semantics. The drawback of this is that since it is web based it requires an internet connection to function. Another disadvantage is that the user interface for Phrasomatic is not optimized for mobile devices, making it difficult to use on a smaller screen.

An application that would combine the browsing abilities of Phrasomatic with the translation capabilities of the GF Offline Translator, would showcase the potential for portable phrasebook applications. Customizable and correctly translated phrases would facilitate communication abroad for many groups of people, including tourists and businessmen.

1.2 Purpose

The goal of this bachelor project is to develop an offline phrasebook application for mobile devices which can translate different linguistic phrases using GF. The application should have a user interface which allows the user to customize phrases in an intuitive and simple way while showing the translation dynamically. In the application it should also be possible to browse through the available phrases in the phrasebook. Both the user interface and the back end of the application should be flexible and dynamic to make it reusable for several grammars.

1.3 Problem description

The main issues to be solved are how to represent the sentences' varying amount of available options in both the user interface and the back-end. These two problems are described below.

1.3.1 Dynamic User Interface

A typical procedure for using a phrasebook would be to locate the desired phrase to be translated and view the translation. One problem with a phrasebook for mobile devices is the limiting screen. This becomes especially apparent when the phrase is customizable with several options. Thus, there is a higher risk of the options cluttering the screen and overwhelming the user.

The problem to be solved is how an application can be created where the user intuitively finds and uses phrases without being overwhelmed by the amount of options available. The user should feel secure when customizing a phrase in order to satisfy their current needs without being overpowered or losing interest in the application.

Another obstacle to overcome is the application's need for instant translation. This means that a new translation should immediately be generated as soon as an option is altered. Not having this dynamic updating would risk an increased frustration for the user. Another aspect to the dynamic updating is that it should be designed in a way that allows for adding new phrases without altering the user interface.

1.3.2 Back end configurability

The back end of the application will mainly depend on GF. As a result of this, the phrases in the application must be represented in a way that allows for text generation with GF as well as being compatible with the user interface. This includes supporting the alteration of phrases in an efficient way.

The application should be expandable with new phrases and phrasebooks without any alteration to the code of the application. Therefore the logical structure of sentences should be stored outside the core code as a configuration. A developer must also be able to add new GF grammars to the project and be able to run the application immediately without altering its code.

1.4 Scope

There are multiple ways in which the application developed in this project could be expanded. However, the project had to be limited due to time restrictions and choices regarding the focus of the project.

In this project, an Android application was developed. This decision was made since the time frame was quite limited and the time was rather spent on creating a good application than developing support for several platforms such as iOS and Windows 10 Mobile.

As a fully functioning translator with text input from the user already exists in the GF-Offline Translator, this application focus on phrasebook traits such as template sentences that could be customized by the user.

2
Methods

In this chapter the methods used throughout the project will be described. This includes research, project planning and practices used.

2.1 Research and Project Planning

In general a project needs a period of initial background research and planning before the work process can begin. The corresponding phase in this project will be described in this section.

2.1.1 Research

Before the application was implemented, several fields were researched in order to get an understanding of what problems had to be solved during the project. A part of this period was spent studying the web application Phrasomatic since one goal of this project was to adapt its features to fit on a smaller screen. Alongside this process, it was investigated what data structure would best represent the sentences.

Due to some unfamiliarity regarding Android development within the group, time had to be spent on learning basic concepts. This included how the foundation of an application is built as well as what components are present for improving user experience. There are also Android design guidelines which the developers required knowledge of to be able to successfully design visual aspects of the application.

2.1.2 Work Procedures

The project was planned to be developed in two major phases, aptly named Development phase 1 and Development phase 2. The goal of Development phase 1 was to develop a working prototype of the application. Initial sketching of the user interface was carried out in parallel to the research and project planning. Thus, a complete design was finished before Development phase 1 started, although small changes were made during the implementation in order to meet new requirements that were identified during the process.

During Development phase 2 the user interface of the application was polished. This included implementing the last parts of the interface as well as designing it according to Android's material design guidelines. Since it was near the project's end small problems in the application were also resolved, such as back-end malfunctions and graphical glitches. Towards the end of Development phase 2, final user tests were performed in order to evaluate the result of the project and to identify key points of improvements for further development on the application.

Inspiration for working methods was taken from Scrum in order to remain effective and productive throughout the whole project. Scrum is an agile working method for system development that allows the project to change during the process in a structured way. In Scrum the development is divided into shorter phases called sprints. Each sprint is planned with the help of a backlog which contains all the tasks that are to be completed during the project. This way of working encourages change along the way and is often improving productivity since there is a sense of team effort that motivates the group [8]

In order to plan the sprints a backlog was constructed from user stories which will be further explained in Section 2.2.1. Weekly meetings were held with the supervisor to outline what features would be implemented during the upcoming week. Smaller meetings were also held within the group to plan the working sessions, thus ensuring that all tasks would be completed.

2.2 Practices

In order to guide the design process and development of the application, a few practices that work well with scrum were adapted. In this section, these will be described starting with personas and user stories followed by user tests. How these practices have been applied in this project will be described in more detail in Chapter 4.

2.2.1 Personas and User Stories

When creating the backlog for the project according to Scrum practices, user stories were applied. However, before constructing user stories, it was important that the user stories covered all necessary functionality of the application. To ensure this, a pair of personas were constructed. Personas are empathic descriptions of fictive persons that are expected to use the product that is developed [9]. These personas were created in order to understand the needs of users that might have different experience in using applications than the developers do. Since our way of working was inspired by agile methods, the personas were adapted to them. In this case, in order for the developers to not get stuck with precise requirements, the personas acted more as a guideline and were not as detailed as they usually are. The benefit of this was that there was more flexibility in the development while still having a clear direction.

Based on these personas the user stories previously mentioned were defined. A user story is a high-level description of a requirement from a user's perspective [10]. Each user story should define what functionality a user wants and why. As such, user stories may be used by the developer in order to set out what requirements the app should have. One of the main benefits of working with user stories is that everyone included in the development of the product have the same understanding of the requirements and what the end product should look like. This prevents unnecessary rework of the product and helps improve productivity [11].

2.2.2 User Tests

In order to make sure that the design of the user interface is appealing to possible users, paper prototype tests were performed. A paper prototype is a representation of the user interface made by pen and paper. This prototype can then be utilized in order to gather information about how users would interact with the particular interface and what they think of it. This form of testing is very cost and time efficient since it does not require any programming in order to test basic interface ideas. An additional advantage is that it is easy to test several different ideas at the same time. Based on the observations from this test, the first iteration of the software prototype was built [12].

When the software prototype had been developed, high fidelity user tests were performed. In this case it means that users tested the product on an actual device. This form of testing provides a clearer view of how the user interacts with the system since the user is able to use the common controls of the specific device. The model that was followed was the “Usability Test Plan Dashboard” suggested by Dr. David Travis [13]. This plan includes the most essential part of usability testing and is adapted for projects with few persons working on the user experience. The plan also gives the developers a structure to keep in mind, that facilitates the construction of a test.

3
Technical Background

The development of the application has been dependent on different technologies. These will be described in this chapter in order to provide a good foundation for understanding the content in this report. This includes Android development, XML and the Grammatical Framework as these are the most important technical parts and build the foundation for this project.

3.1 Android Development

Android is an operating system for mobile devices based on Linux. The multi-user Linux core enables the applications installed on the device to operate isolated from each other. As a result of this, each app only has access to the parts of the system that are crucial for its functionality. If an application requires access to device data such as contacts, Bluetooth or the camera, it needs to ask for the user's permission. This results in an effective usage of the limited memory available in mobile devices [14].

3.1.1 Application Structure

Applications for Android devices are mainly developed in the programming language Java using the Android API. However other languages may be used as well, such as C or C++ by using Android Native Development Kit (NDK) [15].

The parts of the application that the user sees and can interact with is composed of activities. An application can have several different activities with different contents and layouts which the user can switch between. An activity always occupies the whole screen and its layout is defined using XML. In order to further customize the different activities, fragments can be added. Similar to activities, the fragments have their own functionality and layout. However, the difference between them is that a fragment can only be a part of an activity, and it cannot be an independent screen by itself [14] [16].

Every graphical element in the application's user interface should be defined separately from the Java code in resource files. This includes, for example layouts for activities and fragments, predefined values, images and icons. This practice allows for simple adaptions to different screen sizes and orientation. While keeping the appearance separate from the functionality of the activities the graphical components can be modified and replaced without affecting the Java code [14].

3.1.2 Material Design

In Android 5.0 material design was introduced for application development. Material design is a collection of new design elements, aimed to "create a visual language for our users that synthesizes the classic principles of good design with the innovation and possibility of technology and science"[17]. The guidelines for these new elements are described in Android's material design specifications. This document provides the developer with a multitude of design examples and guidelines of how content should be laid out on the screen, including spacing, text size and colour schemes.

3.2 XML

Extensible Markup Language, or XML, is a markup language which can be used to store textual information. In XML markup is done by using start-tags and end-tags to enclose the selected text. These tags constitute the concept of elements in XML. It is also possible for an XML element to contain other XML elements, hence forming a tree structure. The contained elements can be referred to as child elements while the element surrounding it is called the parent element. XML elements can also include more data than the text content it encloses by using attributes. This may be done by assigning variables with string values inside the start tag [18].

Listing 3.1: Example of XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <person name="Alice">
        <age>20</age>
   </person>
</root>
```

As shown in listing 3.1 the elements have a hierarchical relation to each other. For example, the element <age> is the child element to the element <person>, whereas <person> is the parent element to <age>. The element <person> also showcases the usage of attributes. In this case, the <person> element has an attribute "name" with the value "Alice".

Once an XML document is saved and accessible by the program, it is possible to read and parse the XML data by using different APIs. In Java, it is possible to use the Java API for XML Processing (JAXP) in order to handle parsing of an XML document. By using the DocumentBuilder class from said API, it is possible to load an in-memory tree representation of the XML document. The document tree can then be processed, node by node, in order to get and use the data stored in the XML document [19].

3.3 Grammatical Framework

The Grammatical Framework (GF) is a typed functional programming language which has been inspired by the ML language and Haskell by using algebraic datatypes and higher-order functions. It is primarily designed for writing multilingual grammars for natural languages which can be used to generate translations [3].

In GF it is possible to write programs called grammars. A grammar is the definition of several languages and the translation between these. Grammars thus consist of rules and functions that are used to translate, parse and generate a language. The central component of every GF program is the abstract syntax which is a description, independent from natural languages, of the domain. Every abstract syntax can be equipped with one or more concrete syntaxes. Abstract syntaxes are tree-like representations of phrases which work like a model for semantically relevant structures of language. Concrete syntaxes on the other hand relate this tree structure to a linear string representation for a specific language. This is useful as computers have an easier time reading tree structures while humans have a significantly easier time reading string structures [3].

Listing 3.2: Example of abstract syntax

