CHALMERS

GÖTEBORGS UNIVERSITET

Språkövningsapplikation

Applikation för att träna på glosor och dess böjningsformer

Examensarbete på dataingenjörsprogrammet

DAVID JOHNSSON

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2016

Språkövningsapplikation
Applikation för att träna på glosor
och dess böjningsformer
DAVID JOHNSSON

© DAVID JOHNSSON, 2016.

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Förord

Denna rapport beskriver ett examensarbete utfört på Dataingenjörsprogrammet på Chalmers Tekniska Högskola. Arbetet har inte utförts i samarbete med något företag.

Jag vill ge ett stort tack till min handledare Thomas Hallgren som hjälpt mig med detta arbete. Utan hans hjälp hade arbetet varit helt omöjligt för mig att genomföra. Jag vill även tacka Krasimir Angelov för hjälpen med att få JPGF att fungera.

Sammanfattning

Idag finns många tekniska hjälpmedel såsom mobilapplikationer och webbsidor, för personer som vill lära sig främmande språk. Många sådana hjälpmedel låter dock bara användaren träna på glosor i deras grammatiska grundform. De språkstuderande lär sig då inte känna igen orden i andra böjningsformer, något som är viktigt för att lära sig ett språks grammatik. Det här arbetet har gått ut på att skapa en applikation som hjälper språkstuderande, genom att förhöra dem på glosor i olika grammatiska böjningsformer. Arbetet har inte utförts åt eller vid något företag eller annan organisation, utan är ett helt självständigt arbete. Två applikationer har utvecklats, en för vanliga datorer och en för androidtelefoner. Dessa prototyper visar att språkövningsapplikationer av detta slag går att utveckla, även om just de applikationer som utvecklats i arbetet är mycket begränsade med avseende på antal språk, ordklasser och böjningsformer som stöds.

Nyckelord: språk, grammatik, inlärning, GF, Java, Android.

Abstract

There are many tools available today, such as mobile applications and web sites, for people who are studying foreign languages. But many such tools only let the users practice words in their grammatical base form. The foreign language students will then never learn to recognize the words in other inflection forms, something which is important when learning the grammar of a language. The aim of this project has been to create an application that aids foreign language students by word quizzes containing many different inflection forms. The project has not been done for or at any company or other organization, but is a completely independent project. Two applications have been developed, one for personal computers and one for Android smartphones. These prototypes prove that it is possible to create language learning applications like these, however, the created apps are very limited in the number of languages, parts of speech and inflection forms available.

Keywords: languages, grammar, learning, GF, Java, Android.

Innehåll

Förkortningar och ordförklaringar

CSS - Cascading Style Sheets. Programmeringsspråk för design av grafiska gränssnitt.
FXML - XML-baserat språk för design av JavaFX-applikationer.
GF - Grammatical Framework. Ramverk för att skapa flerspråkiga applikationer.
IDE - Integrated Development Environment. Verktyg för utveckling av mjukvara.
JPGF - Med detta menas javabindningen till PGF-biblioteket skrivet i C, inte det äldre biblioteket skrivet i Java med samma namn.
MVC - Model View Controller. Designmönster som används för att skilja på vy och modell.
NLP - Natural Language Processing. Område inom datavetenskapen vilket handlar om att få datorer att förstå och använda mänskliga språk.
Parsa - Försvenskat verb, från engelskans *parse*. Med detta menas att utföra en analys, tolkning eller kategorisering av information, t.ex. att ta ut satsdelarna i en mening.
Skrivbordsapplikation - Med detta menas ett vanligt program för pc-datorer, alltså inte en applikation för smarta telefoner eller andra enheter.

1
Inledning

1.1 Bakgrund

Detta arbete är baserat på ett hobbyprojekt från sommaren 2015. Då skapades en skrivbordsapplikation med tillhörande Androidapplikation som hjälpmedel för språkstuderande som vill repetera glosor. Med skrivbordsapplikationen kan man, allt eftersom man lär sig nya glosor i det språk man studerar, skriva in orden i applikationen, för att sedan göra repetitionsövningar med just de glosorna. Dessa kan synkroniseras till Androidapplikationen så att samma övningar kan utföras där. Övningarna i applikationerna går ut på att översätta de ord som datorn slumpvis visar på skärmen.

Detta projekt är baserat på denna idé, och arbetet består av att utveckla mer avancerade applikationer för repetition av glosor och grammatik. Ingen kod eller annat kommer att kopieras från hobbyprojektet, allting kommer att skapas från grunden.

1.2 Syfte

När man studerar ett språk, lär man sig hela tiden ny grammatik. Om man använder en applikation likt den som beskrevs ovan, vore det till hjälp om de glosor man förhörs på även står i andra grammatiska böjningsformer än grundformen som finns i textbokens gloslista. Det blir tröttsamt att skriva in varje glosa i applikationen flera gånger i olika böjningsformer, särskilt då antalet böjningsformer ökar proportionellt mot hur långt man kommit i sina språkstudier. Att ha mjukvara som automatiskt kan skapa alla möjliga böjningsformer av orden som skrivs in skulle förenkla detta. Det här projektet går ut på att skapa just denna mjukvara. Projektet kommer först och främst vara fokuserat på att utveckla mjukvara som hanterar grammatiken, men det ingår även i arbetet att utveckla en användarvänlig skrivbordsapplikation. Om tid finns skall även en Androidapplikation som har alla eller åtminstone de flesta av skrivbordsapplikationens funktioner utvecklas.

1.3 Mål

Den applikation som utvecklats när arbetet är färdigt ska ha de funktioner som beskrivs i listan nedan.

- Vid uppstart av applikationen anger användaren sitt modersmål samt vilket språk denne studerar.
- Användaren kan sedan mata in glosor med deras översättningar i applikationen och sortera dem efter ordklass. För oregelbundna ord kan flera former av ordet behöva matas in.
- När användaren vill bli förhörd på glosorna anger denne vilka ordklasser han vill bli förhörd på samt vilka grammatiska regler han vill att orden ska böjas efter. Exempelvis ska man kunna välja att bli förhörd på endast verb i preteritum och perfekt, eller endast substantiv i plural.
- Datorn väljer sedan slumpvis ut ord från de valda ordklasserna och visar dessa på skärmen, böjda enligt de angivna böjningsformerna. Användaren skriver då in en översättning av ordet, varpå datorn visar om svaret var rätt eller fel.

1.4 Avgränsningar

Programmet kommer inte att ha stöd för en mängd olika språk, utan endast ett fåtal. Dock är det meningen att utforma programmet på ett sådant sätt att det blir enkelt att i framtiden utöka programmet med stöd för fler språk. Detsamma gäller ordklasser samt böjningsformer. Programmet kommer inte att kunna böja ord i alla böjningsformer som finns i språket, utan endast några få, men stöd för fler böjningsformer ska vara enkelt att skapa vid en eventuell framtida vidareutveckling.

Programmet kommer inte att säljas eller distribueras till skolor eller andra tänkbara målgrupper. Det kommer därför inte att ske några undersökningar eller tester för att ta reda på hur användarvänligt programmet är. Fokus kommer inte att ligga på applikationernas utseende, grafiska design och enkelhet att använda, men dessa kommer till viss del ändå att has i åtanke.

2

Teknisk bakgrund

Här beskrivs och förklaras de ramverk, hjälpmedel och verktyg som används i arbetet. Syftet är att ge läsaren en förståelse av dessa för att förstå resterande delar av rapporten. Speciellt mycket utrymme ges här åt en grundläggande förklaring av GF, vilket utgör en central del i arbetet. Läsaren förväntas ha en grundläggande förståelse för programmering och mjukvaruutveckling.

2.1 GF

GF är ett programmeringsspråk för att skapa flerspråkiga grammatikapplikationer[1]. GF är open-source och utvecklas bland annat på Chalmers. GF är skrivet i Haskell och bygger på funktionell programmering. För att förstå GF:s syntax är det till stor hjälp om man känner till syntaxen hos Haskell eller andra funktionella programmeringsspråk.

Ett program i GF kallas för en *grammatik*. En grammatik innehåller definitioner för ett språk, med dess grammatiska regler och ordlistor. En grammatik kan användas för att parsa (analysera) ord, satser och meningar samt översätta dem. När en textsträng parsas formas ett abstrakt syntaxträd av texten. Noder i trädet kan vara bland annat ord eller satsdelar. Att utifrån ett sådant träd forma en textsträng i ett visst språk, kallas för linearisering.

En grammatik delas upp i flera moduler, en för abstrakt syntax och en eller flera för konkret syntax. I den abstrakta syntaxen deklareras det som är gemensamt för språken i grammatiken, såsom till exempel ordklasser, medan den konkreta syntaxen definierar hur de saker som deklarerats i den abstrakta ska lineariseras till ett visst språk. För att översätta en text från ett språk till ett annat med GF, behöver man först parsa texten med det ena språkets syntax och sedan linearisera med det andra språkets. Varje modul skrivs i en egen fil. Om filen för den abstrakta syntaxen heter Words.gf, kan de konkreta namnges till WordsSwe.gf för svenska, WordsEng.gf för engelska, och så vidare.

