Apputveckling för Roboten Alf

Ett arbete inom MäRI projektet

Kristoffer Kuvaja Adolfsson

Examensarbete
Informationsteknik
2022

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	8551
Författare:	Kristoffer Kuvaja Adolfsson
Arbetets namn:	Apputveckling för Roboten Alf
Handledare (Arcada):	Dennis Biström
Uppdragsgivare:	-

Sammandrag:

Projektet MäRI har för avsikt att ta fram evidensbaserad information om upplevelsen i mötet mellan en social, humanoid robot, vårdtagare och vårdstuderande. För att ta fram informationen måste applikationen Flossa utvecklas till Arcada roboten Alf, en kommersiell, social och humanoid robot från Sanbot vid namn Elf.
Information för apputveckling till sociala, humanoida robotar är däremot bristfällig och för att Flossa skall kunna ta fram information till MäRI projektet måste Alf och applikationen vara praktiskt användbara. Detta innebär att Alf och Flossa måste kunna kommunicera på svenska genom tal och taligenkänning.
Arbetet beskriver utvecklingsprocessen för Flossa och dess 4 programvaror. En serverimplementation, som nyttjar talsyntes- och taligenkänningstjänster från Google för att få roboten Alf att kommunicera på svenska. En webbapplikation, som blir Flossas användargränssnitt baserat på ett nod-träd och som möjliggör interaktionen mellan användare och robot. Samt en streamingserver och en Android applikation, byggd utifrån tillgängliga verktyg från Sanbot, som gemensamt ger roboten Alf liv och rörelser under interaktionen tillsammans med användargränssnittet.
De utvecklade programvarorna vidareutvecklas utifrån pilottester och feedback för att skapa den versionen som slutligen analyseras och användes av MäRI projektet. Android applikationen som utvecklas blir även utvärderad i jämförelse med den medföljande kommersiella applikationen från tillverkaren. Där lyfts brister och fördelar fram med den egna applikationen. Fördelar som möjligheten att utföra automatiserade rörelser på roboten och lyssna på tredjeparts kommandon genom den utvecklade servern.
Diskussion framför sedan förslag för vidareutveckling angående hur Alf och modifikationer av Flossa kan används i andra syften och hur författaren önskar se fortsatt forskning i taligenkänningsramverken Kaldi och Coqiu-ai för att ta fram språkmodeller som alternativ till Googles tjänster för att trygga den digitala utvecklingen av mindre språk som svenskan.

Nyckelord:	MäRI, robotar, SDK, SSE, taligenkänning, talsyntes, webbapplikation
Sidantal:	45
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	8551
Author:	Kristoffer Kuvaja Adolfsson
Title:	App development for the Robot Alf
Supervisor (Arcada):	Dennis Biström
Commissioned by:	-

Abstract:
Project MäRI intends to bring forth evidence-based information regarding the experience when a social, humanoid robot, a care recipient and nursing students meet. To collect this information an application, Flossa, needs to be developed for the Arcada robot Alf, a commercial robot from Sanbot called Elf.
Information for app development on social, humanoid robots is limited and to make sure Flossa can bring forth the appropriate information for the MäRI project Alf and the application must be serviceable. This in turns means that Alf and Flossa must be able to communicate in Swedish through speech synthesis and speech recognition
This thesis describes the development process for the application Flossa and its 4 software. A server implementation that uses speech synthesis and speech recognition services from Google to make the robot Alf communicate in Swedish. A web application that becomes Flossa's graphical user interface, based on a nod-tree that enables the interaction between user and robot. And finally, a streaming server and an Android application, developed from the included tools provided by Sanbot, that together brings life and movements to the robot during the interaction with the user and the graphical user interface.
The developed software is then improved upon based on user tests and feedback to create the version that is analyzed and eventually used by the MäRI project. The developed Android application is compared to the commercial application included from the manufacturer. The developed applications lack for face manipulation is brought forth along with its potential advantages, the ability to automate movements and the integration with a third-party stream like the developed server.
The discussion suggests future development regarding how Alf and a modified version of Flossa can be used for other purposes and how the author wish to see further research into the speech frameworks Kaldi and Coqui-ai to develop language models as alternatives to the used Google services and the secure the digital progress of minor languages like Finnish-Swedish.

Keywords:	MäRI, robots, SDK, SSE, speech recognition, speech synthesis, Web applications
Number of pages:	45
Language:	Swedish
Date of acceptance:	

INNEHÅLL

Figurer

FÖRORD

Tack till min familj för deras stöd, motivering och tålamod, för att de stått ut med mig under arbetsprocessen då jag varit lika responsiv som en robot,
Tack även till MäRI och Christa Tigerstedt, för att ha get mig chansen att ta del av ett projekt med god etik och socialt mervärde och för att hon alltid tagit emot mina idéer och värderat mina åsikter under utvecklingen av Flossa.
Det bör inte heller komma som någon super-hemlis att jag önskar visa en enorm tacksamhet till lektor Dennis Biström för att ha handlett mig och hållit ett gott kritiskt öga över mig under hela projekttiden.
Ett stort tack skall även utses till medlemmarna från MäRI projektet men extra erkännande bör ges till studiekamraten och kollegan Johan Penttinen som med sitt ihärdiga arbete hjälpt mig vidare på rätt spår när jag känt mig vilse och puffat mig över de största utmaningarna med sin kompetens under utvecklingen.

Ord

API - Application Programming Interface

AWS - Amazon Web Services

BLOB - Binary Large OBject

CSC – IT Center for Science

GUI - Graphical User Interface

ROS – Robot Operating System

SDK - Software Development kit

SSE – Server-Sent Events

STT – Speech To Text

TTS – Text To Speech

XML - eXtensible Markup Language

1 INTRODUKTION

Innan penseln uttryckte de första dragen för vad som skulle bli Mona Lisa i tidigt 1500-tal designade och, mot förmodan, konstruerade Leonardo da Vinci *Leonardos robot*, en automation som kunde stå, sitta och röra på armarna, med ett utseende som efterliknade en riddarrustning (Moran, 2007). Enligt historien skall Leonardos robot ha använts under en tillställning av, då da Vincis beskyddare, hertig *Ludovico Sforza*, för att underhålla gäster. Inspirationen som da Vinci redan hade på 1500-talet har sedan förföljt oss fem århundraden framåt, en fantasi om en humanoid följeslagare som kan samspela med oss, människan, i sociala sammanhang. En otröttlig kompanjon med mänskliga attribut för stöd i ärenden och interaktioner. Vid Yrkeshögskolan Arcada finns sådana följeslagare, sociala humanoida robotar som Alf och Snow, men även mer specialiserade robotar som Amy, en restaurangrobot, eller Lilla My, en robot utvecklad för data visualisering.

Om robotar finns redan forskning från läroanstalter i Finland som *The Future of Robotics* (Lius, 2021) eller *Designing service/care robot* (Isoaho, 2016), som framför att robotar kan utvecklas i syfte att stödja människan. Detta med samma princip som digitala verktyg redan gör idag, alltså förutspås robotens tillföra en likartad samhällsnytta som digitaliseringen i framtiden. Men för att använda robotar behövs korrekt programvara och inom forskning för programvara finner vi även resurser från Arcada med bland annat *Developing a cross-platform MVP app with React Native* (Nylund, 2020) eller längre tillbaka med *Utveckling av en digital dosettlåda* (Nyberg, 2016) men till skillnad från endast digitalprogramvara så existerar robotar i vår fysiska omgivning och därför behöver vi identifiera nya processer och metoder för utvecklingen av robotmjukvara och bygga kunskapsbron mellan digital- och robotprogramvara.

För att undersöka ett område där en robot kan nyttjas inleddes projektet MäRI (Människa och Robot-Interaktion) i ett samarbete mellan Åbo Akademi, Experience Lab och Yrkeshögskolan Arcada under andra kvartalet av 2020. Projektet strävar att ta fram evidensbaserad kunskap kring hur en vårdtagare och vårdstuderanden upplever mötet med en social, humanoid robot (Hägglund, 2020). Till forskning används Arcada robotarna

Amy och Alf som har funnits på plats sedan 2020 och roboten Snow sedan 2021 och vid Arcada används Amy främst i restaurangsyfte och undersökningssyfte för bland annat projektet *AI Driven Nordic Health and Welfare*, medan de två robotarna Alf och Snow nyttjas i mer social forskning som projektet MäRI. Vid Experience Lab finns roboten Pepper, roboten som är utvecklad från ett samarbete mellan Frankrike och Kina (Nussey, 2021) och likt Snow och Alf är den mer inriktad på sociala aspekter än servicearbetet Amy är specialiserad för.

Åbo Akademi och Arcada har tillsammans designat applikationen för MäRI och testat den på målgruppen (vårdtagare och vårdstuderanden) för att förstå hur tillit mot sociala, humanoida robotar upplevs och uttrycks. Projektet förväntas vara slutfört 30.6.2022 (Tigerstedt, 2020).

1.1 Frågeställningar

Examensarbetes huvudsakliga mål är att utreda utvecklingsprocessen för att ta fram applikationen Flossa till projektet MäRI för roboten Alf och undersöka om det är möjligt att utveckla applikationen på roboten Alf för elev-teamet på Arcada.

1. Hur ser utvecklingsprocessen ut för en robotapplikation på Alf, hur går processen till och finns de några begränsningar?

Det finns även ett behov att undersöka och utforska vilken programvara som är nödvändig för att uppfylla de krav MäRI projektet ställer för att de skall kunna testa applikationen på de definierade målgrupperna.

2. Vad måste utvecklas för att uppfylla behovet MäRI projektet har för roboten Alf?

Susanne Hägglund poängterar att projektet MäRI skall sammanföra och stärka kunskapen kring sociala robotar i Svensk-Finland (Hägglund, 2020) och vi vet att röststyrda assistenter funnits från stora leverantörer på svenska sedan en tid tillbaka. Till exempel Siri från Apple har funnits på svenska sedan april 2015 (Mörner, 2015) men utöver stora företag med tillgång till kopiösa mängder data är det oklart hur hållbart det är att ut-

veckla en robot och dess applikationer till att servera användare på svenska som MäRI behöver.

