Talstyrning av katalogstruktur

Erik Anderberg
Datavetenskap
Naturvetenskapliga fakulteten
Lunds universitet
cloudius@hotmail.com

Raoul Nilsson
Datavetenskap
Naturvetenskapliga fakulteten
Lunds universitet
lukasdo-main@hotmail.com

Niclas Reisnert
Datavetenskap
Naturvetenskapliga fakulteten
Lunds universitet
n_reisnert@hotmail.com

Abstract

This document describes the evolution process of a voice-controlled directory traverser, from choice of language and platform, to demo program. Described are also the difficulties and limitations encountered during the development process, as well as possible directions for further development.

1 Introduktion

Vår primära målsättning med projektet var att utveckla en fristående så kallad stand-alone applikation som inte skulle vara webbaserad eller beroende av någon online-server för att köra. Anledningen till detta är att vi har velat utveckla en mapptraverserare (liknande utforskaren i Windows fast röststyrd). Projektet hade blivit avsevärt mer komplicerat ifall vi hade försökt tillämpa en nätbaserad lösning på ett program som är menat att hantera ett lokalt filsystem. Ursprungsidén var att på ett eller annat sätt kunna orientera sig med hjälp av tal. Istället för att bygga ett litet scenario för detta orienteringssystem, såsom ett schackbräde eller liknande, så verkade det långt mer intressant att utnyttja datorns redan befintliga mappträd. Detta gör projektet mer praktiskt och lämnar en öppning för mer användbar vidareutveckling.

Ett annat mål vi hade var att programmet skulle använda sig av en dynamiskt genererad XML-grammatik. Med andra ord ville vi att programmets grammatik skulle styras av ett XML-dokument som uppdateras under tiden man använder programmet. Vi vet nu i efter hand att det finns fler än ett sätt att integrera grammatik i talstyrda program, bland annat kan man låta programkoden innehålla grammatiken direkt. Vi ansåg att XML-varianten var mer intressant då den förhöll sig mer relevant till kursens innehåll.

Ett sista mål var att programmet skulle vara körbart i Windows XP och i bästa fall i Windows Vista. Vi ansåg att vi ville utveckla projektet i ett operativsystem som vi var vana vid och hade vi läst att Windows skulle ha ett visst stöd för taligenkänning.

2 Utvecklingsprocessen

I vår planeringsprocess bestämde vi oss tidigt för att göra ett program som med hjälp av taligenkänning skulle kunna traversera en mappstruktur. Så som en utforskare används i Windows. Detta betyder att grammatiken som skulle användas var tvungen att vara dynamisk, varje gång man byter mapp måste man ändra grammatiken. Det tidiga målet var att göra ett program som kunde exekveras lokalt, ingen internetuppkoppling skulle krävas över huvud taget. Utöver detta var våra mål ganska flytande, ta det som det kommer blev det och se om vi kan hitta något verktyg att arbeta med. Vi satte också upp en exempeldialog, om än primitiv:

Människa *startar program och sätter igång taligenkänningen*
Dator svarar med att visa filerna och mapparna
Människa: Open music
Datorn går in i mappen music och visar dess innehåll
Människa: Open albums
Datorn går in i mappen albums och visar dess innehåll
Människa: The Foghat Talk to me baby
Datorn spelar musiken i förinställd mediaspelare

Vi diskuterade huruvida vi skulle konfirmera våra val eller inte, dvs om datorn skulle fråga om det den tolkar var det man menade. Men detta avfärdade vi med att istället välja att man får gå tillbaka om datorn tolkar fel.