```
PQuestion (QWhatName YouFamFemale)
```

In Listing 3.2 an example of abstract syntax is shown. The abstract syntax consists of several abstract functions. These are used to define various principles of sentence in an abstract way. For example, *PQuestion* denotes that the sentence is a question, *QWhatName* denotes a question posing what name a person has and finally *YouFamFemale* signifies a familiar female person as the subject. The abstract functions have both inputs and outputs. For instance, the input of *PQuestion* is the output from *QWhatName*, as seen in Listing 3.2. While the example may not be easily understandable for persons not familiar with GF, the abstract syntax can be linearised to a string in a concrete language. For instance, the abstract syntax in Listing 3.2 would be linearised as:

What is your name? in English,
Comment t'appelles-tu? in French, and
Mikä on nimesi? in Finnish

Even though GF works equally well for implementing grammars for traditional translations between two languages at a time, the real strength of Grammatical Framework lies in the multilingual design of grammars. This means that one grammar can handle several different languages at the same time. A fundamental difference from other compilers is the fact that GF grammars are reversible, i.e. a string in any concrete language can be transformed into an abstract syntax tree and vice versa. Due to this it is possible to translate a phrase from any language supported by the grammar into another language. This is done by parsing the input sentence into an abstract syntax tree and then generating the sentence in the target language from the abstract syntax tree [3].

One use of GF is for development of mobile and web applications. The original GF grammars are written in a high-level functional language which is easy for humans to use but hard for computers to interpret. Writing source level representations of grammars for a multitude of different environments is unnecessarily complicated. This problem is solved by the Portable Grammar Format (PGF) [20] which is an easy to interpret assembly language to which the GF grammars are compiled. As there exists a version of PGF runtime written in C, it is possible to use PGF on Android by using the Android NDK and and a Java Native Interface binding to the C runtime [4]. As such it is possible to use the functionality of grammars in mobile development which has been done with “GF Offline Translator” [5].

4 Design Process

The interaction design of the application developed in this project has been one of the main focal points. In order to achieve a user interface that is both intuitive, easy to use and follows Android's design guidelines the design process has been divided into several steps. In this chapter the design process will be described which includes identifying user needs, initial design and user tests.

4.1 Identifying User Needs

Identifying the user's needs and expectations of an application is a crucial step in the design process. In this project, personas and user stories have been used to target this issue. These will be described in this section.

4.1.1 Personas

The purpose of the application is to provide help in communicating and understanding people correctly when interacting in foreign languages, both when speaking and in writing. Since there are many types of persons that could be possible users of the application the main types had to be defined. When designating the scope it was decided to mainly focus on travelers. The type of travelers that this application was designed for are tourists and business travelers. The personas representing each type of traveler have been created based on data gathered from statistical investigations and are listed below.

> **Tourist**
> Anna, 23, Sweden
>
> Anna is currently studying to become an engineer. Alongside her studies she works at Knowly in order to be able to travel, which is one of her greatest interests. Anna likes to travel to different places every time, and it is possible that the locals do not speak Swedish or English which are the only two languages Anna can speak fluently. Since she is a student she can not afford to use the Internet on her smartphone abroad. She would like to be able to construct her own phrasebooks for different occasions that she knows are grammatically correct in order to be more confident when interacting with other people.

Business Person
James, 38, USA

James is working as an executive assistant at a technology company. He travels a lot alongside his boss all around the world. Even though he enjoys the trips he still finds it important to be able to stay in touch with his family. To do this he always has his laptop and smartphone with him. On these trips he usually have a very tight schedule and need to get to one place to another as quickly as possible. In order to travel fast in a foreign country it is convenient to know a few phrases in the particular language. Since James prefer to use wifi when he is abroad an offline translation application would satisfy this need. Such an application should be easy to use since he often need to find a phrase quickly. It is also important that the phrases are grammatically correct since he encounters many important persons each day.

4.1.2 User Stories

From the personas described in section 4.1.1, user stories were constructed in order to make sure that all the users needs would be covered by the final product. The user stories follow the style of the following example:

5 **Choose phrasebook**
As a traveler, I want to be able to choose from several different phrasebooks depending on the trip I am on.

Conditions of satisfaction
- Nice UI with a grid of default phrasebooks
- When entering a phrasebook the user should see a list of the phrases within the specific phrasebook.

The complete collection of user stories can be found in Appendix A. Each user story is connected to the persona that mainly represents the particular functionality requirement, although many of the requirements would be satisfying for the other persona as well. Some of the user stories have been considered equally important for both personas and are thus connected to both.

The finished user stories were prioritized by importance in order to guide the development and to know what features are the most crucial to be finished. They are prioritized so that the most basic functionality would be implemented early on in development phase 1 and less important features towards the end.

The user stories A-C in Appendix A do not represent a specific features of the application but rather serve as general guidelines and basic requirements. These include that the application should be able to run offline, on smartphones and that the user interface should be intuitive. These are not included in the priority list but should be kept in mind during the whole development process.

4.2 Designs at Different Stages and User Tests

During the course of the project user tests have been performed at different stages of the development. The purpose of the user tests was to find flaws in the user interface and create an application which is easy to use. In this section the user tests and the most important observations will be described in conjunction with designs at different stages in the designing process.

4.2.1 Initial designs

After identifying the users' needs and expectations of the application, initial sketches of the user interface were made. In order to produce as many different concepts as possible, a number of designs were sketched individually in order to avoid inspiration from other sketches. The different designs were developed based on the user stories described in Section 4.1.2. The focus was mainly on the navigation of the application as well as the design of the phrase customization screen.

Two initial sketches

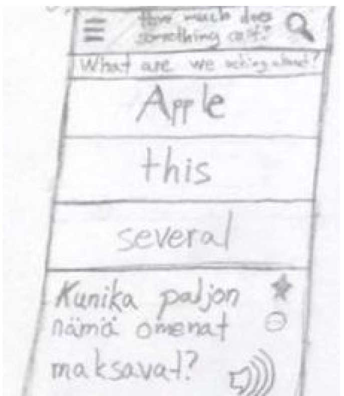

How much does something cost?
What are we talking about?
Apple
this
several
Kunika paljon nämä omenat maksavat?

Figure 4.1

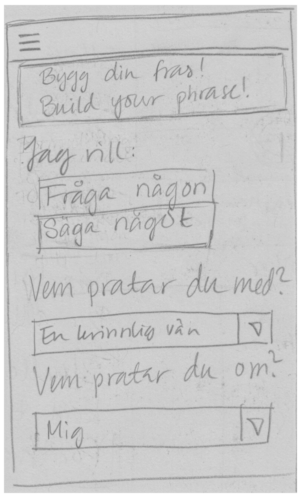


Figure 4.2

The first iterations of sketches varied quite drastically from one another. This served well as a basis for further dialog regarding the design. The discussions mainly revolved around the phrase customization screen as this is the main feature of the application and it required a good design. From the user stories it was apparent that showing more than one pair of languages on the screen would be redundant, as displaying several languages would be more interesting for linguists rather than for ordinary travellers. It was also decided that the placement of the translation should be at the top of the screen, as shown in Figure 4.2, since it was perceived as the most intuitive option when comparing the different designs. An important aspect of this layout was that the translation should always be visible as well as all the available customizations.

Another valuable input when evaluating the designs was regarding representing the functionality for changing language with flags, which was a part of the design shown in Figure 4.1. The issue was mainly about connecting a language to a specific nationality. Since many languages are spoken in many different countries, it could be unclear what language a flag actually represents. Likewise, some people could be offended that their language is represented by another nation's flag. Another disadvantage is that the navigation to the functionality for changing language has a prominent presence on the screen. This was realized when comparing the functionality to our pre-defined user stories. It was discovered that changing language would not be used frequently enough to warrant such a pronounced position. Therefore, it was recommended to consider moving this functionality elsewhere. Based on all these conclusions new designs were made individually.

Two secondary sketches

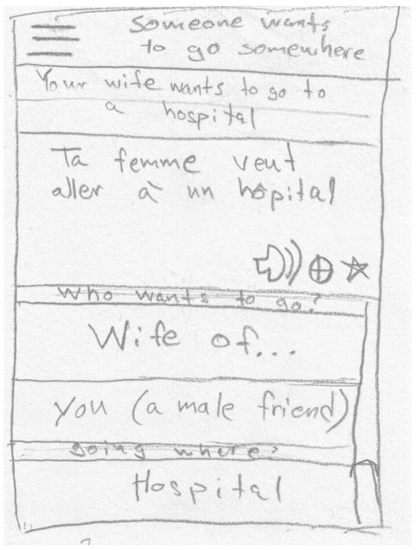

Someone wants to go somewhere
Your wife wants to go to a hospital
Ta femme veut aller à un hôpital
Who wants to go?
Wife of...
You (a male friend)
Going where?
Hospital

Figure 4.3

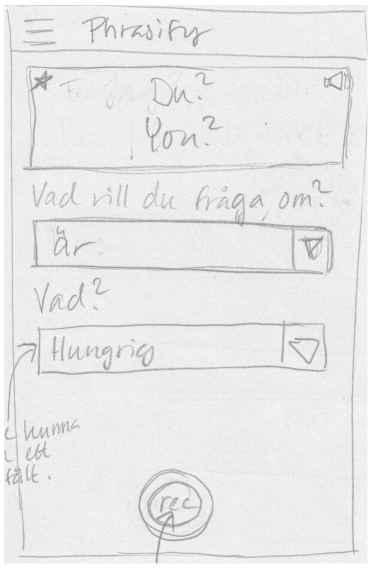


Figure 4.4

In the second iteration the designs were quite similar. The main difference was how the choices were presented to the user. In Figure 4.3, by clicking an option a separate screen woud open displaying all the different choices, while in Figure 4.4, the choices were shown in drop down menus. The argument for the design in Figure 4.3 was that it would be visually appealing to present the options in a clear view without additional distractions. For the drop down menus on the other hand, the advantage was that the translation would be visible while making the choice. Since there were positive aspects for both options, it was decided that they would be tested on potential users in order to know what felt most intuitive to them.

Two initial prototypes

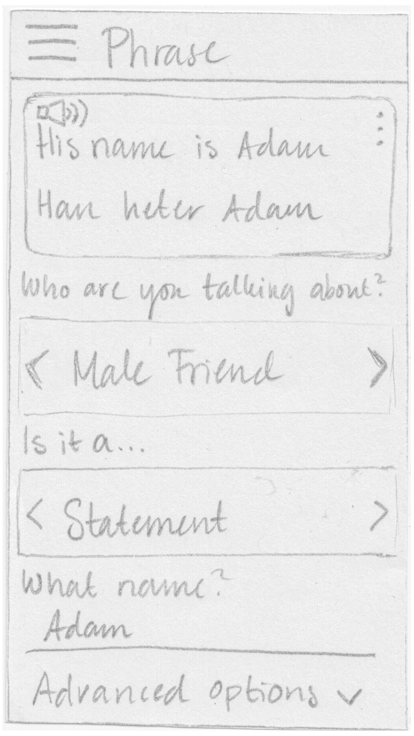


Figure 4.5

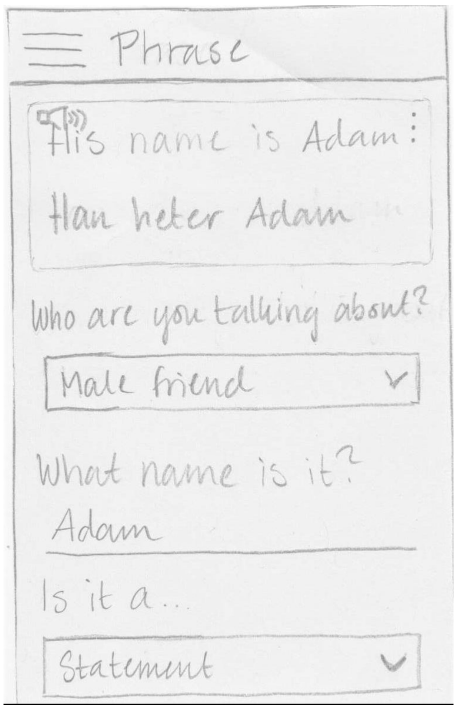


Figure 4.6

The designs in Figure 4.5 and Figure 4.6 have different solutions to the problem that emerges if the amount of customization would not fit on one screen. In the design in Figure 4.5, the thought was to have all options in one screen which could be scrolled. The design in Figure 4.6 on the other hand, did not contain any scroll bar but would show the rest of the options in another screen which the user could swipe to. Additional design conflicts also arose when the advanced options were to be displayed in this screen as well. According to our user stories these options would, in comparison to the standard options, not be used very frequently and should thus not always be shown. A solution to this that were to be tested was to put the advanced options at the end of the scrollable view where they would be available by inflating the advanced options tag.

From the initial design iterations the screens in Figure 4.5 and Figure 4.6 were developed and was a part of the paper prototype which was to be tested. The whole collection of sketches can be found in Appendix B.

4.2.2 Paper Prototype User Test

To assure quality of the initial design, a paper prototype of the application was created. This was used as a basis for tests performed on a number of potential users. The paper prototype included all the views in the application that a user would encounter. Before the test began each person was given a brief explanation of the application so that they would have the same background knowledge. This information along with the test specification and observations are found in Appendix C. After reading the background description they were given a few tasks that they were supposed to complete while explaining how they were thinking. The tasks given were the following:

1. Get started with Swedish as language to translate to.
2. Go customize a phrase of your choice.
3. Make the following customizations of the phrase "Somebody's name is..."
   (a) Change who you are talking about.
   (b) Change the type of the phrase.
   (c) Find the advanced options.
4. Add your own phrasebook and then add a phrase to it.
5. Change the target language.
6. Change the application language.

When all tasks were completed four follow up questions were asked in order to receive feedback that may not have been previously covered. The questions asked were the following:

1. What was your favourite design choice?
2. What was the main problem regarding the design?
3. Is there a function that you feel is missing?
4. Is there a function that you think should be improved?

The user test provided answers to a few of the design problems that had emerged during the first stages of the design process. According to this feedback the design was changed before beginning the development of the software prototype. It was planned to have a screen that would only be shown the first time the application is started on a specific device. On this screen the user would choose origin and target language and the initial thought was that the application language would change to the origin language as well. This functionality was removed after the user test since it was not clear that it would change both application and origin language.

It was decided to use drop down lists for all the options in the customization screen since that alternative had more advantages compared to the previously considered options. Another aspect of the customization screen that emerged during the test was whether the user should scroll through all alteration options available or swipe to the side if all alteration options did not fit on one page. It was discovered that it was more intuitive to swipe to the side than scrolling. As a result of this the scroll was removed from the design. With all this input in mind, the process began moving towards implementing a functioning software prototype.

4.2.3 Software Prototype User Test

As foundation for the software prototype user test, “The Usability Test Plan Dashboard” was used [13]. The whole test specification can be found in Appendix D. Five potential users participated in this test, all of them had similar technical backgrounds and smartphone usage experience. When performing the test, the test persons were asked to perform the following tasks:

- Go to a phrase and alter it.
- Play a phrase using text to speech.
- Change language.
- Add a phrasebook.
- Add a phrase to your phrasebook.
- Find the advanced options in the phrase "Someone is thirsty".

After performing the tasks, they were also asked to answer a questionnaire regarding the application. The questions below were asked in order to get a solid foundation for evaluation of the application rather than making changes to it so late in the project. In the first four questions the users answered with a number between 1 and 10, 1 being bad and 10 being very good. The two last questions were answered in text.

1. What was your overall impression of Parlira?
2. How useful would you find the application while travelling?
3. How did you find the navigation?
4. How did you find the phrase customization screen?
5. What features could be improved? In what way?
6. Is there any feature you are missing in the application?

The test highlighted some design issues that had previously been missed by the developers. The functionality for adding phrases to a phrasebook was only present in the phrase customization screen. But based on the feedback from the test, it was decided that some kind of navigation should be added to the phrasebook as well, in order to facilitate adding phrases to it. This would contribute to better navigation through the application which also was something that should be improved based on the answers from the questionnaire.

It was also agreed upon within the development team that the functionality for changing language could be improved based on the feedback. This functionality would be more intuitively used if it was placed in a more prominent place but in the same time not be visible to the user at all times.

5
Implementation

The final product consists of different components that work together to form the application. In this chapter, the back end and how it is implemented in order to support the alteration of sentences will be described and justified. This includes how the phrases are represented as well as how the back end allows for dynamic updating of the user interface.

5.1 Representing Sentences

The translation functionality for the application was provided by a PGF grammar that contains the abstract functions required for translating the available phrases. A translation can be generated in any supported language by using a string representation of an abstract syntax which is processed by the PGF grammar. For the program to be able to use the PGF grammar the required abstract syntax strings had to be generated. This problem was solved by defining the abstract functions for each available phrase in an XML-file. The phrases are then parsed into a tree-like data structure in Java, here on referred to as a syntax tree, which enables the user to alter different parts of the phrase. In this section, the XML and the syntax tree representations of phrases will be described in more detail.

5.1.1 XML Configuration File

Using XML to define sentences was an intuitive choice as it is possible to specify hierarchical structures similar to the abstract syntaxes that can be found in GF as well as the syntax tree described in Section 5.1.2. By storing the phrases in a XML-file it is also possible to add new sentences post-release, without altering the Java code. As a result, the application becomes more modular.

In the XML-file the abstract syntaxes of each phrase are expressed by different markups. New phrases that are covered by the grammar can be added to the application by updating the XML-file. It should be noted that the XML-file only defines the phrases that are currently available in the application, and does not necessarily represent every possible combination of abstract functions for the currently used grammar. New abstract functions that might be needed for adding new phrases to the application should be added by replacing the PGF grammar used by the application.

In the XML-file, each abstract function that would be represented by a node in the syntax tree is expressed by a tag. A majority of the tags are enclosing other tags which results in hierarchical relationships between the abstract functions.

Listing 5.1: The sentence "Someone speaks a language" represented in XML.