3. Kan vi utveckla en robotapplikation på svenska? Och vilka begränsningar möter vi?

1.2 Teamet på Arcada

Gruppen som arbetar på Arcada för projektet MäRI med utveckling av programvara ägs av Christa Tigerstedt och innehåller ett flertal medlemmar som tagit del av projektet i varierande möjlighet under tidsperioden för projektet. Utvecklingsteamet bestod av Dennis Biström, lektor och huvudansvarig och fyra IT elever i olika assisterande roller, Johannes Edgren, Markus Kankkonen, Johan Penttinen och författaren av detta arbete Kristoffer Kuvaja Adolfsson.

1.3 Om Sanbot Elf

Följande stycke innehåller teknisk information om roboten Alf som i grunden är en robot av typen Sanbot Elf samt kort allmän information om dess utgivare och tillverkare.

1.3.1 Sanbot

Roboten Elf produceras av *Sanbot Innovation* ett högteknologiskt kinesiskt företag. Deras mål är att bryta barriären mellan traditioner och nya utvecklingar. Med över 6 år i branschen och 200 patent inom teknologi söker de sig till att utveckla fler AI lösningar (Sanbot Innovation Technology, 2018).

Sanbot samarbetar med b.la. IBM Watson, Amazon Alexa och Nuance. De säljer robotarna från Sanbot serien, vilket inkluderar Sanbot Elf (den modell Arcada Alf är), Sanbot Nano och Sanbot Max. Robotarna är främst riktade mot industrier inom hotell, restaurang, vård och skolor.

1.3.2 Yodaway

Roboten Alf inköptes av Arcada från företaget Yodaway ApS, ett danskt företag grundat 31 augusti 2018 (Open Corporates, 2022). Företagets mål är att all data, även den i vår fysiska närvaro ska kunna användas på samma/liknande sätt som online data. Deras vision är att fylla den fysiska närvaron med deras egna AI modeller och robotar. Robotarna idag ska fungera som en isbrytare mellan ”dig” och dina kunder och dra uppmärksamhet till ditt varumärke och företag (Yodaway, 2019).

1.3.3 Sanbot Elfs användningsområden

Utöver sin roll på Arcada för hälsovårdsforskning har Sanbot Elf genom Yodaway andra liknande forskningsprojekt även i Danmark (Søgaard, 2021). Yodaway visar även stolt upp sitt partnerskap med större danska bolag som Arla där Sanbot Elf arbetar som roboten *Carla* som ett kommunikationsverktyg och med uppgifter inom marketing (Yodaway, 2020) eller på Lego hotells som en receptionist som uppmanar om handhygien (Yodaway, 2020)

2 HÅRD- OCH MJUKVARUINFORMATION

Kapitlet beskriver hårdvaran applikationen Flossa användes på och den medföljande mjukvaran roboten Sanbot Elf hade vid inköp för utvecklingssyfte och användning.

2.1 Sanbot Elf

Arcada Alf är en Sanbot Elf robot, en specialiserad och intelligent servicerobot utformad speciellt för kommersiellt bruk inom handel, hälsovård, gästfrihet och utbildning (Sanbot Innovation Technology, 2016). Sanbot Elf tillhandahåller även en öppen SDK för utveckling och integrering av egna ändamål.

Sanbot Elf väger 19kg, är 90 cm hög, 42 cm bred och 33 cm djup, Arcadas Alf är av färgen vit, men Sanbot Elf finns även i guldfärg och kan anpassas efter kunders behov med olika dekaler som till exempel företagslogon. Batteritiden för Sanbot Elf är uppskattad till 10h användning och över 24h i vänteläge med sitt 300W litiumbatteri.

Till sensorer har Alf 7 mikrofoner inbyggt i sitt huvud med två RGB kameror, 1 3D sensor och 7st känsel sensorer. På sin kropp has Alf 1 gyroskop sensor, 2st induktions sensorer, en IR kollisions undviknings-sensor, 6st IR mottagare, 4st känsel sensorer och en elektronisk kompass. På varje hand och på benen/bottnen har Alf 4/10 IR kollisions undviknings-sensor, den nederdelen av roboten har även 3 rundstrålande hjul och 2 känselsensorer på varje hand.

Med hjälp av hjulen kan Sanbot Elf röra sig i 360 graders riktning från startposition och med de över 10 kollisions-undviknings-sensorerna kan Sanbot Elf automatiskt undvika hinder och håller sin omgivning och användare säkra. Roboten har även 360 graders ljudlokalisering med röstkontroll och för att spela upp ljud har Alf 2 diskanthögtalare och 1 sub-bashögtalare. (Sanbot Innovation Technology, 2016)

Roboten är utvecklad på ROS, version 1.1, Fire Break från 2010. ROS består av ett flertal bibliotek och verktyg för att utveckla robotapplikationer som drivrutiner och algoritmer (Open Robotics, 2021).

Roboten Alf har en 10,1 tums pekplatta på bröstet med en upplösning på 1080p, 60 Hz och 16:9 format och en fingerkänslighet på 10 punkter. Kamerorna i roboten består av en 8 megapixel HD kamera och en 1 megapixel färgkamera med 140 graders sfärisk yta. Sanbot hänvisar till att roboten kan användas för att hålla presentationer och videosamtal med en större bildskärm än den inkluderade pekplattan genom sin inbyggda projektor bakpå Sanbot Elfs huvud. Laser projektorn har en upplösning på 720p och uppskattas att ge bra bild upp till 65 tums format (Sanbot Innovation Technology, 2016).

Figur 1 Toppen, vänster till höger: Alfs pekplatta, Alfs kamera sensorer och ansikte, Alfs mikrofoner och projektor, Alfs baksida med Arcada dekal, egna foton

2.2 Android enhet

Telefonen som används för att köra Android applikationerna under projektet är en Huawei Honor 8 Lite med modellnumret PRA-LX1. Telefon har 3 GB ram och körs på Android 8.0 (Oreo) med en Cortex-A53 processor som har 8 kärnor, 4 med en klockfrekvens på 2.1 GHz och 4 kärnor med en frekvens 1.7 GHz, telefonen har även 16 gigabytes internminne (Wikipedia, 2021).

2.3 Qlink

Qlink, försedd av Sanbot, är en mobilapplikation som kan kommunicera över nätet med Sanbot robotar. Applikationen har åtkomst till ditt Sanbot-kontos sociala information; som registrerade kontakter eller dina registrerade robotar. Applikationen ger även tillgång till de flesta av robotarnas externa hårdvara som robotens kamera, rörelser och ansiktsuttryck men även en talsyntes där ett indatafält tar emot en text som roboten sedan spelar upp som ljud. Videouppspelningen som fanns tillgänglig från kameran i applikationen saknar ljud men är annars en direktsändning.

Rörelserna kan kontrolleras genom att välja kroppsdel, huvud, fötter (hjul) samt höger eller vänster arm. Samtliga kroppsdelar kan röra sig uppåt och neråt (framåt/bakåt för fötterna) dessutom kan huvudet och fötterna rotera med- eller motsols. Huvudet uppnår en rörlighet på nästan 180 grader.

Ansiktsuttrycken saknar beskrivning men antyder på att simulera vanliga känslor som kärlek, glädje eller sorg. Sammanlagt finns det 18 ansiktsuttryck med små animationer som pulserande hjärtan eller rinnande tårar.

2.4 SDK

Sanbot SDK är en nativ Java-applikation försedd av Sanbot, robotens primära utvecklare. Applikationen installeras direkt på Sanbot Elf och ger åtkomst till samtliga funktioner för roboten, som rörelser, ansikte, kameror, högtalare, musik. Applikationen utforskades men användes väldigt lite i detta arbete då samtliga interaktioner förorsakade kraschar direkt på Sanbot Elf. Teamet misslyckades även med att utveckla applikationen vidare då det inte var möjligt att emulera applikationen på en PC utan specifik emulerings programvara.

2.5 MobileSDK

Android- eller MobileSDK är också utgiven av Sanbot. Denna SDK är en Android applikation som kommunicerar med roboten Alf över internet genom en server belägen på

AWS. Applikationen hade tillgång till kameran på pekplattan, kroppsdelarna, robotens rörelser och Sanbot-kontots sociala kontakter likt de som funnits på Qlink applikationen. MobileSDK:n var även öppen för utveckling och gick att modifiera direkt i Android studio och kunde emuleras på en fysisk telefon eller någon av Android Studios emulerings alternativ.

3 GENOMFÖRANDE

Följande kapitel lägger fram information angående vilken programvara som kommer utvecklas under utvecklingsprocessen och varför programvaran valts att utvecklas. Kapitlet kommer ställa upp de behov och krav MäRI projektet hade för applikationen Flossa och dess tester.

3.1 Krav för Flossa

Roboten Alf har naturligt fysiska begränsningar som kan observeras vid första kontakt, mindre synligt är dock Alfs mjukvarubegränsningar och detta slutarbete avser att utforska dessa begränsningar genom att utveckla applikationen Flossa och nyttja roboten Alfs möjligheter till det yttersta för att uppfylla kraven från MäRI projektet.

Kraven för Flossa sattes av MäRI teamet på Arcada och kan summeras till följande 3 punkter:

1. Design & Funktionalitet
    a. Applikationens design skall godkännas av projektägaren
    b. Applikationen skall vara godtyckligt användbar för forskningstester
    c. En robotviskare skall kunna följa med interaktionen under forskningstester och styra roboten vid behov.
2. Röst och Hörsel
    a. Applikationen skall kunna tala på svenska
    b. Applikationen skall kunna lyssna och förstå, i godtycklig mån, svenska för interaktionens behov

3. Användning av robotens attribut
    a. Robotens attribut ska stödja applikationens interaktion, attribut hänvisar till robotens armar, huvud och ansikte

För att kunna uppfylla kraven till MäRI med applikationen Flossa identifierades behovet av 4 delar, som i sin helhet bildar applikationen Flossa, dessa är:

1. Användargränssnittet, webbapplikationen Alf frontend (webbaserad HTML, CSS och JS applikation)
2. API till roboten, genom MobileSDK (Android Java applikation)
3. API mellan användargränssnittet och robot API:n, SSE server (Python applikation)
4. API för talsyntes och taligenkänning, TTS & STT server (Python applikation) med komplimenterande tredjepartstjänst

3.2 Användning av robotens attribut

MobileSDK:n valdes som utvecklingsplattform då ingen direkt API fanns till roboten Alf och den medföljande SDK:n inte var användbar på grund av instabilitet. Detta uteslöt alla andra alternativ för utveckling av robotens rörelser förutom användningen av MobileSDK:n då användning av robotens attribut krävdes.

