# En utvärdering av Audiolet för portandet av ett röst-drivet program till webben

DD134X Examensarbete inom datalogi, grundnivå
CSC/Tal, Musik och Hörsel
Kungliga Tekniska Högskolan

Johan Fogelström
Roslagstullsbacken 5
114 21 Stockholm
+46761345645
johfog@kth.se

Max Nordlund
Maltesholmsvägen 67
165 55 Hässelby
+46734255178
maxno@kth.se

12 april 2013

**Abstract**

We where asked to write a webapplication, with similar functionality as an existing desktop application. The application can today record sounds from a person, mix it together at the optimal place with another, similar sound and play back the result.

The application used mathematical calculation that the developers where not familiar with, therefore it was decided to focus on the parts that involved recording and playing sound, to pave the way for any future implementation.

The follwing prototypes where implemneted:

1. Read a sound clip from a file.
2. Splicing of two sound clips at different points in the clips.
3. Generate the sound of a guitar.

The result of this evaluation is that web browsers today are not ready for this type of application, but that there is potetial as soon as W3C have standardised the use of a microphone as a source to the existing DSP functions.

**Sammanfattning**

Vi blev ombedda att skriva en webbapplikation, som har liknande funktionalitet som en befintlig applikation på skrivbordet. Applikationen kan idag läsa in ljud från en person, klippa ihop det ljudet på det optimala stället med ett annat, likartat ljud och spela upp resultatet.

Applikationen gjorde matematiska beräkningar som utvecklarna inte är särskilt familjära med, därför sattes fokus på de delarna som involverar inspelning och uppspelning av ljud, för att bana väg för en framtida implementation.

Det implementerades följande prototyper:

1. Läsa in ett ljudklipp från fil.
2. Skarva ihop två ljudklipp på olika punkter i klippen.
3. Generera ljudet av en gitarr sträng.

Resultatet av vår undersökning är att webbläsare och Audiolet ännu inte riktigt är redo för den här typen av applikation, men att det finns potential och webbläsaren är redo när W3C[17] har standardiserat användandet av mikrofon som källa i de befintliga DSP funktionerna.

# Innehåll

# 1 Introduktion

Den här rapporten är resultatet av ett kandidatexamensarbete vid Kungliga Tekniska Högskolan. En anställd vid TMH har en befintlig applikation, skriven i Tcl/Tk, för att hjälpa barn med talsvårigheter att övervinna sitt problem. Problemen som programmet kan hjälpa med är av typen där barnet uttalar första bokstaven fel.

När barnet trycker på en knapp i användargränsnittet för att välja ett ord att uttala. Därefter trycker barnet på en annan knapp för att aktivera mikrofonen kopplad till datorn. Barnet talar in i mikrofonen som lagrar ljudet i minnet. Efter att programmet gjort en serie beräkningar på ljudet spelar det upp det korrekta uttalet som om barnet själv hade sagt det.

Detta sker genom att under beräkningsfasen hitta var första vokalen är i det inspelade ljudet och hur denna vokal låter. Denna information används för att slå upp vilket ljud som är mest lik barnets i en databas av förinspelade ljud. Det förinspelade ljudet och barnets inspelning sätts ihop på det ställe där de låter som mest likt varandra, vilket ger illusionen av att barnet själv sa ordet korrekt.

Denna rapport undersöker möjligheten att porta programmet till webben för att nå ut till det största antalet barn som möjligt. Frågeställningen som utgicks ifrån är "Vilket är det bästa sättet att på ett standardiserat och framtidssäkert sätt att arbeta med ljud i webbläsare?"

## 1.1 Begrepp

**XML** *eXtensible Markup Language*
: Språk för att på ett systematiskt sätt lagra data i en struktur av taggar.

**HTML** *HyperText Markup Language*
: XML liknande språk för att beskriva information och struktur av en webbsida.

**CSS** *Cascading Style Sheets*
: Språk som används för att beskriva hur en webbsida ska visas.

**Tcl/Tk** Programmeringsspråk med tillhörande verktygslåda
: Erbjuder ett sätt att bygga grafiska gränssnitt.

**CSC** *School of Computer Science and Communication* Skolan för datavetenskap och kommunikation på KTH

**TMH** *Tal, Musik och Hörsel*
: Avdelning vid CSC skolan på KTH

**DSP** *Digital Signal Processing*
Samlingsnamn för att modifiera signaler och ljud i datorer.

# 2 Bakgrund

## 2.1 Befintlig Applikation

Den befintliga applikationen, är skriven i Tcl/Tk, ett skriptspråk utvecklat för att snabbt och enkelt att programmera grafiska gränsnitt i. Språket kapslar in enkla aritmetiska uttryck som parametrar till en *expr* funktion som beräknar uttrycket. Detta var något som ingen av utvecklarna hade tidigare erfarenheter av.