```
1    <sentence desc="Someone speaks a language" id="ASpeak">
2        <node child="phraseit">
3            <node syntax="PropAction">
4                <node args="2" syntax="ASpeak">
5                    <option option="Who speaks it?">
6                        <node child="allpersons" />
7                    </option>
8                    <option option="What language?">
9                        <node child="language" />
10                    </option>
11                </node>
12            </node>
13        </node>
14    </sentence>
```

Each sentence is constructed by several elements contained within the markup <sentence>, shown at row 1 in Listing 5.1. Within the <sentence> tag all information for that specific phrase is stored, both for building the syntax tree and the user interface. When creating a syntax tree all abstract functions need to be acquired in the right order. This is provided by the hierarchical structure of each <sentence> by collecting the values of all the "syntax" attributes for a phrase.

Since some parts of the phrase can be customized, the abstract syntax functions for all the options for each alteration available need to be stored. The parts that are customizable are enclosed by a <option> tag, seen at row 5 in Listing 5.1. Instead of listing all the options directly in the specific sentence the options are stored in separate lists that can be reused for several sentences. The fact that a list will be used is denoted by the attribute "child". The value of the attribute will indicate which list of abstract syntax functions will be replacing the tag when parsing the <sentence> to a syntax tree.

Listing 5.2: Child option used by "Someone speaks a language"