MobileSDK:n är en Android applikation byggd på Java med layouts i XML. Utvecklingen av applikationen kommer fortskrida med samma system dvs. Android Studio och Java. Valet gjordes främst för att andra alternativ, som Kotlin, är helt obekanta för utvecklarna och i andra hand för att Java och XML anses flexibla nog för de krav MäRI ställt.

MobileSDK:n som utvecklas kommer ställas sida vid sida med den redan kommersiella applikationen Qlink och jämföras i funktionalitet, hur roboten kan manipuleras, och även i hastighet dvs. den tid det tar för roboten att reagera från det att ett kommando ges till applikationen. Hastigheten kommer simpelt jämföras med författarens observationer i syfte att avgöra om MobileSDK plattformen är godtycklig nog i jämförelse till den

kommersiella applikationen Qlink. Det kommer inte jämföras ifall den utvecklade applikationen kan mäta sig i användarvänlighet eller stabilitet, då detta är utanför projektets referensram eftersom programvaran utvecklas för ett specifikt forskningssyfte och inte som en kommersiell applikation för ekonomisk vinning eller försäljning.

För att utforska robotens rörelsebegränsningar kommer en funktion göras som på måfå skickar rörelser till roboten. Dessa rörelser kommer välja en slumpmässig kroppsdel, en slumpmässig möjlig riktning (som konstaterats tidigare kan en arm endast kan röra sig vertikalt) och under en slumpmässig tidsperiod på mellan 1 och 10 sekunder. 10 sekunder är mer än dubbelt den tid det tar för roboten att röra en kroppsdel från ett ändläge till ett annat vid lägsta hastigheten. Detta förutsätter därmed att ingen rörelse blir för lång eller för kort för att kunna observeras.

3.2.1 SSE server

Valet av MobileSDK:n bildade även behovet av en SSE server för att kunna kommunicera mellan webbapplikationen, Flossas grafiska gränssnitt, och MobileSDK:n. Roboten saknade kapacitet för att orka köra servern och användargränssnittet samtidigt på grund av sin limiterade hårdvara, och därför måste även en värd finnas för servern. Servervärd kommer väljas efter möjlighet och utvecklarnas egna preferenser.

SSE servern kommer byggas med Python på mikro-ramverket Flask (Pallets Project, 2010) då utvecklaren för applikationen var bekant med det från tidigare och det var snabbt och enkelt att komma i gång med för våra definierade mål. Flask och ramverket Waitress nyttjas även i utvecklingen av TTS & STT servern, Waitress används för att behandla HTTPS begäran (Pylons Project, 2022) och valdes av samma orsak som tidigare, bekantskap.

3.3 Röst och Hörsel

Till att börja med behandlades ljud genom Google *webkitSpeechRecognition* (Web Platform Incubator Community Group, 2020) men för att inte låsa webbapplikationen till endast Chrome-baserade webbläsare flyttades ljudbehandlingen över till ramverket

WebRTC (Google, WebRTC team, 2015b). WebRTC användes sedan för att spela in och behandla användarens auditiva respons under interaktionen med roboten.

Googles TTS (Google, 2018a) och STT (Google, 2018b) påträffades i samband *webkitSpeechRecognition*. Dessa tjänster har sina självständiga API:n som kan användas, men för att upprätthålla flexibilitet och stabilitet utvecklades en separat server för att behandla begäran mellan webbapplikationen och Googles TTS respektive STT tjänster. Talsyntes och taligenkännings tjänsterna från Google behölls under utvecklingen då den befintlig integration uppfyllde godtycklig användbarhet för projektet. Valet att inte förhandsinspela robotens röst gjordes då manuskriptet måste vara flexibelt under utvecklingsprocessen.

3.4 Pilottester

Ett dussin experiment utfördes av utomstående personer på Flossa applikationen under november månad 2021, där testades applikationen av frivilliga svensk- och engelsktalande elever från Arcada för att identifiera möjliga problem inför MäRI projektets forskningsintervjuer och för att stödja utvecklingen av att uppfylla en godtycklig applikation. Deltagarna för våra experiment utvärderades eller undersöktes inte utan det enda kriteriet var att deltagarna besatt godtycklig svenska eller engelska och var villiga att delta. Samtliga deltagare var ekonomistuderanden ifrån projektledare Christa Tigerstedts kurs, servicedesign som sysslar med användarcentrerad design.

Deltagarna fick genomföra applikationen en gång enskilt under observation av teamet i e-businesslaboratoriet på Arcada, detta rum valdes för att minska på auditiva störningar samt för att möbleringen innehöll flertal sittplatser, likt ett väntrum på en tandvårdsklinik. Deltagarna gavs sedan följande instruktioner muntligt:

> ”Ni är på en tandvårdsklinik. Där finns roboten Alf, och ni ska nu prova roboten. Vänligen gå fram och tryck på *Start* och lyssna sedan på robotens instruktioner.”

Experimenten utvärderades sedan på personlig basis med kriterierna om applikationen kunde genomföras med speciell vikt på de tidigare definierade målen, design, funktionalitet, röst, hörsel och användning av robotattribut.

När det kommer till val av värd för alla applikationer valdes Heroku och AWS som slutleverantör under målgruppstesterna.

4 RESULTAT

Resultaten består av utvecklingsfasen för Flossa applikationen. Huvudsakligen kommer detta kapitel att bearbeta utvecklingen av de 4 applikationerna som bildar Flossa, dessa är webbapplikationen, SSE servern, TTS & STT servern och MobileSDK:n. Applikationerna kommer beskrivas i de 3 krav, Design & Funktionalitet, Röst & Hörsel och Användning av robotens attribut, som ställdes av MäRI projektet.

4.1 Användning av robotens attribut

MobileSDK:n som kunde manipulera roboten gick att utveckla med hjälp av Java i Android Studio. MobileSDK:n kom färdigt med ett inloggningssystem och autentisering vilket gav möjligheten att logga in till MäRI teamets Sanbot-konto. Väl inloggad fanns en kontaktlista som visade kontots sociala kontakter och ett separat GUI med möjligheten till att välja en av kontots registrerade robotar för att sedan styra vald robot och öppna en direktsänd videoström från kameran.

Från början kunde MobileSDK:n endast strömma video, vilket inte var tillräckligt för att uppfylla behovet; då kravet för robot-viskaren ingick i att vara medveten om miljöns ljud under en interaktion för att kunna agera mot användaren. Den existerande videoklassen konverterades till att skicka audiobytes i sin direktsändning, utöver bara videobytes, med en ändring av det existerande attributet *mIRobot.onOpenVideo*. Det strömmade ljudet konverterades sedan med en *ByteSerializer* funktion till uppspelningsbar audio för applikationen MobileSDK. De strömmade audiobytesen konverterades till audio med en samplingsfrekvens på 16 kHz, 16 bitars format och i monokanal som MobileSDK:n kunde spela upp samtidigt som direktsändningen av videon spelades.

För att kunna manipulera rörelser fanns ett färdigt joystickgränssnitt för att kontrollera riktning och drivkraft. Joystickens läge angavs av ett decimalt värde mellan 0.0 till 1.0 som sedan multiplicerades med joystickens färdriktning för att skapa drivkraftens värde till roboten; detta innebar att när joysticken var i viloläge multipliceras drivkraften med 0.0 och ingen rörelse uppstod hos roboten. Drivkraftsvärdena skickades sedan till en AWS belägen server som upprätthölls av Sanbot innan de sändes vidare till roboten för att utföras.

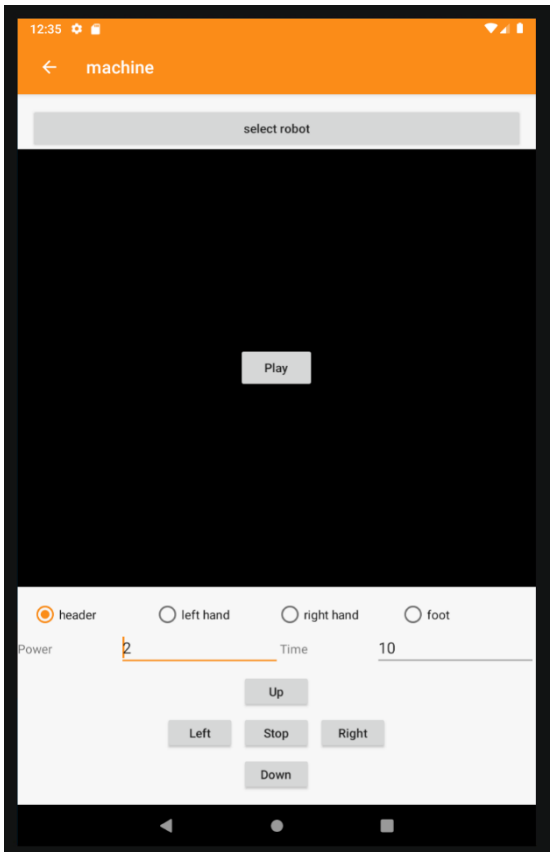
På grund av begränsningarna att behöva begära rörelser genom Sanbots AWS server uteslöts försök att koppla bort inloggningssystemet som var färdigt i MobileSDK:n, det medföljande joystickverktyget saknade dock den precision som behövdes för att kunna programmera förvalda rörelsemönster för att nyttja robotens rörelseattribut till fullo. Dessa rörelsemönster, s.k. *gestures* i Flossa, kunde vara att till exempel vinka genom att röra armen uppåt och neråt upprepade gånger, likt när en person vinkar.

Efter utforskning av koden fann vi ett objekt med 3 egenskaper som skickades mellan MobileSDK:n och AWS servern som instruerade roboten. Objektet hade de följande egenskaperna, *body* som tog emot ett heltal, *orientation* som också tog emot ett heltal och *power* som tog emot ett flyttal. Det existerande även en stopfunktion som stannade robotens aktiva rörelser som tog emot parametern *body*.

