

Universitatea “Politehnica” din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

**Aplicație de transcriere a convorbirilor telefonice pe  
platformă Android (Android telephone calls  
transcription and searchable history)**

**Lucrare de licență**

Prezentată ca cerință parțială pentru obținerea titlului de  
*Inginer în domeniul Electronică și Telecomunicații*  
*programul de studii Tehnologii și Sisteme de Telecomunicații*

Conducător științific

Lect. Horia CUCU

Absolvent

Diana-Iuliana ENESCU

2014



Universitatea "Politehnica" din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației  
Departamentul TELECOMUNICAȚII

Aprobat Director de Departament Telecomunicații:

Prof. Dr. Ing Silviu CIOCHINĂ



**TEMA PROIECTULUI DE DIPLOMĂ**  
**a studentei ENESCU D. Diana-Iuliana, grupa 442C**

1. Titlul temei: Aplicație de transcriere a convorbirilor telefonice pe platformă Android (Android telephone calls transcription and searchable history).
2. Contribuția practică, originală a studentului va consta în:
  - 2.1 Aplicație Server care realizează transcrierea fișierelor audio .wav în text ;
  - 2.2 Aplicație Client pe o platformă Android:
    - Realizarea unei aplicații de înregistrare a apelurilor telefonice direct după linia telefonică și salvarea fișierelor .wav în memoria telefonului;
    - Realizarea unei aplicații de trimitere a fișierelor .wav la server;
  - 2.3 Aplicație Android care administrează lista de contacte și fișierele cu transcrieri;
3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline: Programare Obiect-Orientată, Prelucrarea Digitală a Semnalelor;
4. Realizarea practică/ proiectul rămân în proprietatea: *Laborator de Cercetare Speed*, student *Enescu Diana-Iuliana*;
5. Proprietatea intelectuală asupra proiectului aparține: *Laborator de Cercetare Speed*, student *Enescu Diana-Iuliana*;
6. Locul de desfășurare a activității: *UPB*
7. Data eliberării temei: iunie 2013

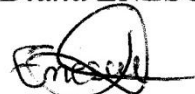
CONDUCĂTOR LUCRARE:

STUDENT:

Lect. Horia CUCU



Diana ENESCU





## Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul “Aplicație de transcriere a convorbirilor telefonice pe platformă Android (Android telephone calls transcription and searchable history)”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Inginerie Electronică și Telecomunicații*, programul de studii *Tehnologii și Sisteme de Telecomunicații*, este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 28.06.2014

Absolvent *Diana ENESCU*

A handwritten signature in black ink, appearing to read 'Enescu', with a large, stylized loop at the end.



## Cuprins

<b>Lista figurilor .....</b>	<b>9</b>
<b>Lista acronimelor .....</b>	<b>11</b>
<b>Introducere .....</b>	<b>13</b>
<b>1. Introducere în Android .....</b>	<b>15</b>
1.1 Istoric .....	15
1.2 Caracteristici Android .....	15
1.3 Arhitectura sistemului Android .....	16
1.3.1 Nucleul Linux .....	17
1.3.2 Librăriile de bază .....	17
1.3.3 Android Runtime .....	17
1.3.4 Aplicații Framework .....	18
1.3.5 Nivel aplicații .....	18
1.4 Elementele componente ale unei aplicații Android .....	18
<b>2. Instrumente de dezvoltare Android .....</b>	<b>23</b>
2.1 Eclipse IDE .....	23
2.2 Android SDK.....	23
2.3 Android Development Tools (ADT) .....	24
2.4 Mașina virtuală Dalvik .....	24
<b>3. Sisteme de recunoaștere automată a vorbirii continue .....</b>	<b>25</b>
3.1 Introducere în recunoașterea automată a vorbirii .....	25
3.2 Resurse necesare în construcția unui sistem RVC .....	26
3.3 Aplicații ale sistemelor de recunoaștere a vorbirii .....	28
<b>4. Proiectare și implementare .....</b>	<b>31</b>
4.1 Prezentarea aplicației .....	31
4.2 Descrierea codului sursă .....	39
4.3 Testarea aplicației pe diferite dispozitive .....	43
<b>Concluzii .....</b>	<b>45</b>
<b>Bibliografie .....</b>	<b>47</b>
<b>Anexa 1 .....</b>	<b>49</b>





## Lista de figuri

Figura 1.1 Arhitectura sistemului Android .....	16
Figura 1.2 Nucleul Linux.....	17
Figura 1.3 Librării de bază.....	17
Figura 1.4 Android Runtime.....	17
Figura 1.5 Aplicații Framework.....	18
Figura 1.6 Nivel Aplicații.....	18
Figura 1.7 Ciclul de viață al unei activități.....	19
Figura 1.8 Interfață fragmentată pe ecran mic.....	21
Figura 1.9 Interfață fragmentată pe ecran lat.....	21
Figura 1.10 Fișierul Manifest.....	22
Figura 2.1 Eclipse IDE.....	23
Figura 3.1 Arhitectura generală a unui sistem de RAV.....	25
Figura 3.2 Resursele necesare în construcția unui sistem RVC.....	26
Figura 3.3 Fonemele limbii române.....	27
Figura 3.4 Aplicație S2T.....	29
Figura 4.1 Interfața grafică a aplicației.....	31
Figura 4.2 Activitate principală: Informații apel.....	32
Figura 4.3 Directorul creat de aplicație în memoria telefonului.....	34
Figura 4.4 Lista cu directoare în interiorul directorului principal.....	35
Figura 4.5 Directoarele generate de contacte.....	35
Figura 4.6 Bara de notificare.....	35
Figura 4.7 Butoane funcții.....	36
Figura 4.8 Fereastră media player.....	36
Figura 4.9 Fereastră transcriere.....	48



## **Lista acronimelor**

AAC – Advanced Audio Coding  
ADT – Android Developer Tools  
AMR – Adaptive Multi-Rate  
API - Application Programming Interface  
CDMA – Code Division Multiple Access  
DTMF – Dual Tone Multi Frequency  
GIF – Graphics Interchange Format  
GPL – General Public License  
GSM – Global System for Mobile Communications  
IDE – Integrated development environment  
JPEG – Joint Photographic Experts Group  
LAN - Local Area Network  
MPEG – Moving Picture Experts Group  
OHA – Open Handset Alliance  
PNG – Portable Network Graphics  
RAV – Recunoaștere automată a vorbirii  
RVC – Recunoașterea vorbirii continue  
S2T – Speech-to-Text  
SDK – Software development kit  
TCP/IP – Transmission Control Protocol / Internet Protocol  
UMTS - Universal Mobile Telecommunications System  
XML – Extensible Markup Language



## Introducere

În această lucrare am descris modul de implementare a unei aplicații pe platformă Android. Aplicația realizează o transcriere a convorbirilor telefonice utilizând tehnologia dezvoltată de Laboratorul de cercetare Speech and Dialogue. Soluția oferită reprezintă o arhitectură de tip client-server, comunicarea între cele două realizându-se prin socluri TCP-IP.

Sistemele de recunoaștere a vorbirii au o arie de aplicabilitate foarte mare, în prezent ele putând fi utilizate pentru transcrierea de text după dictare, completarea vocală a rubricilor unei fișe (fișe medicale, cereri de înscriere, etc.), transcrierea subiectelor discutate într-o emisiune TV/Radio.

La momentul actual, telefoanele mobile pe platformă Android sunt foarte răspândite în lume ceea ce susține utilitatea portării sistemului pe această platformă. Sistemele de recunoașterea a vorbirii pot fi utilizate în activități de rutină, pot ajuta persoane cu probleme locomotorii (persoane care, din diverse motive, nu își pot folosi membrele superioare pentru a scrie).

Am ales implementarea clientului pe platformă Android deoarece este o platformă open-source, bine documentată, iar dezvoltarea unei aplicații nu necesită cumpărarea unei licențe.

Am început cu o scurtă descriere a platformei Android și a instrumentelor utilizate la implementarea aplicației pentru a evidenția caracteristicile și resursele puse la dispoziția dezvoltatorilor. Mai departe am descris modul în care realizează serverul traducerea clipurilor audio și modul de realizare al aplicației. Descrierea aplicației este însoțită de pasaje de cod pentru a evidenția modul în care poate fi implementată practic.

Alegerea acestei teme a fost motivată de interesul personal către dezvoltarea de aplicații și de dorința de a realiza o lucrare cât mai practică, de actualitate.



# CAPITOLUL 1

## Introducere în Android

### 1.1 Istoric

Android este singurul sistem de operare mobil creat de Google, iar mai tarziu de consorțiul comercial Open Handset Alliance și reprezintă o platformă software pentru telefoane mobile și dispozitive bazate pe nucleu Linux. Acesta permite dezvoltatorilor să scrie cod în limbaj Java. Aplicațiile pot fi scrise și în C sau alte limbaje de programare dar acest mod de dezvoltare nu este susținut oficial de Google.[1]

Platforma Android a fost lansată la 5 noiembrie 2007 prin anunțarea fondării unui consorțiu de 48 de companii de hardware, software și de telecomunicații numit Open Handset Alliance (OHA), incluzând companii ca Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Sprint Nextel și Nvidia.[1]

La 9 decembrie 2008, 14 noi membri au aderat la proiectul Android printre care Sony Ericsson, Vodafone Group, Asustek Computer Inc, Toshiba Corp și Garmin Ltd.

Din data de 21 octombrie 2008, Android a fost disponibil ca Open Source astfel că Google a deschis întregul cod sursă sub licența Apache. Sub această licență producătorii sunt liberi să adauge extensii proprietare, fără a le face disponibile comunității open source, în timp ce contribuțiile Google la această platformă rămân open source.

Platforma Android a fost construită pentru a permite dezvoltatorilor să creeze aplicații mobile care să utilizeze toate resursele pe care un telefon le are de oferit. A fost construit pentru a fi cu adevărat deschis. O aplicație poate apela oricare dintre funcționalitățile de bază ale telefonului, cum ar fi efectuarea de apeluri, trimiterea de mesaje text sau folosirea aparatului de fotografiat. Android-ul nu face diferența între aplicațiile de bază ale telefonului și cele create de dezvoltatori. Ele pot fi construite să aibă acces egal la capacitățile telefonului pentru a oferi utilizatorilor un spectru larg de aplicații și servicii. Fiind o platformă open source, aceasta va evolua continuu prin încorporarea tehnologiilor de ultimă generație.<sup>[2]</sup>

### 1.2 Caracteristici Android

Printre caracteristicile principale ale sistemului Android se numără următoarele:

- *Platformă open source.* Android este un produs open source, distribuit sub licența Apache. Această licență este destul de permisivă și oferă libertatea de a copia, distribui și de a modifica în mod liber surse existente fără nici un cost de licențiere, rămânând la alegerea dezvoltatorilor de a distribui sursele modificate.[1] Singura excepție de la această regulă o constituie nucleul Linux care se află sub licență GPL versiunea 2 ce prevede că orice modificare a surselor trebuie să fie făcută publică și distribuită gratuit.

- *Portabilitate.* Platforma Android permite rularea aplicațiilor pe o gamă largă de dispozitive. Programele sunt scrise în Java și executate pe mașina virtuală Dalvik permițând astfel portarea pe ARM, x86 și alte arhitecturi. Mașina virtuală Dalvik reprezintă o implementare specializată de mașină virtuală concepută pentru utilizarea în dispozitivele mobile, desi nu este o mașină virtuală Java standard.

- *Oferă suport pentru grafică 2D și 3D.* Platforma este adaptabilă la configurații mai mari, biblioteci grafice 2D, biblioteci grafice 3D și configurații tradiționale pentru telefoane mobile.

- *Împărțirea pe sarcini(task)*. Aplicațiile Android sunt alcătuite din diferite componente ce pot fi reutilizate și de alte aplicații. Refolosirea de alte componente pentru a ajunge la rezultatul final este cunoscută sub numele de sarcină( *task*) in Android.