2.1.1 Syntax

I den abstrakta syntaxen definieras de så kallade kategorier och funktioner som grammatiken innehåller. Under rubriken `cat` definieras kategorier, vilka är de olika typer av noder som kan förekomma i abstrakta syntaxträd. Under `fun` definieras

funktioner, vilka var och en tillhör en viss kategori.

I den konkreta syntaxen visas hur kategorierna och funktionerna som definierades i den abstrakta syntaxen ska implementeras i ett visst språk. Under rubriken `lincat` beskrivs hur varje kategori ska lineariseras till just det språket. Under `lin` beskrivs hur funktionerna ska lineariseras.

```
abstract Words = {
        cat
                Noun;
                Verb;
        fun
                Dog : Noun;
                Write : Verb;
}
```

Figur 2.1: Exempel på abstrakt syntax.

```
concrete WordsSwe of Words = open ParadigmsSwe, CatSwe in {
        lincat
                Noun = N;
                Verb = V;
        lin
                Dog = mkN "hund";
                Write = mkV "skriva" "skrev" "skrivit";
}
```

Figur 2.2: Exempel på konkret syntax.

Figurerna 2.1 och 2.2 utgör tillsammans ett exempel på en grammatik med två filer, en för abstrakt och en för konkret syntax. Grammatiken innehåller kategorier för två ordklasser (substantiv och verb) samt funktioner för två ord (*hund* och *skriva*). De textsträngar som kan parsas med denna grammatik måste alltså vara något av dessa ord, alla försök att parsa andra texter kommer att misslyckas. Modulen för konkret syntax använder sig av modulerna `ParadigmsSwe` och `CatSwe` från GF Resource Grammar Library (mer om det senare). Texten *hund* skulle vid parsning bilda ett abstrakt syntaxträd med en enda nod som utgörs av funktionen `Dog`. När detta träd ska lineariseras kommer funktionen `Dog` att returnera typen `N`, eftersom `Dog` är av typen `Noun` och `Noun` lineariseras till `N`. `N` är en kategori som är definierad i modulen `CatSwe`.

Funktionen `Dog` lineariseras till `mkN "hund"` där `mkN` är en operation för att skapa substantiv. Varje språk i GF Resource Grammar Library har sådana operationer för att skapa olika ordklasser, `mkV` för verb, `mkAdv` för adverb e.t.c. Det finns flera varianter av varje operation, överlagrade med olika parameterlistor. För regelbundna ord räcker det med ett argument, men för oregelbundna behövs flera former.

Alltså, för ett regelbundet verb som till exempel *spela* räcker det med att skriva `Play = mkV "spela"` i den konkreta GF-filen, men för ett oregelbundet som t.ex. *springa*, behövs `Run = mkV "springa" "sprang" "sprungit"`. I exemplet ovan ser man att *hund* är ett regelbundet substantiv, medan skriva är ett oregelbundet verb. Parameterlistorna för dessa operationer kan skilja sig åt mellan olika språk i GF Resource Grammar Library.

Figurerna 2.3 och 2.4 visar ett mer avancerat exempel. De kategorier och funktioner som fanns i det förra exemplet finns även med här, men här finns också många fler tillagda.

```
abstract Words = {
  flags
    startcat=Word;
  cat
    Word; Noun; Verb;
    VerbForm; NounForm;
  fun
    VF : Verb -> VerbForm -> Word;
    NF : Noun -> NounForm -> Word;

    Infinitive : VerbForm;
    Past : VerbForm;
    PPart: VerbForm;

    Sing : NounForm;
    Plur : NounForm;

    Dog : Noun ;
    Write : Verb ;
}
```

Figur 2.3: Mer avancerat exempel på abstrakt syntax.

```
concrete WordsSwe of Words = open CommonScand, CatSwe, ParadigmsSwe in
   {
  lincat
    Word = Str;
    Noun = N;
    Verb = V;

    VerbForm = {vf:VForm;s:Str};
    NounForm = {nf:Number;s:Str};
  lin
    Infinitive = {vf=VI (VInfin Act);s=""};
    Past = {vf=VF (VPret Act);s=""};
    PPart = {vf=VI (VSupin Act);s=""};

    Sing = {nf=Sg;s=""};
    Plur = {nf=Pl;s=""};

    VF v f = f.s++v.s ! f.vf;
    NF n f = f.s++n.s ! f.nf ! Indef ! Nom;

    Dog = mkN "hund";
    Write = mkV "skriva" "skrev" "skrivit";
}
```

Figur 2.4: Mer avancerat exempel på konkret syntax.

Kategorin `Word` har i filen för abstrakt syntax satts som `startcat`. Det betyder att de abstrakta syntaxträd som skapas med hjälp av den här grammatiken kommer att ha en rot (startnod) av typen `Word`. Funktionerna `VF` och `NF` kan användas för att skapa nya `Word`. Funktionerna tar två argument, det första är `Verb` respektive `Noun`, och det andra `VerbForm` respektive `NounForm`. För att skapa ett ord behövs alltså själva ordet plus en böjningsform av dess ordklass. Därför innehåller denna grammatik funktioner som representerar ord (`Dog` och `Write`) samt funktioner som representerar böjningsformer (`Infinitive`, `Past`, `PPart`, `Sing` och `Plur`). För att den här grammatiken ska kunna böja ord i fler former, kan man alltså definiera fler sådana funktioner.

GF-syntaxen i modulen `WordsSwe` är komplex och inte nödvändig att förstå för att förstå senare delar av rapporten. Kort kan nämnas att funktionerna som definierar böjningsformer lineariseras som records, innehållandes parametrar (till exempel `VI (VInfin Act)`) från GF Resource Grammar Library. Funktionerna `NF` och `VF` lineariseras genom att skapa ett `Word` av ett ord och en viss böjningsform. Med syntaxen `! Indef ! Nom` menas att alla svenska substantiv som skapas kommer att vara i obestämd form nominativ. Den som vill veta mer om hur GF fungerar rekommenderas besöka GF:s hemsida[1] och läsa Grammatical Framework Tutorial[2] eller GF Library Tutorial[3].

2.1.2 GF Resource Grammar Library

GF Resource Grammar Library är GF:s standardbibliotek som innehåller grammatiska regler, morfologi och syntax för ca 40 språk. Användbara moduler i biblioteket är Paradigms för morfologi och Syntax för meningsbyggnad (bl.a. används modulen `ParadigmnSwe` i exemplet ovan). Modulen `Prelude` innehåller diverse strängoperationer som kan vara användbara om man vill bygga sitt eget bibliotek, till exempel om man vill skapa grammatiker med språk som ännu inte finns i biblioteket.

2.1.3 JPGF

Moduler skrivs i klartext i filer av formatet `.gf`, men när de ska användas kompileras de till GFO-filer (GF-Object) för snabbare hantering. När man kör GF som ett fristående program (GF shell) används dessa filer, men vill man bygga in grammatiken i ett program skrivet i något annat språk, säg Python eller Java, måste filerna kompileras till formatet `.pgf`. PGF-filer är portabla grammatiker som innehåller all nödvändig information samlad från flera GF-filer. För att läsa PGF-filer behövs ett bibliotek för språket i fråga. Bibliotek finns för Java, JavaScript, Python, C och Haskell. Biblioteket för Java, som används i detta arbete, heter JPGF[4]. JPGF innehåller bland annat klasser och metoder för att få listor över kategorier och funktioner från PGF-filer, samt för att parsa textsträngar och linearisera abstrakta syntaxträd.

2.1.4 GF Cloud Service API

GF Cloud Service API är ett API som erbjuder GF:s tjänster och funktionalitet via nätet[5]. Genom HTTP GET och POST kan man utnyttja GF utan att ha det installerat på den lokala enheten. Med GF Cloud Service API kan man bland annat ladda upp filer, kompilera dem samt utföra kommandon från GF Shell.

Att kompilera PGF-filer kan ske, om GF är installerat på den lokala enheten, genom att köra kommandot

```
$ gf -make file1.gf file2.gf ...
```

Om GF däremot inte är installerat kan kompileringen utföras via internet. En katalog för kompilering i GF Cloud kan skapas, och när de GF-filer som ska kompileras laddats upp kan kommandot

```
/cloud?dir=...&command=make&path1=source1&path2=source2&...
```

användas för att skapa en PGF-fil, vilken sedan kan laddas ned och användas på enheten.