Egenskapen *body* tog emot ett av fyra Id:n, 1001, 1002, 1003, 1004 vilka representerade huvudet, vänster arm, höger arm och fötterna. *Orientation* tog emot id:n 1001, 1002, 1003, 1004 vilka representerade upp, ner, vänster och höger riktning medan egenskapen *power* tog emot ett flyttal mellan 1.0 och 8.0 för att styra rörelsens hastighet, det spekulerades inom teamet att *power* egenskapen hölls som ett flyttal av utgivaren för att kunna multipliceras med joystickens position, eftersom det värdet var ett decimalvärde mellan 0.0 och 1.0.

För att utveckla mer kontrollerade och bestämda rörelser ändrades MobileSDK:ns gränssnitt för att tillåta preciserade värden med hjälp av knappar där varje riktning kunde väljas individuellt i stället för med en joystick. Ett indatafält med värden för rö-

relsens *power* och ett annat indatafält med en tid millisekunder som simulerade joystickens aktivitet med värdet 1.0 och tiden räknades i en separat tråd (Java *thread*) i Java programmeringen. Detta gjorde att en tid kunde preciseras för hur länge en rörelse skulle utföras innan stopfunktionen kallades som avslutade rörelsen.



Figur 2 Skärmdump av användargränssnittet från ett utvecklingsskede av MobileSDK

En funktion kallad *getSchwifty* utvecklades sedan för att demonstrera och konceptualisera rörelsemönster samt för att uppskatta om roboten Alf kunde utföra flera rörelser samtidigt utan avbrott eller kraschar. Funktionen kallade en slumpmässig kroppsdel, riktning och *power* från listor med värden, listorna var anpassade för varje kroppsdel. Dessa rörelser kördes sedan i nya trådar och skickades till funktionen som hanterade vidarebefordring av rörelser till AWS servern.

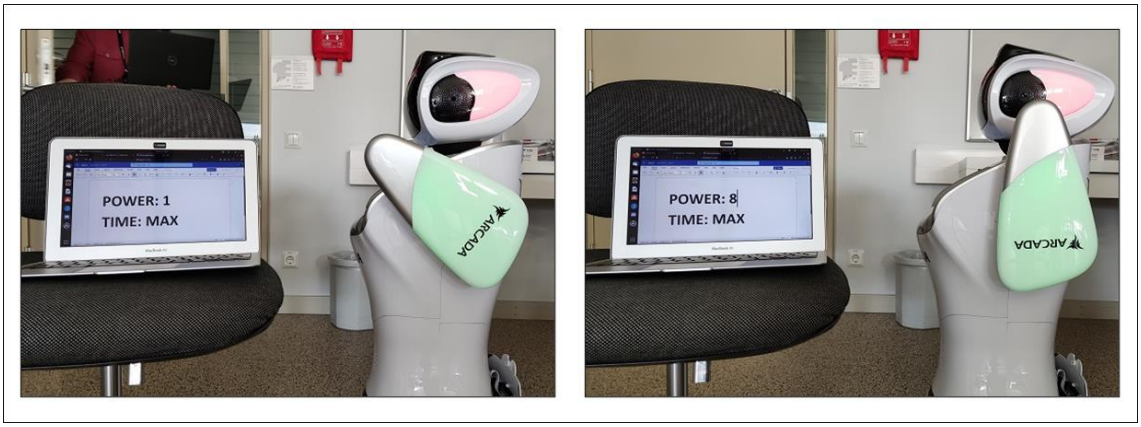
Funktionen *getSchwifty* fick en tillfällig knapp i mitten av användargränssnitten och testades sedermera på Alf. Vid de första observationer uppstod rörelserna mer sällan och sporadiska än förväntat och efter närmare iakttagelser hittades det att den separata trå-

den för att stoppa rörelser utfördes för snabbt och stängde följande rörelse på grund av fördröjningen som uppstod över nätet. Därför ändrades funktionen till att fördröjas i 100 millisekunder, detta gjorde att följande rörelse inte stängdes av föregående anrop innan rörelsen kunde utföras.

```
1.  public void getSchwifty(MoveInfo moveInfo) {
2.          //It is time to get schwifty
3.          MoveInfo tempInfo = moveInfo.clone(); //temp to not corrupt original call
4.          Thread t = new Thread(() -> {
5.              long start = System.currentTimeMillis();
6.              long end = start + 30000;
7.              while (System.currentTimeMillis() < end) {
8.                  int randomPower = getRandomPower();
9.                  int randomTime = getRandomTime(randomPower);
10.                 tempInfo.setDistance(randomTime);
11.                 int body = getRandomBody();
12.                 tempInfo.setBody(body);
13.                 int orientation = getRandomOrientation(body);
14.                 tempInfo.setDirection(orientation);
15.                 move(body, orientation, randomTime);
16.                 String info = tempInfo.toString();
17.                 Log.i("MoveInfo", info + " randomTime: " + randomTime);
18.                 sleepStop(randomTime, body);
19.             }
20.         });
21.         t.start();
22.     }
```

Figur 3 getSchwifty funktionen från MobileSDK

För att skapa rörelsescheman dokumenterade vi rörelser för varje kroppsdel, utgående från den maximala motsatta starta-position på roboten i 10 sekund/10 000 millisekunder med varierande värden *power* (1 till 8). Med bilddokumentation kunde MobileSDK:ns API-anrop översättas till riktiga rörelser och rörelsemönster planerades sedan, så kallade *gestures* i Flossa applikationen.



Figur 4 Bilddokumentation av armrörelse Power 1 (vänster) och Power 8 (höger), egna foton

En *gesture* för att vinka skrevs sedermera i MobileSDK:n för att enkelt kunna anropas vid hälsning/början av Flossa interaktionen. Detta rörelseschema grupperades sedan med den tidigare *getSchwifty* funktionen och kunde anropas utöver de standardiserade rörelserna.

```
1.      public void wave(int hand) {
2.          Thread t = new Thread(() -> {
3.              move(hand, RobotCmd.CMD_MOVE_UP, 8f);
4.              sleepStop(2000, hand);
5.              for (int i = 0; i < 3; i++) {
6.                  move(hand, RobotCmd.CMD_MOVE_DOWN, 3f);
7.                  sleepStop(1000, hand);
8.                  move(hand, RobotCmd.CMD_MOVE_UP, 8f);
9.                  sleepStop(1500, hand);
10.             }
11.             move(hand, RobotCmd.CMD_MOVE_DOWN, 4f);
12.         });
13.         t.start();
14.     }
```

Figur 5 Funktion för att vinka från MobileSDK:n

För att användare av Flossa applikationen skulle kunna anropa rörelser direkt från roboten Alf till MobileSDK:n på Android telefonen från det grafiska användargränssnittet byggdes en ström med *okhttp-eventsource* som tillåter konsumtion av Server Sent Events (SSE) genom en API byggt på *OkHttp* (LaunchDarkly, 2022). Funktionen i MobileSDK:n som skulle lyssna på strömmen utvecklades utifrån ett fungerande exempel som lyssnade på Wikipedias ändringslog (Wikimedia, 2022). Efter att ha implementerat exemplet konfigurerades MobileSDK:n att lyssna på den skapta SSE servern som strömmade ut rörelsekommandon. Funktionen konfigurerades sedan att starta när en ägare och robot blivit valda i applikationen och knappen Play tryckts. MobileSDK:n lyssnade sedan efter rörelsekommandon och om dessa hördes kunde de skickas vidare till den relevanta funktionen, en rörelse eller ett rörelseschema, för att få roboten att röra sig direkt från användargränssnittet.

4.1.1 SSE server

För att roboten Alf skulle kunna kommunicera med användaren och använda sina rörelser under interaktionen med Flossas användargränssnitt byggdes en mellanhand som förde information vid interaktionerna på gränssnittet till MobileSDK:n. Denna mellanhand är en server byggd i Python, med mikro-ramverket Flask. Servern byggdes för att kunna ta emot JSON objekt via HTTP POST-metoden. Dessa objekt strömmas sedermera konstant ut på den adress MobileSDK:n lyssnade efter.

Servern tar emot ett JSON objekt som använder sig av samma kroppsdelar och riktningar som MobileSDK:ns rörelse funktioner, dock översatta till mer deskriptiva värden i stället för siffer-id:n, till exempel översattes 1001 till *head*.

```
1.  {
2.    bodyPart: String
3.    gesture: String
4.    direction: String
5.    distance: int
6.  }
```

Figur 6 JSON objekt som SSE servern tar emot

Python servern laddades först upp på värden Heroku för att snabbt kunna demonstreras i början av projektet, utveckling gick sedermera vidare till en mer permanent lösning. Först försökte teamet att värda servern på plats vid Arcada genom en Raspberry PI 3B. Dock på grund av säkerhetsprotokollen i HTTPS misslyckades försöket då webbapplikationen och servern inte tilläts kommunicera med användargränssnittet utan signerade HTTPS certifikat. Detta i kombination med problem att upprätthålla permanenta internetlösningar genom skolans brandvägg gjorde att servern slutligen flyttades till AWS där säkerhetsprotokoll fanns färdiga och skolans brandvägg inte behövde konfigureras.

4.2 Röst och Hörsel

För att Alf skall kunna tala och höra byggdes en serverapplikation på Flask, samma mikro-ramverk som användes för SSE servern. Applikationen använde sig även av

Waitress, ett simpelt ramverk för att snabbt komma i gång med en WSGI (Web Server Gateway Interface) för Python applikationer. Serverns uppdrag är att agera mellanhand för användargränssnittet, som användaren kommer interagera med, och vår tredjeparts röst- och hörseltjänster.

Som röst används talsyntes, *text to speech* och taligenkänning, *automatic speech recognition*. För att implementera dessa system användes Google TTS och Google STT.