- *Posibilitatea de a stoca date sub forma unor baze de date SQLite.*
- *Conectivitate*. Platforma Android suportă o gamă largă de tehnologii de conectivitate precum GSM, CDMA, UMTS, Bluetooth, Wi-Fi.
- *Suport media audio/video/imagine*: MPEG-4, H.264, MP3, AAC, AMR, JPEG, PNG, GIF.

Dezvoltatorii au la dispoziție o serie de unelte pentru dezvoltarea aplicațiilor precum emulator, unelte de depanare(debugging), pentru măsurarea performanțelor aplicațiilor și posibilitatea de integrare cu Eclipse IDE.

Fiecare versiune de Android lansată (nivel API) aduce îmbunătățiri componentelor existente precum si funcționalități noi care sa eficientizeze utilizarea resurselor fizice ale dispozitivelor.

### 1.3 Arhitectura sistemului Android

Sistemul Android dispune de o arhitectură alcătuită din 5 niveluri ce comunica între ele:

- Nucleu Linux (Linux Kernel)
- Librării de bază
- Android RUNTIME
- Aplicații Framework
- Aplicații

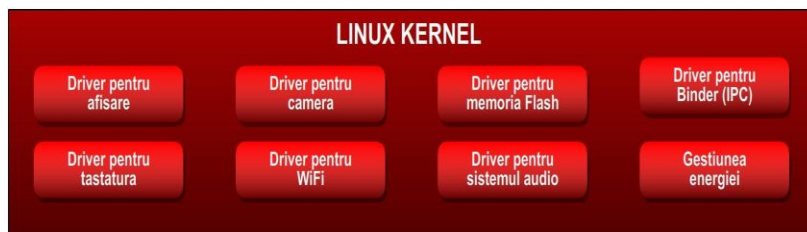


Fig.1.1 Arhitectura sistemului Android



### 1.3.1 Nucleul Linux (Linux Kernel)

Primul nivel al arhitecturii sistemului Android îl constituie *Nucleul Linux*. Acesta se află la baza arhitecturii și asigură funcționalitățile de bază ale sistemului precum gestionarea memoriei, gestionarea proceselor, rețelelor, a sistemului de fișiere și a driverelor (driver pentru afișaj, cameră, memorie Flash, etc).[2]



**Fig. 1.2 Nucleul Linux**

### 1.3.2 Librăriile de bază

Al doilea nivel reprezintă bibliotecile de bază (native) Android și constă dintr-un set de librării C/C++ ce stau la baza funcționării sistemului. Printre acestea se numără bibliotecile responsabile de stocarea și gestionarea bazelor de date (SQLite), biblioteci ce oferă suport pentru formate audio și video (Media Framework), etc.



**Fig. 1.3 Librării de bază**

### 1.3.3 Android Runtime

Android Runtime conține mașina virtuală Dalvik și bibliotecile Java. Mașina Virtuală Dalvik este o componentă principală a acestui nivel ce permite rularea fiecărei aplicații într-un proces propriu.



**Fig.1.4 Android Runtime**

### 1.3.4 Aplicații Framework

Nivelul de aplicații Framework este cel cu care lucrează direct programatorul, oferind dezvoltatorilor toate funcționalitățile și resursele oferite de sistem. Acest nivel este preinstalat în Android și este organizat pe componente pentru a putea extinde și crea noi componente.

Cele mai importante componente sunt:

- Gestiunea activității (Activity Manager): coordonează și controlează ciclul de viață al aplicațiilor.
- Providerul de conținut (Content Provider): este întâlnit doar la arhitectura Android și oferă posibilitatea de a accesa date din alte aplicații.



Fig. 1.5 Aplicații Framework

### 1.3.5 Nivel Aplicații

Nivelul Aplicații reprezintă ultimul nivel din arhitectura sistemului Android și cuprinde toate aplicațiile ce folosesc interfața cu utilizatorul precum contacte, telefon, Browser, etc.

Fiecare aplicație rulează într-un proces propriu, oferind astfel securitate maximă și protecție între aplicații în cazul în care o aplicație se blochează.



Fig.1.6 Nivel Aplicații

## 1.4 Elementele componente ale unei aplicații Android

O aplicație Android este o unitate instalabilă care poate fi pornită și utilizată independent de alte aplicații. Aceasta poate avea o singură clasă care este instanțiată de îndată ce este pornită aplicația și este ultima componentă care este executată la oprirea aplicației.

O aplicație Android este formată din componente software și fișiere de resurse. Componentele unei aplicații Android pot accesa componentele unei alte aplicații pe baza unei descrieri de sarcină (Intent). Astfel se pot crea sarcini executate între aplicații. Integrarea acestor componente se poate face astfel încât aplicația să ruleze impecabil chiar dacă componentele suplimentare nu sunt instalate sau există componente diferite care îndeplinesc aceeași sarcină.

Componentele de bază care sunt folosite pentru construirea unei aplicații sunt:<sup>[3]</sup>

- **Activități (Activity)**

Componentele de tip Activity sunt componente responsabile de prezentarea unei interfețe grafice utilizatorului, înregistrarea și procesarea comenzilor utilizatorului. O aplicație poate avea mai multe componente de tip Activity, una dintre ele fiind considerată activitate principală.

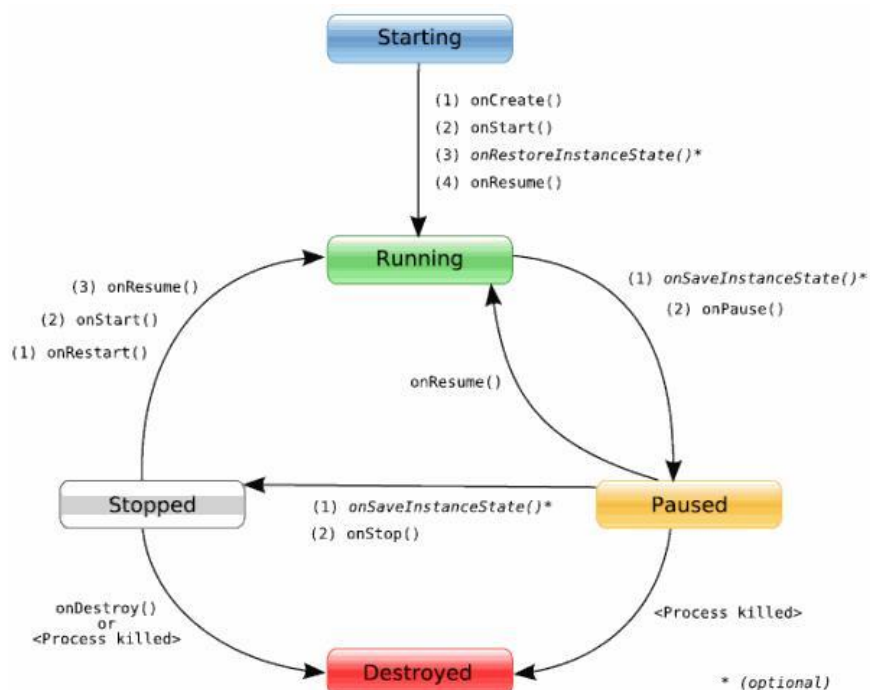
Fiecare obiect de tip Activity are un ciclu de viață care descrie starea în care se află activitatea la un moment dat:

- stare activă (Running). Activitatea este afișată pe ecranul telefonului, utilizatorul interacționează direct cu activitatea prin intermediul interfeței dispozitivului.[3]

- stare de așteptare (Paused). Activitatea nu se mai află în prim plan, utilizatorul nu mai interacționează cu aplicația.

- starea de întrerupere (Stopped). Activitatea nu mai este utilizată și nici nu mai este vizibilă. Pentru a putea fi reactivată, activitatea trebuie să fie repornită.

- starea de distrugere (Destroyed). Activitatea este distrusă și memoria este eliberată deoarece nu mai este necesară.[3]



**Fig.1.7 Ciclul de viață al unei activități**

O singură activitate poate fi afișată în prim-plan la un moment dat. Sistemul este cel care gestionează stările și tranzițiile. Acesta va anunța când se modifică starea activității curente sau este lansată o altă aplicație.[3] Pentru evenimentele de tip tranziție sunt apelate următoarele metode:

- onCreate(Bundle) - este apelată atunci când activitatea este creată. Folosirea argumentului Bundle oferă posibilitatea de a restabili starea salvată într-o sesiune anterioară.

- onStart() - metoda este apelată atunci când activitatea urmează să fie afișată în prim-plan.

- onResume() - este apelată atunci când activitatea este vizibilă, utilizatorul poate interacționa cu aceasta.

- onRestart() - este apelată atunci când activitatea revine în prim-plan dintr-o stare oprită.

- onPause() - metoda este apelată atunci când o altă activitate este adusă în prim-plan, activitatea curentă fiind mutată în fundal.

- onStop() - metoda este apelată atunci când activitatea nu mai este utilizată, utilizatorul interacționând cu altă activitate.

-onDestroy() - apelată atunci când activitatea este distrusă, memoria este eliberată.

-onRestoreInstanceState(Bundle) - apelată în cazul în care activitatea este inițializată cu datele dintr-o stare anterioară.

-onSaveInstanceState(Bundle) - metoda este apelată pentru a salva starea curentă a activității.

#### • **Intent**

Obiectele de tip Intent sunt mesaje asincrone care permit aplicațiilor să solicite funcționalități de la alte componente Android. Cu ajutorul obiectelor de tip Intent, este posibilă comunicarea în timpul rulării cu diverse componente aflate fie în interiorul aplicației, fie localizate în alte aplicații. Printre componentele ce pot fi apelate prin intermediul obiectelor de tip Intent se numără servicii, activități etc.

O activitate poate apela direct o componentă (Intent explicit) sau poate cere sistemului să evalueze componentele înregistrate pe baza datelor din Intent (Intent implicit).

#### • **Servicii**

O componentă de tip Service este o componentă care se execută în fundal, fără interacțiune directă cu utilizatorul și al cărei ciclu de viață este independent de cel al altor componente.[3] Odată pornit, serviciul respectiv își execută în mod implicit în cadrul firului de execuție principal sarcinile pe care le are de făcut chiar dacă componenta care l-a pornit inițial este distrusă. Serviciul este folosit atunci când aplicația are de efectuat o operație de lungă durată care nu interacționează cu utilizatorul sau pentru a furniza funcționalități pentru alte aplicații.

#### • **Furnizorul de conținut (Content Provider)**

O componentă de tip Content Provider este un obiect din cadrul unei aplicații care oferă o interfață structurată la datele aplicației. Cu ajutorul acesteia aplicația poate partaja date cu alte aplicații.[3] Sistemul Android conține o bază de date SQLite în care se pot stoca datele care vor fi accesate cu ajutorul componentelor Content Provider. Datele partajate pot fi imagini, fișiere text, video, audio.

#### • **Receptori de anunțuri (Broadcast Receivers)**

Un obiect de tip receptor de anunțuri este o componentă Android care preia mesaje de tip broadcast. Aceste mesaje pot fi transmise fie de alte aplicații pentru a anunța finalizarea/începerea unei operații, fie de sistem pentru a anunța modificarea parametrilor sistemului (baterie, memorie, semnal, etc.). Mesajele de broadcast sunt de obicei obiecte de tip Intent.

Există două mari clase de mesaje ce pot fi recepționate:

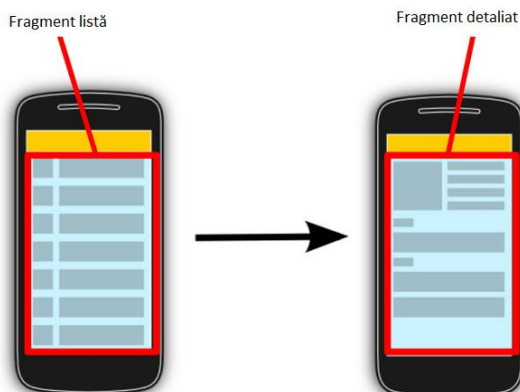
-Mesaje normale (trimise cu context.sendBroadcast). Acestea sunt complet asincrone și sunt transmise într-o ordine aleatoare, de multe ori în același timp. Acest lucru este mai eficient dar rezultatele nu pot fi folosite de receptoare.