### 2.1.1 Beroenden

Koden har ett grundläggande beroende till Snack, ett ljudbibliotek utvecklat i C av TMH. För att bilda oss en uppfattning av vad det används till egentligen, och hur mycket av det som används, undersöktes de refererade delarnas källkod[1]. Det var en begränsad del av biblioteket som faktiskt användes. Det verkade som att det skulle vara enkelt att lyfta ut ur biblioteket och därmed bli av med alla beroenden. Dock är de delarna mestadels programmerad på franska.

### 2.1.2 Algoritmer

Programmet använder olika sorters avståndsberäkningar för att räkna avstånd mellan ljudfiler, primärt euklidiskt, Mahalanobiskt avstånd och Mel-frequency cepstral koefficienter. Det euklidiska avståndet är enkelt att implementera, men det Mahalanobiska avståndet kräver mer ingående kunskap om det förväntade ljuden eftersom den väger in variansen i datan. Det är fortfarande genomförbart att implementera det, eftersom den nödvändiga informationen redan finns beräknad.

**MFCC** Mel-frequency cepstral koefficienterna, eller MFCC, är koefficienterna i en representation av kraftspektrumet för ett ljudklipp. MFCC transformera ljudet med en Fourier transform, konverterar resultatet till Mel-skalan, en logaritmisk skala och transformerar slutgiltigen med en diskret cosinus transform. Resultatet är koefficienterna av Mel-frequency ceptral spektrumet som beskriver en ton- och uttalslös representation av ljudet som är användbar i röstigenkänning.[2].

### 2.1.3 Grafiskt gränsnitt

Gränsnittet mot användarna (barnen) är gjort i Tcl/Tks inbyggda GUI bibliotek.

## 2.2 Webbteknologier för ljud

Det finns två fundamentalt olika sätt att spela upp ljud i webbläsare. De som använder insticksmoduler, och de som inte gör det. Insticksmoduler är på väg att fasas ut men har fortfarande en plats för sin stora användarbas och bakåtkompatibilitet med gamla webbläsarversioner.

### 2.2.1 Plugin-baserade

**Silverlight** Silverlight är en plugin utvecklad av Microsoft som ett alternativ till Java-applets. Silverlight har bra stöd för mediauppspelning. Men eftersom den är utvecklad av Microsoft har den dåligt stöd på alternativa operativsystem som Macintosh och Linux[4].

**Java** Java är en språk utvecklat av Sun Microsystems och ägs idag av Oracle. Java är känd för sitt motto "Compile once, run everywhere" och används i en stor mängd hårdvara och mjukvara i olika former. Java i webbläsare arbetar med så kallade "Applets" som är ett vanligt java program, men som inte kör *main* först utan *init* och *start*. En applet kan rita ut sitt gränsnitt i webbläsaren genom de inkluderade AWT och Swing biblioteken för att göra grafiska gränsnitt.

**Flash** Flash är en plattform för uppspelning av animeringar från Adobe. Flash har en stor installationsbas, delvis för att den här funnits på marknaden sedan mitten av 1990-talet och därför haft god tid på sig att ackumulera och delvis för att Googles webbläsare Chrome kommer med en inbyggd installation[6].

### 2.2.2 Web Audio

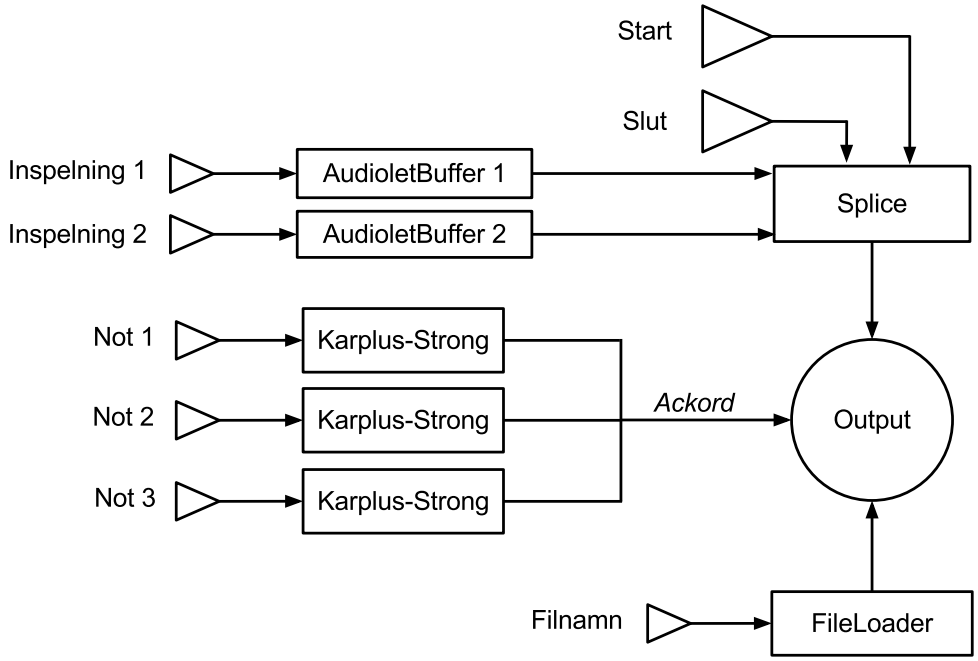
Den rena JavaScript lösningen baserad på en standard från W3C[7]. Web Audio API är ett mängd av högnivå JavaScript funktioner för att modifiera och spela upp ljud i webbläsare. Eftersom det är en standard kommer samtliga webbläsare att med tiden stödja den här metoden. Arbetssättet är att skapa olika noder som sedan länkas samman, för att till slut hamna i en ljuduppspelnings nod som API:t tillhandahåller.