4.2.1 Talsyntes

Innan talsyntesen togs i bruk nyttjades en testsida för att prova Google TTS tjänst olika röstparametrar. Valen bestod av följande; språk, kön, frekvens, talhastighet och tonhöjd (Edgren, 2021). Parametrarnas värden valdes tillsammans inom teamet av utvecklarna och godkändes sedermera även av ägaren för projektet, valet blev följande, för Flossa applikationen version 1; *gender = FEMALE, language = sv-SE, hertz = 16 000, pitch = -10, rate = 1*

Servern som mellanhand kräver autentisering till Googles *Cloud API* och en personlig nyckel måste sättas upp. Applikationen möjliggör sedan att textdata kan skickas som ett JSON objekt från webbapplikationen med POST-metoden. JSON-objektet servern tar emot analyseras och delas upp för att sändas vidare till Googles tjänst, tjänsten returnerar sedan en ljudfil i MP3 format och MP3 filen skickas sedan tillbaka genom mellanhanden och till sist når den användargränssnittet där roboten spelar upp ljudfilen för att skapa Alfs röst. Utöver text nyttjas även pauser med hjälp av taggen *<break time=”seconds”/>* i manuskriptet för att öka hörförståelsen i robotens tal.

```
1.  "server_url?ReqString=" + '<speak>' + din text <break time=”1s”/> här + '</speak>'
    + "&lang=sv-SE&rate=1.4"
```

Figur 7 Exempel på talsyntes förfrågan till TTS & STT servern från användargränssnittet

4.2.2 Taligenkänning

I och med att servermellanhanden redan nyttjar en autentiseringsnyckel från Google utvecklades samma applikation till att även behandla ASR med ett nytt vägval. En liknande process användes som vid talsyntes, dvs. användargränssnittet skickade en BLOB med audiobytes till servern där den formaterades med korrekt metadata, till exempel språk, och vidarebefordrades sedan till Googles STT tjänst. Tjänsten i sin tur returnerade en teckensträng på vad som uppfattats från BLOB:n. Teckensträngen färdades sedan tillbaka till användargränssnittet och roboten.

```
1.  def tts():
2.
3.      TDIR = os.path.dirname(__file__)
4.      filename = "output.mp3"
5.      ReqString = request.args.get('ReqString')
6.      rate = request.args.get('rate') or 1
7.      pitch = request.args.get('pitch') or -10
8.      hertz = request.args.get('hertz') or 16000
9.      lang = request.args.get('lang') or "sv-SE"
10.     gender = request.args.get('gender') or "FEMALE"
11.     ReqString = urllib.parse.unquote(str(ReqString))
12.     hertz = int(hertz)
13.     rate = float(rate)
14.     pitch = float(pitch)
15.     lang = str(lang)
16.     print("String: " + str(ReqString))
17.     print("Speechrate: " + str(rate))
18.     print("Hertz: " + str(hertz))
19.     print("Pitch: " + str(pitch))
20.     print("Lang: " + str(lang))
21.     print("Gender: " + str(gender))
22.
23.     if(ReqString):
24.         response = CreateTTS(ReqString, rate, pitch, hertz, lang, gender)
25.         with open(filename, "wb") as out:
26.         # Write the response to the output file.
27.             out.write(response.audio_content)
28.         path = os.path.join(TDIR + filename)
29.         return send_from_directory(TDIR, path, as_attachment=True)
```

Figur 8 Funktion från TTS & STT servern

Med hjälp av Googles STT parameter *grammar*, kunde vi skicka medföljande nyckelord från webbapplikationen. Denna grammatik ökade chansen för tjänsten att identifiera ljudet i BLOB:n som den angivna grammatiken i parametern. I vårt fall gäller grammatiken de ord som finns som alternativ till svar för användaren i vårt användargränssnitt. Praktiskt innebär det att om användaren svarar på en ”Ja” eller ”Nej” fråga med ”Jo” och vi sänder grammatiken ”Ja” och ”Nej” till tjänsten kommer tjänsten lägga extra vikt

i att identifiera ordet som ”Ja”. Alla våra förvalda interaktioner från användargränssnittet skickades därför även med till mellanhandsservern som teckensträngar och formaterades om till grammatikparametern för tjänsten.

4.3 Design och funktionalitet

MäRI projektets strävade att redan i ett tidigt skede granska möjligheterna med roboten Alf och applikationen Flossa, tala och hörförmåga skulle utvärderas men möjligheten att visa upp webbinnehåll och videon skulle även fastslås. Likväl var det önskvärt att söka lösningar om Alf kunde påbörja Flossa applikationens interaktion automatiskt när användare närmade sig roboten.

För att uppfylla design kravet var det viktigt att användargränssnittet var flexibelt och smidigt och kunde kommunicera med våra andra applikationer, till detta valdes därför en webbapplikation. Webbapplikationens utveckling delas in i ett pre-alfa eller konceptstadie kapitel och följande angående de viktigaste delarna av applikationen som förbättrades upp till leveransversionen som användes för MäRI projektets målgruppstester.

4.3.1 Webbapplikationens konceptstadium

Pre-alfaversionen av webbapplikationen demonstrerades inför hela MäRI teamet inklusive partnerna från Åbo Akademi och Experience Lab. Där visades en populär Youtube video, *Rick Astley - Never Gonna Give You Up* (Astley, 2009), upp i webbapplikationen för att uppskatta videouppspelningsförmågan då den korrekta videon med tandtråds instruktioner ännu inte fanns tillgänglig. För att implementera taligenkänning användes *webkitSpeechRecognition* som lyssnade på användaren och skickade en BLOB med ljud till vår TTS & STT server. Manuskripttexten för interaktionen skickades även den till TTS & STT servern för att omvandlas till syntetiserat ljud på Googles tjänster och ge roboten en röst. Själva interaktionsflödet byggdes i form av ett nod-träd med två olika klasser, en klass för interaktion och en klass för att avsluta applikationen och visa ovannämnda video.

Applikationen kunde till en början endast köras i Chrome med flaggan *Must run on Chrome with flag –autoplay-policy=no-user-gesture-required*, för att skapa en sömlös användarerfarenhet där användaren inte manuellt måste starta videon.

Figur 9 Skärmdump av användargränssnittet vid demonstrationsskedet

Webbapplikationen implementerade även ansiktsigenkänning under pre-alfaversionen med ramverket Picojs (Markuš, 2021). Detta användes för att utforska ifall en separat videoström kunde identifiera ett ansikte i webbapplikationen och sedermera starta interaktionen när ett ansikte identifierats under en längre tid.

Trots lyckad ansiktsidentifiering och automatisk start slopades idén kort därefter då Sanbot Elf inte stadigt kunde driva både användargränssnittet och ansiktsigenkänning utan att påverka prestandan för webbapplikationen så pass mycket att kravet på en godtycklig interaktion inte kunde uppnås.

Efter att den första iterationen av hela Flossa applikationen demonstrerats under det interna mötet tillsammans med våra samarbetspartner, beslöts det att utvecklingen skulle fortsätta med de utforskade metoderna. Applikationen skulle även öppnas för att kunna användas på fler plattformar, dvs. inte endast Google Chrome. Också designen skulle vidareutvecklas för webbapplikationen till en gemensam design och en gemensam design skulle väljas vid ett senare tillfälle.

4.3.2 Applikationens vidareutveckling

Webbapplikationens design genomgick många iterationer och med de flexibla webb-verktyg som använts, HTML, CSS och JS, var det möjligt att ha många parallella designförslag för att sedan kombinera de bästa attributen av varje design till den slutgiltiga produkten.

Bland annat utvecklades en modal för att hålla videoelement, knappar som dynamiskt kunde gömmas och visas under interaktionen och ny text för interaktionen. Det slutgiltiga utseendet och manuskriptet beslöts sedan av projektets parter utifrån många designiterationer. Design kravet ändrades till att utveckla ett gemensamt utseende med samma interaktion hos båda parterna för att kunna kombinera data från olika studier och tester.

För att kunna utveckla den interaktion MäRI krävde för sitt ändamål måste främst webbapplikationen utvecklas att stödja ett mer omfattande nod-träd för interaktionen då två noder inte längre var tillräckligt till följande fem noder:

1. Question
    a. En fråga formuleras med text följt av 2 till 3 svars alternativ, dynamiskt anpassande efter antalet frågor med tillhörande JavaScript logik för stilisering. Varje svarsalternativ har sedan möjlighet att fortsätta till sin egen separata nod.
2. TrickQuestion
    a. En dubblett av ovanstående klass, men utan förmågan att gå vidare till olika noder, i stället fortsätter alltid varje svarsalternativ till samma nod.
3. Monologue
    a. En nod där endast en text formuleras som roboten talar, följande nod spelas upp utan att användaren behöver agera med applikationen.
4. Video
    a. En nod som tar emot ett videoobjekt och en text. Videon kan spelas med eller utan ljud och videons start- och ändpunkt kan konfigureras med en

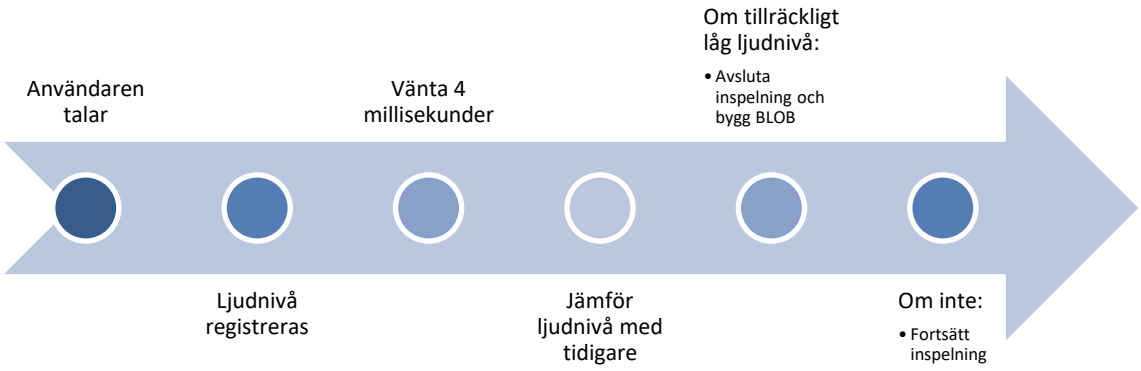
extra parameter. Noden avslutas dynamiskt när det sista talet utförts eller videon upphör. Noden går sedan automatiskt vidare till följande nod.

5. EndTree
    a. Noden formulerar tal med en text och avslutar interaktionen. Interaktionen och webbapplikationen avslutas med en annan funktion som laddar om webbapplikationen med JavaScript funktionen *window.reload()*.

För att flytta webbapplikationen från att endast vara användbar på Google Chrome till att stödja flera webbläsare avvecklades Google *webkitSpeechRecognition*, som inte stödes i till exempel Firefox, till open-source alternativet WebRTC.