-Mesaje comandate (transmise cu Context.sendOrderedBroadcast). Acestea sunt livrate pe rând la un receptor. Pe măsură ce receptorul execută codul, rezultatul poate fi propagat la receptorul următor sau se poate renunța complet la Broadcast și astfel rezultatul nu poate fi transmis către alt receptor. Ordinea în care sunt transmise este controlată de un atribut numit prioritate (android:priority attribute). Receptoarele cu aceeași prioritate vor fi rulate într-o ordine arbitrară.

#### • **Fragment**

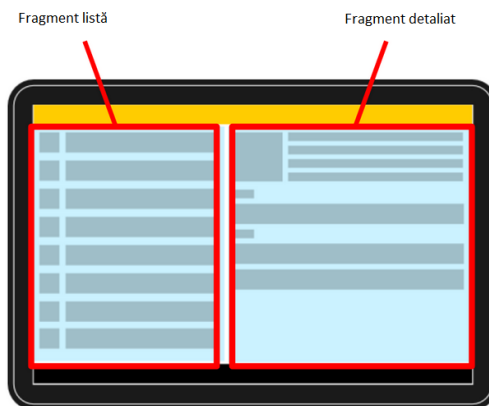
Un fragment reprezintă un comportament sau o porțiune de interfață cu utilizatorul din cadrul unei activități. Se pot combina mai multe fragmente într-o singură activitate pentru a construi o interfață multi-panou sau se poate reutiliza un fragment în mai multe activități.

Imaginea următoare arată o astfel de implementare. Pe un ecran mai mic arată doar un fragment și permite utilizatorului să navigheze printr-un alt fragment.<sup>[4]</sup>



**Fig.1.8 Interfață fragmentată pe ecran mic**

Pe un ecran mai lat sunt afișate imediat cele două fragmente.



**Fig.1.9 Interfață fragmentată pe ecran lat**

Obiectele de tip fragment au propriul ciclu de viață. Un fragment trebuie să fie întotdeauna încorporat într-o activitate, ciclul de viață al fragmentului fiind afectat în mod direct de ciclul de viață al activității gazdă. De exemplu, dacă activitatea este întreruptă atunci și fragmentele incluse în aceasta sunt întrerupte. Ciclul de viață al fragmentului diferă de cel al activității atunci când activitatea se rulează, fiecare fragment putând fi manipulat independent (adăugare, eliminare, modificare de fragment, etc.).

Ca și structură, componenta de tip fragment este foarte similară cu cea de tip Activity, în cadrul lor regăsindu-se pe lângă metode specifice și metodele `onCreate(Bundle)`, `onStart()`, `onResume()`, etc.

- ***Fisierul Manifest***

Fiecare aplicație trebuie să aibă un fișier `AndroidManifest.xml` în directorul principal. Acest fișier conține informații referitoare la toate componentele, permisiile, serviciile și librăriile utilizate în aplicație.<sup>[4]</sup>

Imaginea de mai jos prezintă structura generală a unui fișier manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <uses-permission />
  <permission />
  <permission-tree />
  <permission-group />
  <instrumentation />
  <uses-sdk />
  <uses-configuration />
  <uses-feature />
  <supports-screens />
  <compatible-screens />
  <supports-gl-texture />

  <application>

    <activity>
      <intent-filter>
        <action />
        <category />
        <data />
      </intent-filter>
      <meta-data />
    </activity>

    <activity-alias>
      <intent-filter> . . . </intent-filter>
      <meta-data />
    </activity-alias>

    <service>
      <intent-filter> . . . </intent-filter>
      <meta-data />
    </service>

    <receiver>
      <intent-filter> . . . </intent-filter>
      <meta-data />
    </receiver>

    <provider>
      <grant-uri-permission />
      <meta-data />
      <path-permission />
    </provider>