### 2.2.3 Native Client

Googles system för att köra kompilerad kod i webbläsaren[3]. Fungerar i dagsläget bara i Google Chrome och är idag begränsade till distibution genom Chrome Web Store. Firefox uttalade sig 2010 att de inte tänker stödja tekniken, eftersom de vill främja lösningar som inte använder kompilerad kod[6]. Erbjuder inget eget sätt att få tillgång till mikrofon, utan är beroende av andra lösningar för det.

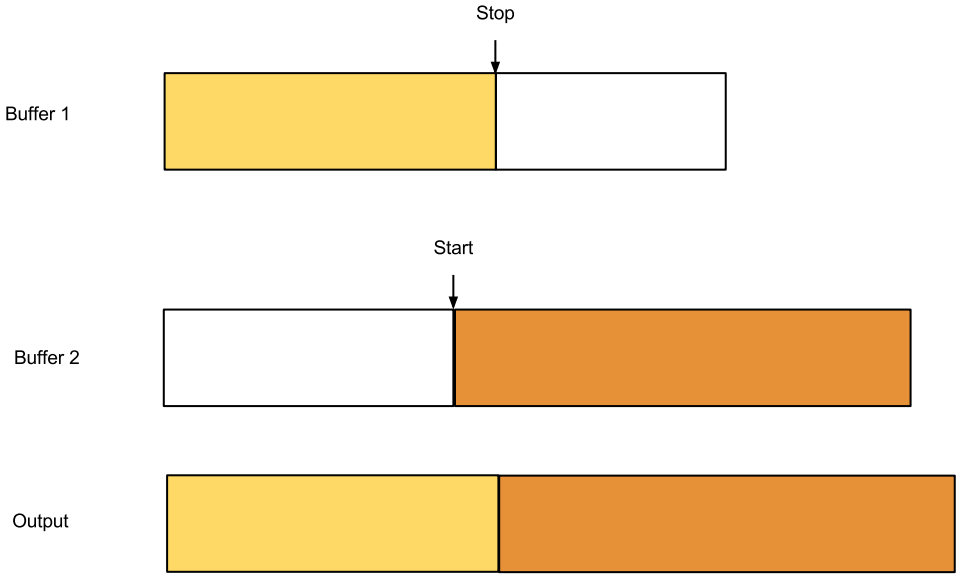
# 3 Metodik

Denna rapport illusterar hur en webbapplikation kan processera ljud och sedan spela upp resultatet genom implementation av ett antal enkla prototyper. För detta används biblioteket Audiolet[13] för att processera ljud. Det är ett ramverk för att använda ljud i webbapplikationer. Den tillåter att man enkelt och tydligt kan sätta ihop delkomponenter för att processa ljud i en dataflödesgraf. Den erbjuder mer funktionalitet än en ren Web Audio API lösning samt är anpassad att fungera i olika implementationer av Web Audio API.




Figur 1: Överblick över synten

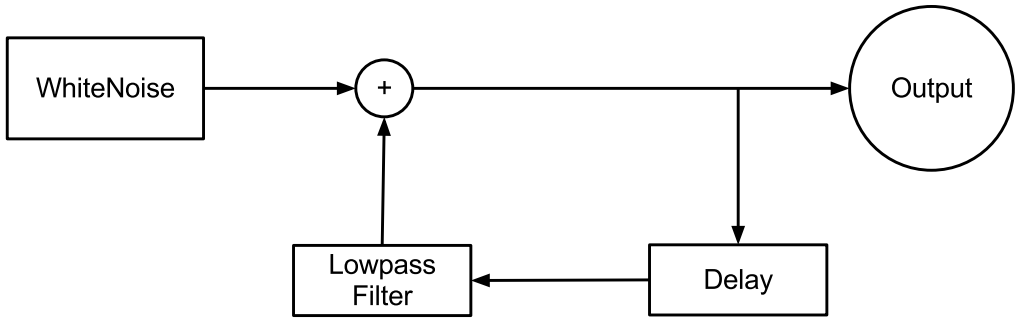
För att stresstesta prestandan på JavaScript-motorn i en modern webbläsare byggs en enkel synth som emulerar sex gitarrsträngar. En liten del av programmet portas också till webben, närmare bestämt ihopskarvningen av två ljudklipp laddade från fil för att utvärdera Audiolet som ljudbibliotek.