WebRTC funktionaliteten utvecklades från ett exempel att strömma användarens audiovolym (Google WebRTC Team, 2015a). Detta exempel valdes för att undersöka möjligheten för automatisk aktivering av Flossa interaktionen vid tal av nyckelord från användaren som en hälsning. Som med slopandet av idén med ansiktsigenkänning, övergavs även denna implementation senare av samma orsak då robotens behov av att konstant lyssna på sin omgivning blev för tungt för robotens processor och Flossa interaktionen påverkades negativt och kravet på en godtycklig interaktion kunde inte upprätthållas.

WebRTC exemplet anpassades vidare för webbapplikationen, likt implementationen av *webkitSpeechRecognition* funktionen, skapades en BLOB som sändes till TTS & STT servern. Med soundmeter funktionen från WebRTC kunde BLOB:n justeras när ljudnivån blev tillräckligt hög eller låg för att avsluta inspelningen och sända BLOB:n med audiobytes till servern. En algoritm utvecklades från WebRTC exemplet som var fjärde millisekund jämförde nuvarande ljudnivå med tidigare ljudnivån. Om en tillräckligt stor ändring i ljudnivån uppstod skulle inspelning påbörjas, inspelning slutar sedan när nuvarande ljudnivå var lägre än tidigare ljudnivån plus 0.02 enheter. Algoritmen och enheten 0.02 framkom av praktiska tester från utvecklarna på funktionen och roboten.



Figur 10 Schema för logik av BLOB bildande med WebRTC

Andra alternativ utforskades även för både talsyntes och taligenkänning men ekosystemet runt Google behölls då andra lösningar inte uppfyllde kravet för en godtycklig interaktion. Bland andra lösningar fanns Kaldi, Coqui och Mozilla projektet Common Voice. Ingen av alternativen kunde erbjuda både en talsyntes och en taligenkännings lösning inom ramarna för projektet då de saknade modeller för det svenska språket under utvecklingsskedet för Flossa.

Webbapplikationen genomgick även en större omstrukturering där mycket överflödig kod togs bort och förtydligades, variabler fick namnbyte, buggar fixades och mindre funktioner implementerades, som en knapp på applikationen för att kunna maximera användargränssnittet på skärmen.

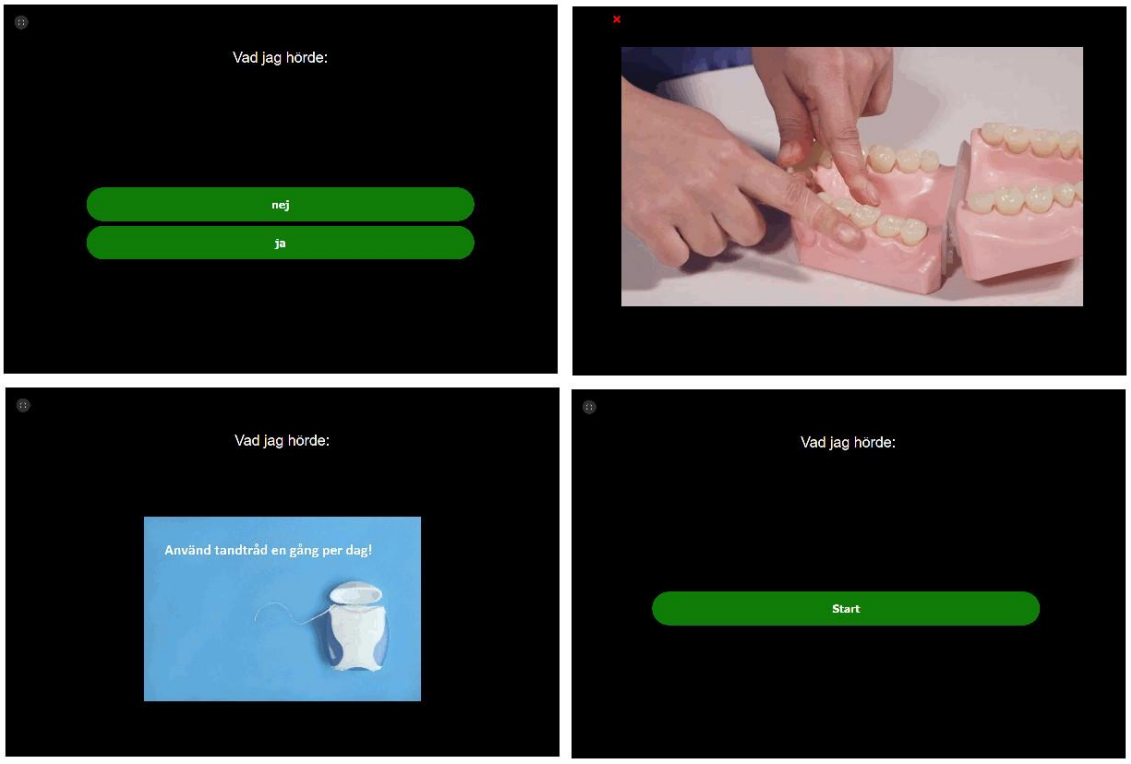
Utöver detta utvecklades en version av webbapplikationen på engelska efter förfrågan av projekt ägaren med den motivering att pandemin (Covid-19) gjort det extra svårt att få tillstånd för tester men genom att nyttja flera språkliga versioner av applikationen kunde personer utan svenska språkkunskaper testa applikationen och ge feedback.

För att utveckla den engelska versionen bröts texten ut ur nod objekten till variabler i en separat JavaScripts fil. Variablerna översättes sedan till det språk som behövdes, i detta

fall engelska. De tjänster vi använde från Google kunde även leverera taligenkänning på engelska och TTS & STT servern uppdaterades därför genom att öppna ytterligare ett vägval för STT på engelska, detta vägval var en direkt kopia av tidigare funktionen *def tts()* men med parameter ändringen för language_code till ”en-US”.

Talsyntesen kunde sedan direkt modifieras på webbapplikationen med argumentet *lang* från ”se-SV” till ”en-US” i sin talsyntes förfrågan och skickas till TTS & STT servern och vidare till Google TTS, på så vis ändrades talsyntesen från att inte tala engelska med en svensk dialekt utan i stället engelska med amerikansk dialekt.

Bakgrunden som Åbo Akademi försett projektet med korrigerades även till engelska med bildredigeringsmjukvara och webbapplikationen laddades upp på en separata URL för att kunna användas vid behov på roboten Alf.



Figur 11 Skärmdumpar av Flossa V1 användargränssnittet

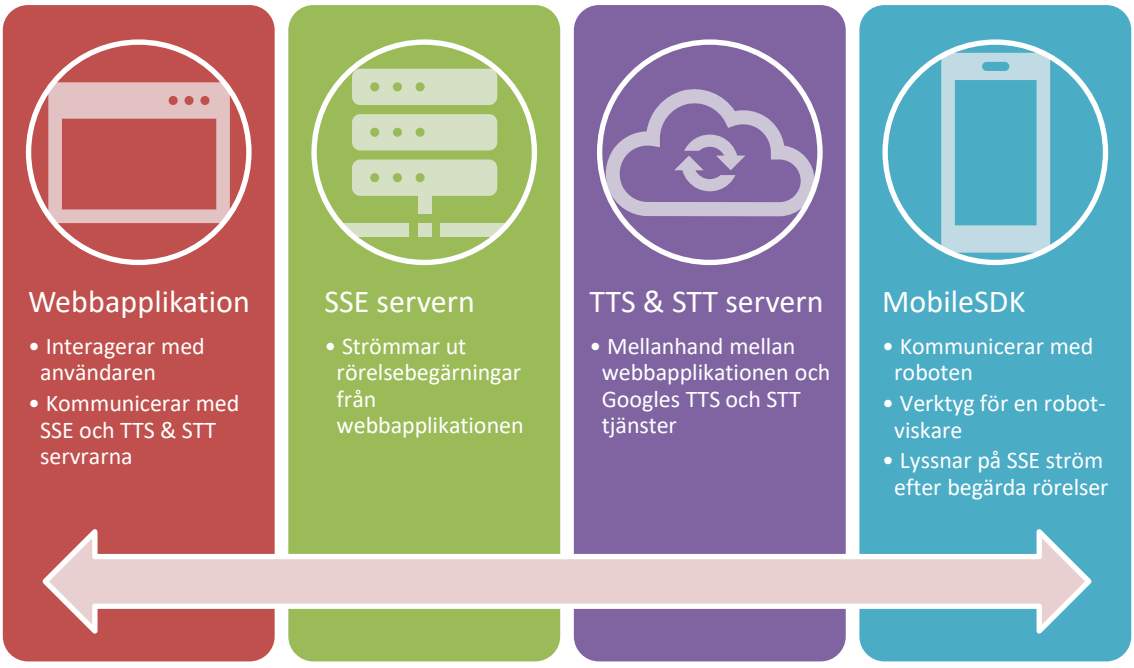
Webbapplikationens design och utseende kom inte att förändras ytterligare, utan endast processen av att dokumentera och upprätthålla applikationerna genom att fixa eventuella buggar återstod innan MäRIs målgruppstester.

Utifrån pilottesterna identifierades ett flertal buggar och ett frustrerande moment i designen; övertydlig repetition av instruktionerna vid misstolkning. Detta korrigerades med att inte upprepa hela frågan utan endast använda sig av en ursäktsfras när ett tal utanför de angivna svaren identifierats från en användare.

Den slutgiltiga interaktionen för webbapplikationen blev således en startknapp följt av en hälsning och en vinkning från roboten som i sin tur ledde in till en förfrågan på om användaren ville lära sig om tandhygien. Därifrån visades en instruktionsvideo angående tandtråd som roboten uppmärksammade med en huvudrörelse neråt. Efter videon lyfte roboten på huvudet och ställde tre följdfrågor angående materialet från videon. Alla frågor under interaktioner kunde besvaras genom att tala eller genom att välja något av de synliga alternativen på pekplattan.