    <uses-library />
  </application>
</manifest>
```

**Fig.1.10 Fișierul Manifest**

O permisiune reprezintă o restricție care limitează accesul la date sau la o parte din cod pentru a proteja datele confidențiale ale utilizatorului. Fiecare permisiune este identificată printr-o etichetă unică. De multe ori eticheta indică acțiunea care este limitată.

#### • **Procese și fire de execuție**

La pornirea unei aplicații se crează automat un proces Linux cu un singur fir de execuție numit fir principal de execuție (main thread). În cadrul acestui proces sunt executate toate componentele și instrucțiunile asociate acestora. În cazul în care o componentă este pornită când există deja un proces principal atunci componenta va rula în procesul deja existent. În cazul în care aplicația are de efectuat o operație de lungă durată sau o operație care ar afecta performanțele acesteia se asociază un nou proces care să îndeplinească respectivul set de instrucțiuni, creând astfel fire suplimentare de execuție pentru orice proces.

Procesele sunt executate pe o perioadă mai lungă de timp, ele fiind oprite atunci când sistemul are nevoie de memorie sau are de executat procese cu o prioritate mai mare. Pentru a determina ce proces trebuie oprit, sistemul organizează toate procesele în funcție de importanță.

#### • **Resurse Android**

O aplicație Android este alcătuită din fișiere ce conțin codul sursă și fișiere cu resurse. Resursele sunt separate de codul sursă și reprezintă o colecție de fișiere video, audio, imagini, text folosite pentru a crea o interfață vizuală cât mai bogată. Accesarea acestora se face prin intermediul codului sursă. Separarea resurselor permite dezvoltatorului să creeze interfețe adaptate la diferitele configurații de dispozitive.

## CAPITOLUL 2

### Instrumente de dezvoltare Android

#### 2.1 Eclipse IDE

Eclipse este un mediu de dezvoltare integrat (IDE) utilizat pentru a dezvolta aplicații scrise în cea mai mare parte în Java.

Cea mai mică unitate funcțională a platformei Eclipse care poate fi dezvoltată și transmisă separat se numește plug-in. Aceste plug-in-uri sunt folosite pentru a oferi toate funcționalitățile necesare. Cu excepția unui mic nucleu Runtime, totul în Eclipse este plug-in. Acest lucru permite o îmbunătățire constantă a codului deja existent deoarece fiecare plug-in nou dezvoltat se integrează perfect. Eclipse oferă o mare varietate de plug-in-uri utilizate pentru a oferi numeroase facilități și opțiuni.

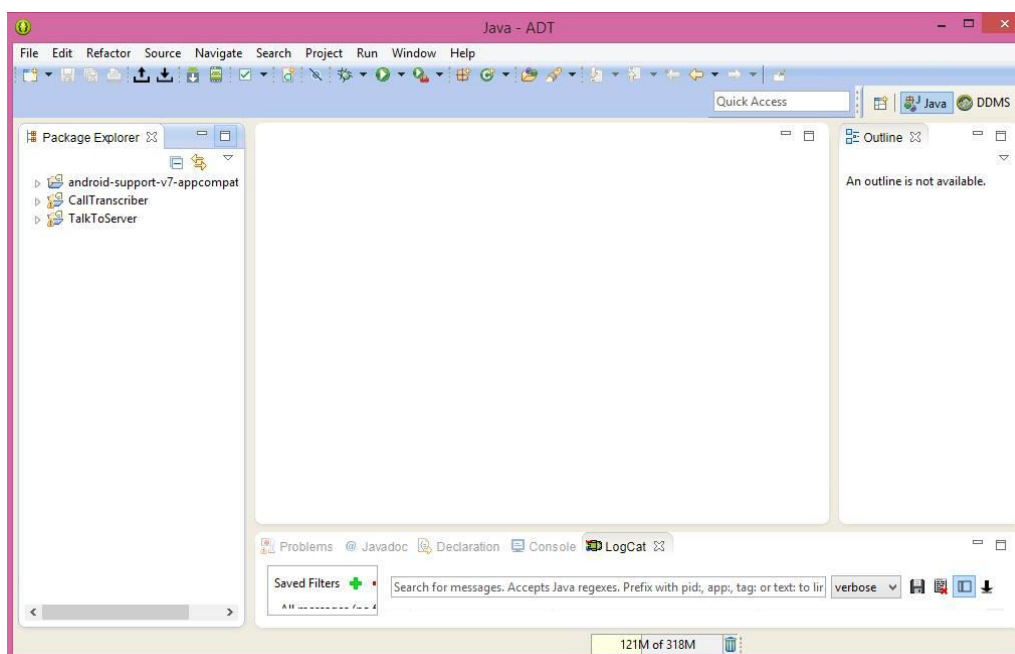


Fig. 2.1 Eclipse IDE

#### 2.2 Android SDK

Android Software Development este procesul prin care sunt create noi aplicații pentru sistemul de operare Android. Aplicațiile sunt scrise în limbajul de programare Java cu ajutorul kitului de dezvoltare software Android (Android SDK).

Android SDK conține toate instrumentele necesare pentru a crea și compila o aplicație Android.[4] Pe lângă acestea, Android SDK mai pune la dispoziția dezvoltatorilor un emulator de telefon, mostre de coduri, tutoriale pentru începători și instrumente de depanare. Platformele de dezvoltare sprijinite în prezent includ calculatoare care rulează Linux, Mac și Windows (XP sau variantă mai nouă).

Android SDK sprijină și dezvoltarea de aplicații compatibile cu versiuni mai vechi ale platformei Android. Instrumentele de dezvoltare sunt componente puse la dispoziția dezvoltatorilor

tot timpul, astfel și versiunile mai vechi pot fi descărcate și utilizate pentru a testa aplicația. Există o platformă SDK pentru fiecare versiune de Android lansată, care poate fi aleasă ca și platformă țintă pentru aplicație.

Aplicațiile Android sunt arhivate în format .apk și depozitate în folderul “/data/app “ (folder care nu este accesibil decât utilizatorilor care au acces la root). Pachetul .apk conține fișierele .dex (executabile Dalvik), resurse etc.

Android SDK conține :

- **Android API:** Nucleul SDK-ului îl reprezintă bibliotecile API Android care permit dezvoltatorilor să acceseze stiva Android. Aceste biblioteci sunt cele folosite de Google la dezvoltarea aplicațiilor native.
- **Instrumente de dezvoltare:** Android SDK conține mai multe instrumente de dezvoltare care permit dezvoltatorilor să compileze și să depaneze aplicații.
- **Emulator Android:** Emulatorul Android este un dispozitiv complet interactiv care simulează configurația hardware a dispozitivului. Cu ajutorul emulatorului se poate observa modul în care aplicația se va comporta pe un dispozitiv Android.
- **Documentație completă:** SDK-ul include informații detaliate, referințe de cod și instrucțiuni de utilizare a claselor și explicații referitoare la ideile de bază ale dezvoltării Android.
- **Suport online Android:** Datorită platformei open-source, s-a dezvoltat rapid o comunitate de dezvoltatori activă prin intermediul căreia se discută mereu idei noi și se oferă suport tuturor membrilor.

### 2.3 Android Development Tool (ADT)

Android Development Tool (ADT) este un plug-in pentru Eclipse IDE, proiectat pentru a oferi un mediu puternic, integrat pentru a dezvolta aplicații. ADT extinde capacitățile oferite de Eclipse pentru a permite crearea rapidă de noi proiecte, depanarea și exportarea de proiecte deja existente.[4]

Dezvoltarea aplicațiilor în Eclipse cu ADT este cel mai rapid mod de dezvoltare pentru începători deoarece oferă instrumentele necesare pentru a edita fișiere XML și depanare.

### 2.4 Mașina Virtuală Dalvik

Unul din elementele cheie ale sistemului Android este mașina virtuală Dalvik. Mașina virtuală Dalvik este un mediu software proiectat astfel încât să permită multiplelor instanțe să ruleze eficient pe un singur dispozitiv, fiind astfel o parte integrată a sistemului Android. Programele sunt de obicei scrise în Java, compilate în bytecode pentru mașina virtuală, apoi traduse în bytecode Dalvik și stocate în fișier .dex (executabil dalvik).

Dalvik este o mașină virtuală personalizată, pe bază de registru, proiectată pentru sistemele care sunt limitate în ceea ce privește memoria, bateria și viteza procesorului. Aceasta folosește nucleul Linux al dispozitivului pentru a realiza funcționalități low-level legate de securitate, fire de execuție și managementul memoriei și proceselor.



## CAPITOLUL 3

### Sisteme de recunoaștere automată a vorbirii continue

#### 3.1 Introducere în recunoașterea automată a vorbirii

Recunoașterea automată a vorbirii (RAV) este un proces care vizează transformarea unui semnal audio ce conține vorbire într-o succesiune de cuvinte. Textul format cu aceste cuvinte trebuie să reproducă cât mai bine conținutul fișierului audio transcris. Procesul de recunoaștere a vorbirii își propune să producă informații de natură semantică, să producă propoziții cu sens, nu doar o înșiruire de cuvinte. [5]

Arhitectura generală a unui sistem de recunoaștere automată a vorbirii este prezentată în figura următoare:

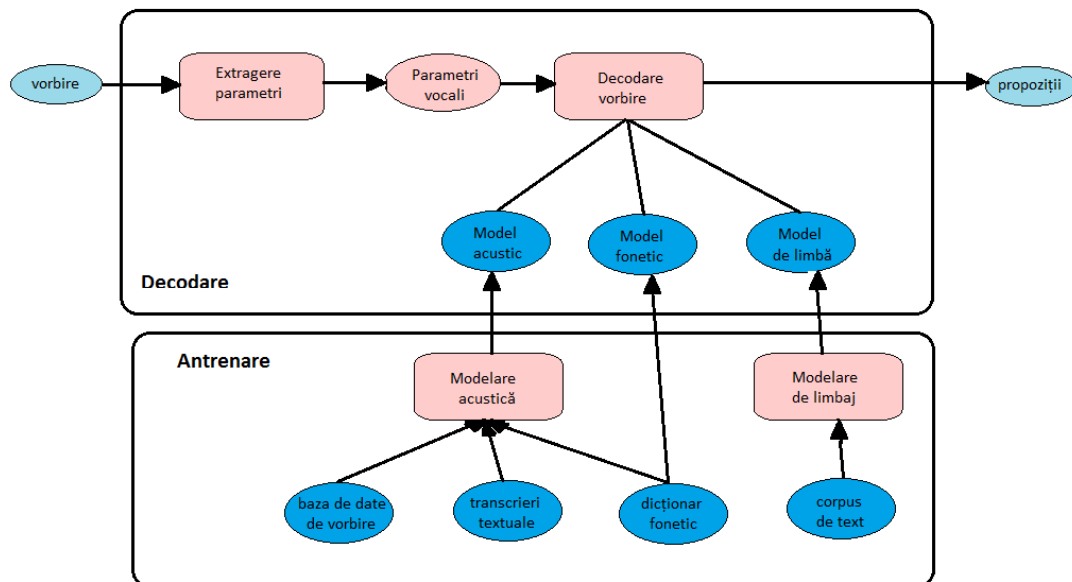


Fig. 3.1 Arhitectura generală a unui sistem de RAV

Procesul de recunoaștere automată a vorbirii utilizează o serie de parametri extrași din semnalul vocal și modele acustice, fonetice și lingvistice deja dezvoltate.

Modelul acustic se ocupă de estimarea probabilității mesajului vorbit având ca intrare o succesiune de cuvinte. Acest model utilizează unități acustice de bază sub-lexicale (foneme) sau unități sub-fonetice (senone) în locul cuvintelor deoarece există un număr prea mare de cuvinte diferite într-o limbă pentru care nu există modele deja antrenate și nici date de antrenare disponibile. Astfel modelul acustic este format dintr-un set de modele pentru foneme (sau senone) care se combină în timpul procesului de decodare pentru a forma modele pentru cuvinte și apoi modele pentru succesiuni de cuvinte.[5]

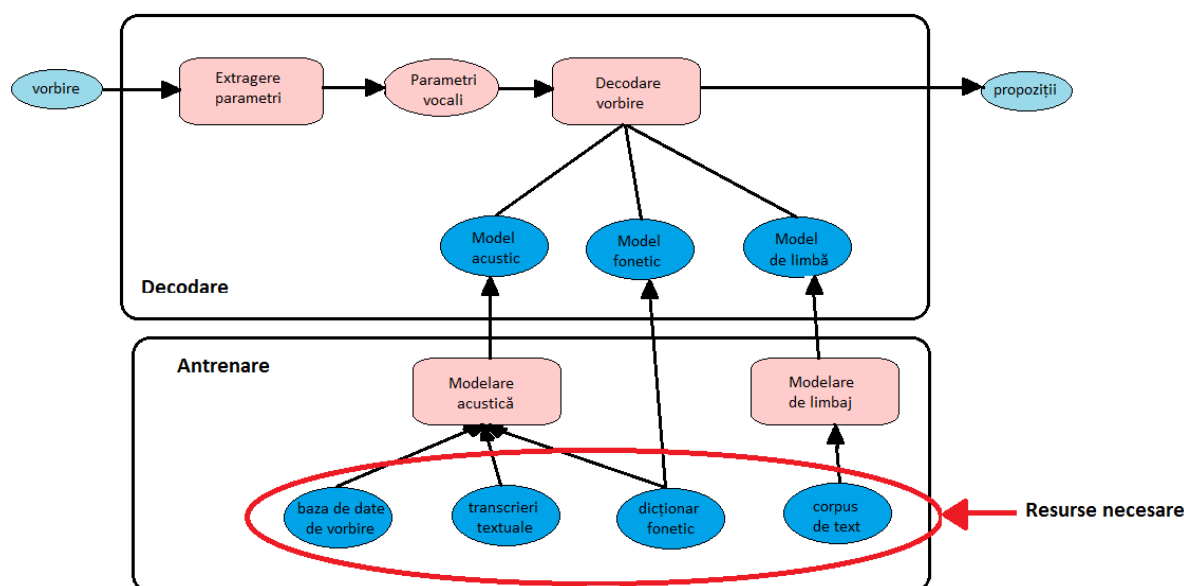
Modelul acustic se construiește pe baza unei colecții de fișiere audio înregistrate. Fiecare clip audio are asociat o transcriere text a mesajelor vorbite și un dicționar fonetic ce cuprinde toate cuvintele regăsite în transcriere. În cazul sistemelor cu vocabular extins se folosesc corpusuri de text cu dimensiuni cât mai mari și cât mai adaptate la domeniul din care fac parte mesajele vocale. Aceste corpusuri sunt utilizate pentru a crea modele de limbă statistice.[5]

Modelul de limbă este folosit pentru a estima probabilitatea ca o succesiune de cuvinte să alcătuiască o propoziție validă a limbii. Utilizând acest model se asociază fiecărei succesiuni de cuvinte o probabilitate și în funcție de aceasta se decide care este fraza cea mai apropiată de fraza vorbită.

Modelul fonetic are rolul de a uni modelul acustic cu modelul de limbă, acesta fiind, de cele mai multe ori, un dicționar de pronunție care asociază fiecărui cuvânt din vocabular una sau mai multe secvențe de foneme.

### 3.2 Resurse necesare în construcția unui sistem de recunoaștere a vorbirii continue

Sistemele de recunoaștere a vorbirii continue (RVC) transformă semnalul vocal în text cu ajutorul modelelor (acustic, fonetic și lingvistic) dezvoltate în prealabil.



**Fig. 3.2 Resurse necesare în construcția unui sistem RVC**

Modelul acustic care modelează fonemele limbii se construiește pe baza unui set de clipuri audio înregistrate, a transcrierilor aferente și a unui dicționar fonetic care să specifice modul de pronunție al cuvintelor din transcrierile textuale.

Fonemul este unitatea de sunet fundamentală care ajută la diferențierea cuvintelor. Prin modificarea unui singur fonem al unui cuvânt se generează fie un cuvânt cu sens diferit, fie un cuvânt inexistent. În limba română se folosesc 7 vocale de bază și două împrumutate, 4 semi-vocale și 22 de consoane.[5]