2.2 JavaFX

JavaFX är ett bibliotek för att bygga grafiska applikationer i Java för olika plattformar. Eftersom JavaFX stöder alla de vanligaste operativsystemen (Windows, Mac OS X, Linux) är det ett självklart val för detta projekt. JavaFX kan sägas vara

en efterträdare till Swing, vilket är ett äldre grafikbiblioteket för Java. Oracle (utvecklaren till både Swing och JavaFX) vill gradvis avveckla Swing och ersätta det med JavaFX[6], men i dagsläget används båda. För en utvecklare som är van vid Swing är övergången till JavaFX inget stort steg. Det går till och med att kombinera komponenter från Swing och JavaFX i samma applikation för den som vill göra en mjukare övergång.

Skrivbordsapplikationer gjorda i JavaFX består av en `Stage` med en eller flera `Scenes`. En `Stage` är den yttersta behållaren för allt innehåll i programmet. Till den hör fönstrets kanter och standardknapparna för att stänga, minimera och maximera fönstret. I denna sätts sedan en `Scene` som innehåller alla grafiska komponenter som knappar, textfält eller andra behållare. JavaFX är alltså gjort så att man ska föreställa sig programmet som en teater, där man har en fast plattform som fylls med olika scener.

I en JavaFX-applikation som är byggd enligt designmönstret MVC, kan vyn skapas antingen i Java eller med det XML-baserade språket FXML. Genom att använda FXML för att designa det grafiska gränssnittet blir det mycket enkelt att skilja vyn från modell och kontroller. FXML-filer binds samman med Javafiler genom att i början av filen definiera vilken Javaklass som agerar kontroller för just den vyn. Elementen i FXML-filen motsvarar Javaobjekt, vilka kan injiceras i Javakoden med annotationen `@FXML`.

2.3 Android

Android är ett operativsystem för bland annat mobiltelefoner. Eftersom det idag är bland de största operativsystemen för mobila enheter ges det ingen ingående beskrivning här. Värt att säga är dock att applikationsutveckling för Android främst sker i Java, vanligtvis i IDE:n Android Studio[7]. Android Studio är byggt på IntelliJ IDEA[8], vilket är en IDE för bland annat Javautveckling.

3
Metod

I detta kapitel visas hur arbetet lagts upp för att nå målet, vilket är att skapa en Javamodul för att hantera grammatik samt en skrivbordsapplikation som använder denna. Projektet inleds med att söka efter ett lämpligt NLP-verktyg. När det hittats, ägnas större delen av arbetet åt att programmera den mjukvara som är gemensam för de båda plattformarna. Därefter utvecklas skrivbordsapplikationen, sedan Androidapplikationen om tid finns. Om ytterligare tid finns efter att detta är gjort, kan mer avancerade funktioner utvecklas. IntelliJ och Android Studio används som IDE:er för utvecklingen. Till vissa Javaklasser skrivs JUnit-tester för att öka deras tillförlitlighet. För interaktionen med användaren och det grafiska gränssnittet kan inga sådana tester göras.

Den mjukvara som är gemensam för de båda plattformarna ska kunna hantera grammatik, analysera ord, böja dem i olika grammatiska former, samt översätta orden till olika språk. För att underlätta detta bör ett bibliotek, ramverk eller annat hjälpmedel för NLP användas. NLP är ett avancerat område och att skapa ett eget bibliotek för ändamålet från grunden vore mycket komplext och bör undvikas vid ett projekt som detta. I värsta fall, om inget bibliotek som passar ändamålet hittas, kan ett eget minimalt bibliotek behöva utvecklas för att kunna hantera grammatiken. Oavsett vilket bibliotek som används ska mjukvaran för grammatiken vara samlad i en Javamodul som kan importeras till de båda applikationerna.

Skrivbordsapplikationen skrivs i Java och använder sig av ovan nämnda modul. Som beskrevs i inledningen ska man kunna lägga till ord och träna på dessa, samt välja vilka grammatiska former man vill träna på. Androidappen kan ha samma grundläggande funktionalitet, men möjligheten att lägga till nya ord är inte en nödvändig funktion i mobilapplikationen. Alternativt kan ord skrivas in i skrivbordsapplikationen och därifrån synkroniseras till telefonen.

Om tid finns kvar efter att detta är skapat, kan applikationerna utökas med möjlighet att översätta (enklare) meningar och fraser, istället för enbart enstaka ord. Applikationerna ska då kunna forma meningar bestående av de glosor som finns inmatade, böjda i de grammatiska former som användaren valt.

Om allt detta uppnås skulle den slutliga produkten utan större förändringar kunna användas i skolundervisning, där en lärare kan mata in glosor och regler i skrivbordsapplikationen och eleverna träna på dessa genom mobilappen. Läraren skulle till exempel kunna ge sina elever en ny gloslista varje vecka och tillåta att de böjs

enligt de grammatiska regler eleverna dittills lärt sig.

4

Genomförande och designval

I detta kapitel beskrivs hur arbetet har gått till och hur mjukvaran har implementerats för att lösa de problem som stötts på. Stort fokus läggs i detta kapitel på hur de delar av GF som förklarades i avsnitt 2.1 har använts för att nå målen, samt hur mjukvaran i sin slutliga form fungerar från en programmerares perspektiv. I kapitel 5 beskrivs de färdiga applikationerna mer detaljerat från användarens perspektiv, med illustrationer.

4.1 Sammanfattning av arbetsgången

Javamodulen som hanterar grammatiken använder sig av GF som hjälpmedel för NLP. Vid projektets början gjordes efterforskning om andra verktyg för NLP som eventuellt kunde användas. Vissa av dessa hade bara stöd för engelska, andra verkade vara onödigt komplexa och stora för det som behövdes i det här arbetet, och ytterligare andra hade minimal dokumentation. Sökandet efter andra NLP-verktyg än GF upphörde därför snart. GF kan utföra just de uppgifter som behövdes, har stöd för många olika språk och kan enkelt användas i Javaapplikationer med hjälp av JPGF. Med en handledare som själv är en av utvecklarna av GF framstod valet som självklart. Det finns säkerligen andra NLP-verktyg som hade fungerat lika bra som GF, men då NLP är avancerat och alla sådana verktyg oundvikligen är komplexa, kändes det tryggt att använda något som handledaren enkelt kunde hjälpa till med.

Grammatikmodulen var tänkt att utvecklas som ett eget projekt, helt skilt från applikationerna. Men då det i början av arbetet uppstod tekniska problem vid utvecklingen, inleddes arbetet med skrivbordsapplikationen innan modulen var färdig. Grammatikmodulen och applikationen började alltså programmeras parallellt, redan tidigt i arbetet, i samma IntelliJ-projekt. När dessa var färdiga programmerades Andoidapplikationen i Android Studio. Applikationerna programmerades enligt designmönstret MVC. Vy och kontroller skiljer sig mellan de båda applikationerna, men modellen är densamma. Av orsaker som nämns senare i detta kapitel kunde inte modellen (alltså grammatikmodulen) kopieras rakt av från IntelliJ-projektet till Android Studio-projektet.

4.2 Grammatikmodul

Några viktiga Javaklasser som får grammatiken att fungera är `GfFileEditor`, `GrammarManager`, `Word` och `ResourceManager`. Klassdiagrammet i appendix A visar visar dessa och ytterligare några klasser; deras metoder; och hur de relaterar till övriga delar av applikationen. Dessa klassers uppgifter förklaras i avsnitten från 4.2.1 till och med 4.2.6.

För att utnyttja GF:s funktionalitet i Java behövs biblioteket JPGF, vilket kan hämta ut data ur PGF-filer. En av de viktigaste funktionerna i applikationen är att låta användaren lägga till nya ord att öva på, och eftersom det inte går att skriva in ny information till en redan kompilerad PGF-fil, måste alltså orden skrivas till en GF-fil som sedan kompileras till PGF. Det finns två möjliga sätt att utföra sparandet och kompileringen. Antingen lagras alla ord separat från GF, till exempel i en vanlig textfil eller databas. När nya ord läggs till, kompileras då nya GF-filer utifrån listan med alla ord, vilka sedan i sin tur kompileras till en PGF-fil. En annan lösning vore att enbart lagra orden i GF-filer, utan ytterligare textfiler. Vid varje ny insättning av ord parsas då GF-filen, varpå ordet skrivs in på en ny rad på rätt plats i filen, vilken sedan kompileras till PGF. Den första lösningen använder något mer minne, eftersom orden lagras i både GF-filer och på annat sätt. GF-filerna är dock bara ett mellansteg och kan raderas så fort PGF-filen skapats. De två metoderna kan alltså ses som ungefär lika resurskrävande. Den andra lösningen är den som används i grammatikmodulen. Modulen har utrustats med Javaklassen `GfFileEditor`, vilken kan skapa nya GF-filer med grundläggande innehåll, parsa och lägga till/ta bort textrader på specificerade platser i dem, samt `GrammarManager`, vilken kan kompilera filerna till PGF.