Figur 2: Ihopskarvning av två ljudklipp

Ihopskarvning väljs för att genom att skarva ihop två ljudklipp vid rätt tidpunkter kan Tcl/Tk programmet idag få det att låta som om barnet sa ordet korrekt. Detta gör att när barnet med talfel säger "tung" kan programmet sätta ihop barnets "ung" med det förinspelade "ku" från ett annat barn så att det låter som att barnet med talfel själv sa "kung". Utvecklarena har inte kunskap eller tid nog att implementera den matematiska delen av programmet, men har implementerat den komponenten[2] som sätter ihop ljudklippen.

Karplus-Strong[3] är en enkel algoritm för att syntesera ljudet av att knäppa en sträng, som till exempel på en gitarr.




Figur 3: Karplus-Strong algoritm[14] för gitarr syntetisering

# 4 Resultat

För att snabba upp utvecklingen av prototyperna, gjordes valet att inte utveckla direkt i JavaScript, utan istället i CoffeeScript, som sedan kompileras till JavaScript.

## 4.1 Gitarren

Audiolets lämplighet och prestandan hos moderna webbläsares JavaScript-motorer utvärderades genom att implementera Karplus-Strongs algoritm. Algoritmen ska gå att implementera i en miljö som funktionellt påminner om Audiolets miljö[15]. Dock blev resultatet inte som förväntat.

Färdiga moduler försöktes alltså dras nytta av men Audiolets struktur tycks inte tillåta cykler i dataflödesgrafen vilken är en nödvändighet för Karplus-Strongs algoritm. Därför implementerades algoritmen med en egen Audiolet nod.

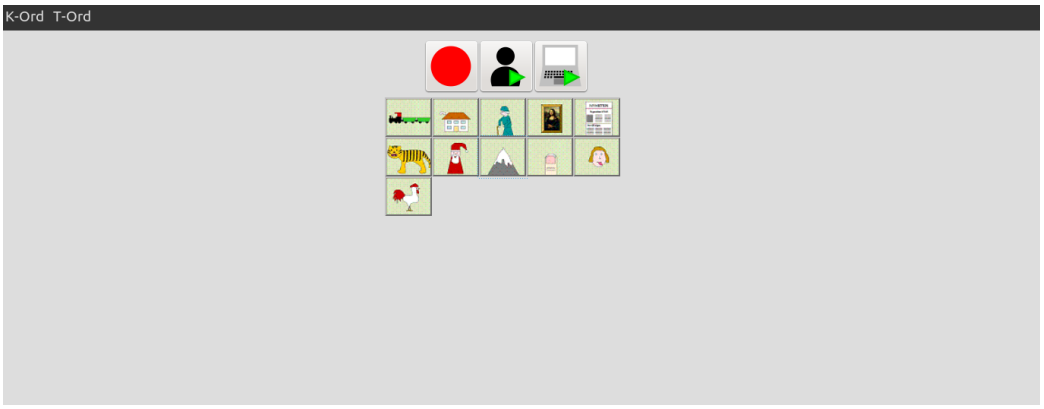
## 4.2 Ihopskarvningen

Givet två buffrar med ljuddata och två tidpunkter, stopp och start, spelar algoritmen upp ljudet från den första bufferten fram till stopptidpunkten och därefter ljudet från den andra bufferten från starttidpunkten och framåt. Detta görs genom att kopiera det relevanta ljuddatan från de två buffrarna till en ny, gemensam buffer som sedan spelas upp.

Detta implementerades som en egen Audiolet nod vilket medför att den går enkelt att återanvända.

### 4.3 Gränsnittet

Efter en undersökning om hur det grafiska gränsnittet skulle kunna byggas om valdes en design inspirerad av den befintliga. Detta gränsnitt är byggt i Jade och Stylus, som erbjuder en alternativ syntax till HTML5 och CSS3, vilket de också kompileras till. HTML5 och CSS är stabila webbstandarder[17]. Detta medför att gränsnittet fungerar idag och i framtiden.




Figur 4: Skärmdump över användargränsnittet

En del av de ikoner som följde med programmet har byts ut mot liknande ikoner i Scalable Vector Graphics format (SVG)[16]. Dessa tar mindre plats och skalar godtyckligt stort. De har även en mer semantisk betydelse, vilket kan göra programmet mer lättanpassat till barn med speciella behov. De andra bilderna kan konverteras till samma format, men bedömdes utanför den här rapportens räckvidd.

## 5 Diskussion

Eftersom denna rapport har för avsikt få programmet att nå ut till en större publik, måste lösningen för ljudbearbetning ha ett brett stöd i dagens och morgondagens webbläsare.