5 ANALYS

Utvecklingsprocessen för Flossa applikationen följer de generiska utvecklingsstadierna vi lärt oss under utbildningen vid Arcada, *pre-alfa*, *alfa*, *beta*, *gamma* och *slutgiltig produkt*. För att få Flossa att fungera på roboten Alf behövdes det utvecklas mer än en applikation, ett användargränssnitt i form av en webbapplikation som roboten Alf visar upp på sin pekplatta och som användaren interagerar med. Ytterligare applikationer behövdes för att få roboten att uppfatta ljud och röst genom taligenkänning och för att kunna kommunicera med användare genom talsyntes. Applikationen agerade som mellanhand och nyttjade en tredjeparts tjänst, Google, som omvandlade text till ljud och ljud till text. Det behövs även en API för webbapplikationen som möjliggör kommunikation med ytterligare en serverapplikation som därefter strömmar ut instruktioner till Flossas mobila utvecklingsverktyget som sedan kan styra roboten. Tillsammans bildar de fyra applikationerna Flossa.



Figur 12 Flossas 4 applikationer

Ett viktigt förlopp som bör uppmärksammas inför andra projekt är att utvecklingsprocessen, jämfört med mer generell programvaruutveckling, hade en betydande brist i emulering för tester. Även om varje delapplikation inom Flossa kunde emuleras eller enskilt testas gick det inte att få roboten Alf emulerad. Detta innebar att MobileSDK:ns

funktionalitet som var central för interaktionen av Flossa alltid måste testas på plats med roboten Alf och med den rådande pandemin som var aktiv under stora delar av 2021 och 2022 fanns det tydliga svårigheter i det behovet.

I jämförelse med den kommersiella Qlink applikationen begränsades MobileSDK:n med att inte kunna ändra på roboten Alfs ansiktsuttryck och att inte tillhandhålla sin egen talsyntes. Bristen av talsyntes på MobileSDK:n kan åtgärdas med utveckling, då fungerande funktioner för talsyntes redan existerar i Flossa och en anpassning till Java bör vara fullt möjlig. Men för att utveckla ansiktsuttrycken som fanns i Qlink applikationen påträffades ingen API i MobileSDK:n vilket innebär att det kan vara en brist som inte gå att åtgärda. Reaktionstiderna för att kommunicera rörelserna till roboten Alf observerades dock inte vara märkbart skilda, vilket innebär att den skapade implementation kan mäta sig med den kommersiella. Funktionalitet som utvecklats utöver Qlink applikationen är möjligheten till att lyssna på den strömmade videons audio, möjligheten att ta emot rörelseförfrågningar från en tredjeparts applikation (webbapplikationen) och möjligheten att precisera rörelserna med ett mer detaljerat användargränssnitt.

Eftersom källkoden för roboten Alf var låst blev det uteslutet att bygga egna miljöer för emulering eller automatiska tester då ingen inom teamet hade expertis eller kunskap att ta reda på vad eller vilken kod som gick in och ut ur roboten från AWS servern. En lösning som diskuterades inom teamet var att sätta upp kameror för att direktsända miljön där roboten befann sig för att kunna observera roboten via en direktsändning under utvecklingen. Detta ansågs dock inte praktiskt eftersom roboten Alf befann sig på en offentlig plats, Yrkeshögskolan Arcada, och en öppen direktsändning under dygnets alla timmar rakt in i högskolans lokaler skapar betänkliga sekretess- och säkerhetsrisker.

När det kommer till webbapplikationen uppfyller den ett specifikt syfte med att instruera angående tandtråd men har även några bristande moment, som till exempel lösningen att tvinga en sidomladdning vid interaktionsslut. Webbapplikation i sig är även väldigt flexibel och stommen, funktionaliteten med ett nod-träd, kan nyttjas och anpassas vidare för andra syften tillsammans med roboten Alf.

Back-end programmen, SSE och TTS & STT, fyllde sina behov och genomgick väldigt få ändringar när de väl var implementerade. Den största upplevda bristen var att hitta en bra värd för dem. Båda programmen hystes i testskedet på Heroku. Medan MobileSDK:n användes på Honor8 Lite Telefonen. Webbapplikationens två språkversioner hystes på den egna Arcada domänen som elever vid Arcada har tillgång till.

Med alla 4 delar av applikationen i fungerande form kunde pilottester utföras i den kontrollerade miljön på Arcada. Testerna utfördes mestadels av enskilda användare och pilottesterna på applikationen och samtliga personer kom igenom applikationen med röst och/eller touch kontroller på endera den svenska eller engelska versionen. Dock måste ett test startas om på grund av en bugg, som senare identifierades och fixades och ett annat test måste ha robotens instruktioner från webbapplikationen repeterade av en forskningsassistent, då testpersonen hade nedsatt hörselförmåga.

6 DISKUSSION

1. Hur ser utvecklingsprocessen ut för en robotapplikation på Alf, hur går processen till och finns de några begränsningar?

Trots att utvecklingsprocessen för Flossa varit generisk vill jag personligen uppmärksamma att Flossa och dess utvecklingsprocess endast utförts på en typ av robot, en Sanbot Elf, och att det finns tusentals olika robotplattformer att arbeta på. Detta arbete är därmed en väldigt liten undersökning inom en väldigt stor industri och utser inte att göra någon till expert.

Som utvecklare blir man även snabbt blind för sina egna applikationer och i jämförelse med tidigare utvecklingsprocesser som skol- eller hobbyprojekt skall vikten av utomstående tester inte underskattas. I min mening skulle Flossa inte varit lika finslipad som den var inför MäRI projektets målgruppstester om det inte varit för pilottesterna. Jag anser dock att det varit önskvärt med ännu fler pilottester under utveckling, i hopp om att förbättra applikationerna vidare. Men realistiskt sett var detta något den rådande pandemin försvårade betänkligt. Pandemin satte även käppar i hjulen för mer kreativa

idéer jag hade önskat, som att besöka andra offentliga miljöer med roboten, som en tandvårdsklinik eller ett annat utbildnings campus för mer feedback till utvecklingen av Flossa.

Det skulle därmed vara varmt välkommet för fortsatta studier att undersöka testmiljöer för framtida robotutvecklingar. Går det att emulera robotar i en virtuell testmiljö och kan man därmed överkomma svårigheterna till att samla testdata under restriktiva perioder som en pandemi?

När det kommer till de skillnader mellan den kommersiella Qlink applikationen och vår MobileSDK hade vår applikation fördelen av att kunna utveckla egna rörelsescheman s.k. *gestures* och lyssna på en ström för rörelser. En styrka med detta är möjligheten att strömma ut instruktioner till många applikationer samtidigt, jag kan spekulera i att en svärm av Alf robotar kanske kan styras eller en koreografisk uppvisning går att utveckla för underhållningsbruk.

Personligen har jag själv anpassat och använt webbapplikationen och Alf under två andra tillställningar, en filminspelning till MäRI projektets webbseminarium där ett flertal *monologue* noder användes för att skapa robotens repliker. Alf och jag befann oss även på öppet hus dagen vid Arcada den 25 november 2021. Webbapplikationen justerades så att Alf hälsade på besökare och presenterade 3 alternativ, ett påstående, en fråga eller en video. Påståendet och frågan inriktade sig på fakta om Arcada medan videoalternativet spelade upp *Arcada – Hands on Tomorrow* (ArcadaUAS, 2018). Jag ser absolut en framtida användning för Alf på Arcada där interaktionen från webbapplikationen kan skräddarsys till olika evenemang eller till och med för att stödja i undervisningssyfte som någon gärna får utforska i framtiden.

2. Vad måste utvecklas för att uppfylla behovet MäRI projektet har för roboten Alf?

Analyseringen av Flossa underströk att fyra applikationer behövdes för att uppfylla kraven från MäRI projektet. Att SDK plattformen för nativ robotutveckling inte fungerade var en stor orsak till det numeriska behovet och detta trots att roboten kommersialisera-

des med ett öppet och utvecklingsvänligt ekosystem (Sanbot Innovation Technology, 2016). Därför rekommendera jag starkt att granska lösningar från andra håll än just den leverantör Alf kommer ifrån. En större förundersökning bör även göras för att se hur anpassningsbar en robotprodukt faktiskt är för det önskade syftet innan inköp görs. Till exempel har roboten Reachy från Frankrike en helt öppen API i Python som finns publicerad på Github med manualer och dokumentation på engelska (Pollen Robotics, 2022). Detta är en klart starkare utgångspunkt än den för Alf i min mening. Dessutom kan troligen Reachy roboten, med hjälp av en separat pekplatta, utföra Flossa interaktionen lika bra som den specialiserade sociala och humanoida roboten Alf.

När det kommer till val av värd för applikationerna uppfyllde Heroku sitt syfte men det hade varit önskvärt att flytta programmen till CSC:s Rahti-tjänst (CSC – IT Center for Science, 2021), en tjänst som teamet utforskade efter målgruppstesterna och som jag i framtiden varmt rekommenderar för andra projekt från Arcada.

För att fortsätta diskussion kring upplevda brister var det tydligt under projektet att taligenkänningen var den del som orsakade övervägande problem för både användare och utvecklare. Det var problematiskt att försöka bygga algoritmer och skriva kod till både *webkitSpeechRecognition* och WebRTC som kunde effektivt avsluta inspelning av tal under interaktion mellan maskin och människa och sedan samtidigt bygga och skicka en BLOB för taligenkänning. Detta orsakade många svårigheter, främst då människor inte kommunicerar monotont vilket även Shreya Amin påpekar ytterligare med att förklara att en enkel fras som ”Hej! Hur mår du?” kan sägas på många olika sätt, i många olika tonlägen, hastigheter och dialekter och att en maskin inte bara hör, utan måste också konvertera ljudsignaler från människor till digitala signaler. Maskinen måste sedan även ha en förståelse för eventuella bakgrundsljud samtidigt som den försöker identifiera talet från användaren. (Amin, 2019)

Taligenkänning, *Automatic Speech Recognition*, har däremot under de senaste åren blivit ett mer och mer aktivt ämne och studier från bland annat grannlandet Sverige utförs kontinuerligt med till exempel *Automatic Speech Recognition Model for Swedish using Kaldi* (Wang, 2020) eller Kungliga bibliotekets arbete med BERT (Haffenden, 2020). Kaldi undersöktes även under projektets gång i MäRI teamet men även ramverket Co-

qui-ai och både ramverken ger möjligheter till att träna sina egna modeller. Det skulle ha varit önskvärt i efterhand att gått längre under utvecklingen med Coqui-ai eller Kaldi men tyvärr var båda dessa verktyg för tunga för roboten Alf att köras lokalt och dessutom började utvecklingen tränga in i andra projekt på Arcada. Coqui-ai ramverket hade däremot exempel på röstolkning genom en serverström i stället för en BLOB och detta upplevdes ge en betänkligt mer responsiv upplevelse och skulle varit ett önskvärt alternativ till Google-lösningen Flossa använde.