```
    <child id="phraseit">
        <option option="How do you want to phrase it?">
            <node desc="As a statement" syntax="PSentence">
                <node syntax="SProp"/>
            </node>
            <node desc="As a question" syntax="PQuestion">
                <node syntax="QProp"/>
            </node>
            <node desc="As a negation" syntax="PSentence">
                 <node syntax="SPropNot"/>
             </node>
        </option>
     </child>
```

One example of a list of options for a specific alteration is "phraseit" shown in Listing 5.2. This list contains all the information for the option where the user decides if they want to turn a phrase into a statement, question or negation.

Some syntax functions need multiple arguments in order for the grammar to interpret the abstract syntax correctly. To facilitate this problem, an argument attribute called "args" was created. This attribute denotes how many arguments the abstract function represented by the node should have. If this attribute is missing the function requires one input as long as it has a child. In Listing 5.1, the node with the syntax function *ASpeak* will be given two inputs: one node from the child node set "allpersons" and one from the child node set "language".

The majority of the text present in the user interface is acquired from the XML file in order to make it possible to add phrases without altering Java code. The first thing the user interface has to know is the name of all the phrases for displaying them in a list the user can choose from. The name of a phrase is present in the "desc" attribute present in row 1 in Listing 5.1. When the user has selected a phrase the interface also needs information of what alterations can be made and what options are available. The available alterations, or the questions that are asked the user, can be found in an attribute in the tag <option>. While the options are found in the "desc" attributes in the "child" lists.

To illustrate how the XML can describe the abstract syntax we can study a basic example. Take the English phrase "I speak Bulgarian", which has the following abstract syntax:

PSentence (SProp (PropAction (ASpeak IMale (LangNat Bulgarian))))

The abstract syntax above would be a possible result from the XML in Listing 5.1. Here the child "phraseit" has resulted in *PSentence (SProp* while the other lists provide input for the *ASpeak*-node.

5.1.2 Syntax Tree Structure

The information stored in an XML-file mainly describes how a sentence should be structured. In order to utilise that information, the syntax tree class was created. A syntax tree is a tree-like data structure designed to be a representation of a sentence in Java that supports alteration of the specific sentence. In short, a sentence described in the XML-file is parsed into a syntax tree by an XML-parser. Each node in the syntax tree represents an abstract function. As such, an abstract syntax can be built from the information stored in the syntax tree.

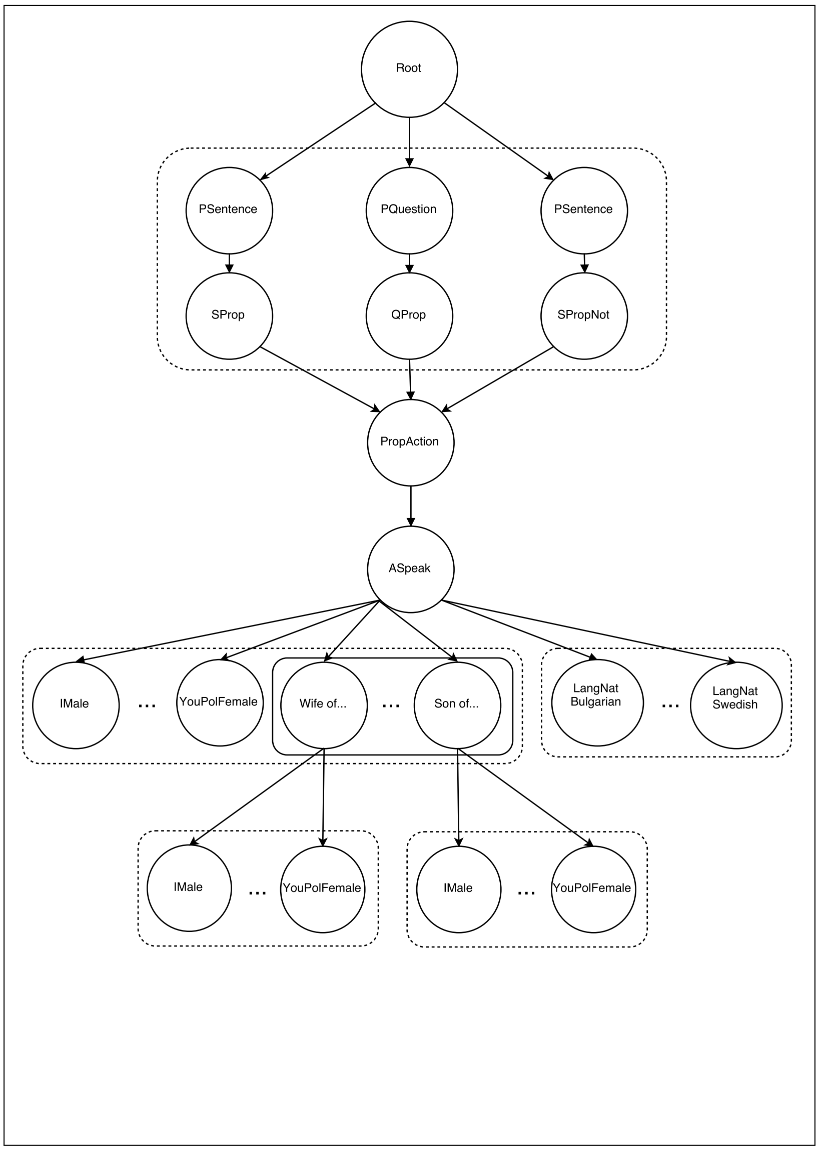


Figure 5.1: A visual representation of a syntax tree for the template sentence "Someone speaks a language"

A node in the syntax tree may have a reference to a set of child nodes. This set of nodes represents the possible inputs for the abstract syntax function in the parent node. In Figure 5.1 these types of node sets are enclosed by dotted squares.

In order to build a correct abstract syntax, at most one node from each set of child nodes can be used. What nodes that will be used in each child node set is decided by the user in the user interface. In order to keep track of what node is currently selected, that node is stored in the parent node. The selected nodes in each set are then used for walking the tree and building the abstract syntax.

To illustrate these ideas the example from Section 5.1.1 is continued, i.e. the abstract syntax for the sentence "I speak Bulgarian":

PSentence (SProp (PropAction (ASpeak IMale (LangNat Bulgarian))))

The syntax functions "PropAction" and "ASpeak" are static since they are alone at their respective level in the syntax tree, as can be seen in Figure 5.1. The other functions of this abstract syntax are parts of node sets and thus replaceable by another node from the same set. The functions "PSentence" and "SProp" are chosen from the set of nodes at the top of the tree present in Figure 5.1. If the user changed this option and wants to create a question instead, the nodes of the syntax functions "PQuestion" and "QProp" will be used instead.

Using sets of nodes to represent the potential choices a user can choose works well for the most part. However, for abstract functions regarding numbers, a set of choices is not sufficient. The main reason why is that to represent numbers in GF several abstract functions are needed [21]. For example, the number 18 is represented by the abstract syntax:

num (pot2as3 (pot1as2 (pot1to19 n8)))

This case can not be represented by a set of nodes due to the large amount of abstract functions involved. In order to solve this problem, a different node was created. This node does not represent a single abstract function but rather generate the requested sequence of abstract functions from an integer acquired from user input.

5.2 Dynamic Updating

The user interface was designed to immediately accustom a sentence according to the user's choices without the need for an update button. In practice this means that whenever the user changes an option for a particular sentence, both the origin translation and target translation are immediately updated. Some of the options enables more customization that appears immediately when that specific option is chosen. As an example, when the option "Wife Of. . . " is selected the GUI adds a new option to enable the user to choose which person's wife we are talking about.

When the user chooses a phrase, that specific phrase is parsed from the XML-file into a syntax tree. The GUI is then built based on the information that is stored in the syntax tree, including how many parts of the specific phrase that can be altered and what options should be included. Every syntax tree has a list of options which holds this information and makes it accessible for the GUI. Whenever a option is changed, the user interface communicates this change to the back end. When the back end is notified that a change has occurred, it asks the currently active syntax tree to generate a new options data structure as described above. This means that the syntax tree changes the selected node for the specific option that has been altered. After receiving this new structure the user interface rebuilds itself to accommodate this updated list of options and the newly parsed translations.

Since the phrases all have different structures and different alterable parts, the GUI will look different for almost every phrase. This is made possible by nested fragments in the phrase customization screen. Every separate option, for example the subject of the sentence, is accommodated in one fragment. Then, depending on how many alterations are available, a number of these fragments, each representing a separate option, will be added to the container fragment that constitutes the bottom part of this screen. These fragments will also be different depending on what type of input a specific alteration requires, for example a drop down list or a number input field.

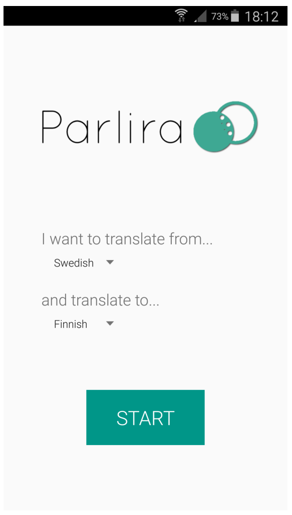
6
The Final Product

In this chapter, the result of this project will be presented. The functionality and abilities of the application will be described, as well as the features that the user can utilise.

6.1 Application

The result of this project is an application called Parlira. In Parlira, a user can select a type of sentence and modify it through its dynamic user interface in order to change the outputted translation. The user can choose to translate between 22 different languages and the majority of these are compatible with the text-to-speech functionality. At the project's completion the number of possible sentences was 27. However, it's possible for a developer to add even more sentences by modifying the XML document defining the sentences. Parlira also works without the need of an internet connection. During the final weeks of the project, the total installed size of Parlira was approximately 8 MB.

6.2 Features of Parlira



I want to translate from...
Swedish
and translate to...
Finnish

Figure 6.1: First usage screen

When first starting the application, the user is presented with a “First usage” screen shown in Figure 6.1. This screen only appears the very first time the application is used on the specific device. In this screen, the user is prompted to select the desired origin and target languages that will be used for translating. This choice is made by using drop down lists. By clicking the "START" button the choice is confirmed and the application progresses.

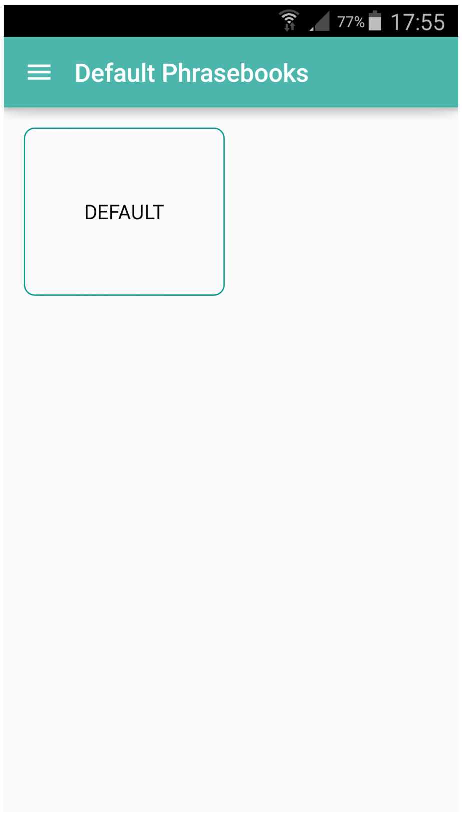
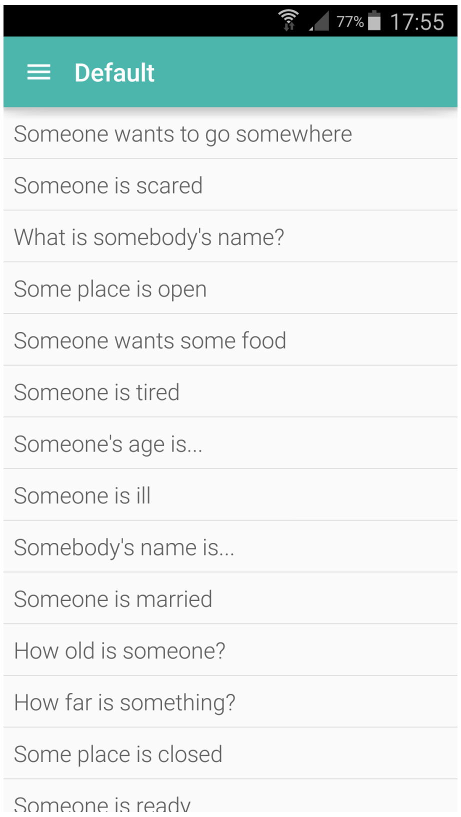


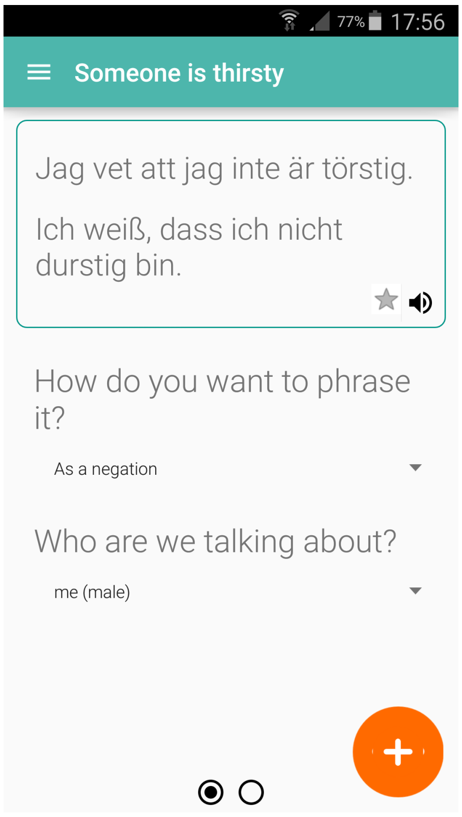
Figure 6.2: Default Phrasebooks



Someone wants to go somewhere
Someone is scared
What is somebody's name?
Some place is open
Someone wants some food
Someone is tired
Someone's age is...
Someone is ill
Somebody's name is...
Someone is married
How old is someone?
How far is something?
Some place is closed
Someone is ready

Figure 6.3: Default Phraselist