Utvecklingen av Silverlight tycks ha stannat av, med version 5 som släpptes 2011 med en livslängd långt längre än tidigare versioner, därför är det troligt att Microsoft ha beslutat sig för att Silverlight är färdigutvecklat[8].

Javas insticksmodul har haft problem med säkerheten på sista tiden och därför en oviss framtid framför sig i webbläsarna.[5] Adobe har erkänt att Flash kommer att fasas ut från webben inom den närmsta tiden[9]. Men Flash är fortfarande en möjlighet ur kompatibilitetssyfte, och det finns exempel på bibliotek som faller tillbaka på Flash om det behövs[12].

Av de anledningarna har de pluginbaserade lösningarna en tveksam framtidsutsikt i webbläsarna. Därför valdes den typen av lösningar bort, i fördel för en av W3C standardiserad lösning i form av Web Audio API.

Tyvärr är Web Audio API är ännu inte implementerat i alla moderna webbläsare, och framförallt inte på exakt samma sätt. Med tiden kommer detta problem att försvinna och genom att använda sig av ett bibliotek[13] som slätar över skillnaderna märks det mindre av nu. Prestandan för JavaScript i moderna webbläsare tycks baserat på våra tester vara fullt tillräcklig för detta program[10], men kan orsaka problem i gamla webbläsare. Dessutom kan det behövas extra arbete för kompatibilitet med det standardiserade API:t.

Det begränsade stödet för Native Client betyder att även om det öppnas upp för att inte behöva levereras över Chrome Web Store, så är det begränsat till Google Chrome, och därför olämpligt för det här projektet. Tekniken hade varit bra, eftersom den tillåtit oss att lyfta ut kodblock från Snack, och därför snabbat upp utvecklingen en aning, men kostnaden av att vara begränsad till en specifik webbläsare är för hög.

Efter att ha tittat på källkoden för den befintliga applikationen, drogs slutsatsen att den inte är värd att återanvända i sin nuvarande form i skapandet av en webbapplikation. Den är skriven på ett sätt som gör att det mesta måste skrivas om i alla fall för att passa webben.

Ett alternativ till att porta allt till JavaScript vore att genomföra alla beräkningar på en server version av original programmet. Det bedömdes dock att transporten av ljud fram och tillbaka mellan klienterna och servern, skulle introducera för höga krav på bandbredd. Dessutom var det valda språket, Tcl/Tk långt i från optimalt för skrivandet av den server som skulle behövas, vilket betyder att koden ändå skulle behövs skrivas om i ett annat språk för att lättare kunna underhålla servern i framtiden.

Även om MFCC beräkningen är simpel att beskriva, involverar den två matematiskt tunga transformer, varav en tycks inte finnas färdigimplementerad i JavaScript. Utvecklarna kom fram till att de hade varken tiden eller kapaciteten att implementera den delen, men dem är direkt nödvändig för att applikationen skall fungera korrekt. Därför måste en framtida implementation reda ut den beräkningen, eller hitta ett annat, likvärdigt sätt att bedöma två inspelningar.

Implementationen av splice kommer av hur originalprogrammet fungerar. Genom att räkna ut de optimala skärningspunkterna låter det som om barnet sa ordet korrekt när det i själva verket var ett annat barn som sa den första konsonanten och det första barnets inspelning av resten av ordet ihopklistrat. Alltså sätter den ovan nämnda algoritmen ihop de två ljuden utan att försöka släta över skillnader eller göra en mjuk övergång[19]. Istället är det upp till användaren, som i framtiden kan vara samma algoritm som hittar de optimala

skärningspunkterna, att välja var de två ljudklippen ska sättas ihop.

Det nya gränsnittet innehåller inte nödvändigtvis all funktionalitet från originalet, mestadels för att designen gjordes ur minne från den demonstration som gjordes av original utvecklaren i början av projektet, tillsammans med den enda gången som programmet demonstrerats. Valet att göra det i HTML/CSS är naturligt, eftersom det skulle köras i en webbläsare, vilket betyder HTML/CSS, eller att använda rendreringskraften hos en insticksmodul, så som Java eller Flash. Men, det hade introducerat ytterliga beroende, och betytt ett medvetet val att undvika etablerade webbstandarder.

# 6 Slutsats

Webbläsare idag är på gränsen till att vara redo för den här typen av applikation. W3C har nyligen standardiserat metoden att få tillgång till mikrofonen i Web Audio API, men stödet finns idag bara i de senaste versionerna av de moderna webbläsarna.

Det eventuella problemet med beräkningshastighet i JavaScript för avancerad ljudprocessering visades inte vara ett problem. Vid test av Karplus-Strongs alla spelades ljudet upp i med en bitrate av 32 bitar i 44 kHz. Vid uppspelning av ett ackord ger detta $6 \cdot 44\,000 \approx 1,3$ miljoner funktionsanrop till Karplus-Strongs algoritm per sekund, som kallar på ytterligare funktioner. Detta gjordes utan problem.