För att hjälpa användaren hålla ordning på olika gloslistor har applikationen utrustats med möjlighet att skapa flera s.k. sessioner. Om man exempelvis studerar flera språk kan man skapa en session per språk, men studerar man endast ett kan man till exempel skapa en session per kapitel i läroboken. Varje session hanterar två språk: användarens modersmål (native) samt språket man vill träna på (foreign). All information om existerande sessioner lagras i en mapp, vari varje session har sin egen mapp innehållandes tre GF-filer: en abstrakt syntax, en konkret syntax för native och en konkret syntax för foreign. I dessa mappar hamnar också de PGF- och GFO-filer som kompileras. Skapandet och hanteringen av sessionsmappar sköts av `GrammarManager`. Det är också möjligt att skapa sessioner ”manuellt”, utan att använda applikationens funktion för sessionsskapande. En session består ju bara av en mapp med nödvändiga GF-filer, så om dessa skapas och placeras i rätt mapp, kan sessionen öppnas och användas av applikationen, oavsett på vilket sätt de har skapats.

4.2.1 XML-resurser

I GF Resource Library ingår runt 40 språk, men alla dessa går dessvärre inte att träna på med denna applikation. När användaren skapar en ny session ska applika-

tionen visa en lista för användaren med de språk som finns att tillgå. Dessa språk måste finnas angivna i någon fil som programmet kan läsa från. Eftersom GF Resource Library inte tillhandahåller någon sådan fil, har den nödvändiga filen skapats i form av en XML-resurs. Denna och andra XML-filer innehåller även annan nödvändig information om språken, som till exempel namn på de operationer som skapar ord i GF Resource Library (`mkN`, `mkAdv` e.t.c), namn på viktiga kategorier och funktioner, samt annan information som applikationen måste känna till för att hantera GF-filer på rätt sätt.

Applikationen har endast stöd för språken svenska och engelska; ordklasserna substantiv och verb; samt böjningsformerna singular/plural för substantiv och infinitiv/preteritum/perfekt för verb. Vilka språk, ordklasser och böjningsformer som stöds beror på vad som har definierats i XML-resurserna. Om filerna utökades med information om andra språk, ordklasser eller böjningsformer, skulle dessa kunna användas i applikationen.

XML-resursernas data är tillgänglig genom Javaklassen `ResourceManager`, vilken använder sig av XPath för att hämta filernas data. `ResourceManager` är en hjälpklass som används av flera andra klasser, dels av klasser i grammatikmodulen för att hantera GF-filer på rätt sätt, men även av kontrollerklasserna för att visa korrekt information för användaren. Figur 4.1 visar ett kort utdrag ur en av XML-filerna, vilket innehåller nödvändig information om hur verb ska hanteras i GF.

```
<cat name="Verb" oper="mkV"  cat="VFormFun" paradigm-cat="V">
        <inflection name="Infinitive"/>
        <inflection name="Past"/>
        <inflection name="PPart"/>
</cat>
```

Figur 4.1: Exempel från en XML-resurs som beskriver information om verb.

4.2.2 Skapande av nya GF-filer

Skapandet (och raderingen) av sessioner sker genom anrop av statiska metoder i klassen `GrammarManager`. När en ny session skapats, anropar denna klass i sin tur statiska metoder i `GfFileEditor` som fyller de nyskapade filerna med sitt grundläggande innehåll, det vill säga kategorier och funktioner som definierar ordklasser och böjningsformer, men ej glosor. Glosfunktioner läggs först till i filerna när användaren skriver in ord i applikationen. Innehållet i de nyskapade filerna visas i figur 4.2.

Innehållet i figur 4.2 är för övrigt grunden för den grammatik som visades i exemplet i figurerna 2.3 och 2.4. I det exemplet fanns också två glosfunktioner, för glosorna *hund* och *skriva*.

```
abstract Words =  {
flags startcat = Word ;
cat Word ;
Noun ;NounForm ;
Verb ;VerbForm ;

fun
NFormFun :Noun -> NounForm -> Word ;
Sing : NounForm ;
Plur : NounForm ;

VFormFun :Verb -> VerbForm -> Word ;
Infinitive : VerbForm ;
Past : VerbForm ;
PPart : VerbForm ;

}

concrete WordsEng of Words = open ResEng, CatEng, ParadigmsEng in {
lincat Word = Str ;Noun = N ;
NounForm = {nf:Number;s:Str};
Verb = V ;
VerbForm = {vf:VForm;s:Str};

lin
NFormFun n f = f.s++n.s ! f.nf ! Nom  ;
Sing = {nf=Sg ; s=""} ;
Plur = {nf=Pl ; s=""} ;
VFormFun v f = f.s++v.s ! f.vf  ;
Infinitive = {vf=VInf ; s=""} ;
Past = {vf=VPast ; s=""} ;
PPart = {vf=VPPart ; s=""} ;
}
```

Figur 4.2: Innehållet i filerna för abstrakt respektive konkret syntax, efter att en ny session har skapats.

4.2.3 Redigering av GF-filer

Klassen `GfFileEditor` måste parsa innehållet i en GF-fil innan den kan sätta in och ta bort textsträngar på rätt plats i filen. För varje fil som ska redigeras måste ett nytt objekt av klassen instansieras. Vid instansieringen parsar objektet den specificerade filen och sparar allt dess innehåll i en datastruktur av trädliknande struktur. Objektet anropas sedan för varje ändring som ska göras i filen, och dessa ändringar utförs genom att element läggs till eller raderas i trädstrukturen. När ändringar utförts anropas en metod för att spara filen. Denna metod tar innehållet i datastrukturen och skapar en ny GF-fil av dess innehåll som skriver över den gamla.

Användaren ska kunna skriva in glosor att träna på. Dessa ord måste skrivas in i filerna för abstrakt och konkret syntax. Om t.ex. substantivet *text* skrivs in som ett nytt ord, måste filen för abstrakt syntax utökas med en rad som `Text : Noun` och

den konkreta syntaxen med `Text = mkN "text"` för de båda språken. Namnen på de operationer som ska användas (`mkN`, `mkV` e.t.c.) står definierade i den XML-resurs som nämndes tidigare. `GfFileEditor` har ingen metod för att redigera en rad som redan finns i GF-filen. Om en rad behöver ändras, sker detta genom att hela raden tas bort för att sedan ersättas av en ny.

Glosor läggs till som funktioner i GF-filen, och varje funktion måste ges ett namn. Dessa namn är dolda för användaren, så namngivningen måste ske automatiskt. Om ordet som skrivs in vore *text*, blir funktionsnamnet i regel samma ord fast med stor bokstav, i det här fallet *Text*. Det finns dock problem med att generera funktionsnamn utifrån ordet självt. Om substantivet *fish* och verbet *fish* skrivs in, kommer en namnkrock uppstå, vilket leder till kompileringsfel. Samma sak inträffar om ord som *word* eller *noun* läggs till som glosor. För att undvika detta ges funktionerna numrerade namn: `fun0`, `fun1` o.s.v.

4.2.4 Kompilering av GF-filer

När GF-filer har ändrats behöver en ny PGF-fil kompileras. Detta kan ske på två sätt, antingen med hjälp av GF Cloud Service eller på den lokala enheten. Om GF inte är installerat på den lokala enheten kan PGF-filer kompileras med den metod som beskrevs i 2.1.4. Applikationen behöver skapa en egen katalog för kompilering i GF Cloud, till vilken GF-filer laddas upp och från vilken PGF-filer laddas ned för att användas på enheten.

`GrammarManager` är den Javaklass som ansvarar för att kompilera filer. Då kompilering ska ske via molntjänsten var tanken att `GrammarManager` skulle använda sig av hjälpklassen `GfCloudProxy`, vilken dock i nuläget inte är färdigskriven. På grund av detta kan applikationen inte använda sig av molntjänsten, utan tvingas ha GF installerat lokalt på enheten.

4.2.5 Läsning från PGF-filer

För att utläsa informationen i en PGF-fil används biblioteken JPGF. Det tog mycket längre tid än väntat att påbörja arbetet med JPGF, eftersom det under installationen uppstod problem med datorn. Ett par veckor för sent löstes dessa problem med handledarens hjälp och först då kunde arbetet på grammatikmodulen gå vidare i normal takt.

JPGF används för att utläsa information från en PGF-fil i form av listor över alla kategorier och funktioner i grammatiken. JPGF används även för att linearisera abstrakta syntaxträd och på så sätt få fram den text som ska visas för användaren. Figur 4.3 visar ett (något modifierat) avsnitt ur javaklassen `Word`, vilken (som namnet antyder) representerar ett ord. Klassen innehåller bland annat metoder som använder JPGF för att linearisera till båda språken, native och foreign.

```
private Concr nativeConcr;
private String function;
private String category;

public String getNativeInflectionFormByName(String inflectionName){
  Expr function = new Expr(function, new Expr[0]);
  Expr form = new Expr(inflectionName, new Expr[0]);
  Expr word = new Expr(ResourceManager.getPartOfSpeechLinCatByName(
     category), function, form);
  return nativeConcr.linearize(word);
}
```

Figur 4.3: Javametod för linearisering av ord.