After having selected and confirmed translation languages, the home screen in Figure 6.2 comes into view. This screen will also be the default starting screen in future usage of the application. From the home screen, the user can chose the “default” phrasebook. This is a phrasebook containing all possible phrases the application currently supports. It is in this screen new phrasebooks will be added by developers when extending the application with new phrases. Clicking the “default” phrasebook leads the user to the screen shown in Figure 6.3 which shows all the phrases in that phrasebook.



Jag vet att jag inte är törstig.

Ich weiß, dass ich nicht durstig bin.

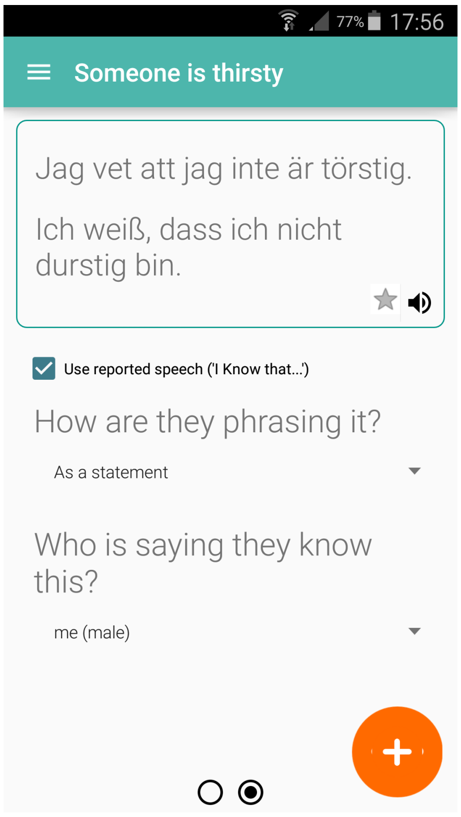
How do you want to phrase it?

As a negation

Who are we talking about?

me (male)

Figure 6.4: Standard options



Jag vet att jag inte är törstig.

Ich weiß, dass ich nicht durstig bin.

✓ Use reported speech ('I Know that...')

How are they phrasing it?

As a statement

Who is saying they know this?

me (male)

Figure 6.5: Advanced options

Once a phrase has been selected, the user is presented with the customization and translation screen for that specific phrase, shown in Figure 6.4. By using the different options presented at the bottom part of the screen, the user can change the outputted translation. This is done instantly, as soon as option is changed. Most of the choices are made by selecting an option from drop down lists. For numbers however, the choice will be made using a slider or manual input depending on what the user prefers. Some phrases have advanced options that become available by swiping the bottom part of the screen. If it is possible to switch to advanced option, it is indicated by the two dots at the bottom of the screen. The advanced options is applied when checking the check-box present beneath the translation box, as seen in Figure 6.5.

Parlira can also read the translation to the user by using text-to-speech (TTS). By pressing the TTS icon in the translation box, the user can play the phrase in the target language. However, the quality may vary as the smartphone running the application needs to have TTS engine support for the requested language. In this translation box there is also a button for marking the current phrase as a favourite. When a phrase is a favourite it can be found in the phrasebook favourites shown in the screen "My Phrasebooks".

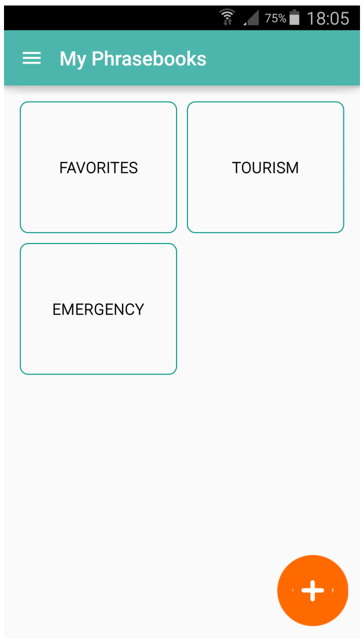


Figure 6.6: My Phrasebooks

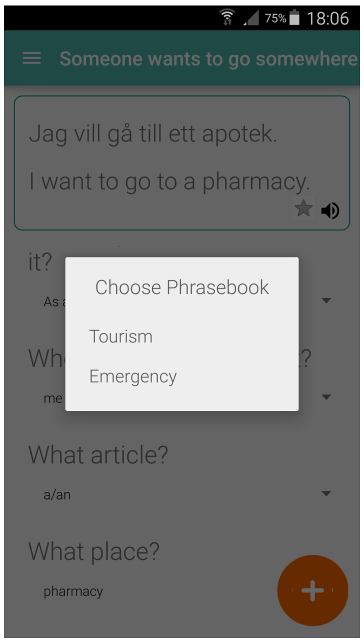

75% 18:06
Someone wants to go somewhere
Jag vill gå till ett apotek.
I want to go to a pharmacy.
Choose Phrasebook
Tourism
Emergency
What article?
a/an
What place?
pharmacy
+

Figure 6.7: Adding to phrasebook

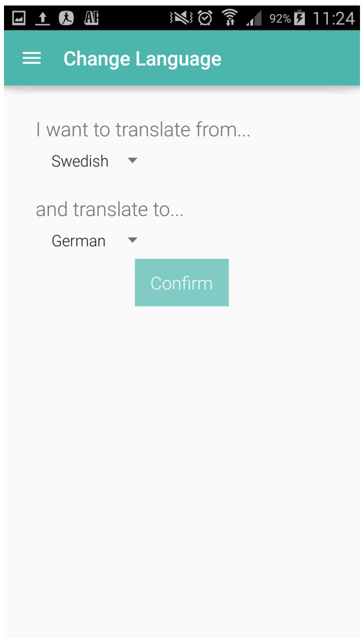


Figure 6.8: Change language

One feature that Parlira has is the functionality to select and save phrases in custom phrasebooks. A user can create and name a custom phrasebook in the application, shown in Figure 6.6, by navigating to "My Phrasebooks" in the navigational drawer. When a phrasebook has been created it is possible to add a phrase to it. This is done from the phrase customization screen by clicking the floating action button in the bottom right corner. When the user clicks the button a dialog appears asking the user what phrasebook to add the current phrase to, which is shown in Figure 6.7. The phrasebooks along with their phrases are saved between sessions.
Finally the user can change the origin and target language of the application through the "Change language" screen displayed in Figure 6.8. The available languages are displayed in two drop-down lists, one for the origin language and one for the target language. Upon pressing "confirm", the users choices are applied and the user is returned to the previously visited screen which is now adapted for the new languages.

7 Discussion

During the project the plans that were made in the beginning changed due to different circumstances. In this chapter, the methods that were chosen in the project will be reviewed, such as how well the methods worked and how they could have been improved. Apart from this, a reflection regarding the results of the project will end the chapter.

7.1 Method Discussion

At the beginning of the project, the methods used in the development process were established. Every method has advantages and disadvantages which will be discussed in this section. These practices have been divided into two categories: Design Process methods and Implementation Process methods.

7.1.1 Design Process

In order for the initial design to be accurate in incorporating the necessary functionality, a number of personas were developed from research about potential users. Although these personas gave insight of the potential users and their expectations, no user-type was different enough to warrant special treatment of the design. If this method was to be performed again, a larger set of data would be recommendable to assure that the personas were correctly described. To achieve this a questionnaire could have been done to assure that the personas would be correctly defined.

Based on the insights that were gained from the personas, the team constructed user stories. Although the user stories have been useful during the development, the motivations behind them might not be accurate since the personas on which they were based were vaguely defined. Despite this, the user stories gave a good overview of what functionalities were to be implemented. They were also helpful when ensuring that no functionality had been forgotten. When evaluating the designs, it was helpful to look at the user stories in parallel to the user tests to understand what functionalities should be improved.

The first stages of the development involved designing the user interface. The team chose to make individual sketches in two iterations with evaluation discussions after each one. Working in this way helped when trying to come up with new ideas and having many different styles for different screens. This was especially helpful when designing the phrase customization screen, as this approach led to a lot of different ideas which in turn allowed for an optimal solution by combining these.

Both the paper prototype user tests and the software prototype user tests were equally important and contributed to the design process in similar ways. They were exceptionally helpful in identifying problematic areas in the application since those are sometimes difficult to spot as a developer. One aspect that might have made the user tests even more helpful would be if the test subjects had a more diverse background. The persons that tested the prototypes had similar background and were around the same age. If more persons with more diverse backgrounds and different ages were included in the tests, the test results may have been different. This might have given a more fair picture of reality, and thus giving more valuable input. However due to time constraints and the small size of the development team, time could not be spared to test with a larger set of subjects.

7.1.2 Implementation Process

In the early stages of development, the implementation was planned to be in sprints, just as the process had been in the design phase. But during this process, the plan that was set up was not followed as well as it should have been. As a result of this a lot of the implementation on several different parts was done simultaneously. However, when the core translation implementation was finished, it was decided to implement each new feature in short sprints. This was made possible in part due to that many functionalities had been started on during the less structured phase and thus needed very little time to actually incorporate into the existing product. As such, some sprints would be completed in a day, such as the advanced options. Other could take some more time, although none took longer than a week to complete, this to ensure that we kept progressing in the project.

As a result from not working in proper sprints the code obtained a quite rigid structure as some small functionality would not be planned in advance. This lead to many separate methods being created that served a similar purpose, which could easily have been avoided if more specific requirements had been set up in advance for each new functionality.

Although the shorter sprints for additional functionality ensured that all functionality were implemented and working, it was somewhat poorly executed in regards to planning. In our case, we implemented functionality such as advanced options before we had completed the ability for users to create personalized phrasebooks. This was a functionality that according to our user stories were much more desired than the ability to create more complex sentences.

Something that could have helped the team in the development were more specific milestones. The milestones used in this project were: completing a user interface design, a functioning software prototype and a tweaked software prototype. These were set due to the difficulty to predict what would make good milestones in advance.

When the project came to an end, it was discovered that better milestones could have been created, for example being able to alter a phrase and incorporating advanced options.

7.2 Result Discussion

Before the development of Parlira began, a few requirements were established which the application was going to fulfill at the end of the project. In this section, it will be discussed how well the application meets these requirements and what aspects of the final product that could be improved.

7.2.1 The Application