"Vilket är det bästa sättet att på ett standardiserat och framtidssäkert sätt att arbeta med ljud i webbläsare?" Frågan som ställdes i början kan därför idag besvaras som att det idag är möjligt att med Web Audio API skriva en webbapplikation som arbetar med ljud, på liknande förutsättningar som en lokal applikation.

Problemet är att det finns inga alternativ, vilket är både bra och dåligt. Det betyder att det finns ett monopol på hur gränsnittet ska se ut och vad det kan göra, men eftersom allt måste gå genom en webbläsare på något sätt, så betyder ett gemensamt gränsnitt att alla webbläsare har samma möjlighet att köra eventuella applikationer som använder det. Under den tid som rapporten skrevs har situationen förändras från en situation som krävt extra installation av tredjepartsprogramvara till en ny situation där endast bristande kompatibilitet mellan webbläsare och plattformar har hindrat en fullständig portning.

För att fortsätta arbetet att porta applikationen till webben, föreslås det att man tittar på portandet av Snack biblioteket skrivet i C till JavaScript, samt tittar på hur samlingen av ljudklipp som samlats in kan komprimeras och organiseras om.

# 7 Uttalande om samarbete

I den här rapporten har Max Nordlund skrivit majoriteten av koden, formaliserat rapporten samt korrekturläst. Johan Fogelström har skrivit majoriteten av rapporten, gett kommentarer på koden och letat källor.

# Referenser

[1] Snack Source Code
Avdelningen för tal, musik och hörsel (TMH).
http://www.speech.kth.se/snack/dist/snack2.2.10.tar.gz Hämtad: 2013-02-16

[2] The MFCC
http://www.cic.unb.br/~lamar/te073/Aulas/mfcc.pdf
Hämtad: 2013-04-12

[3] Google Developers NaCl site
https://developers.google.com/native-client/

[4] Microsoft Silverlight
http://www.microsoft.com/silverlight/

[5] Java Vulnerability
http://www.informationweek.com/security/application-security/java-vulnerability-affects-1-billion-plu/240007985

[6] DZone - Mozilla Rejects Native Code Approach of Chrome's NaCl
http://css.dzone.com/articles/mozilla-rejects-native-code
Hämtad: 2013-03-05

[7] W3C - Web Audio API
https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html
Hämtad: 2013-03-05

[8] Microsoft Support Lifecycle - Silverlight
http://support.microsoft.com/lifecycle/default.aspx?LN=sv&x=16&y=15&c2=12905
Hämtad: 2013-03-05

[9] Adobe Bows in Apple Feud
http://online.wsj.com/article/SB10001424052970204358004577027923205009352.html
Hämtad: 2013-04-11

[10] Is JavaScript Faster Than C?
http://onlinevillage.blogspot.se/2011/03/is-JavaScript-is-faster-than-c.html
Hämtad: 2013-03-05

[11] The Computer Language Benchmarks Game - JavaScript V8 vs. Java 7
http://benchmarksgame.alioth.debian.org/u64/benchmark.php?test=all&lang=v8&lang2=java
Hämtad: 2013-03-05

[12] A cross browser JavaScript library to bring native audio sample output to the underlying OS's audio system directly from JavaScript.
https://github.com/grantgalitz/XAudioJS
Hämtad: 2013-03-05

[13] A JavaScript library for real-time audio synthesis and composition from within the browser
http://oampo.github.com/Audiolet/
Hämtad: 2013-03-07

[14] Physical Modelling Synthesis
*Nicky Hind*
https://ccrma.stanford.edu/software/clm/compmus/clm-tutorials/pm.html
Hämtad: 2013-03-07

[15] Physical Modeling
http://www.cim.mcgill.ca/~clark/nordmodularbook/nm_physical.html
Hämtad: 2013-04-11

[16] SVG
http://www.w3.org/TR/SVG/
Hämtad: 2013-04-08

[17] W3C
World Wide Web Consortium
http://www.w3.org/
Hämtad: 2013-04-10

[18] Capturing Audio & Video in HTML5
http:\www.html5rocks.com/en/tutorials/getusermedia/intro

Senast updaterad: 2012-06-20
Hämtad: 2013-04-11

[19] Get Started with Web Audio API
`http:\www.html5rocks.com/en/tutorials/webaudio/intro`
Publicerad: 2011-10-14
Hämtad: 2013-04-11

[20] Chmod Code Repository
`https://github.com/maxnordlund/chmod`
Hämtad: 2013-04-12

# Figurer

# A User Requirements Document

Syftet med applikationen är att hjälpa barn med talsvårigheter att uttala orden korrekt

## A.1 Tillgänglighet

Applikationen ska nå ut till så många som möjligt med så låga systemkrav som det går, därför ska applikationen vara en webbapplikation.