Klasserna `Concr` och `Expr` kommer från JPGF. Objektet `nativeConcr` är ett Javaobjekt som representerar modulen för konkret syntax för användarens modersmål och alla `Expr`-objekt representerar abstrakta syntaxträd. Genom att använda `Concr`-objekt för olika språk vid lineariseringen kan man skapa textsträngar i olika språk. När en form av ordet ska visas anropas metoden i figur 4.3, med namnet på den böjningsform som ska genereras som argument. För att skapa ett abstrakt syntaxträd som representerar ett ord med hjälp av den här grammatiken, behövs två argument: funktionen för själva ordet samt en böjningsform. Dessa skapas på de två första raderna i metoden. Den tredje raden använder dessa två som argument för att skapa själva ordet, vilket sedan linealiseras till ett strängobjekt som returneras på sista raden.

4.2.6 Översättning

Huvudsyftet med applikationen är att träna på glosor. Detta sker genom att applikationen visar ord på skärmen som användaren ska översätta. När ett nytt ord ska visas, anropas `GrammarManager` som slumpmässigt väljer ut en av funktionerna i PGF-filen och returnerar den i form av ett `Word`-objekt. Detta ord linealiseras till det språk användaren ska översätta från, och visas därefter på skärmen. När användaren sedan angivit sin översättning, måste svaret jämföras med flera olika textsträngar för att se om det är korrekt. Det ord som visas på skärmen kan nämligen ha flera korrekta översättningar.

Ibland kan ett ord se likadant ut i olika böjningsformer, t.ex. *träd* (singular) och *träd* (plural). När användaren ska översätta ett ord från svenska till engelska, och det ord som visas på skärmen är *träd*, måste både *tree* och *trees* vara godkända svar. Applikationen tar hänsyn till detta och godkänner båda svaren. Användaren har möjlighet att välja vilka böjningsformer som ska förekomma i övningarna. Man kan till exempel välja att bara få övningar som innehåller substantiv i plural. I detta fall kan man tänka sig att programmet inte skulle ta hänsyn till att ord kan se lika ut i olika former, utan endast utgå från den form som användaren angivit. Som ett exempel: säg att användaren tränar på att översätta ord från engelska till svenska och väljer att bara öva på verb i infinitiv. På skärmen visar ordet *read*. Detta kan

översättas med *läsa*, *läste* eller *läst*. Användaren har visserligen valt att endast låta verb i infinitiv förekomma, och därför är ordet på skärmen alltså lineariserat i infinitiv (*läsa* är då rätt svar), men faktum kvarstår att *läste* och *läst* fortfarande är korrekta översättningar av ordet som visas. Alltså kommer programmet att markera även dessa som godkända svar trots allt. Ibland kan två helt olika ord se likadana ut i ett annat språk. Ordet *pil* på svenska kan översättas med *arrow*, *dart* eller *willow*. I dessa fall tar programmet också hänsyn till att användarens svar kan vara en korrekt översättning av ett annat ord än det som lineariserats och visas på skärmen. Om en funktion för ordet *arrow* valts ut och visas som *pil* på skärmen, kommer svaren *dart* och *willow* också vara godkända svar.

För att få de just nämnda funktionerna att fungera används JPGF:s parsefunktion. I de situationer då en textsträng kan tolkas på flera olika sätt (t.ex. *pil* som i exemplet ovan), kan man vid parsning av ordet iterera över resultatet från parsefunktionen för att se alla möjliga korrekta tolkningar. Både ordet som visas på skärmen och det svar användaren ger kan tolkas på olika sätt. När applikationen ska kontrollera svaret, anropas en metod i `Word`-objektet för det ord som visas på skärmen med användarens svar som argument. Metoden jämför varje möjlig parsning av ordet som visas med varje möjlig parsning av användarens svar. Om någon av dessa matchar, returneras `true`, varpå användaren får se att svaret var korrekt.

4.3 Skrivbordsapplikation

Applikationen har byggts med JavaFX enligt designmönstret MVC. Fördelen med JavaFX är att det är mycket enkelt att separera modell, vy och kontroller, eftersom vyn skapas i FXML och det övriga i Java. För den grafiska designen har även CSS används för att skapa ett tilltalande och modernt utseende.

Applikationen är uppdelad i ett antal vyer: start, övning, redigering och statistik. Varje vy består av en FXML-fil och en kontroller. Vid uppstart av applikationen skapas en `Stage` med en `Scene`, vilken fylls med de komponenter som är specificerade i starvyns FXML-fil. När en ny vy ska visas töms scenen och fylls med nya komponenter enligt FXML-filen för den vyn som ska visas.

Kontrollerklasserna använder sig av flera klasser från grammatikmodulen. Klassen `Model` agerar som ett interface som förbinder kontrollerklasserna med bland annat `GrammarManager`. `Model` ger även kontrollerklasserna tillgång till statistik för användarens översättningar. Objekt av typen `Word`, som visserligen har stark koppling till grammatikmodulen, används även mycket i kontrollerklasserna, när ord ska visas i olika böjningsformer och översättningar.

4.4 Androidapplikation

Androidapplikationen är skapad i Android Studio. Applikationen har i stort sett alla funktioner som skrivbordsapplikationen har, förutom möjlighet att lägga till och ta bort ord från listan samt att se statistik. Tanken var att mobilapplikationen skulle använda exakt samma grammatikmodul som skrivbordsapplikationen, men det var inte möjligt av två skäl. För det första sker filhanteringen annorlunda för Android, då filer som Androidapplikationen använder sig av ska placeras i vissa mappar. När kod från skrivbordsapplikationen kopierades till Android Studio behövde alltså några rader i de Javaklasser som hanterar filer skrivas om.

För det andra finns det inget enkelt sätt att installera GF på Android, vilket behövs för att kompilera PGF-filer. Därför måste androidapplikationen (och eventuellt andra mobilapplikationer som använder GF) kompilera filer med hjälp av GF Cloud Service API. En mobilapplikation skulle kunna skapa GF-filer precis som vanliga textfiler, ladda upp dem till GF Cloud, för att sedan ladda ner en kompilerad PGF-fil som kan användas med JPGF, om det finns installerat. Finns inte JPGF är det inte möjligt att använda PGF-filer över huvud taget i applikationen. Som redan nämnts är Javaklassen för kommunikation med GF:s molntjänst inte färdigskriven, vilken gör att Androidapplikationen inte har någon som helst möjlighet att kompilera GF-filer.

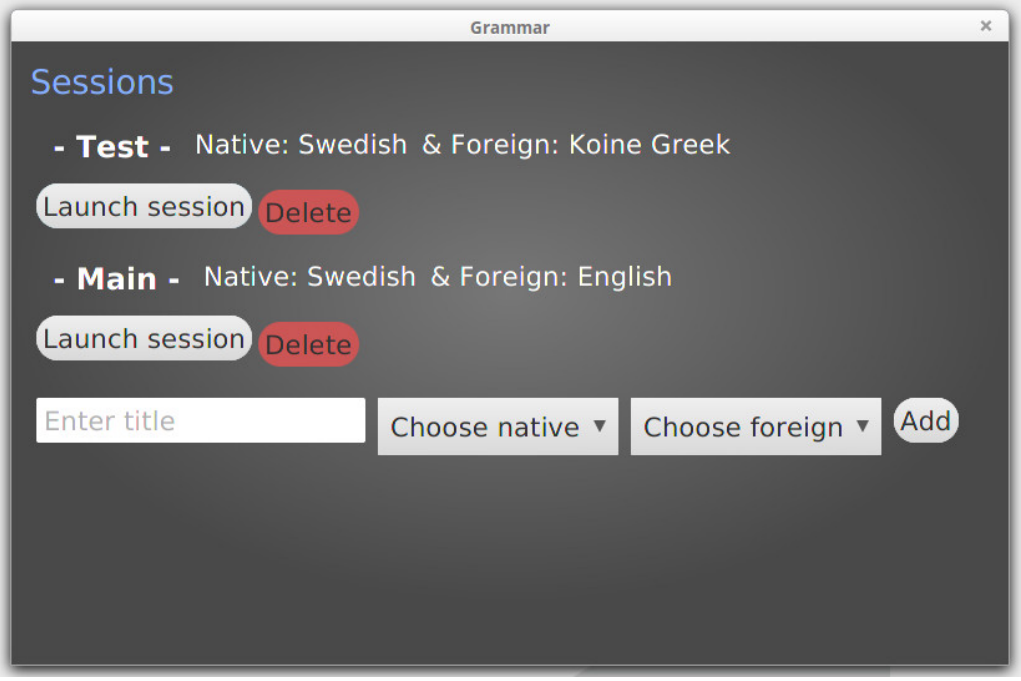
Androidapplikationen har därför programmerats för att fungera helt utan GF. Applikationen är gjord som ett komplement till skrivbordsapplikationen och kan inte användas utan den. Utan möjlighet till kompilering av filer är det ingen mening med att låta användaren lägga till nya ord. Nya ord kan alltså bara läggas till genom skrivbordsapplikationen, vilken sedan trådlöst kan sända informationen till mobilapplikationen. Datan som sänds skulle kunna vara en helt vanlig textfil, eller något annat vanligt förekommande dataformat såsom JSON eller XML. Men i det här fallet, då båda applikationerna kör Java, består informationen för enkelhets skull av ett serialiserat fält av Javaobjekt. Dessa objekt är samlingar med ord och deras böjningsformer.