Att hitta verktyg visade sig vara svårare än vi ursprungligen trodde. Våra tankar var att det skulle finnas något Open Source projekt som man kunde ta till användning. Detta fanns, men var tyvärr undermåligt utvecklade eller hade helt stagnerat. Exempelvis läste vi om Javas Speech Recognition[1], som inte verkade speciellt bra. AT&T[2] har ett projekt igång som utvecklar taligenkänning, detta var tyvärr inriktat på mobiltelefoner och inte applicerbart på vårt projekt. TellMe[3] var ett annat sätt som vi utforskade, detta var kommunikationsbaserat via mobil, man laddade upp sin XML-kod, även känt som VoiceXML[4], och sen ringde man upp ett nummer för att använda det. Väldigt intressant och användbart men tyvärr var detta inte heller något som gick att applicera på det vi tänkte göra. Istället vände vi oss till Microsoft och deras Speech API[5]. Att välja Microsoft betyder att man får tampas med en uppsjö av olika versioner, vi kunde skära ner det snabbt till två olika Speech API 5.1 och 5.3. Skillnaderna mellan dessa uppfattade vi från början inte som så stora, förutom att 5.3 var för Vista, vilket då gjorde valet med 5.1 enkelt.
Till att börja med fick vi sätta upp en utvecklingsmiljö. Till vår hjälp hade vi Visual Studio[6] som ganska smärtfritt fick alla bibliotek att fungera för oss efter lite manuellt intrång. För att göra det bekvämt för oss valde vi C# då vi är väl bekanta med Java var detta det mest likvärdiga. Med hjälp av ett exempel[7] som demonstrerade hur man använde igenkänningsmotorn var vi i farten. För att lära oss hur det fungerade gjorde vi personliga modifieringar och ändringar i exemplet tills vi var mer insatta i hur man använde sig av API:n.
Att välja Visual Studio gav oss vissa fördelar, det är enkelt och snabbt att skapa gränssnitt. Vi började i den ändan för att sen lägga på funktionaliteten. Detta för att enkelt kunna testa om det gjorde som tänkt. Att debugga tal är en konst, då man inte riktigt vet om den känt igen det man ville att den skulle känna igen, åtminstone inte i början. De första försöken var primitiva, vi lade in vissa mappar och såg om den kunde markera dessa. Efter det gick vi vidare till att ladda in alla mapparna i C-roten, när det funkade lade vi till filerna. Vår ursprungliga plan var att exekvera de filer som man säger, men vi insåg att detta inte skulle hinnas med under projektets tidsplan, vi valde att istället bara markera dem. Sedan kom det mest avancerade och kritiska för att uppnå vårt mål. Den dynamiskt förändrande grammatiken. Nu skulle vi inte bara känna igen om det var filer eller mappar som sades utan också öppna mappar, rensa den gamla grammatiken och läsa in ny, samt införa ett upp-kommando så man kunde ta sig tillbaka. Vi hade redan skillnader mellan filer och mappar, vi införde snabbt ett kommando för att ändra sökvägen när vi skulle läsa in mappar till grammatiken genom att helt enkelt utöka sökvägen från c:. Sen tog vi den enkla vägen och rensade hela grammatiken och läste in den på nytt. På så vis slapp vi efterhängande mappar och filer i grammatiken som inte blev ersatta med nya regler.

Reglerna för grammatiken var än så länge primitiv. Vi lade in mapparna och filerna som de var som regler direkt in i grammatiken, det var enkelt men inte särskilt kraftfullt. När vi utforskade de andra verktygen användes XML-notation, vilket är det standardiserade sättet att uttrycka en grammatik. Det nya målet att använda XML kände vi var betydande för projektet och insikten i hur saker och ting fungerar.

Vi fick gräva i API :erna för att hitta vad vi ville. Det gick snabbt att hitta att Speech API stödde XML, men tyvärr bara i 5.3. Detta var inget som vi lagt möda på att undersöka tidigare. 5.3 har väldigt bra stöd för den standardiserade (W3C standard) XML-notationen SRGS (Speech Recognition Grammar Specification)[8]. Vi behövde detta i 5.1. I den versionen hittade vi en primitiv variant av SRGS, väldigt likt fast med andra taggar. Som kan ses i källkoden (Bilaga 1) så är det denna taggning som använts. Då vår grammatik inte var i behov av att vara så avancerad är den heller inte så svår att förstå.

Exempelgrammatik:

```
<GRAMMAR LANGID="409">
  <RULE NAME="commands" TOPLEVEL="ACTIVE">
      <P>return</P>
  </RULE>
  <RULE NAME="toplevel" TOPLEVEL="ACTIVE">
      <O>open</O>
      <L>
          <P>programs</P>
          <P>windows</P>
      </L>
  </RULE>
</GRAMMAR>
```

Reglerna grupperas och man specificerar varje regel för sig. ”return” är en regel för sig oberoende av vilken mapp man befinner sig i. Den andra regeln beskriver sedan hur det är valfritt att säga ”open” innan man säger något mappnamn.