## A.2 Låg Bandbredd

Som en fortsättning på tillgänglighetskravet, så skall applikationen inte kräva mycket bandbredd för att fungera, eftersom applikationen skall kunna köras på platser med lägre uppkopplingskapacitet.

## A.3 Använder aktuella standarder

Applikationen ska helst inte kräva installation av ytterligare komponenter av användaren, så som insticksmoduler till webbläsaren.

## A.4 Snabb respons

Användaren skall inte behöva vänta mer en ett fåtal sekunder från det att inspelningen, tills dess att resultatet spelas upp.

# B Kod

## B.1 Karplus-Strong

```
class KarplusStrong extends AudioletNode
  constructor: (audiolet, frequency) ->
    super audiolet, 1, 1
    @linkNumberOfOutputChannels 0, 0
    @period = Math.floor @audiolet.device.sampleRate /
        frequency
    @buffers = []
    @firstRun = true
    @index  = 0

  generate: (inputBuffers, outputBuffers) ->
    input  = @inputs[0]
    output = @outputs[0]

    numberOfChannels = input.samples.length
    numberOfBuffers  = @buffers.length
    for i in [0...numberOfChannels]
      if i >= numberOfBuffers
        @buffers.push new Float32Array @period

      buffer = @buffers[i]
      if @firstRun
        output.samples[i] = input.samples[i]
      else
        if @index is 0
          prev = @period - 1
        else
          prev = @index - 1
        output.samples[i] = (buffer[@index] +
            buffer[prev]) / 2
      buffer[@index] = output.samples[i]

    @index++
    if @index >= @period
      @index = 0
      @firstRun = false

```

```
  toString: -> "Karplus-Strong"
37
class GitarrString extends AudioletGroup
39  constructor: (audiolet, frequency) ->
      super audiolet, 0, 1
41
      @noise = new WhiteNoise @audiolet
43    @gain = new Gain @audiolet
      @envelope = new PercussiveEnvelope @audiolet, 0,
         0.5, 3,
45      ( -> @audiolet.scheduler.addRelative 0,
           @remove.bind @ ).bind @

47    @main  = new KarplusStrong @audiolet, frequency
      @noise.connect @main
49
      @main.connect @gain
51    @envelope.connect @gain, 0, 1
      @gain.connect @outputs[0]
53
class Range
55  base: [ "A", "A#", "B", "C", "C#", "D", "D#", "E",
      "F", "F#", "G", "G#" ]

57  constructor: (id, @isNote=true) ->
      @input  = $("##{id}_input")
59    @output = $("##{id}_output")
      @input.on "change", @change
61    @change null

63  change: (event) =>
      @value = parseFloat do @input.val
65    @output.text if @isNote then do @toString else
         @value
      return true
67
  get: ->
69    # See
         http://en.wikipedia.org/wiki/Piano_key_frequencies
      return 440 * Math.pow 2, (@value - 49) / 12
71
```

```
  set: (note) ->
73  normalize = (n) -> Math.ceil Math.floor(n / 4) /
      (n / 4)
    digit  = parseInt note.slice note.length - 1
75  letter = 1 + @base.indexOf note.slice 0,
      note.length - 1
    @input.val letter + 12 * (digit - normalize letter)
77  @change null
    return note
79
  toString: -> "#{@base[(@value - 1) %
    12]}#{Math.floor (@value + 8) / 12}"
81
$ ->
83  frequency = new Range "frequency", false
  # CM11 (major eleventh)
85  window.notes = "C4-E4-G4-B4-D5-F#5".split "-"
  for i in [0..5]
87  range = new Range "tune_#{i}"
    range.set notes[i]
89  notes[i] = range

91  $play = $("#play")
  $play.on "click", (event) ->
93  audiolet = new Audiolet
    frequencyPattern = new PSequence (do note.get for
      note in notes), 1
95  durationPattern  = new PChoose [0.1, 0.15, 0.2,
      0.25], 1
    audiolet.scheduler.play [frequencyPattern], 0.01,
      (tune) ->
97  string = new GitarrString audiolet, tune
      string.connect audiolet.output
99  return true
```