Dicționarul fonetic stă și la baza dezvoltării modelului fonetic utilizat în procesul de decodare. În cazul în care un cuvânt permite mai multe pronunții atunci acestea sunt comasate formând modelul de pronunție al respectivului cuvânt.

Fonemul			Exemple de cuvinte	
Tip	Simbol IPA	Simbol intern	Forma scrisă	Forma fonetică
vocale	a	a	sat	s a t
	e	e	mare	m a r e
	i	i	lift	l i f t
	o	o	loc	l o c
	u	u	șut	s l u t
	ə	a1	gură	g u r a1
	ɨ	i2	între	i2 n t r e
vocale împrumutate	ʏ	y	ecru	e c r y
	ø	o2	bleu	b l o2
semivocale	ɛ	e1	deal	d e1 a1
	j	i3	fiară	f i3 a r a1
	ɔ	o1	oase	o1 a s e
	w	w	sau	s a w
consoane	c	k2	chem	k2 e m
	b	b	bar	b a r
	p	p	par	p a r
	k	k	acum	a k u m
	tʃ	k1	cenușă	k1 e n u s i a1
	g	g	galben	g a l b e n
	ʒ	g1	girafă	g1 i r a f a1
	ʒ	g2	unghi	u n g2
	d	d	dar	d a r
	t	t	tot	t o t
	f	f	fața	f a t i a
	v	v	vapor	v a p o r
	h	h	harta	h a r t a
	ʒ	j	ajutor	a j u t o r
	ʃ	s1	coș	k o s1
	l	l	lac	l a c
	m	m	măr	m a i r
	n	n	nas	n a s
	s	s	sare	s a r e
	z	z	zar	z a r
	r	r	risc	r i s k
	ts	t1	țăran	t1 a i r a n
consoană palatalizată	ʎ	i1	tari	t a r i1

**Fig 3.3 Fonemele limbii române [5]**

Un dicționar fonetic este un instrument lingvistic care specifică modul în care se pronunță cuvintele unei limbi, face corespondența între forma scrisă și cea fonetică. În sistemul de recunoaștere a vorbirii continue, dicționarul fonetic are rolul de a face legătura între modelul acustic și modelul de limbă, astfel că acesta trebuie să conțină toate cuvintele și transcrierile fonetice utilizate într-o anumită sarcină de recunoaștere.

Baza de date de vorbire este folosită la antrenarea modelului acustic și cuprinde următoarele componente:

- un set de clipuri audio ce conțin vorbire salvate în format .wav;
- un set de fișiere text ce conțin transcrierile textuale ale clipurilor audio ;
- informații suplimentare privind stilul și domeniul vorbirii;

Baza de date de vorbire reprezintă un element important în aprecierea performanțelor sistemului de recunoaștere a vorbirii. Calitatea acesteia este evaluată în funcție de stilul vorbirii (cuvinte izolate, vorbire continuă citită, vorbire convențională), dimensiunea bazei de date (număr de ore vorbite, număr de vorbitori) și de variabilitate (calitate înregistrări, zgomot de fundal, variabilitatea vorbitorilor).[5]

Achiziționarea unei baze de date de vorbire se poate face fie prin înregistrarea unor texte predefinite, fie prin etichetarea unor materiale audio ce conțin vorbire.

Modelul de limbă utilizat într-un sistem de recunoaștere a vorbirii continue utilizează un corpus de text de dimensiuni mari din care se extrag statisticile caracteristice limbii. Aceste statistici sunt utilizate în procesul de decodare pentru a atribui probabilități diverselor cuvinte și secvențe de cuvinte propuse de modelul acustic.

Sistemele de recunoaștere a vorbirii continue utilizează modele statistice pentru a modela pronunția fonemelor, probabilitățile de apariție ale cuvintelor și succesiunilor de cuvinte dintr-o limbă. Pentru antrenarea acestor modele este necesară achiziționarea unei cantități mare de date reprezentative pentru fonemul ce trebuie modelat, un dicționar fonetic care să specifice modul de pronunție al cuvintelor limbii și corpusuri de text pentru modelarea statisticii apariției cuvintelor.

### 3.3 Aplicații ale sistemelor de recunoaștere a vorbirii

Recunoașterea vorbirii are o gamă largă de aplicabilitate. Aceste aplicații se pot grupa în 3 categorii mari:

- Dictarea: Aceasta este cea mai evidentă aplicație a sistemelor de recunoaștere a vorbirii având ca scop traducerea unui semnal vocal. În cele mai multe cazuri, la dictare se consideră că materialul care urmează a fi citit este pregătit dinainte, limba folosită este cea literară, iar condițiile de înregistrare și calitatea achiziționării semnalului vocal sunt bune.

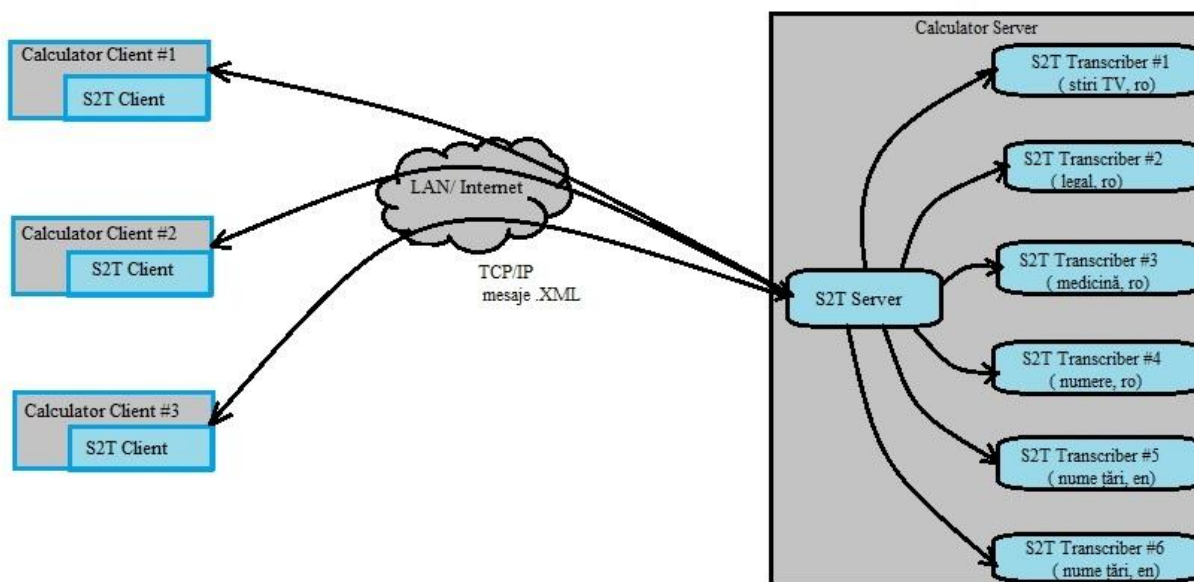
- Indexarea audio: Indexarea audio presupune transcrierea și indexarea materialului audio înregistrat la conferințe, la emisiuni radio/TV, etc. În acest caz limbajul vorbit folosit este neuniform, tinde spre vorbire spontană.[6]

- Dialog om-mașină: Cele mai simple aplicații care se găsesc pe piață sunt similare aplicațiilor bazate pe răspunsuri prin DTMF (din engleză Dual Tone Multi Frequency). Acestea necesită un nivel scăzut de înțelegere a limbajului natural și un vocabular mic.[6] Un exemplu de o astfel de aplicație este navigarea printr-un meniu. Există și aplicații bazate pe dialog care folosesc vocabulare mari. Alte aplicații sunt apelarea prin nume, informații despre starea vremii, nume, adrese și numere de telefon. Aceste aplicații pot fi servicii bazate pe telefonie. În cazul acestor aplicații nu se poate impune dinainte calitatea microfonului și a sistemului de achiziție a semnalului vocal, pot apărea probleme datorate benzii înguste a semnalului vocal și a perturbațiilor în canalul de telecomunicații, astfel că sistemul trebuie să fie cât mai robust pentru a putea realiza recunoașterea în condiții variate.

O aplicație de recunoaștere a vorbirii o reprezintă soluția software de transcriere Speech-to-Text (S2T) creată de Speech and Dialogue Research Laboratory. Aceasta transformă vorbirea dintr-un fișier sau dintr-un stream audio în text. În prezent, sistemul permite procesarea fișierelor în limba română și în limba engleză.

Arhitectura acestei soluții este de tip client-server. Aplicațiile S2T-Client și S2T-Server se pot afla pe sisteme hardware diferite, dar trebuie să comunice prin intermediul unei rețele locale (LAN) sau prin intermediul Internetului. Aplicația client poate fi dezvoltată prin orice tehnologie care permite comunicarea prin socket-uri TCP-IP cu aplicația server, comunicația realizându-se printr-un protocol bazat pe mesaje XML.

Aplicația S2T – Server poate fi configurată pentru a transforma în text diverse tipuri de vorbire, din diverse domenii sau diferite limbi. Aplicația S2T-Server poate fi configurată să instanțieze mai multe motoare de transcrieri (S2T-Transcriber)[5], fiecare astfel de motor deserving un domeniu de transcriere diferit (ex. știri în limba română, vorbire medicală în limba română, nume de țări pronunțate în engleză, etc.). Serverul poate interacționa cu mai mulți clienți simultan sau în ordinea în care au cerut transcrierea.



**Fig. 3.4 Aplicație S2T**

Motorul de transcriere S2T-Transcriber realizează transcrierea vorbirii în text utilizând modele antrenate anterior: model acustic, modelul de limbă statistic și modelul fonetic.

Motorul de transcriere S2T-Transcriber pentru vorbire continuă în limba română are următoarele caracteristici:

- Transcriere speech-to-text specifică limbii române (transcrierea conține cuvinte cu diacritice, cuvinte separate cu cratimă, nume proprii și acronime românești etc.);
- Transcriere pentru fișiere în format .wav și .mp3;
- identificarea vorbitorului dintr-o listă predefinită (necesită antrenare prealabilă);
- transcriere cu acuratețe bună pentru fișierele cu raport semnal zgomot de până la 15 dB;
- eroare de transcriere de maxim 20% pentru vorbire citită și 30% pentru vorbire spontană, în condiții de liniște;
- timp de procesare mic;
- posibilitatea adaptării la vorbitor și la domeniul de recunoaștere;



## Capitolul 4

### Proiectare și implementare

#### 4.1 Prezentarea aplicației

Aplicația de transcriere a convorbirilor reprezintă o implementare practică pe platformă Android a soluției software de recunoaștere a vorbirii, Transcriere Speech-to-Text (S2T), creată în prealabil de Speech and Dialogue Research Laboratory.

S2T are la bază o arhitectură de tip client-server. Aplicația S2T-Client se poate implementa pe orice platformă ce permite comunicarea prin intermediul unei rețele locale (LAN) sau prin intermediul internetului cu S2T-Server. Am ales implementarea clientului pe o platformă mobilă Android deoarece în decursul ultimilor ani popularitatea telefoanelor mobile ce rulează pe această platformă a avut o creștere rapidă.

Aplicația S2T-Client "CallTranscriber" implementată pe platformă Android are o interfață grafică cât mai simplă dar în același timp prezintă o gamă de funcționalități care permit utilizatorului să urmărească progresul operației de transcriere, să vizualizeze fișierul cu textul conversației telefonice, să asculte conversațiile înregistrate și să oprească/pornească serviciul de înregistrare și transcriere.

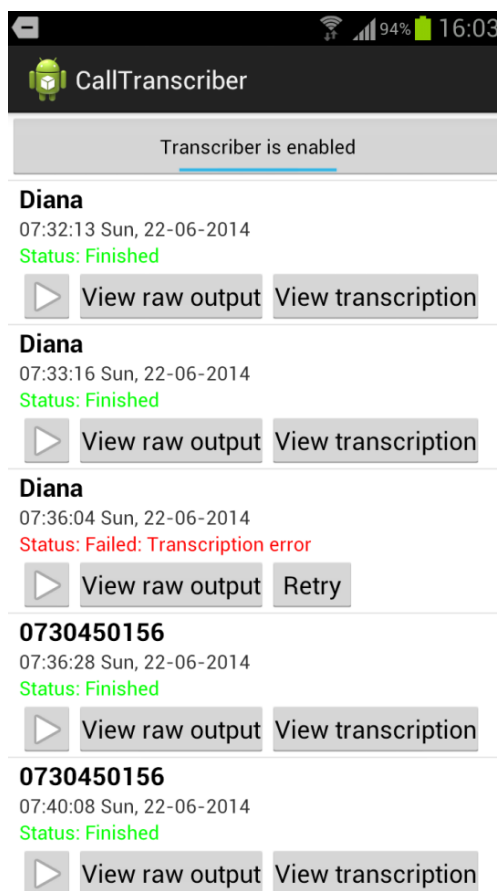
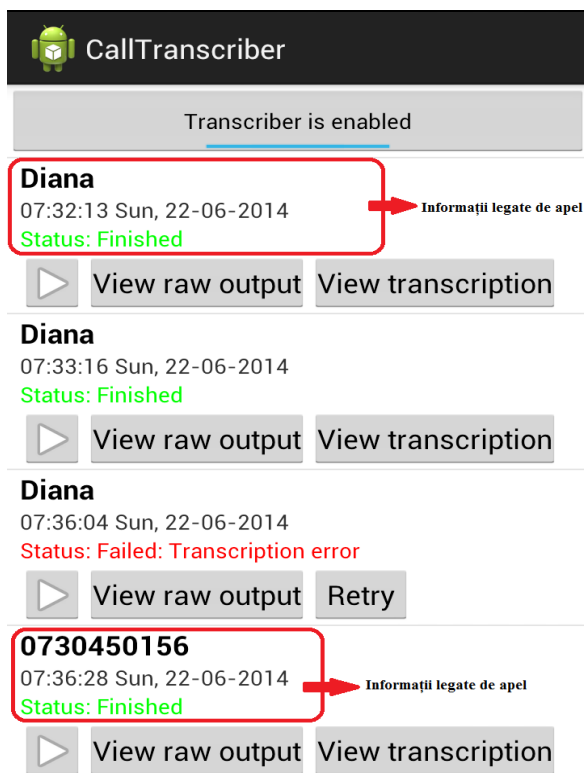


Fig. 4.1 Interfață grafică a aplicației

Aplicația permite utilizatorului să verifice ce apeluri au fost înregistrate și traduse deoarece activitatea principală afișează o listă cu toate informațiile necesare. Pentru fiecare apel este specificat numele apelantului/apelatului în cazul în care acesta este salvat în telefon sau numărul de telefon în caz contrar, data și ora la care a fost efectuat apelul și starea operației de transcriere.



**Fig. 4.2 Activitate principală: Informații apel**

Starea operației de transcriere (Status) poate avea mai multe valori în funcție de stadiul în care se află aplicația. Acesta poate lua următoarele valori:

- 0 – Starting;
- 1 – Pending;
- 2 – Failed: Authentication error;
- 3 – Failed: Transcription error (server error);
- 4 – Failed: No internet connection;
- 5 – Failed: Incomplete transcription (server error);
- 6 – Finished;

Câmpul "Status" este folosit pentru a semnaliza utilizatorului evoluția procesului de transcriere.