Vi förändrade programmet till att skriva våra regler till en XML-fil med notationen här ovan istället för att

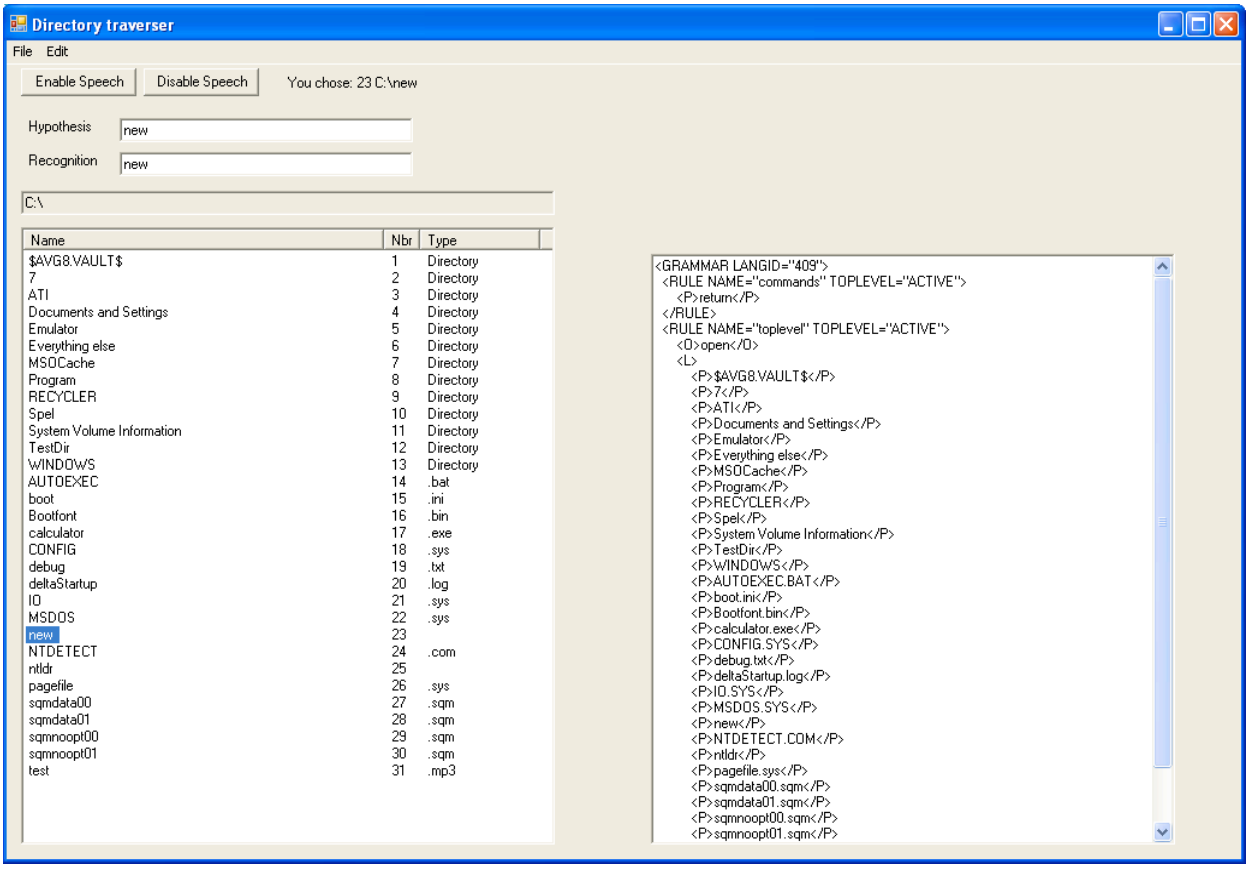

Illustration 1: Exempel på en typisk körning

lägga in de direkt in i grammatiken. Sen istället för att läsa in alla regler var för sig läser vi enbart in XML-filen. Resten av programmet kunde vara oförändrat.

2.1 Resultat

Om vi upprepar målen vi haft med projektet:

- Mapptraverserare som kan exekvera filer
- Lokalt körbar utan internetuppkoppling
- Körbart på XP och möjligtvis Vista
- Dynamiskt genererad XML-grammatik
- Index för att öppna svåruttalade namn eller duplicerade filnamn

Dessa mål lyckades vi uppfylla till stora delar, resultatet blev väldigt likt våra första tankar (Illustration 1). Vad som inte blev implementerat var exekveringen av filer samt indexering. Det förstnämnda fallerade på grund av tidsbrist. Det sistnämnda fallerade eftersom igenkänningen inte kunde översätta uttal till siffror, detta hade vi i så fall fått hårdkoda in och tiden brast för detta.
För att erfara resultaten bör man exekvera programmet och prata med det (Källkoden finns i Bilaga 1). Vad som ställer till det är oftast att datorn inte är tränad till ens röst, vilket vi kommer närmare inpå i nästa avsnitt.