During this project Phrasomatic has been a target in both functionality and in requirements. At the end of the project, most of what can be done in Phrasomatic can also be done in Parlira, although there are some differences between them. One being the function to see the abstract syntax used for generating translations found in Phrasomatic. It should be stated however, that this function is only really valuable to developers. In fact, we as developers have been using that function feverishly when creating sentence representations in the XML document However, for actual users seeing the abstract syntax is not that useful and with the user stories in mind, this functionality can be considered completely irrelevant.

On the subject of implementation, there are some things that could have been done differently. In the back end for instance, the data structure representing sentences could have been designed in another way. One issue with the currently implemented syntax tree, is that it could be much smaller and more optimized than it currently is. In order to prevent redundancies such as multiple instances of the same set of children in the syntax tree, the XML parser could have been made better or another data structure could have been used instead.

It is also important to note that, although the core of a sentence representation is completely adaptive, the advanced options are not. In the current implementation the only way to state that a advanced option exists is a small note in the <sentence> tag which directs to the specific advanced option for reported speech. The parsing of advanced options are also tailored specifically for the reported speech option. If the application should be extended with more advanced options then the parsing of said option should probably be defined in a configuration-file. For the sentences, a smarter solution for adding multiple or different advanced options should be implemented. Sadly this was discovered to late in the development process and thus not implemented.

Using XML to build phrases also posed some difficulties in the development phase. While using XML worked well in conjunction with a tree structure, as both representations have a hierarchical structure, there were some problems in regards of efficiency. In the current implementation the XML document has child elements defined in a separate section. This was done so that the the child elements could be reused in other sentences. However, this approach resulted in the XML document becoming cluttered with definitions, making it more difficult navigating the XML document. If the developers were more well-versed with XML, another method for solving the repeated usage problem could have been used.

An area which could see improvements is the time it takes to start the application. The start-up time, while not unreasonable high, could be detrimental for the user experience. Due to the many potential causes for the long start-up time, we were not able to improve on this issue before deadline.

Apart from the problems in the back end described above, there are some features that could be improved in order to make the experience better for the user. As described in section 4.2.3 many of the test persons thought the functionality to add phrases to a custom phrasebook was not intuitive. One part of the problem described could be improved by adding a floating action button to the screen that shows all the included phrases in a phrasebook. The button would indicate that the user is able to add something in the currently viewed screen. Hence, it would be a better option for adding phrases to the phrasebook. The other part of the problem was that the floating action button in the customization screen could have been better placed. One suggested solution to this was to add a small button beside the favourite and text-to-speech buttons and remove the floating action button. The advantages of this solution are that all actions the user can perform with the phrase are collected in the same place and that the floating action button will no longer cover parts of the customization. A floating action button was initially chosen since adding a phrase to a phrasebook would be a frequently used feature according to the user stories.

From the user tests, a problem regarding the functionality for changing translation languages was indicated. A few of the test persons felt that the placement of it was not optimal and could be more intuitively placed elsewhere. One could argue that this functionality should be a part of the customization screen, but this is not a good option since the user might want to change languages from other views in the application as well. One solution that was discussed within the team, was to have two drop down lists in the navigation drawer where you can choose languages. This way, the functionality is easy to reach from any part of the application.

7.2.2 Further Development

Parlira could be further developed in many different ways in order to improve the user experience. One thing that was initially discussed, was the inclusion of application languages. This is intuitive to include, since Parlira is a phrasebook application and people from different countries should be able to use it. The functionality was not implemented since it was of low priority in comparison to other features that have been included, and would require manual translation of UI elements.

Another functionality that was requested during the user tests, was to be able to add phrases that are not already included in the application. These phrases would not be customizable but would still serve a great purpose since it gives the user more freedom in what he or she wants to use the application for.

7.2.3 Parlira in Society

The work produced from this project could be used in society in different ways. Parlira provides correctly translated phrases without the need for internet connection, which makes it useful in many parts of the world. The fact that it supports many languages is also quite unique and it saves the user the need for downloading several language packs for different languages. As the application was designed to be extensible, the value it provides for its user could be even greater if it would be further developed.

The development of the application could be valuable for other developers as well. Certain information displayed in this thesis details advantages and pitfalls during the development, which could be beneficial to other developers building similar applications. The representation of sentences in XML could also be interesting to developers, as the potential application area could be greater than just translating.

8
Conclusion

In this report the development of Parlira, an android application for mobile devices inspired by Phrasomatic, has been described and discussed. The goal of the project was to develop an offline translation application using GF, that would support alteration of phrases. An important aspect of the application was that the user interface would be intuitive and dynamic in order for the user to have a pleasant experience. The application was also supposed to be extensible, meaning that it should be possible to add new phrases without modifying the Java code. Additional requirements were added after user tests. These were the ability to construct personalized phrasebooks and to edit these.

After the project's end, the application met all of the previously stated requirements in various degrees. The application built uses configuration files as well as tree data structures to model phrases. These models enable the user to alter the phrases. The back end also transforms these models to a format that can be translated using GF resources. The usability of the application was also an important aspect of the development and the end product mostly fulfills this requirement. However, there are some features that could be improved and further developed.

Working with GF has revealed a new way of translation and the vast possibilities for improvement of machine translation in the future, that was unknown to the group prior to the project. Even though no one worked directly with GF, the theoretical knowledge has not only given the necessary insight to further delve into the field but also woken interests in natural language processing and further work in the field.

Bibliography

[1] Google Inc, “Google Translate.” `http://translate.google.com`, 2016. Accessed: 2016-05-16.

[2] P. Koehn, *Statistical Machine Translation.* Cambridge: Cambridge University Press, 2009.

[3] A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars.* CSLI Publications, 2011.

[4] K. Angelov, *The Mechanics of the Grammatical Framework.* PhD thesis, Chalmers University of Technology and Göteborg University, 2011.

[5] K. Angelov, B. Bringert, and A. Ranta, “Speech-Enabled Hybrid Multilingual Translation for Mobile Devices,” in *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 41–44, Association for Computational Linguistics, 2014.

[6] A. Ranta, R. Enache, and G. Détrez, “Controlled Language for Everyday Use: The MOLTO Phrasebook,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7175 LNAI, pp. 115–136, 2012.

[7] M. B. Měchura, “Phrasomatic.” `http://phrasomatic.net/`. Accessed: 2016-04-26.

[8] I. Goldstein, *Scrum shortcuts without cutting corners : agile tactics, tools, & tips.* Upper Saddle River, NJ: Addison-Wesley, 2014.

[9] L. Nielsen, *Personas - User Focused Design.* 2013.

[10] M. Cohn, *User stories applied: For agile software development.* 2004.

[11] M. Daneva and O. Pastor, eds., *Requirements Engineering: Foundation for Software Quality*, vol. 9619 of *Lecture Notes in Computer Science.* Cham: Springer International Publishing, 2016.

[12] C. Snyder, *Paper prototyping : the fast and easy way to design and refine user interfaces.* Morgan Kaufmann Pub., 2003.

[13] D. Travis, “The 1-page usability test plan.” `http://www.userfocus.co.uk/articles/usability_test_plan_dashboard.html`. Accessed: 2016-05-16.