```
switch (cursor.getInt(statusIndex)) {
    case TranscriptionProvider.STATUS_STARTING:
        statusText = "Starting";
        color = Color.CYAN;
        break;
```



```

case TranscriptionProvider.STATUS_PENDING:
    statusText = "Pending";
    color = Color.BLUE;
    break;
case TranscriptionProvider.STATUS_DONE:
    statusText = "Finished";
    showTranscription = true;
    break;
case TranscriptionProvider.STATUS_FAILED_AUTHENTICATION:
    statusText = "Failed: Authentication error";
    showRetry = true;
    color = Color.RED;
    break;
case TranscriptionProvider.STATUS_FAILED_INTERNET:
    statusText = "Failed: No internet connection";
    showRetry = true;
    color = Color.RED;
    break;
case TranscriptionProvider.STATUS_FAILED_TRANSCRIPTION:
    statusText = "Failed: Transcription error";
    showRetry = true;
    color = Color.RED;
    break;
case TranscriptionProvider.STATUS_INCOMPLETE_TRANSCRIPTION:
    statusText = "Failed: Incomplete transcription";
    showRetry = true;
    color = Color.RED;
    break;
}

status.setText("Status: " + statusText);
status.setTextColor(color);

```

În codul prezentat mai sus se poate observa că mesajul câmpului Status și culoarea cu care este afișat acesta se schimbă în funcție de rezultatul transcrierii. Culoarele folosite sunt:

- verde în cazul în care transcrierea a fost realizată cu succes;
- albastru pentru cazul când transcrierea este în curs;
- cyan este folosită pentru faza de inițiere a transcrierii;
- roșu pentru cazul în care apar erori și transcrierea nu este realizată;

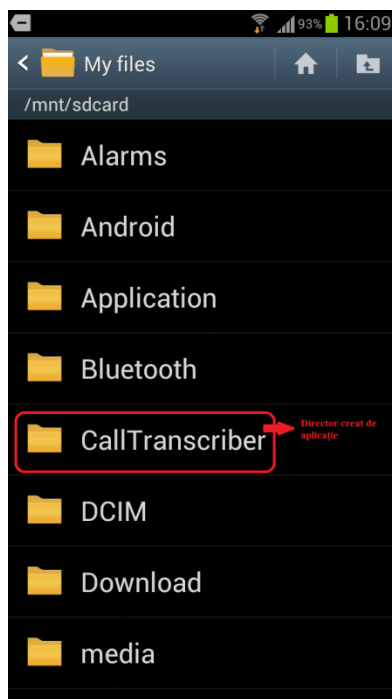
Pentru a oferi utilizatorului posibilitatea de a opri sau de a porni serviciul de transcriere și înregistrare a apelului telefonic am adăugat și un buton de tip On/Off. Pentru a permite serviciului să își seteze preferințele, la prima instalare a aplicației, butonul este ascuns și activat. După primul apel acest buton devine vizibil și funcțional.

```
// Let's just make sure the service didn't crash without setting the preference
appropriately

    if (mTranscriberToggler.isChecked())
        getActivity().startService(new Intent(getActivity(),
            TranscriberService.class));

// Set up toggling capabilities
mTranscriberToggler.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        Intent transcriber = new Intent(getActivity(), TranscriberService.class);
        if (isChecked) getActivity().startService(transcriber);
        else getActivity().stopService(transcriber);
    }
});
```

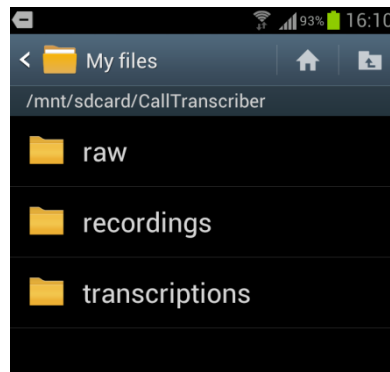
Atunci când serviciul este activat și este detectat un apel se crează un director în memoria telefonului în care sunt stocate toate fișierele necesare aplicației. Acest director se crează la prima apelare a aplicației sau dacă acesta a fost șters de către utilizator.



**Fig. 4.3** Directorul creat de aplicație în memoria telefonului

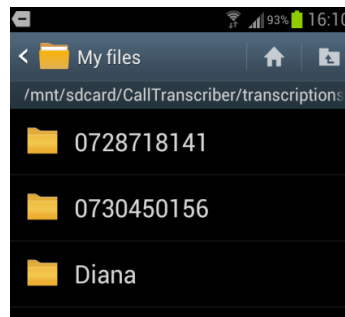
În interiorul directorului principal "CallTranscriber" se crează o listă de directoare:

- recordings – stochează toate clipurile audio ale convorbirilor înregistrate în format .wav;
- transcriptions – stochează transcrierile convorbirilor primite de la server în format .txt;
- raw – stochează toate informațiile primite de la server printre care și transcrierile conversațiilor;



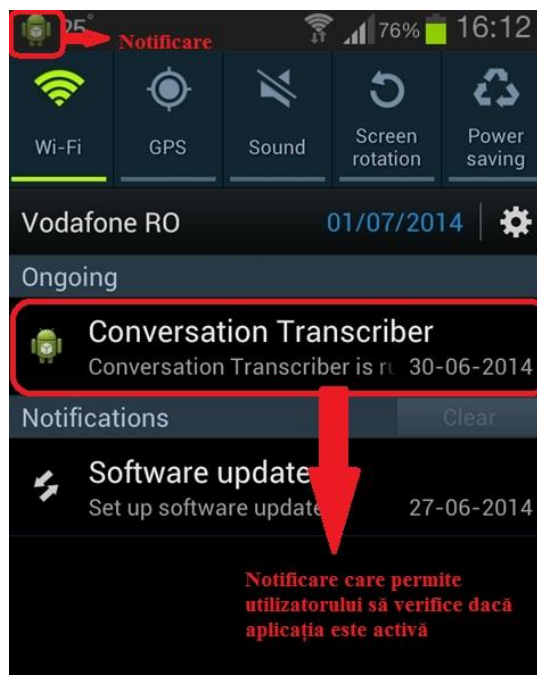
**Fig. 4.4 Lista cu directoare în interiorul directorului principal**

Directoarele enumerate mai sus sunt, la rândul lor, împărțite în directoare pentru a-i permite utilizatorului să găsească mai ușor transcrierea sau clipul audio căutat. Aceste directoare sunt denumite după contact și conțin doar fișierele generate de acesta.



**Fig. 4.5 Directoarele generate de contacte**

Un alt element care este vizibil atunci când serviciul este activ îl reprezintă apariția unei notificări în Status Bar. Această notificare permite serviciului să ruleze în background ca un serviciu de foreground. Deoarece serviciul are prioritatea unei aplicații de foreground, acesta nu poate fi oprit de Runtime atunci când aplicația nu mai este în prim plan (de exemplu atunci când se efectuează un apel). Dacă se face click pe notificare se deschide aplicația, activitatea principală.



**Fig. 4.6 Bara de notificare**

Pentru a crea notificarea care apare în Status Bar am utilizat următorul cod:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.ic_launcher)

    .setContentTitle(resources.getString(R.string.notification_title))

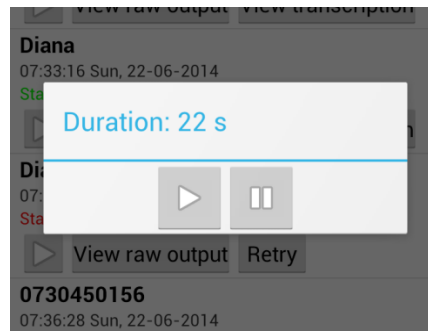
    .setContentText(resources.getString(R.string.notification_message));
startForeground(ONGOING_NOTIFICATION_ID, mBuilder.build());
```

Fiecare intrare în lista prezentată în activitatea principală este însoțită de o serie de funcționalități, funcționalități care facilitează accesul utilizatorului la resursele aplicației.



**Fig.4.7 Butoane funcții**

Una dintre aceste funcționalități o reprezintă accesarea clipului audio direct din ecranul aplicației principale. Atunci când este apăsat butonul "Play" se deschide o fereastră de dialog în care este afișată durata clipului audio și două butoane media player care permit redarea audio.



**Fig. 4.8 Fereastră media player**

Pentru a implementa această funcționalitate am creat o clasă separată, "PlayAudioDialogFrament", al cărei constructor primește ca parametru adresa clipului audio ce trebuie redat.

```
static PlayAudioDialogFrament newInstance(String audioFilePath) {
    PlayAudioDialogFrament f = new PlayAudioDialogFrament();
    // Supply audio file path as an argument
    Bundle args = new Bundle();
    args.putString("path", audioFilePath);
    f.setArguments(args);
    return f;
}
```

La fiecare apel se crează o instanță de tip MediaPlayer cu ajutorul căreia se redă înregistrarea dorită din directorul cu clipuri audio. În cazul în care clipul nu există se afișează un mesaj de eroare, "Cannot play audio file!".

```

mAudioFilePath = getArguments().getString("path");
// prepare the media player (might take some time)
mMediaPlayer = new MediaPlayer();
mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
try {
    mMediaPlayer.setDataSource(getActivity().getApplicationContext(),
        Uri.parse("file:/// " + mAudioFilePath));
    mMediaPlayer.prepare();
} catch (Exception e) {
    Toast.makeText(getActivity(), "Cannot play audio file!",
        Toast.LENGTH_SHORT).show();
    getDialog().dismiss();
    return;
}

```

Codul următor evidențiază modul de construire a interfeței corespunzătoare ferestrei de redare și a opțiunilor oferite de aceasta.

```

View v = inflater.inflate(R.layout.audio_play, container, false);

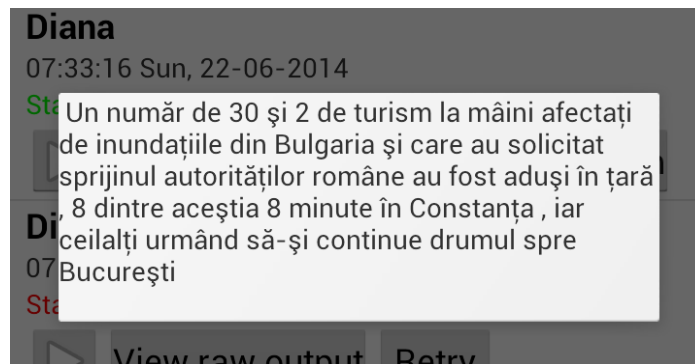
getDialog().setTitle("Duration: " + (new SimpleDateFormat("ss").format(new
Date( mMediaPlayer.getDuration())) + " s"));

ImageButton play = (ImageButton) v.findViewById(R.id.play);
ImageButton pause = (ImageButton) v.findViewById(R.id.pause);
// play audio
play.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mMediaPlayer.start();
    }
});
// pause audio
pause.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        mMediaPlayer.pause();
    }
});
return v;

```

Butonul "View raw output" permite utilizatorului să vizualizeze protocolul de comunicare între server și telefon. Această funcționalitate îi oferă dezvoltatorului o modalitate de debug astfel încât acesta poate verifica de unde apar eventuale erori în procesul de transcriere.

Butonul "View transcription" permite utilizatorului să verifice rezultatul transcrierii deoarece atunci când este apăsat se deschide o fereastră de dialog ce conține fisierul text aferent clipului audio. Acest buton este afișat doar în momentul în care transcrierea este completă, în caz contrar este afișat un buton "Retry" care îi oferă utilizatorului posibilitatea de a încerca să stabilească o altă conexiune la server.