Överföringen mellan enheterna sker direkt genom att mobilen ansluter till datorn genom ett helt vanligt objekt av javaklassen `Socket`, varvid överföringen sker genom Javas objektströmmar. Ett problem att lösa var dock hur mobilen kan få reda på datorns IP-adress. Ett sätt vore att låta datorn skicka ett broadcast med sin adress till alla enheter i det lokala nätet. Lösningen som användes till slut är dock en annan. För att få datorns IP-adress måste mobilen först skanna en QR-kod på datorn. Denna kod visas först när man klickar på knappen *Sync* i övningsvyn. QR-koden innehåller datorns IP-adress, vilket gör det mycket smidigt för mobilen att få reda på adressen att ansluta till. I mobilens startvy finns en knapp för att öppna skanningsvyn. QR-koden genereras med hjälp av open sourcebiblioteket ZXing[9] och skanning i Androidappen med open sourcebiblioteket ZBar[10]. För att överföra en session måste användaren (1) starta sessionen på datorn, (2) klicka på knappen för synkronisering och (3) öppna scanningsvyn på mobilen och skanna koden. Så fort

koden är skannad startas överföringen, QR-koden försvinner från datorns skärm, och mobilapplikationen frågar användaren vad som ska göras med den överförda informationen.

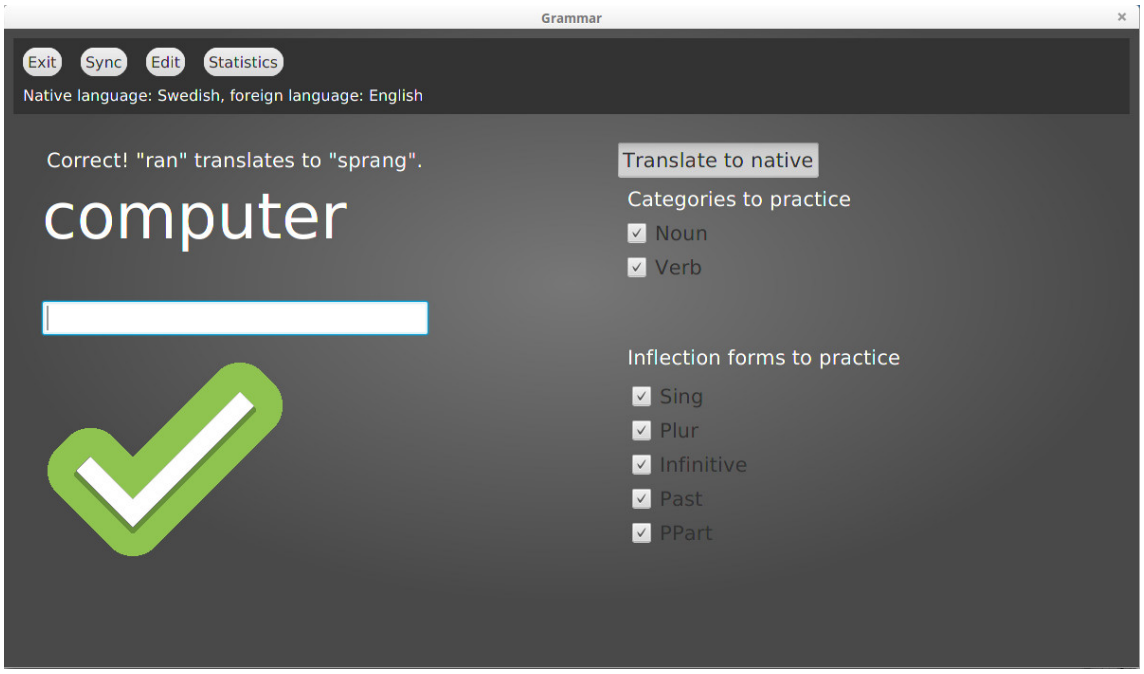
5
Resultat

Som beskrevs i avsnitt 4.3 består skrivbordsapplikationen av de fyra vyerna start, övning, redigering samt statistik. För att navigera mellan vyerna finns ett fält med knappar längst upp i fönstret. I fältet finns också en knapp för att avsluta sessionen. Detta fält finns dock inte i startvyn, eftersom ingen session ännu startats så länge startvyn visas.

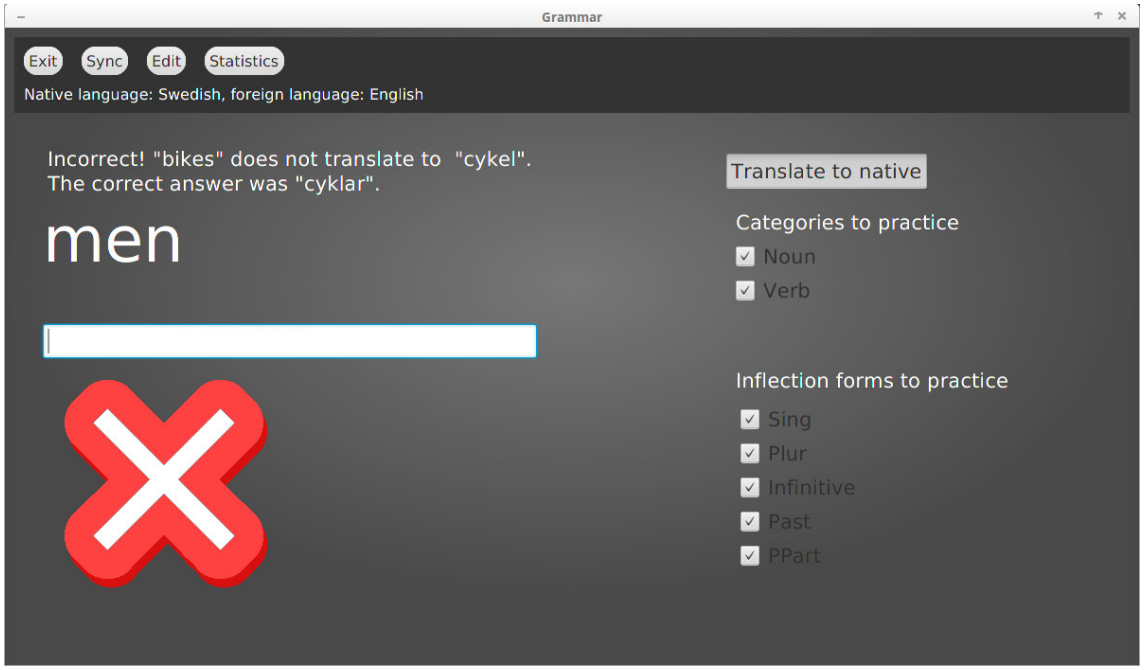


Figur 5.1: Startvyn.

I startvyn (se figur 5.1) finns en lista över alla skapade sessioner, med deras namn och språk. I startvyn syns sessionerna **Test** med språken svenska och koinegrekiska, samt **Main** med svenska och engelska. I avsnitt 4.2.1 nämndes att applikationen bara kan skapa sessioner med språken svenska och engelska. Sessionen med koinegrekiska har därför skapats ”manuellt” så som beskrevs i avsnitt 4.2. Nedanför listan med sessioner finns fält som kan fyllas i för att skapa en ny session. Vid varje session finns knappar för att radera eller starta sessionen. När en session öppnas visas övningsvyn.



Figur 5.2: Övningsvyn vid korrekt svar.



Figur 5.3: Övningsvyn vid felaktigt svar.