Förutom den uppenbara svårigheten med filnamn som är helt omöjliga att uttala så finns svårigheter så som när man har mappar och filer med samma namn, eller flera filer med samma namn men olika ändelser. Man är inte tvungen att säga ändelsen på filen men om man vill specificera fil kan man göra det för att få de resultat man vill. Båda dessa problem hade förenklats om indexeringen fungerat.

Vi har valt att göra det frivilligt att säga ”open” före man vill öppna något. Detta för det oftast är intuitivt att bara säga namnet, men i visa speciella fall kan man vara tvungen att säga ”open” för att kunna komma vidare. I vårt fall är backa kommandot ”return”, har man en mapp som heter ”return” måste man säga ”open” innan för att kunna öppna den.

Illustration 2 visar flödet vid en typisk körning. Det kan tilläggas att om taligenkänningen inte hittar någon hypotes som matchar någon regel så kommer ingenting att hända, det blir en timeout efter ca en sekund.

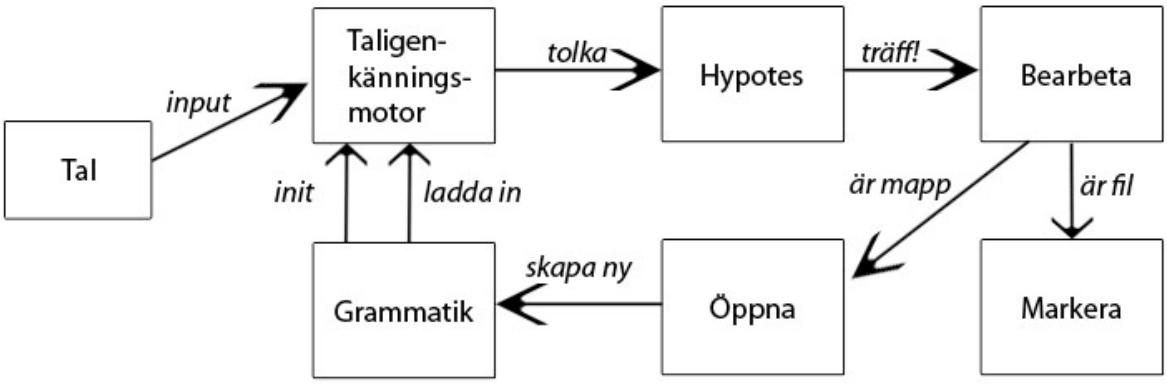

Illustration 2: Flödesschema för taligenkänning

2.2 Svårigheter

Det svåra med att använda taligenkänning i Windowsmiljö (framförallt Windows XP) är att hitta användbar information om hur man skall gå till väga. Större delen av projektet har handlat om att försöka hitta väsentlig information vilket närmast kan beskrivas som att utforska en djungel. Anledningen till bristen på information är delvis att taligenkänning till Windows XP anses vara något förlegat då Windows Vista har taligenkänningsstöd inbyggt redan från grunden. Windows XP kräver att man installerar Speech API Speech Software Development Kit 5.1 (SAPI 5.1 SDK) medan Windows Vista innehåller SAPI 5.3. En av skillnaderna mellan dessa SAPI är stödet för XML-grammatik. SAPI 5.3 har stöd för Speech Recognition Grammar Specification (SRGS) samt den nyare Speech Synthesis Markup Language (SSML) som båda är specifikationer som ingår i W3Cs ramverk för talstyrning (en tredje och minst lika aktuell specifikation som ingår i W3Cs ramverk är VoiceXML). SAPI 5.1 har stöd för en mer primitiv variant av SRGS-grammatiken, vilket vi fick använda oss av. I och med att SAPI 5.1 anses vara föråldrad så är dokumentationen klart mer begränsad än till SAPI 5.3.

Den grammatik vi använde oss av hade ingen direkt igenkänning av siffror. Talar man in till exempel ”seven” så tolkas det som ordet och inte som siffran 7. Om man vill att det skall tolkas som själva siffran så får man hantera detta i programkoden.