[14] Google Inc, “Application Fundamentals.” `https://developer.android.com/guide/components/fundamentals.html`, 2016. Accessed: 2016-05-30.

[15] Google Inc, “Android NDK.” `https://developer.android.com/ndk/`, 2016. Accessed: 2016-05-16.

[16] Google Inc, “Fragments.” `https://developer.android.com/guide/components/fragments.html`, 2016. Accessed: 2016-05-30.

[17] Google Inc, “Introduction - Material Design.” `https://www.google.com/design/spec/material-design/introduction.html`, 2016. Accessed: 2016-05-16.

[18] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition).” `https://www.w3.org/TR/xml/`, 2008. Accessed: 2016-05-16.

[19] Oracle, “Lesson: Introduction to JAXP.” `http://docs.oracle.com/javase/tutorial/jaxp/intro/index.html`, 2015. Accessed: 2016-05-16.

[20] K. Angelov, B. Bringert, and A. Ranta, “PGF: A portable run-time format for type-theoretical grammars,” *Journal of Logic, Language, and Information*, vol. 19, no. 2, pp. 201–228, 2010.

[21] H. Hammarström and A. Ranta, “Cardinal Numerals Revisited in GF,” in *Workshop On Numerals In The World’s Languages. 29 - 30 March 2004 Leipzig, Germany*, 2004.

A
User stories

The user stories used to guide the development of Parlira is listed below. They are ordered by priority with the most important as number 1. The stories A-C are the general guidelines mentioned in section 4.1.2 and are not prioritized but are equally important.

A Offline

As a traveler, I want to be able to use this application fully without the need of an internet connection since it is expensive and limited abroad.

Conditions of satisfaction
- All functionality provided is available offline.

B Android Phone

As a traveler, I want to be able to use this application on my android phone so that it is always available.

Conditions of satisfaction
- All functionality provided is available and suited for the small screen size.

C Intuitive and Dynamic User Interface

As a businessperson I want to have a simple and easily used interface so that I can customize my phrase quickly.

Conditions of satisfaction
- The user interface is intuitive for most persons
- The content is placed on the screen in a way that makes it easy to comprehend
- The colour scheme is is pleasing to look at and helps highlighting important components
- The content is updated directly when an interaction with the interface should result in the interface changing.

1 Choose origin language

As a user, I want to be able to choose an origin language to translate from when I start using the application.

2 Choose target language

As a user, I want to be able to pick target language to translate to when I start using the application.

Conditions of satisfaction 1&2
- First usage activity is complete
- UI is finished with correct colour scheme
- A logo is present
- The target and origin language settings should be altered depending on what choice has been made
- The application language should be changed into the origin language chosen.

3 Change target language

As a user, I want to be able to change target language within the application.

4 Change origin language

As a user, I want to be able to change origin language within the application.

Conditions of satisfaction 3&4
- An screen for language settings should be finished
- UI is finished with correct colour scheme
- The language settings should be altered depending on what choice has been made

5 Choose phrasebook

As a traveler, I want to be able to choose from several different phrasebooks depending on the trip I am on.

Conditions of satisfaction
- Nice UI with a grid of default phrasebooks
- When entering a phrasebook the user should see a list of the phrases within the specific phrasebook.

6 Choose a phrase

As a traveler, I want to be able to choose a phrase from a phrasebook so that I know what phrases are available for translation and alteration.

Conditions of satisfaction
- All phrases should be shown in a list.
- When clicking a phrase the user should enter the phrase customization screen.
- In the default phrasebooks the phrases are shown in their template form, i.e. "Someone's name is".
- In the user's own phrasebooks the phrases are shown in their customized form, i.e. "My name is".

7 Customize a phrase

As a businessperson, I want customize a phrase and get the corresponding translation in order to get a socially correct phrase.

Conditions of satisfaction
- The options should be made using appropriate input methods.
- When a choice is made the translation should be updated immediately without having to confirm it.
- The advanced options should be placed intuitively.

8 Text to speech

As a tourist, I want to use speech synthesis to say the translated output since it may be difficult for me to read the foreign languages.

Conditions of satisfaction
- There should be a button present in the customization screen that plays the target phrase when pressed.
- The target phrase should be spoken in the corresponding language.

9 Create Phrasebook

As a tourist, I want to be able to create my own phrasebooks in order to facilitate the interaction in a specific situations.

Conditions of satisfaction
- There should be a clear indication in the user interface of how to add a new phrasebook.
- The user should be able to give the phrasebook a name.
- The phrasebook should appear immediately in a grid.

10 Delete Phrasebook

As a user, I want to delete phrasebooks that I have created in case they are not relevant anymore.

Conditions of satisfaction
- There is a clear indication in the user interface of how to delete a phrasebook.
- The phrasebook disappears from the grid of phrasebooks immediately.

11 Add to phrasebook

As a user, I want to add a phrase to my own phrasebook.

Conditions of satisfaction
- There is a clear indication in the user interface of how to add a phrase to a phrasebook.
- There is feedback present from the user interface that the phrase has been added.

12 Delete from phrasebook

As a user, I want to delete phrases from my phrasebook.

Conditions of satisfaction
- There is a clear indication in the user interface how to delete a phrase.
- The phrase disappears from the phrasebook immediately.

13 Save between sessions

As a user, I want the application to save my settings between sessions so that I do not have to make the same settings every time.

Conditions of satisfaction
- Language settings and My Phrasebooks are saved between sessions.
- The startup screen is only shown the first time the application is used on the specific device.

14 Favourites

As a user, I want to save a (customized)phrase to favourites so that I do not have to build commonly used phrases several times.

Conditions of satisfaction
- There is a clear indication in the user interface of how to add a phrase to the favourites
- There is a phrasebook called Favourites where all favourite phrases are added.

15 Application Language

As a user I want to be able to choose another language that English for the application language so that I can understand it.

Conditions of satisfaction
- The user can switch between English and Swedish for application language.
- There is a finished screen for changing application language

B
Sketches

In this document sketches from the design process are presented.

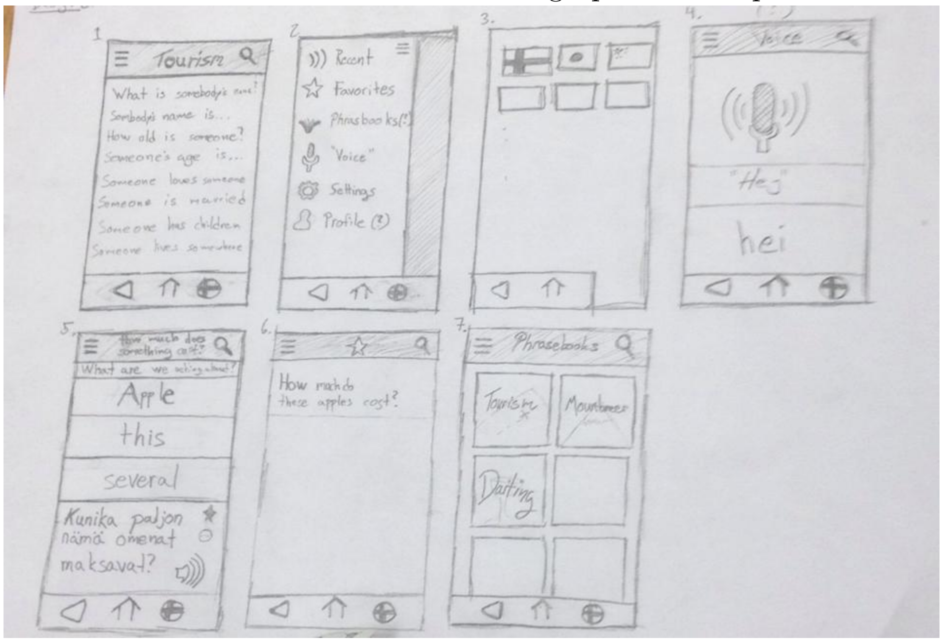


Figure B.1: Sketches from first draft

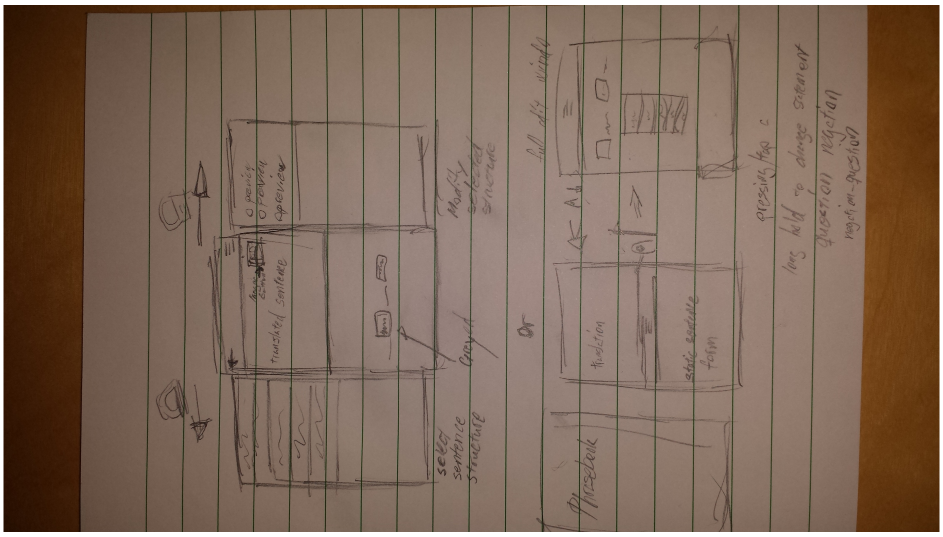


Figure B.2: Sketches from first draft

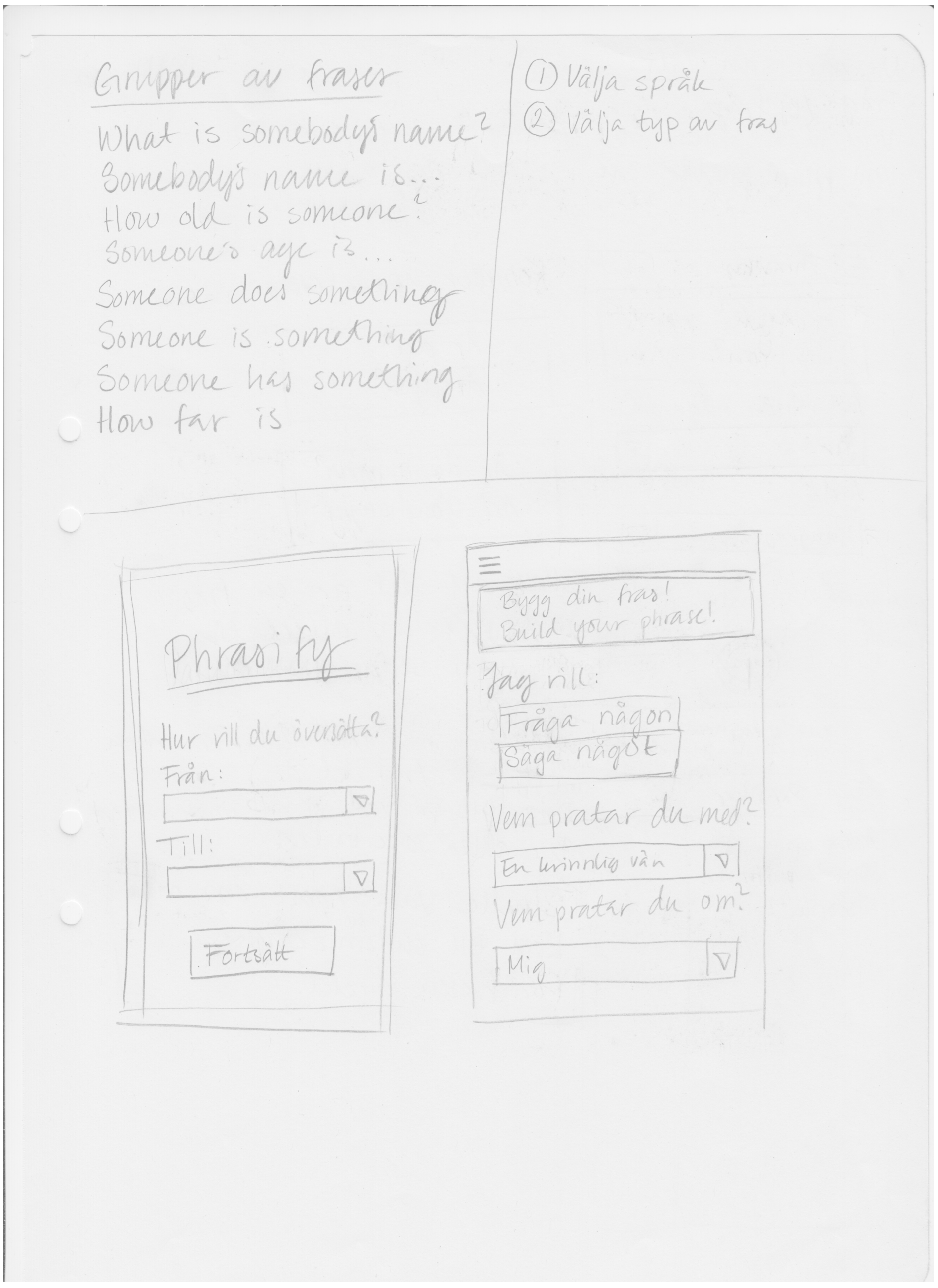


Figure B.3: Sketches from first draft

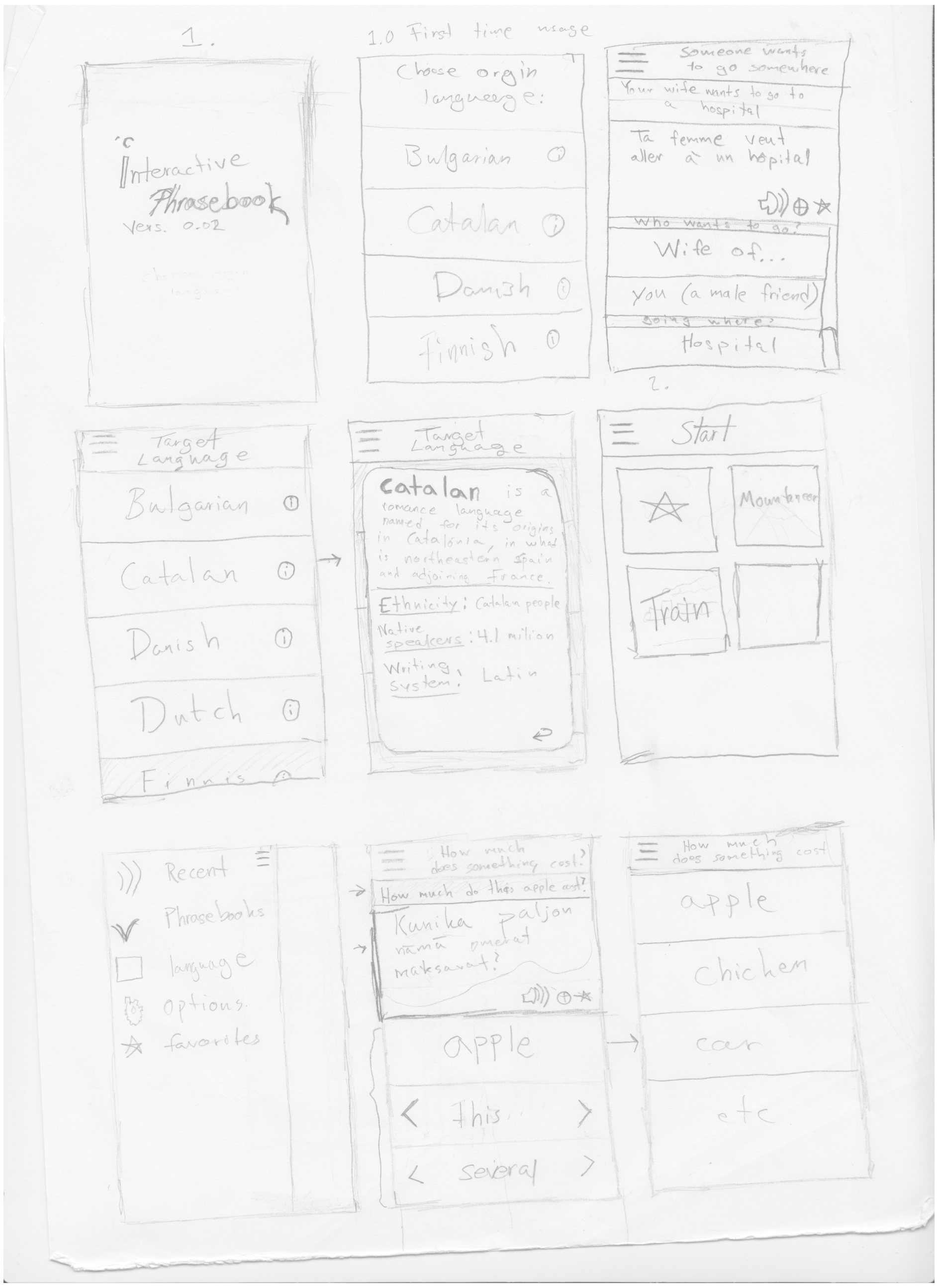

1.
Interactive Phrasebook
Vers. 0.02
1.0 First time usage
Choose orgin language:
Bulgarian
Catalan
Danish
Finnish
Someone wants to go somewhere
Your wife wants to go to a hospital
Ta femme veut aller à un hôpital
Who wants to go?
Wife of...
You (a male friend)
Going where?
Hospital
2.
Target Language
Bulgarian
Catalan
Danish
Dutch
Finnis
Target Language
Catalan is a romance language named for its origins in Catalonia, in what is northeastern Spain and adjoining France.
Ethnicity: Catalan people
Native speakers: 4.1 milion
Writing system: Latin
Start
Mountaineer
Train
Recent
Phrasebooks
language
options
favorites
How much does something cost?
How much do this apple cost?
Kuinka paljon nämä omenat maksavat?
apple
This
several
How much does something cost
apple
chicken
car
etc

Figure B.4: Sketches from second draft

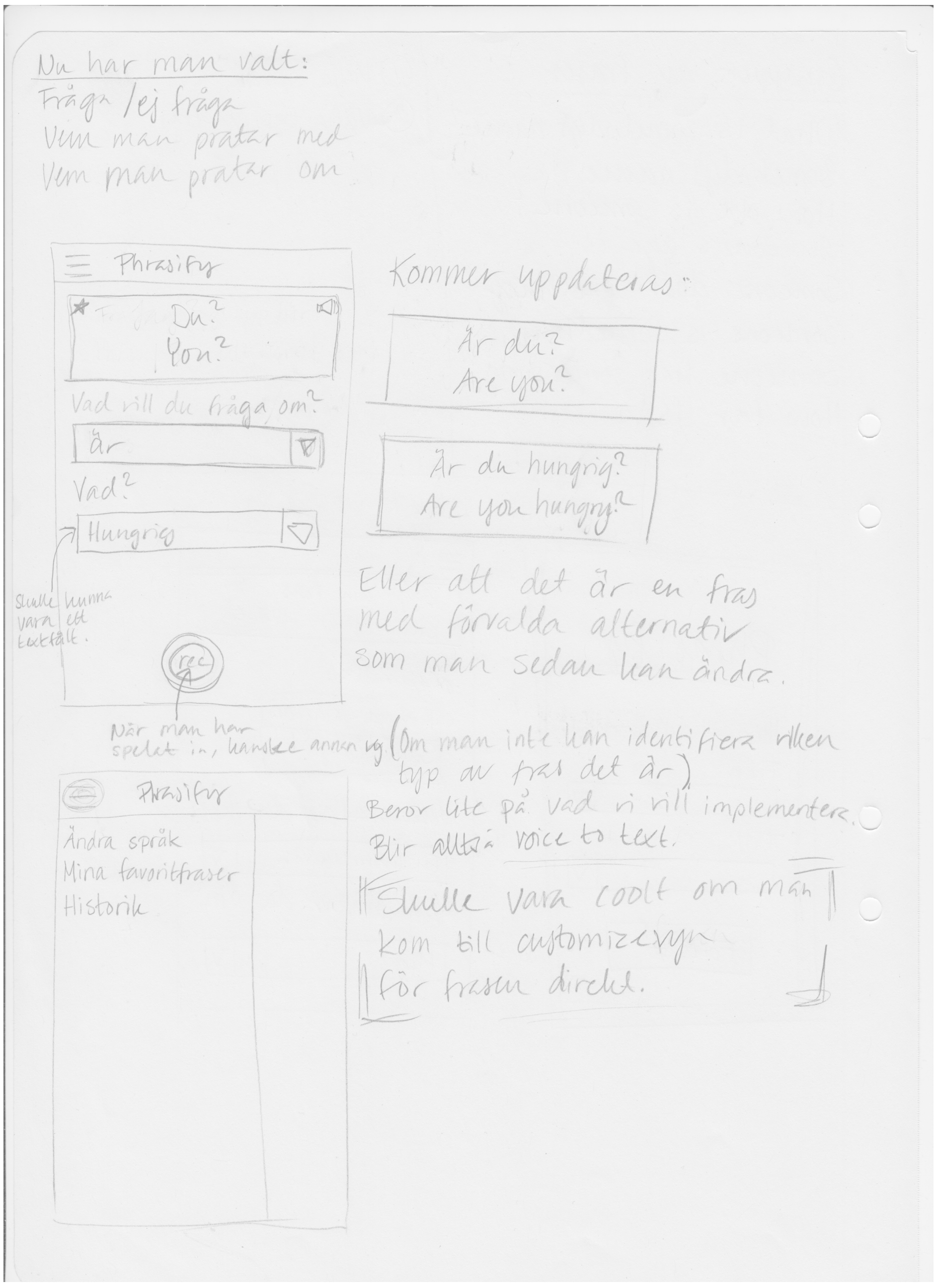

Nu har man valt:
Fråga /ej fråga
Vem man pratar med
Vem man pratar om
Phrasify
Du?
You?
Vad vill du fråga om?
är
Vad?
Hungrig
Skulle kunna vara ett textfält.
rec
När man har spelat in, kanske annan vy.
Kommer uppdateras:
Är du?
Are you?
Är du hungrig?
Are you hungry?
Eller att det är en fras med förvalda alternativ som man sedan kan ändra.
(Om man inte kan identifiera vilken typ av fras det är)
Beror lite på vad vi vill implementera.
Blir alltså voice to text.
Skulle vara coolt om man kom till customizevyn för frasen direkt.
Phrasify
Ändra språk
Mina favoritfraser
Historik

Figure B.5: Sketches from second draft

C
Paper Prototypes

In this document the whole test specification for the paper prototype user test described in Section 4.2.2 is presented. Pictures of the prototype is included and at the end the observations that were made are summarized.

C.0.1 Paper Prototype Test Specification

This is a simple prototype of the user interface of a translation application for Android devices. You will be asked to perform a series of tasks which are essential for the application. For each task you will explain your thinking and point out design choices that you find unintuitive or that could be improved in some way.