En implementation av någon av dessa, Kaldi eller Coqui-ai, skulle i teorin kunna öka chansen till förståelse mellan användare och robot bara vi använder rätt rösttolkningsmodeller. Om det i framtiden skulle skapas metoder eller utvecklas färdigt tränade röstigenkänningsmodeller som kan nyttjas specifikt för Flossas behov skulle jag starkt rekommendera att övergå till dessa ramverk.

I min mening är den huvudsaklig anledningen med att använda andra tjänster än Googles att våra språk kommer digitaliseras mer och mer och om vi inte själva gör arbetet med taligenkänning kommer endast kommersiella lösningar, ofta baserade på endast engelska, bli övervägande dominanta. Mindre språk, som finska, svenska och finlandssvenska, riskerar därmed att tyna bort i den digitala världen. Detta är också innan vi reflekterar över att både Kaldi och Coqui-ai är öppna ramverk i jämförelse med Google-tjänsten, som Flossa använder, där vår data hamnar helt innanför IT-gigantens egna väggar. Något som i min mening utgör en stor brist, då vi som användare och utvecklare aldrig riktigt kan vara säkra på vart vår data lagras eller vad den används till.

3. Kan vi utveckla en robotapplikation på svenska? Och vilka begränsningar möter vi?

Den slutsatsen leder bra in i om robotapplikationer går att utvecklas på svenska, svaret efter analysen var ett tydligt ja. Ett utvecklingsteam bör dock vara beredd på möjligheten att bli låst till vissa tjänster, som till exempel Google då, andra mindre ramverk som Kaldi eller Coqui-ai saknar godtyckliga svenska modeller i nuläget. Det kan även vara värt att belysa att trots att svenska är Arcadas huvudsakliga utbildningsspråk såg vi en del av pilottesterna på engelska, därför är det viktigt att hitta testpersoner som är villiga

att prova applikationen på just det språk som används under utvecklingen för att uppnå bästa resultat.

Eftersom MobileSDK:n, så likt många andra Android applikationer, bygger sitt användargränssnitt med XML innebär per automatik att det är möjligt att omvandla applikationen till det språk som önskas bara korrekta översättningstjänster används, detta utfördes redan smått under projektet då en del av SDK:n var på kinesiska.

Trots att frågan i sig är ganska klar för utvecklingen av programvara för robotar på svenska bör samma begränsning som diskuterades ovan observeras då talsyntes men framför allt taligenkänning är outvecklat inom det svenska området utöver de stora it-företagsnamnen som Apple, Microsoft, Amazon, Meta (tidigare Facebook) eller Google. Därför skulle det vara högst önskvärt om en annan vill utveckla en modell för någon av de förslagna ramverken, Kaldi eller Coqui-ai för att stärka digitaliseringen av finlandssvenskan.

KÄLLOR

Amin, S., 2019. *Speech recognition is hard.* [Online]
Available at: https://towardsdatascience.com/speech-recognition-is-hard-part-1-258e813b6eb7
[Hämtad 30.4.2022].

ArcadaUAS, 2018. *Arcada – Hands on Tomorrow.* [Online]
Available at: https://www.youtube.com/watch?v=xlYszCPQiuY
[Hämtad 30.4.2022].

Astley, R., 2009. *Never Gonna Give You Up.* [Online]
Available at: https://www.youtube.com/watch?v=dQw4w9WgXcQ
[Hämtad 30.4.2022].

Computer Sweden och IDG Lotsson, Anders, n.d. *IT-ord.* [Online]
Available at: https://it-ord.idg.se/
[Hämtad 30.4.2022].

CSC – IT Center for Science, 2021. *Rahti container cloud.* [Online]
Available at: https://rahti.csc.fi/
[Hämtad 30.4.2022].

Edgren, J., 2021. *Enter text and press Play / Get.* [Online]
Available at: https://www.dinmamma.fi/alftts/
[Hämtad 30.4.2022].

Google WebRTC Team, 2015a. *Audio stream volume.* [Online]
Available at: https://webrtc.github.io/samples/src/content/getusermedia/volume/
[Hämtad 30.4.2022].

Google, WebRTC team, 2015b. *WebRTC.* [Online]
Available at: https://webrtc.org/
[Hämtad 30.4.2022].

Google, 2018a. *Text-to-Speech.* [Online]
Available at: https://cloud.google.com/text-to-speech
[Hämtad 30.4.2022].

Google, 2018b. *Speech-to-Text: Automatic Speech Recognition.* [Online]
Available at: https://cloud.google.com/speech-to-text
[Hämtad 30.4.2022].

Haffenden, M., 2020. *Playing with Words at the National Library of Sweden -- Making a Swedish BERT.* [Online]
Available at: https://github.com/Kungbib/swedish-bert-models
[Hämtad 30.4.2022].

Hägglund, S., 2020. *MäRI - Sociala robotar som agenter i vården.* [Online]
Available at: https://www.abo.fi/projekt/mari-sociala-robotar-som-agenter-i-varden/
[Hämtad 30.4.2022].

Isoaho, P., 2016. *Designing service/care robot.* [Online]
Available at: https://www.theseus.fi/handle/10024/113254
[Hämtad 30.4.2022].

LaunchDarkly, 2022. *okhttp-eventsource.* [Online]
Available at: https://github.com/launchdarkly/okhttp-eventsource
[Hämtad 30.4.2022].

Lius, L., 2021. *Theseus, The Future of Robotics.* [Online]
Available at: https://www.theseus.fi/handle/10024/371594
[Hämtad 30.4.2022].

Markuš, N., 2021. *picojs.* [Online]
Available at: https://github.com/nenadmarkus/picojs
[Hämtad 30.4.2022].

Moran, M., 2007. *The da Vinci Robot.* [Online]
Available at:

https://www.researchgate.net/publication/6594114_The_da_Vinci_Robot
[Hämtad 30.4.2022].

Mörner, E., 2015. *Så funkar Siri på svenska.* [Online]
Available at: https://www.mobil.se/appar/s-funkar-siri-p-svenska
[Hämtad 30.4.2022].

Nussey, S., 2021. *Softbank shrinks robotics business, stops Pepper.* [Online]
Available at: https://www.reuters.com/technology/exclusive-softbank-shrinks-robotics-business-stops-pepper-production-sources-2021-06-28/
[Hämtad 30.4.2022].

Nyberg, M., 2016. *Theseus, Utveckling av en digital dosettlåda.* [Online]
Available at: https://www.theseus.fi/handle/10024/113560
[Hämtad 30.4.2022].

Nylund, T., 2020. *Thesus, Developing a cross-platform MVP app with React Native.* [Online]
Available at: https://www.theseus.fi/handle/10024/355335
[Hämtad 30.4.2022].

Open Corporates, 2022. *Yodaway ApS.* [Online]
Available at: https://opencorporates.com/companies/dk/39840871
[Hämtad 30.4.2022].

Open Robotics, 2021. *Why ROS? It's the fastest way to build a robot!.* [Online]
Available at: https://www.ros.org/blog/why-ros/
[Hämtad 30.4.2022].

Pallets Project, 2010. *Flask web development, one drop at a time.* [Online]
Available at: https://flask.palletsprojects.com/en/2.1.x/
[Hämtad 30.4.2022].

Pollen Robotics, 2022. *Reachy by Pollen Robotics, an open source programmable humanoid robot.* [Online]
Available at: https://www.pollen-robotics.com/
[Hämtad 30.4.2022].

Pylons Project, 2022. *Waitress.* [Online]
Available at: https://docs.pylonsproject.org/projects/waitress/en/latest/
[Hämtad 30.4.2022].

Sanbot Innovation Technology, 2016. *Patented Sanbot Robot, AI Robot Design.* [Online]
Available at: https://en.csjbot.com/content/11/1298.html
[Hämtad 30.4.2022].

Sanbot Innovation Technology, 2018. *Brand Story.* [Online]
Available at: http://en.sanbot.com/about/brand-story
[Hämtad 30.4.2022].

af Malthe Bendix Søgaard, M. B., 2021. *Aalborg Kommune afprøver robot på demensplejehjem.* [Online]
Available at: https://www.carenet.nu/nyheder/?id=3951
[Hämtad 30.4.2022].

Tigerstedt, C., 2020. *MäRI -Människa robot interaktion (Den sociala roboten).* [Online]
Available at: https://www.arcada.fi/sv/forskning/projekt/mari-manniska-robot-interaktion-den-sociala-roboten
[Hämtad 30.4.2022].

Wang, Y., 2020. *Automatic Speech Recognition Model for Swedish Using Kaldi.* [Online]
Available at: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:1498905
[Hämtad 30.4.2022].

Web Platform Incubator Community Group, 2020. *Web Speech API.* [Online]
Available at: https://wicg.github.io/speech-api/
[Hämtad 30.4.2022].

Wikimedia, 2022. *Event Platform/EventStreams.* [Online]
Available at: https://wikitech.wikimedia.org/wiki/Event_Platform/EventStreams
[Hämtad 30.4.2022].

Wikipedia, 2021. *Honor 8 Lite.* [Online]
Available at: https://fi.wikipedia.org/wiki/Honor_8_Lite
[Hämtad 30.4.2022].

Yodaway, 2019. *About us.* [Online]
Available at: https://yodaway.com/en/#about-us
[Hämtad 30.4.2022].

Yodaway, 2020. *Facebook.* [Online]
Available at: https://zh-cn.facebook.com/yodaway/posts/623541595100339
[Hämtad 30.4.2022].

Yodaway, 2020. *Hos LEGOLAND Hotel & Conference [Sanbot Robotten Karen, påminder og påminder om håndsprit]*. [Online]
Available at: https://www.youtube.com/shorts/MMTd7QwoaAw
[Hämtad 30.4.2022].