När man gör talstyrda program i Windows XP är det bra ifall man först tränar upp taligenkänningen. Detta innebär att man läser upp texter så att akustiken, ljudpitchen och fonemidentifiering kan justeras. På sätt kan man reducera antalet missförstånd som kan uppstå. Under projektets gång märkte vi vikten av en upptränad taligenkänning. Stundtals verkade det som att en otränad Speech Engine inte förstod ett dyft av vad man hade sagt. Genom att pröva taligenkänning i programmet Dictation Pad som så kan man själv bedöma när träningen och kalibreringen verkar vara komplett. Dictation Pad är ett program som skriver ned det som den hör en säga (det medföljer SAPI 5.1 SDK).

2.3 Vidareutveckling

Här följer en rad olika vidareutvecklingar som skulle vara möjliga:

- Exekvera filer. Möjligheten att kunna köra filer med röstkommando.
- Sökning av filer och mappar.
- Indexering av filer. Detta skulle kunna vara användbart då vissa fil- eller mappnamn är svåra att uttala. Det skulle då helt enkel räcka med att säga indexnumret.
- Text-to-Speech svar (TTS) vilket medför att programmet kan svara på frågor man ställer såsom i vilken mapp en viss fil befinner sig.
- Hantering av filer med samma namn men med olika filändelser. Man skulle till exempel här kunna använda sig av indexering. Om två filer har samma namn så ber programmet en tala det indexnummer filen har som man menar.
- Konfirmation, att programmet ber konfirmera det val man gjort.

3 Taligenkänning idag

3.1 Effektivitet och tillgänglighet

De flesta plattformar har bara stöd för de allra största språken, såsom engelska, franska, tyska, spanska, japanska och kinesiska, samt dialekter därav.

För att datorn med något mått av säkerhet skall kunna identifiera vad det är man säger, så måste den tränas. Detta innebär att den intenderade användaren får läsa ett antal texter upprepade gånger, så att pro-

grammet kan lära sig hur användaren uttalar olika ord eller fonem. Även om programmet då blir ”personligt” tränat, så kommer det fungera avsevärt bättre för användare med liknande dialekter och uttal.

Från och med Windows Vista är taligenkänning en integrerad del av Windows-operativsystemen och kan där användas för att ersätta eller komplementera både mus och tangentbord.

3.2 Användningsområden

Taligenkänning är applicerbart på många områden där behovet finns av att ha ett snabbare eller mer effektivt interface än mus och/eller tangentbord. Man måste dock försäkra sig om att inga kritiska moment enbart utförs m.h.a. taligenkänning, då det alltid finns en chans att programmet inte förstår vad man försöker säga.

Ett exempel på taligenkänning som många människor har kontakt med i sin vardag är den formen av dialogsystem som finns hos många telefonbaserade tjänster, t.ex. SJs biljettbokningstjänst., ett annat är alla de nya mobilapplikationer som går att styra med rösten.

Det de flesta framgångsrika applikationer har gemensamt är att de har jämförelsevis små och icke-ambiguösa grammatiker.

4 Slutsats

Programmet är en god demonstration av vad som man lätt kan åstadkomma. Trots bristen på användbar dokumentation för SAPI 5.1 så uppfattades taligenkänning i Windows XP som stabilt och lättanvänt. Man blir dock motarbetad att arbeta med lokala applikationer, marknaden är intresserad av telefonväxlar och mobilapplikationer som kräver server-klientlösningar. Att döma av vårt resulterade program är det fullt möjligt att skapa lokala applikationer som fungerar. En sak är säker, intresset för taligenkänning är stort och på framgång. Att det är under pågående utveckling av både Microsoft och AT&T:s sida inom område torde vara bevis för det.

5 Referenser

1 http://jvoicexml.sourceforge.net/

2 http://www.research.att.com/viewProject.cfm?prjID=355

3 https://studio.tellme.com/

4 http://www.voicexml.org/

5 http://www.microsoft.com/speech/speech2007/default .mspx

6 http://msdn.microsoft.com/sv-se/vs2008/products/default.aspx

7 http://www.c-sharpcorner.com/uploadfile/ssrinivas/speeechrecognitionusingcsharp11222005054918am/speeechrecognitionusingcsharp.aspx

8 http://www.w3.org/TR/speech-grammar/