The idea of the application is that you should be able to choose a phrase from a list which you can then customize for the specific situation you want to use it in. This phrase is then dynamically translated to a language of your choice. Have in mind that this is the first time you use the application.

Tasks to be completed:

1. Get started with Swedish as language to translate to.
2. Go customize a phrase of your choice
3. Make the following customizations of the phrase "Somebody's name is..."
   (a) Change who you are talking about
   (b) Change the type of the phrase
   (c) Find the advanced options
4. Add your own phrasebook and then add a phrase to it.
5. Change the target language.
6. Change the application language.

Follow-up questions:

1. What was your favourite design choice?
2. What was the main problem regarding the design?
3. Is there a function that you feel is missing?
4. Is there a function that you think should be improved?

The paper prototype that was used is shown below:

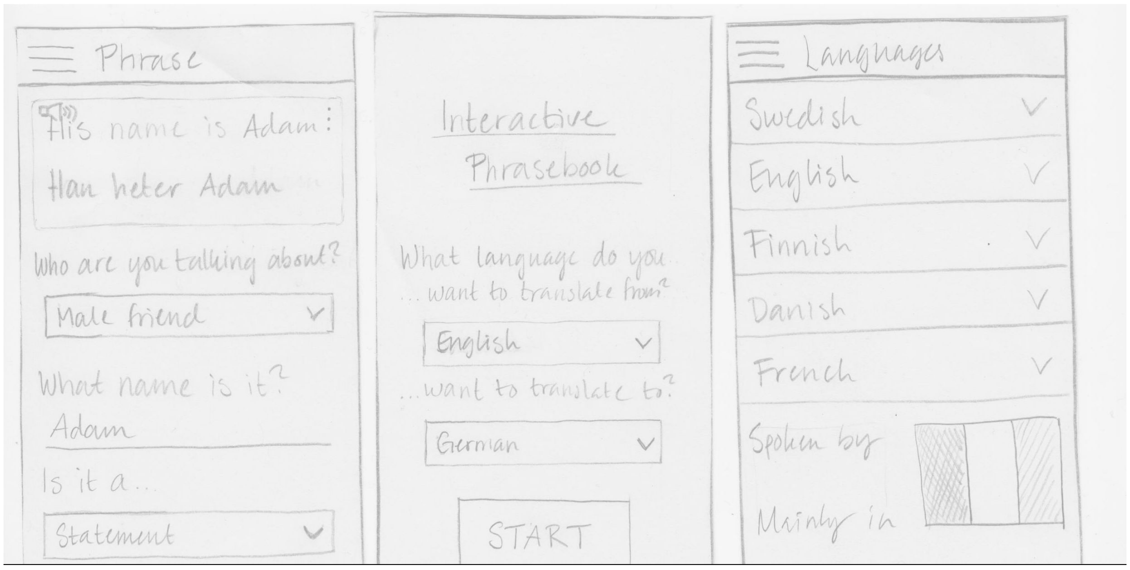


Figure C.1: The phrase customization screen, first usage screen, and language screen

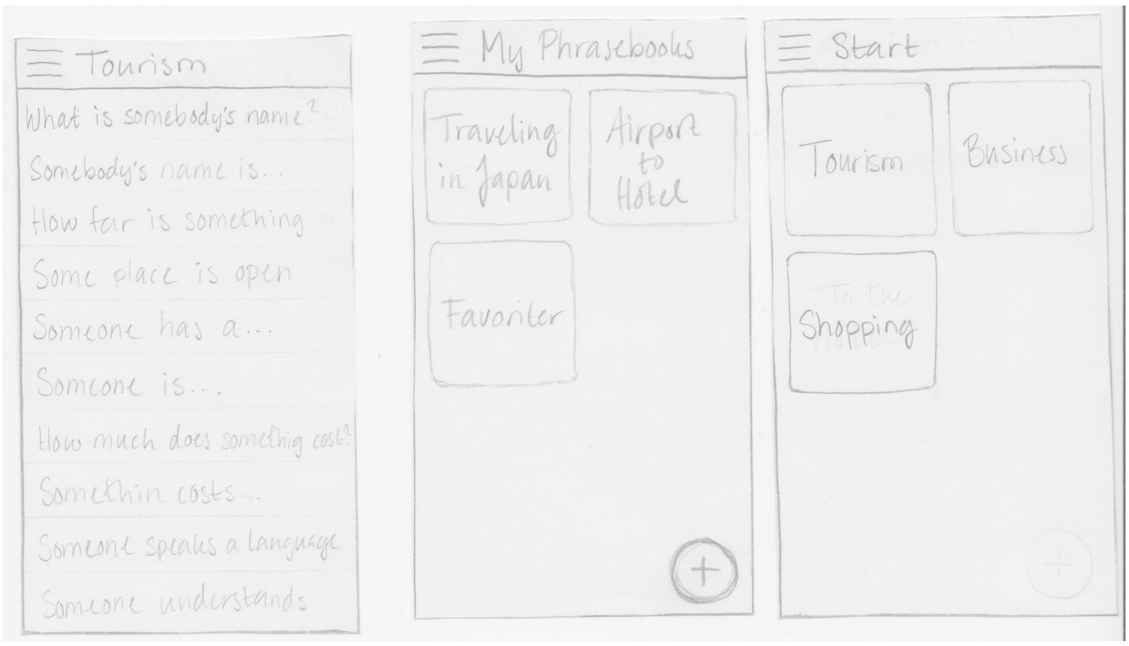


Figure C.2: The phrase list screen, m screen, and language screen

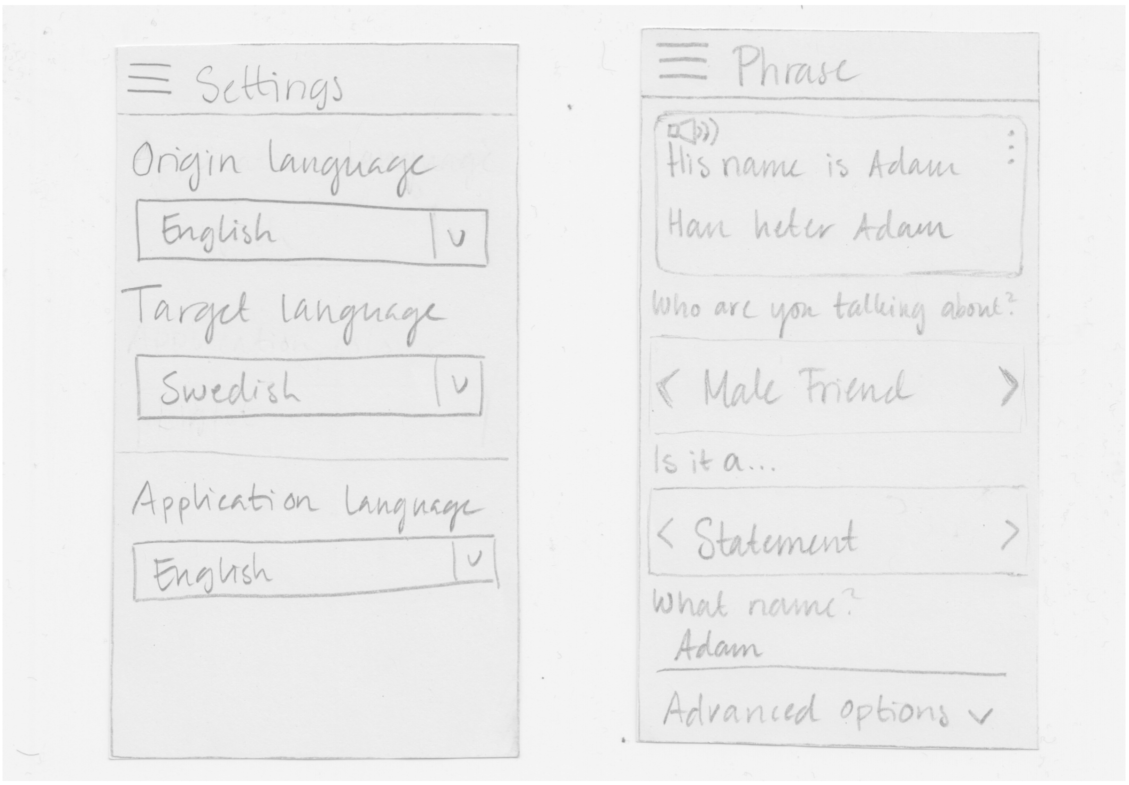


Figure C.3: The settings screen and the phrase customization screen

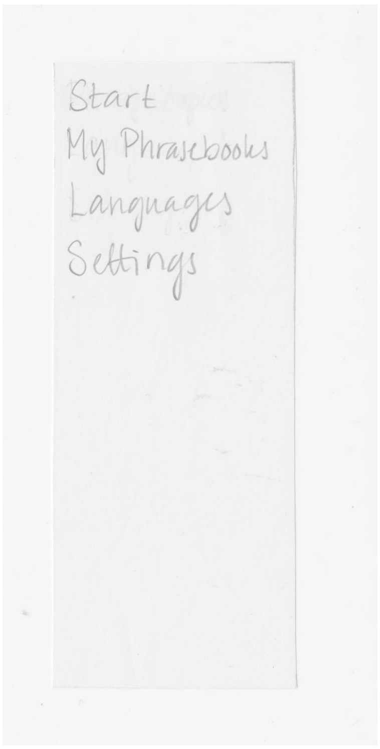


Figure C.4: The navigation drawer

C.0.2 Summary of Observations

The first task was included in order to test the initial screen, shown in Figure D.1 that will only be visible when the user installs the application on a device for the first time. At this stage in the design process the origin language chosen in this initial screen would be the language of the application. This was something that one of the persons did not understand, as this person did not grasp that this should be a language that the user is supposed to know since the label said "What language do you translate from?". The other persons did not have an issue with this since they had in mind that the application is supposed to be similar to a phrasebook.

Task 2 and 3 are similar but in order to cover both the navigation and the customization both felt relevant. For task number 3 two different views were tested, shown in figure D.1 and D.3, in order to find out which one was most appreciated by the users. None of the users had any problem with either finding the customization screen or to customize a phrase. The users preferred to make the choices using a drop down list since it would be too many options to swipe between and they did not like the idea to switch to another screen where the options were shown. They also felt that it was a good idea to have the bottom part swipeable since it gives the user an idea of how many options there are to be made.

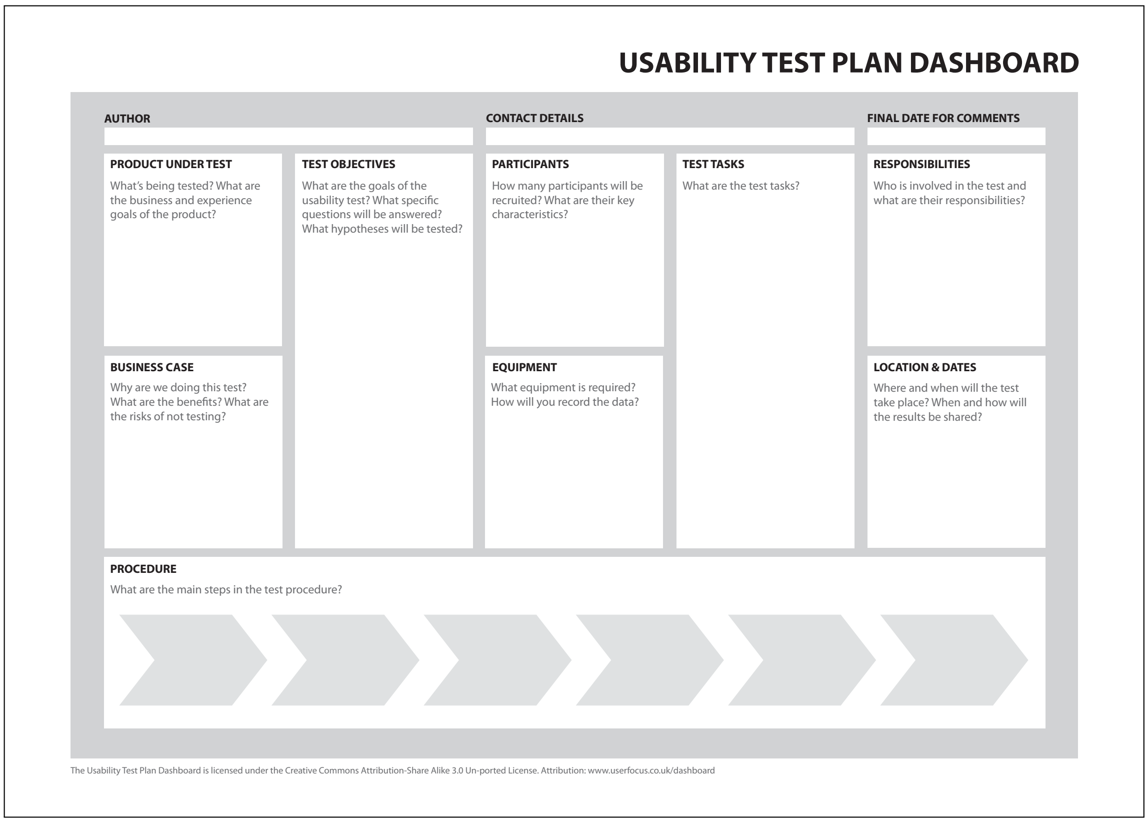
When performing task 4 two of the persons had some difficulties regarding adding a phrase to a phrasebook when being in the phrasebook screen shown in Figure D.2. The reason was that the navigation was quite confusing and that it did not make sense to have an "add to phrasebook-button" in that screen. Even though the navigation was confusing it was obvious for all persons how to add a phrase to a phrasebook when being in the customization screen.

Task 5 and 6 were included in order to test the functions of changing languages on different functionalities in the application and if it was clear what language would be changed. All three persons found it easy to find both screens for changing languages. These screens are shown in Figure D.1. It was clear what would change the application language. Task 5, on the other hand, was fairly unclear since the term target language was used. One person said it was easy to find the functionality when the test conductor explained what the term meant.

D

Usability Test Plan

D.1 Test Specification



USABILITY TEST PLAN DASHBOARD

AUTHOR

CONTACT DETAILS

FINAL DATE FOR COMMENTS

PRODUCT UNDER TEST
What's being tested? What are the business and experience goals of the product?

TEST OBJECTIVES
What are the goals of the usability test? What specific questions will be answered? What hypotheses will be tested?

PARTICIPANTS
How many participants will be recruited? What are their key characteristics?

TEST TASKS
What are the test tasks?

RESPONSIBILITIES
Who is involved in the test and what are their responsibilities?

BUSINESS CASE
Why are we doing this test? What are the benefits? What are the risks of not testing?

EQUIPMENT
What equipment is required? How will you record the data?

LOCATION & DATES
Where and when will the test take place? When and how will the results be shared?

PROCEDURE
What are the main steps in the test procedure?

The Usability Test Plan Dashboard is licensed under the Creative Commons Attribution-Share Alike 3.0 Un-ported License. Attribution: www.userfocus.co.uk/dashboard

Figure D.1: "The Usability Test Plan Dashboard" by `http://www.userfocus.co.uk/dashboard` is licensed under CC BY-SA 3.0

D.2 Product Under Test

In this test, the application Parlira will be tested. Parlira aims to facilitate communication for travelers visiting foreign countries where they have difficulties making themselves understood. The application provides a set of template sentences that can be altered to fit a specific situation.

D.3 Business Case

From the test the developers want to receive information regarding what potential users think of the application and its features and how well they feel the interaction works. The specific questions that require answers are:

- Is the user interface intuitive?
- Is it clear to the user how to use the features within the application?
- Are there any malfunctions that has to be fixed?
- What features could be improved?

D.4 Participants

Five persons will participate in this test. The subjects have similar knowledge in smartphone usage and languages in general. They have all been on journeys where they had to communicate in a foreign language and know what difficulties that could appear.

D.5 Equipment

The only equipment will be a smartphone with the application Parlira installed.

D.6 Responsibilities

When the test is conducted by one person from the development team who will be present to observe how the test persons interact with the application. This person will document the observations and possible malfunctions that will appear.

D.7 Location and Dates

This is not applicable for this test since the test will be performed immediately upon request.

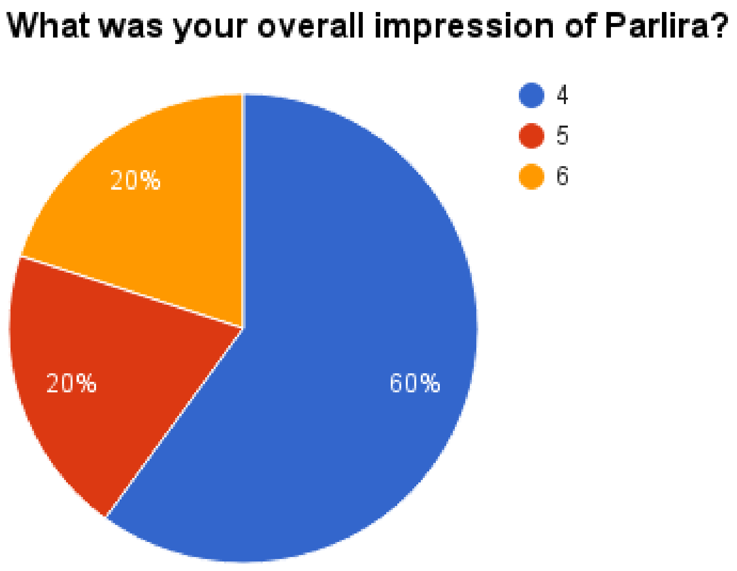
D.8 Follow-up Questionnaire

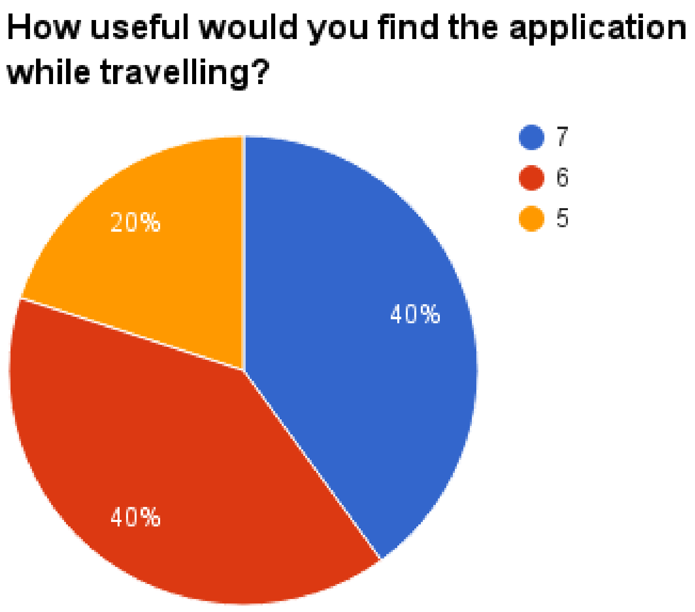
All questions requires an answer between 1 and 10, 1 being bad and 10 being very good.

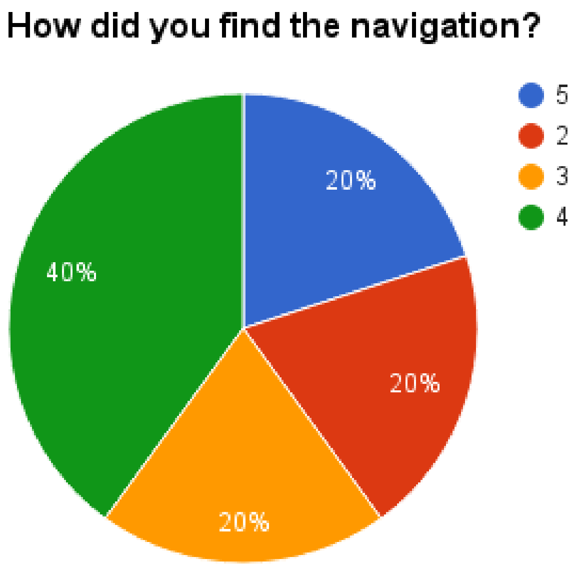
- What was your overall impression of Parlira?
- How useful would you find the application while travelling?
- How did you find the navigation?
- How well did you find the phrase customization screen?
- What features could be improved? In what way?
- Is there any feature you are missing in the application?

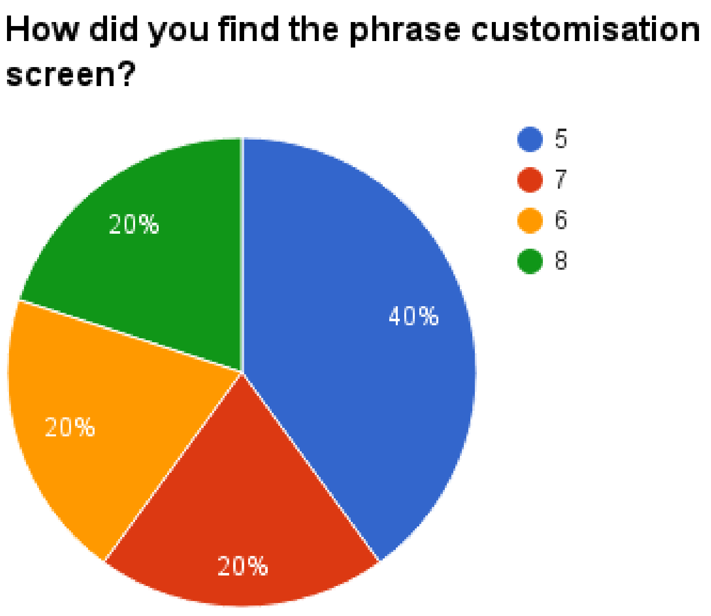
D.9 Summary of Results

The first step in the test for the test persons was to perform the tasks earlier specified. Most of the tasks were completed without any problems. The task where the user would add a phrasebook was problematic for all test persons. They were asked to perform this task directly after having added their own phrasebook so they would still be in the screen for "My Phrasebooks". It was very unclear how they would add a phrase to the phrasebook they had just created. Everyone started by entering the phrasebook but did not find functionality there to add a phrase which all of them thought would be intuitive. After a while all of them found the floating action button in the phrase customization screen and were able to add a phrase to their phrasebook.




How useful would you find the application while travelling?
● 7
● 6
● 5
40%
40%
20%


How did you find the navigation?
● 5
● 2
● 3
● 4
20%
20%
20%
40%



What features could be improved?
The main thing that all users mentioned as something that should be improved is how you add phrases to a phrasebook since it was problematic for all of them to find it. It could be improved by adding a button somewhere in the screen when you have entered a phrasebook. The two users that tried out the application some more than just the tasks pointed out that the user should not be able to save exactly the same phrase twice to a phrasebook. These two persons also found the functionality that it is possible to delete a phrasebook by holding in on the phrasebook. They thought this was very unclear and that some kind of indication that the phrasebook will disappear should be added. Another thing that was brought up by most of the users was that the functionality for changing language should not be in the navigational drawer. This functionality should be easier to reach although it should not be present in the phrase customization screen at all times. Also some of the headlines and titles in the application should be improved, such as renaming the default phrasebook to "Default phrases" instead of "Default".

Is there any feature you are missing in the application?
For two of the users it was unclear that it was only possible to add phrases that already exist in the application to a phrasebook. They thought it would be a nice feature if the user could add other phrases as well, but only as text that could not be customized. Another feature that was discussed was a search function in the phrasebooks so that it would be easier to find a phrase if you know exactly what phrase you are looking for.