**Fig. 4.9 Fereastră transcriere**

**(Text rostit: ”Un număr de 32 de turiști români afectați de inundațiile din Bulgaria și care au solicitat sprijinul autorităților române au fost aduși în țară, 8 dintre aceștia oprindu-se în Constanța, iar ceilalți urmând să-și continue drumul spre București.”)**

Pentru fereastra de afișare a transcrierilor și a răspunsului de la server (raw output) am utilizat clasa "ViewerFragment" creată special pentru afișarea unui mesaj text. Constructorul acestei clase primește ca parametru adresa fișierului text în care este stocat mesajul.

```
static ViewerFragment newInstance(String filePath) {
    ViewerFragment f = new ViewerFragment();
    // Supply file path as an argument
    Bundle args = new Bundle();
    args.putString("path", filePath);
    f.setArguments(args);
    return f;
}
```

Se verifică existența fișierului, se citește linie cu linie conținutul acestuia și se afișează textul într-o fereastră specială. Fereastra se dimensionează în funcție de dimensiunea textului, astfel că pentru texte de dimensiuni mari am implementat un ScrollView.

```
if (file.exists()) {
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(new FileReader(file));
        String line = null;
        while ((line = reader.readLine()) != null) {
            message.append(line).append("\n");
        }
    } catch (Exception e) {
        message.append("Exception reading " + mFilePath + ": " + e);
    } else
        message.append("No such file: " + mFilePath);
}
```

```
ScrollView scroller = new ScrollView(getActivity());
scroller.setLayoutParams(new FrameLayout.LayoutParams(
    FrameLayout.LayoutParams.MATCH_PARENT,
    FrameLayout.LayoutParams.MATCH_PARENT));
TextView text = new TextView(getActivity());
```

```

text.setText(message.toString());
scroller.addView(text, new FrameLayout.LayoutParams(
    FrameLayout.LayoutParams.MATCH_PARENT,
    FrameLayout.LayoutParams.MATCH_PARENT));
// Create the dialog
return new AlertDialog.Builder(getActivity()).setView(scroller).create();

```

## 4.2 Descrierea codului sursă

Componenta principală a acestei aplicații este reprezentată de clasa "TranscriberService". Acesta este un serviciu de background ce rulează ca foreground. În acest fel se garantează că serviciul poate rula non-stop cu prioritatea unei aplicații care este în foreground, aplicația nu va fi oprită de Runtime atunci când trece în background. Pentru a putea manevra cantități mari de informații serviciul rulează în propriul proces. Acest lucru se observă în fișierul `AndroidManifest.xml`:

```
android:process=":transcriber"
```

TranscriberService are următoarele responsabilități:

- Înregistrează apelurile de intrare/de ieșire;
- Salvează clipurile audio în memoria telefonului;
- Trimite înregistrările la server și preia transcrierile transmise de acesta;
- Salvează informațiile primite de la server și transcrierile în memoria telefonului;

Pentru a monitoriza ciclul de viață a unui apel am folosit `PhoneStateListener`:

```

mPhoneStateListener = new PhoneStateListener() {
    public void onCallStateChanged(int state, String incomingNumber) {
        switch (state) {
            case TelephonyManager.CALL_STATE_OFFHOOK:
                startRecordingCall();
                break;
            case TelephonyManager.CALL_STATE_IDLE:
                stopRecordingCall();
                break;
            case TelephonyManager.CALL_STATE_RINGING:
                setupIncomingCall(incomingNumber);
                break;
        }
    }
}

```

Din codul de mai sus se pot observa cele 3 stări posibile ale telefonului:

- În apel: Când telefonul se află în această stare începe funcția de înregistrare a apelului;
- În repaus: Telefonul nu este angajat în apel;
- În timp ce sună: În această stare se preia numărul apelantului;

Pentru a prelua numărul celui apelat am folosit `OutgoingCallReceiver` deoarece `PhoneStateListener` nu permite captarea numărului în cazul unui apel de ieșire.

```

mOutgoingCallReceiver = new OutgoingCallReceiver();
registerReceiver(mOutgoingCallReceiver, new IntentFilter(
    Intent.ACTION_NEW_OUTGOING_CALL));

```

```

mTelephonyManager = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
mTelephonyManager.listen(mPhoneStateListener,
    PhoneStateListener.LISTEN_CALL_STATE);

```

Atunci când telefonul este în apel (CALL\_STATE\_OFFHOOK) începe înregistrarea apelului prin apelarea metodei `startRecordingCall()` care instanțiază o componentă de tip `AudioRecordTask(String contact)`. Aplicația este setată implicit pentru a înregistra ambele sensuri ale convorbirii (VOICE\_CALL).

```

mAudioRecorder = new AudioRecord(MediaRecorder.AudioSource.VOICE_CALL,
    RECORDER_SAMPLERATE, RECORDER_CHANNELS,
    RECORDER_AUDIO_ENCODING, mAudioBufferSize);

```

Înregistrarea celor două sensuri ale convorbirii nu este permisă pe toate telefoanele, depinzând de producătorii telefonului. Înregistrarea celor două sensuri presupune preluarea streamului audio direct după linia telefonică pentru a putea astfel înregistra ambi parteneri la apel. În cazul în care telefonul prezintă această limitare, se instanțiază un `AudioRecorder` care înregistrează de la microfon terminalului pe care rulează aplicația preluând astfel textul rostit de una din cele două persoane angajate în apel.

```

mAudioRecorder = new AudioRecord(MediaRecorder.AudioSource.MIC,
    RECORDER_SAMPLERATE, RECORDER_CHANNELS,
    RECORDER_AUDIO_ENCODING, mAudioBufferSize);

```

Datele acumulate de la `AudioRecorder` sunt stocate într-un fișier temporal. Pentru a crea fișierul final în format .wav se crează și de adaugă într-un fișier header-ul WAV și apoi datele din fișierul temporal.

```

fos = new FileOutputStream(mAudioRecordFile);
fos.write(header, 0, 44);
fos.flush();
while (fis.read(data) != -1) { fos.write(data); }
fos.flush();

```

După ce a fost creat fișierul în format .wav, fișierul temporal este șters deoarece toate datele care se aflau în el se regăsesc acum în fișierul final.

```
temp.delete();
```

În momentul în care telefonul trece în starea de repaus (CALL\_STATE\_IDLE) se oprește înregistrarea apelului și se eliberează `AudioRecord`-erul.

```

mIsRecording = false;
mAudioRecorder.stop();
mAudioRecorder.release();

```

Tot în această stare se actualizează baza de date ce conține informațiile legate de convorbirile înregistrate și începe comunicarea cu serverul (`TranscriberTask`).

```

values.put(TranscriptionProvider.KEY_CALLER, mCaller);
values.put(TranscriptionProvider.KEY_TIME, mTime);
values.put(TranscriptionProvider.KEY_AUDIO_FILE,
    mAudioRecordFile.getAbsolutePath());
values.put(TranscriptionProvider.KEY_TRANSCRIPTION, "-");
values.put(TranscriptionProvider.KEY_TRANSCRIPTION_FILE,
    mTranscriptionFile.getAbsolutePath());

```



```

values.put(TranscriptionProvider.KEY_RAW_OUTPUT,
           mRawOutputFile.getAbsolutePath());
values.put(TranscriptionProvider.KEY_STATUS,
           TranscriptionProvider.STATUS_STARTING);
long id = Long.parseLong(mContentResolver
                        .insert(TranscriptionProvider.CONTENT_URI, values)
                        .getPathSegments().get(1));

// Start the TranscriberTask for the current transcription
new TranscriberTask(mContentResolver, id).execute(recording,
           mRawOutputFile, mTranscriptionFile);

```

Baza de date (TranscriptionProvider) se ocupă de stocarea informațiilor necesare pentru a realiza transcrierea și de asemenea stabilește o legătură coerentă între TranscriberService și interfața principală. Informațiile care sunt stocate în această bază de date sunt numărul de identificare unic, numărul de telefon/numele interlocutorului, momentul la care a început convorbirea, adresele fișierelor ce conțin clipurile audio, transcrierile și informațiile complete primite de la server și transcrierea finală dacă aceasta a fost realizată.

TranscriberTask este responsabil de comunicarea cu S2T-Server. Această sarcină este realizată în mai multe etape pentru a asigura o funcționare cât mai corectă:

- Deschiderea unui socket către server: În această etapă se crează un socket cu ajutorul adresei și portului asociate serverului și se stabilesc cele două căi de comunicare.

```

socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
outputStream = new XMLOutputStream(socket.getOutputStream());
inputStream = new XMLInputStream(socket.getInputStream());
rawOutputStream = new FileOutputStream(rawOutput);

```

- Autentificare: Pentru a realiza autentificarea se trimite o cerere către server ce conține username-ul și parola și se așteaptă răspunsul acestuia. Dacă răspunsul a fost pozitiv atunci se poate trece la pasul următor.

```

Document requestDocument = XMLBuilder.createAuthenticateRequest(
    username, parola);
transformer.transform(new DOMSource(requestDocument),
    new StreamResult(outputStream));
outputStream.send();
// Receive authentication response
inputStream.receive();
Document responseDocument = documentBuilder.parse(inputStream);
Element responseElement = responseDocument.getDocumentElement();
boolean authenticated = responseElement.getAttribute(
    ProtocolConfig.ATTRIBUTE_RESULT).equals("OK");

```

- Cerere de port audio: Dacă autentificarea a fost realizată cu succes se poate trece la etapa următoare și se solicită un port. Portul primit ca răspuns este folosit pentru a transmite fișierul audio către server pentru a fi tradus.

```

// Send a getAudioDataPortRequest
requestDocument = XMLBuilder.createGetAudioDataPortRequest();

```

```
// Receive the getAudioDataResponse XML
    inputStream.receive();
    Socket audioDataSocket = new Socket(SERVER_ADDRESS,
        audioDataPort);
    OutputStream audioDataOutputStream = audioDataSocket
        .getOutputStream();
```

- Cerere de transcriber: Pentru a realiza transcrierea serverul trebuie să atribuie un motor de transcriere (S2T-Transcriber). Fiecare motor de transcriere deservește un domeniu diferit (știri, vorbire medicală, nume de țări, etc.). Pentru a realiza o aplicație cât mai optimă din punctul de vedere al utilizării resurselor și al bateriei am fixat numărul maxim de cereri de atribuire a unui transcriber la 5.

```
// Request a transcriber
    boolean receivedStartTranscriptionAck = false;
    int numTries = 0;
    int maxNumTries = 5; // Allow only 5 re-tries in case the server is
busy
    while (!receivedStartTranscriptionAck) {
        // Send a transcription request
        requestDocument = XMLBuilder.createGetTranscriptionRequest(
            0, "PCM_SIGNED", "narrow",
            new TranscriptionOptions(true, true, true, true,
            true));

        // Receive a startTranscriptionAck or a
        // transcriberTemporarilyUnavailableError
        inputStream.receive();
        if (++numTries > maxNumTries)
            break;
```

- Trimitere fișier audio: După ce serverul a atribuit un motor de transcriere se poate trece la următoarea etapă și anume trimiterea fișierului audio pe portul primit de la server.

```
File audioFile = new File(audioFileName);
InputStream audioFileStream = new FileInputStream(audioFile);
byte[] buffer = new byte[8192];
int length;

while ((length = audioFileStream.read(buffer)) != -1) {
    audioDataOutputStream.write(buffer, 0, length);
}

audioDataOutputStream.close();
audioDataSocket.close();
audioFileStream.close();
```

- Recepționare transcriere și scrierea în fișier: După ce s-a finalizat transcrierea se salvează răspunsul în fișierul corespunzător fișierului audio.

```
// Receive several getTranscriptionResponses.
    boolean receivedDoneTranscriptionAck = false;
```

```

StringBuilder builder = new StringBuilder();
while (!receivedDoneTranscriptionAck) {
    try { inputStream.receive();
    if (responseElement.getNodeName().equals(

        ProtocolConfig.RESPONSE_GET_TRANSCRIPTION)) {

        builder.append(" ")

            .append(responseElement
                .getAttribute(ProtocolConfig.ATTRIBUTE_BEST_PROCESSED_T
                    EXT));

// write the final transcription here

        PrintWriter writer = new PrintWriter(transcription);
        writer.println(builder.toString());
        writer.flush();
        writer.close();

```

Pe parcursul acestor etape TranscriberTask actualizează baza de date pentru a oferi informații referitoare la stadiul în care se află transcrierea ( status: started, pending, finished, error).

### 4.3 Testarea aplicației pe diferite dispozitive

Pentru a verifica modul de funcționare am instalat aplicația pe diferite telefoane Android și am analizat calitatea și modul de înregistrare al apelurilor. Telefoanele utilizate sunt Samsung Galaxy SIII, Samsung Galaxy SIII Mini, HTC Desire X și Huawei G300.

Dintre telefoanele enumerate mai sus doar Samsung S III permite înregistrarea apelului direct de la linia telefonică, celelalte permițând doar înregistrarea de la microfon adică doar a persoanei care deține aplicația. Înregistrarea directă a liniei telefonice este restricționată pe unele telefoane deoarece în unele țări este ilegală înregistrarea unei persoane fără acordul acesteia.

Pe toate telefoanele am obținut aceeași calitate a înregistrărilor. Am folosit același text pentru testare. Deoarece trei dintre telefoanele testate permit doar înregistrarea de la microfon am decis să transform conversația într-un monolog.