Övningsvyn (se figur 5.2) är den viktigaste vyn, eftersom själva glos- och grammatikövningen tar plats här. Det ord som användaren ska översätta visas stort och tydligt. När en översättning angivits visas en grön bock (se figur 5.2) eller ett rött kryss (se figur 5.3) under ordet, samt en mening som säger om svaret var rätt eller fel ovanför ordet. I figur 5.3 syns att det ord som användaren just skrivit var *cykel*, men att den rätta översättningen skulle vara *cyklar*, och att nästa ord att översätta

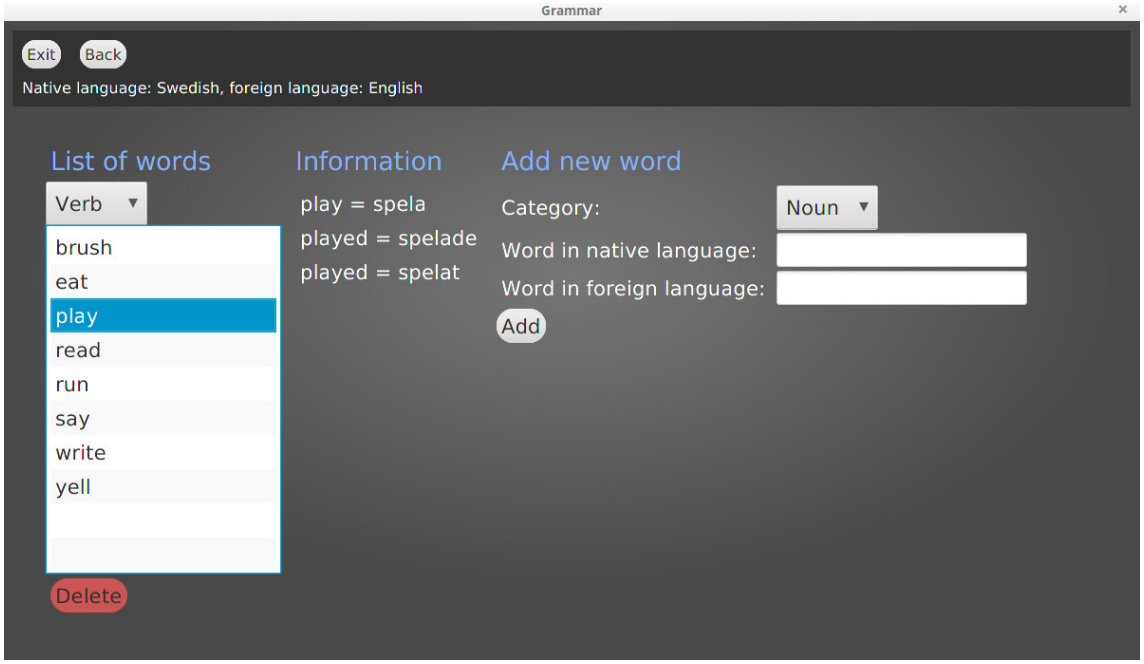
är *men*.

Till höger om allt detta kan användaren konfigurera alternativ för övningarna. Längst upp i kolumnen med inställningar kan man växla mellan att översätta ord från sitt modersmål (*Native*) till språket man vill träna på (*Foreign*) eller tvärtom. Under detta kan man välja vilka ordklasser och böjningsformer som ska förekomma.

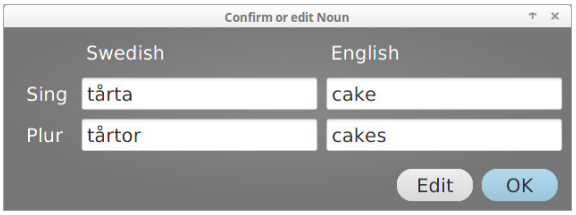
Knappen för att synkronisera en session till mobilapplikationen syns i fältet i fönstrets övre del, men bara i övningsvyn. När denna knapp trycks visas ett fönster med QR-koden (se figur 5.4) som kan skannas med mobilappen. Fönstret stängs så fort överföringen har avslutats.

Figur 5.4: QR-koden för synkronisering.

I redigeringsvyn (se figur 5.5) finns listor över alla sparade glosor, sorterade efter ordklass. Genom att klicka på något av dessa ord visas till höger om listan information om ordets alla böjningsformer, i båda språken. Längst till höger i vyn kan användaren skriva in nya glosor.



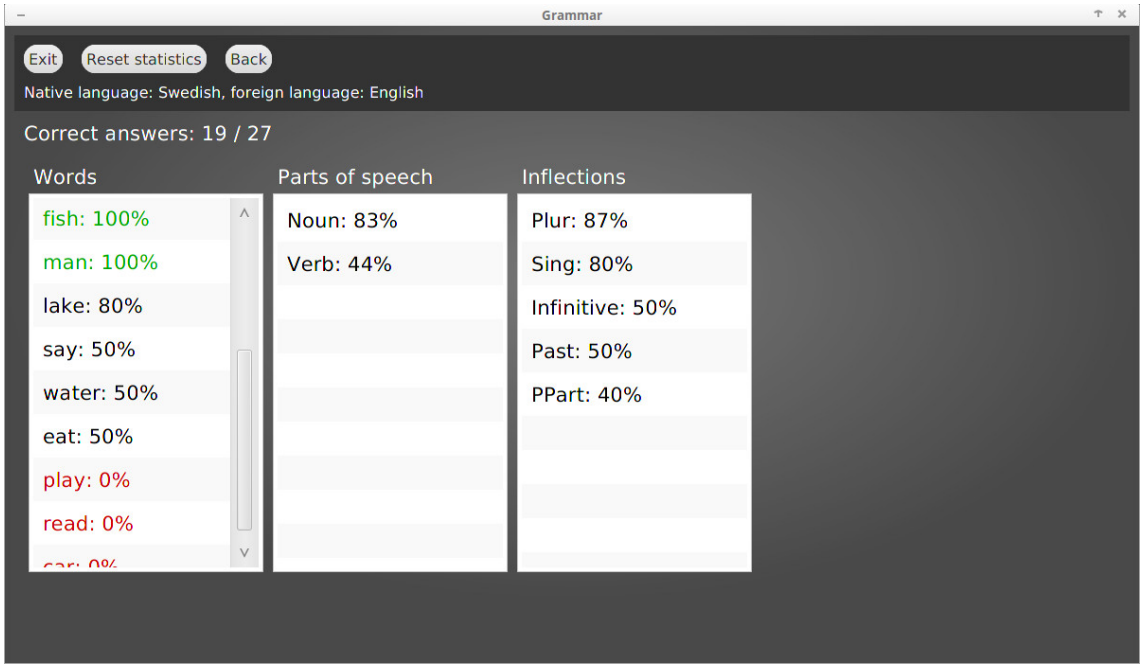
Figur 5.5: Redigeringsvyn.



Figur 5.6: Dialogruta med böjningar.

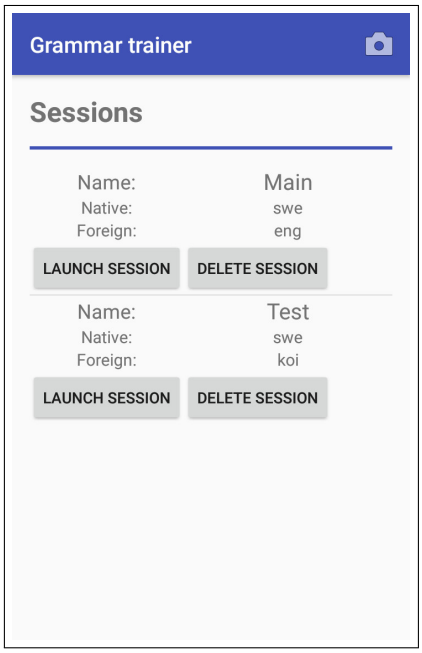
När en ny glosa läggs till visas en dialogruta med alla möjliga böjningsformer av ordet. Dialogrutan (se figur 5.6) uppmanar användaren att kontrollera om formerna är korrekta. Om ordet är regelbundet är alla korrekta och användaren behöver bara stänga ned fönstret, men är det oregelbundet kan användaren redigera böjningsformerna, så att de står korrekt skrivna, innan fönstret stängs ned.

I statistikvyn (se figur 5.7) visas tre listor: ord, ordklasser och böjningsformer. I varje lista visas de saker som användaren blivit förhörd på, sorterade efter den procentuella andelen korrekta svar. Ovanför dessa listor visas det totala antalet översättningar som gjorts samt det totala antalet rätta svar.

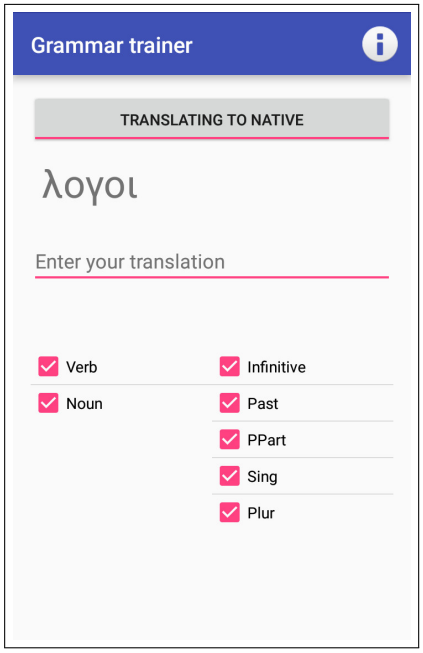


Figur 5.7: Statistikvyn.