Kod/gitarr.coffee

## B.2 Ihopskarvning och filinläsning

```
class Range
  constructor: (id) ->
    @input  = $("##{id} input")
    @output = $("##{id} output")
    @input.on "change", @change
    @change null

  change: (event) =>
    @value = parseFloat do @input.val
    @output.text "#{Math.floor(@value / 60)}:#{(@value
      % 60).toFixed 2}"
    return true

  max: (value) -> @input.attr "max", value

class Switch extends AudioletNode
  constructor: (audiolet, time) ->
    super audiolet, 0, 1
    @linkNumberOfOutputChannels 0, 0
    @length = Math.floor audiolet.device.sampleRate *
      time
    @index  = 0

  generate: (inputBuffers, outputBuffers) ->
    output = @outputs[0]

    numberOfChannels = output.samples.length
    for i in [0..numberOfChannels - 1]
      output.samples[i] = @index / @length

    if @index < @length
      @index++

  toString: -> "Switch"

class Fade extends AudioletGroup
  FADE_TIME = 1.0

  constructor: (audiolet, first, second, time) ->
```

```
    super audiolet, 0, 1

    @first  = new BufferPlayer audiolet, first
    @second = new BufferPlayer audiolet, second
    @switch = new Switch audiolet, FADE_TIME
    @delay  = new Delay audiolet, time, time
    @cross  = new CrossFade audiolet, 0

    @switch.connect @delay
    @first.connect  @cross, 0, 0
    @second.connect @cross, 0, 1
    @delay.connect  @cross, 0, 2
    @cross.connect  @outputs[0]

class Splice extends AudioletGroup
  constructor: (audiolet, first, second, stop, start)
    ->
    super audiolet, 0, 1

    to   = Math.floor first.sampleRate * stop
    from = Math.floor second.sampleRate * start

    @buffer = new AudioletBuffer
      first.numberOfChannels, to + second.length -
      from
    @buffer.setSection first, to, 0, 0
    @buffer.setSection second, second.length - from,
      from, to
    @player = new BufferPlayer audiolet, @buffer,
      first.sampleRate / audiolet.device.sampleRate

    @player.connect @outputs[0]

$ ->
  stop  = new Range "stop"
  start = new Range "start"
  $play = $("#play")

  load = (path) ->
    deferred = new $.Deferred
    buffer   = new AudioletBuffer 1, 0
```

```
    request  = new AudioFileRequest
      "/audio/#{path}.wav", true
75  request.onSuccess = (decoded) ->
      buffer.length = decoded.length
77    buffer.numberOfChannels = decoded.channels.length
      buffer.unslicedChannels = decoded.channels
79    buffer.channels = decoded.channels
      buffer.channelOffset = 0
81    buffer.sampleRate = decoded.sampleRate
      deferred.resolve buffer
83
    request.onFailure = ->
85    deferred.reject "Could not load", path

87  do request.send
    return deferred
89
  $play.attr "disabled", "disabled"
91 $.when(load("karta"), load("kaka")).done (first,
     second) ->
    audiolet = new Audiolet
93  stop.max first.length / first.sampleRate
    start.max second.length / second.sampleRate
95  $play.removeAttr "disabled"
    $play.on "click", (event) ->
97    splice = new Splice audiolet, first, second,
        stop.value, start.value
      splice.connect audiolet.output
99    return true
```

Kod/splice.coffee

## B.3 Användargränsnitt

```
1  extends layout

3  append styles
     link(rel="stylesheet", href="/style/ui.css")
5
   block content
7    nav
       a(href="/ui#k") K–Ord
9      a(href="/ui#t") T–Ord

11   article
       section#controller
13       button
           #recordButton
15           svg(xmlns="http://www.w3.org/2000/svg")
               circle#record(cx="25", cy="25", r="22")
17
         button
19         #playButton
             svg(xmlns="http://www.w3.org/2000/svg")
21             circle#head(cx="25", cy="14", r="10")
               path#shoulders(d="M10,47 C0.5,18 49.5,18
                   40,47")
23             polygon.play(points="30,30 30,45 45,37")

25       button
           #compPlayButton
27           svg(xmlns="http://www.w3.org/2000/svg")
               - var x = 7
29             - var y = 3
               - var width = 50
31             - var height = 30
               - var keyboardHeight = width - height - y
33
               rect#screen(x="#{x}", y="#{y}",
                   width="#{width-2*x}",
                   height="#{height-2*y}")
35             polygon#plate(points="#{x-1},#{height-2*y}
                   #{width-x+1},#{height-2*y}
```

```
                #{width-x+5},#{height+keyboardHeight}␣
                #{x-5},#{height+keyboardHeight}")
            g
37          - for(var y = 30; y < 36; y += 3)
              - for(var x = 14; x < 34; x += 2)
39              polygon(transform="skewX(#{x-24})",points="#{x},#{y}␣
                  #{x+2},#{y}␣#{x+2},#{y+2}␣
                  #{x},#{y+2}")
            - for(var x = 14; x < 20; x += 2)
41              polygon(transform="skewX(#{x-24})",points="#{x},36␣
                  #{x+2},36␣#{x+2},38␣#{x},38")
            polygon(points="18,36␣30,36␣30,38␣18,38")
43          rect#pad(x="18", y="39", width="14",
                height="6")
            polygon.play(points="30,30␣30,45␣45,37")
45
    section#ordlist
47    for word in t_words
          button(style='background-image:␣
            url("/images/#{word.toLowerCase()}_thumb.GIF")')=
            word
```

Kod/ui.jade