Androidapplikationen har fyra vyer: start, övning, skanning och ordlista. Det finns ingen vy för att se statistik. De två första är i princip samma som hos skrivbordsapplikationen, de har samma funktioner men har självklart annat grafiskt utseende. I startvyn (se figur 5.8) finns i fältet längst upp en knapp för att öppna skanningsvyn, vilken används då en session ska synkroniseras från datorn till telefonen. Det finns ingen figur som visar skanningsvyn, då den vyn endast visar kamerans sökare (det som kameran är riktad mot). I övningsvyn (se figur 5.9) finns i fältet längst upp en knapp som leder till listvyn.

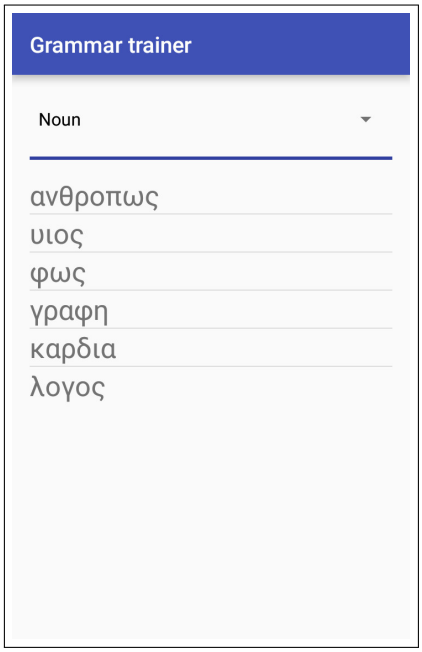


Figur 5.8: Startvyn i mobilapplikationen.

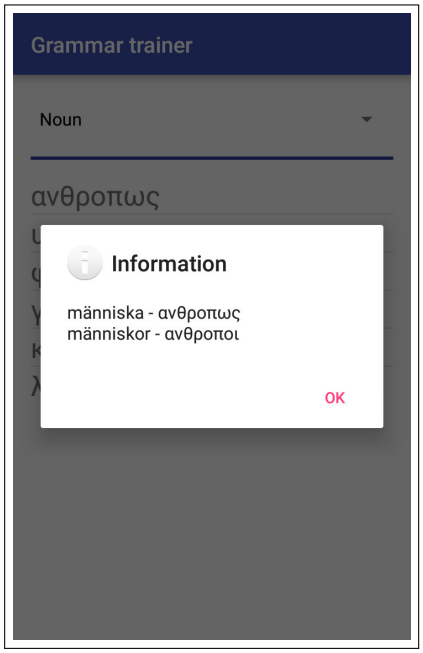


Figur 5.9: Övningsvyn i mobilapplikationen.

Den fjärde vyn i applikationen (se figurerna 5.10 och 5.11) är som skrivbordsapplikationens redigeringsvy, d.v.s. en lista över alla ord i varje ordklass, men utan möjlighet att lägga till nya ord. Genom att klicka på ett ord i listan får man upp ett fönster med information om alla böjningsformer (se figur 5.11).



Figur 5.10: Listvyn i mobilapplikationen.



Figur 5.11: Dialogrutan som visas när ett ord i listan markeras.

6
Diskussion

6.1 Uppnåddes målen?

Uppnåddes de mål som satts upp vid projektet början? Ja, de funktioner som beskrevs i inledningen finns implementerade. Applikationerna ger användaren möjlighet att träna på de glosor användaren själv skrivit in, genom att böja ord i olika grammatiska former. Användaren har kontroll över vilka grammatiska former som ska förekomma i övningarna. Detta fungerar i de båda applikationerna. Det är dock bara skrivbordsapplikationen som låter användaren skriva in nya glosor. Android-applikationen är beroende av skrivbordsapplikationen för att trådlöst få listan med glosor att träna på.

6.2 Kritik av arbetet

Trots att de mål som satts upp har uppnåtts, anser jag tyvärr inte arbetet som helt lyckat, framför allt eftersom det inte är så utbyggbart som önskades. Under arbetets första halva, hade jag flera tekniska problem med mina datorer vilket saktade ner arbetet. Detta berodde bland annat på min ovana att arbeta på större projekt i Linuxmiljö. Genom detta försvann mycket tid som annars kunde ha lagts på själva programmeringen.

Möjligheten att översätta hela meningar istället för bara enstaka ord fanns det ingen tid att implementera. Att i efterhand försöka implementera detta skulle nog vara omöjligt, på grund av de orsaker som diskuteras nedan. Men det råder ingen tvekan om att en applikation med stöd för hela meningar kan göras med GF. GF har stöd för det, problemet är att min applikation inte har det. Den skulle behöva göras om från grunden för att få det att fungera.

Var GF lämpligt som hjälpmedel för NLP i projektet? GF passar visserligen utmärkt att använda i den typ av applikationer som gjorts i projektet, men däremot hade GF en hög inlärningströskel, vilket tvingade mig att tänka om helt och hållet flera gånger, allt eftersom jag lärde mig mer om hur GF var uppbyggt. Den förlorade tiden i första delen av projektet ledde också till att andra delen var mer fokuserad på att snabbt få klart en fungerande prototyp, än att göra ett stabilt, utbyggbart system. GF var alltså mycket lämpligt som NLP-verktyg, men dess fulla potential lyckades inte utnyttjas.

Det känns som om programmet är fyllt med ad hoc-lösningar (lösningar som endast fungerar i en specifik situation), vilket är orsaken till att det inte är så expanderbart som önskades. Tanken var att det skulle vara smidigt att utveckla stöd för fler böjningsformer, fler ordklasser och fler språk. Som beskrevs i 4.2.1, definieras denna information i en XML-fil. I vissa fall kan det vara mycket enkelt att till exempel lägga till stöd för en ny ordklass, genom att helt enkelt utöka filen med några få rader. Men i andra fall är det omöjligt, eftersom vissa antaganden om hur språken är uppbyggda i GF Resource Library har gjorts. Ett exempel är att applikationen förutsätter att parameterlistorna för operationen `mkN` är identiska i alla språk, vilket är ett felaktigt antagande. Denna och andra liknande ad hoc-lösningar gör att projektet inte känns helt lyckat. Genom att utöka XML-resurserna med information om parameterlistorna hos alla språks operationer, skulle detta till viss del åtgärdas. Däremot skulle då applikationen behöva utsökas med ett betydligt mer avancerat gränssnitt för inmatning av information om de nya ord som läggs till.

De övriga delarna av arbetet, användningen av JavaFX och Android, var det inga problem med. Eftersom jag har tidigare erfarenhet av programmering i Android och Swing, var det inga större hinder som dök upp i arbetet med användargränssnittet. JavaFX var ett mycket lämpligt hjälpmedel för att bygga skrivbordsapplikationen.

6.3 Framtida utveckling

Kommer jag att använda applikationen, när jag studerar språk i framtiden? Eftersom den inte just nu har stöd för fler språk än svenska och engelska blir svaret nej. För att träna på engelska finns det dessutom redan andra glosträningsappar, två exempel är Glosboken[11] och Memrise[12]. Dessa har visserligen inte stöd för att automatiskt böja ord, men har andra funktioner som gör dem mer tilltalande att använda, till exempel stavnings- och uttalsövningar och möjlighet att dela övningar med andra användare. Dessa använder sig inte av NLP, utan hanterar därför ord som simpla textsträngar utan att böja dem i olika former.

Kommer jag fortsätta utvecklingen av applikationen? Som sagt har jag gjort felaktiga antaganden, vilket i princip omöjliggör en direkt vidareutveckling av mjukvaran så som den är i sitt nuvarande tillstånd. Den behöver göras om från grunden innan en seriös vidareutveckling kan påbörjas. Eventuellt kommer jag att ta mig an arbetet att göra om applikationen i framtiden.

Referenser

[1] Grammatical Framework www.grammaticalframework.org (2016-05-30)
[2] Ranta A. Grammatical Framework Tutorial www.grammaticalframework.org/doc/tutorial/gf-tutorial.html (2016-04-29)
[3] Ranta A. Creating Linguistic Resources with the Grammatical Framework www.grammaticalframework.org/lib/doc/synopsis.html (2016-04-29)
[4] PGF library API (Java) www.grammaticalframework.org/doc/java-api/index.html (2016-04-15)
[5] GF Cloud Service API cloud.grammaticalframework.org/gf-cloud-api.html (2016-04-20)
[6] Oracle. JavaFX Frequently Asked Questions www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html (2016-04-15)
[7] Android Studio developer.android.com/studio/index.html (2016-04-31)
[8] Jet Brains www.jetbrains.com/idea/ (2016-04-31)
[9] ZXing github.com/zxing/zxing (2016-05-18)
[10] ZBar github.com/ZBar/ZBar (2016-05-18)
[11] Glosboken www.glosboken.se (2016-06-31)
[12] Memrise www.memrise.com (2016-06-31)

Appendix A

Klassdiagram som visar de viktigaste klasserna i grammatikmodulen, och hur dessa är sammankopplade med skrivbordsapplikationen.