## Concluzii

Din analiza făcută în capitolele anterioare se poate ajunge la concluzia că realizarea unei aplicații de transcriere a convorbirilor pe o platformă Android prezintă o serie de avantaje și dezavantaje.

Unul din principalele avantaje ale folosirii platformei Android îl constituie numărul mare de informații, resurse, mostre de cod și numărul mare de dezvoltatori cu experiență dispuși să ajute atunci când întâmpini o problemă.

Dezvoltarea de noi aplicații nu necesită achiziționarea unei licențe care costă, dezvoltarea fiind gratuită și la îndemâna oricărei persoane care dorește să încerce.

Dezavantajul principal îl constituie modul de înregistrare al convorbirilor. Ideal ar fi ca înregistrarea să se facă direct după linia telefonică pentru a putea înregistra ambele sensuri ale unei conversații. Acest lucru nu este posibil pe foarte multe telefoane, accesul la linia telefonică fiind restricționat de producători fiind ilegală înregistrarea în unele țări.

Un alt neajuns al aplicației este legat de calitatea transcrierilor. Sistemul de recunoaștere este antrenat cu clipuri audio înregistrate în condiții de laborator. În cazul acestei aplicații nu se poate impune dinainte calitatea microfonului și a sistemului de achiziție a semnalului vocal astfel că apar distorsiuni care îngreunează recunoașterea. Un alt element care crează probleme este reprezentat de perturbațiile în canalul de telecomunicații și de banda îngustă a semnalului vocal.

Am constatat că se obține o transcriere satisfăcătoare atunci când se vorbește clar și relativ tare. În timpul unei conversații normale traducerea oferită de server nu corespunde cu vorbirea din clipul audio.

Pentru a se putea realiza o transcriere corectă în orice situație trebuie antrenat sistemul de recunoaștere cu fișiere achiziționate prin înregistrarea convorbirilor telefonice.

Contribuția personală constă în realizarea unei aplicații pe platformă Android ce permite utilizarea funcționalităților oferite de sistemul de recunoaștere pe telefonul mobil. Aceasta constă în realizarea unei interfețe grafice, sistemul de înregistrare a apelurilor în format .wav, realizarea unei variante compatibile cu platforma Android a protocolului de conectare la server și crearea unei baze de date pentru gestionarea fișierelor înregistrate și a transcrierilor.



## Bibliografie

- [1] [http://ro.wikipedia.org/wiki/Android \(sistem de operare\)](http://ro.wikipedia.org/wiki/Android_(sistem_de_operare))
- [2] <http://ocw.cs.pub.ro/courses/pdsd/labs/00>
- [3] <http://www.itsolutions.eu/2011/09/08/android-tutorial-concepte-activitati-si-resurse-ale-unei-aplicatii-android/>
- [4] <http://www.vogella.com/tutorials/Android/article.html>
- [5] Lect. Horia Cucu, *Proiect de cercetare-dezvoltare în Tehnologia Vorbirii*
- [6] Andi Buzo, Recunoașterea Limbajului Vorbit în Rețele de Telecomunicații Mobile





## Anexa 1

### MainActivity

```
package ro.pub.calltranscriber;
import java.text.SimpleDateFormat;
import java.util.Date;
import
android.support.v7.app.ActionBarActivity;
import android.support.v4.app.Fragment;
import
android.support.v4.app.FragmentTransaction;
import
android.support.v4.app.ListFragment;
import
android.support.v4.app.LoaderManager;
import
android.support.v4.content.CursorLoader;
import android.support.v4.content.Loader;
import
android.support.v4.widget.CursorAdapter;
import
android.support.v4.widget.SimpleCursorAdapter;
import android.annotation.SuppressLint;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.database.Cursor;
import android.graphics.Color;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.CompoundButton;
import
android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;

/**
 * The main UI interface of the
application:
 * 1. Allows toggling the
TranscriberService
 * 2. Displays all transcriptions
 * Note: even if the TranscriberService
is stopped, the transcriptions are still
 * displayed (but any operations on them
are inactive).
 */
public class MainActivity extends
ActionBarActivity {
    @Override

    protected void onCreate(Bundle
savedInstanceState) {

        super.onCreate(savedInstanceState)
;

        setContentView(R.layout.activity_m
ain);

        if (savedInstanceState ==
null) {
            getSupportFragmentManager().beginTransaction()

                .add(R.id.container, new
PlaceholderFragment()).commit();
        }
    }

    /**
     * The PlaceholderFragment
displays the list of transcriptions. It
also
     * displays a list header with a
toggle button for enabling/disabling the
     * TranscriberService.
     */
    public static class
PlaceholderFragment extends ListFragment
implements

        SharedPreferences.OnSharedPreferen
ceChangeListener,

        LoaderManager.LoaderCallbacks<Curs
or> {

        private ToggleButton
mTranscriberToggler = null;
        private SimpleCursorAdapter
mAdapter;

        public PlaceholderFragment() {
        }

        @Override
        public void
onActivityCreated(Bundle
savedInstanceState) {
            super.onActivityCreated(savedInstanceStat
e);

            // Give some text to display if there is
no data
            setEmptyText("No available
transcriptions");

            // Create an empty adapter to be used to
display loaded dates

            mAdapter = new
TranscriptionAdapter(getActivity(),
null);

            setListAdapter(mAdapter);
        }
    }
}
```

```

// Start out with a progress indicator
setListShown(false);

// Prepare the loader. Either re-connect
with an existing one, or start a new one.

        getLoaderManager().initLoader(LOAD
_TRANSSCRIPTIONS, null, this);
    }
@Override
public View onCreateView(LayoutInflater
inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View rootView =
super.onCreateView(inflater, container,

        savedInstanceState);

// Register for the TranscriberService's
preferences to be able to tell when it
becomes active/inactive
    SharedPreferences prefs =
getActivity().getSharedPreferences(

TranscriberService.TRANSCRIBER_PREFERENCE
S,

        MODE_MULTI_PROCESS |
MODE_PRIVATE);

    prefs.registerOnSharedPreferenceCh
angeListener(this);

    Resources resources =
getActivity().getResources();
// Setup the toggle button which will
control the lifetime of the
TranscriberService
        mTranscriberToggler =
new ToggleButton(getActivity());

        mTranscriberToggler.setTextOn(reso
urces

        .getString(R.string.transcriber_on
));
mTranscriberToggler.setTextOff(resources

        .getString(R.string.transcriber_of
f));

// Set the toggle button as the
transcription header list
        ((ListView)
rootView.findViewById(android.R.id.list))

        .addHeaderView(mTranscriberToggler
);

        mTranscriberToggler.setChecked(
prefs.getBoolean(

        TranscriberService.KEY_TRANSCRIBER
_RUNNING, false));

// Let's just make sure the service
didn't crash without setting the
preference appropriately
        if
(mTranscriberToggler.isChecked())

            getActivity().startService(

                new Intent(getActivity(),
TranscriberService.class));

// Set up toggling capabilities
        mTranscriberToggler

        .setOnCheckedChangeListener(new
OnCheckedChangeListener() {

@Override
public void
onCheckedChanged(CompoundButton
buttonView,boolean isChecked) {

            Intent transcriber = new
Intent(getActivity(),
TranscriberService.class);
            if (isChecked)
                getActivity().startService(transcr
iber)
            else
                getActivity().stopService(transcri
ber);
        }
    });
return rootView;
}

@Override
public void onSharedPreferenceChanged(

        SharedPreferences
sharedPreferences, String key) {
// Alert this Fragment when the
TracriberService lifetime changes
if
(key.equals(TranscriberService.KEY_TRANSC
RIBER_RUNNING)) {
    if (mTranscriberToggler != null) {
        mTranscriberToggler.setChecked(
sharedPreferences.getBoolean(key,
false));
    }
}

        private static final int
LOAD_TRANSCRIPTIONS = 0;
@Override
public Loader<Cursor> onCreateLoader(int
loaderId, Bundle bundle) {
    switch (loaderId) {
    case LOAD_TRANSCRIPTIONS: {
// Use a Cursor Loader as it will auto-
update the ListFragment (even when the
contents change)
return new CursorLoader(getActivity(),

        TranscriptionProvider.CONTENT_URI,
null, null, null,null);
    }
}

```

```

default:
    return null;
}
@Override
public void onLoadFinished(Loader<Cursor>
loader, Cursor cursor) {
    // Swap the new cursor in. (The framework
    will take care of closing the old cursor
    once we return.)

    mAdapter.swapCursor(cursor);

    // Add the beginning the list is empty so
    start the Transcriber
    if (cursor.getCount() == 0) {

        getActivity().startService(

            new Intent(getActivity(),
TranscriberService.class));

        Toast.makeText(getActivity(),
"Transcriber is running!",

            Toast.LENGTH_SHORT).show();
    }
    // The list should now be shown.
    if (isResumed()) {

        setListShown(true);
    }
    else {
        setListShownNoAnimation(true);
    }
}
@Override
public void onLoaderReset(Loader<Cursor>
loader) {
    // This is called when the last Cursor
    provided to onLoadFinished() above is
    about to be closed. We need to make sure
    we are no
    longer using it.
    mAdapter.swapCursor(null);
}
// Custom SQLite database list adapter
for our transcriptions
    // Each transcription will
be displayed as follows:
// 1. Caller ID
// 2. Time of call
// 3. Status of transcribing
// 4. Play the recording
// 5. Show raw output of server
communication
// 6. Show transcription (in case of
success)
// 6'. Retry running the task (in case
it failed). Retrying a task will only
work when the TranscriberService is
running

private class TranscriptionAdapter
extends SimpleCursorAdapter {
private final LayoutInflater mInflater;
public TranscriptionAdapter(Context
context, Cursor cursor) {
    // Registered as a content observer to
    auto-update when the content changes

```

```

super(context, R.layout.transcription,
cursor, new String[] {},
new int[] {}),
CursorAdapter.FLAG_REGISTER_CONTENT_OBSER
VER);
mInflater = LayoutInflater.from(context);
}
@Override
public View onCreateView(Context context,
Cursor cursor, ViewGroup parent) {
    return
mInflater.inflate(R.layout.transcription,
null);
}
// Binding each UI list item with an
entry in the transcription list
@SuppressLint("SimpleDateFormat")
@Override
public void bindView(View view, Context
context, Cursor cursor) {

    super.bindView(view, context,
cursor);

    // Set up UI components
        TextView caller = (TextView)
view.findViewById(R.id.caller);
        TextView time = (TextView)
view.findViewById(R.id.time);
        TextView status = (TextView)
view.findViewById(R.id.status);
        ImageButton play = (ImageButton)
view

        .findViewById(R.id.play_audio);
        Button showRawOutput = (Button)
view

        .findViewById(R.id.show_output);
        Button show = (Button)
view.findViewById(R.id.show_whatever);

    // Retrieve table columns indexes in the
database cursor
    int callerIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_CALLER);
    int timeIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_TIME);
    int statusIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_STATUS);
    int audioFileIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_AUDIO_FILE);
    int rawOutputIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_RAW_OUTPUT);
    int transcriptionIndex =
cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_TRANSCRIPTION_FILE);

```

```

int idIndex = cursor

        .getColumnIndexOrThrow(Transcripti
onProvider.KEY_ID);

        caller.setText(cursor.getString(ca
llerIndex));
time.setText(new
SimpleDateFormat("hh:mm:ss E, dd-MM-
yyyy")
.format(new
Date(cursor.getLong(timeIndex))));
String statusText = "Unknown";
int color = Color.GREEN;
boolean showRetry = false;
boolean showTranscription = false;

// Setup the status of the transcription
switch (cursor.getInt(statusIndex)) {
case
TranscriptionProvider.STATUS_STARTING:

    statusText = "Starting";
    color = Color.CYAN;
    break;

case
TranscriptionProvider.STATUS_PENDING:

    statusText = "Pending";
    color = Color.BLUE;
    break;

case TranscriptionProvider.STATUS_DONE:

    statusText = "Finished";
    showTranscription = true;
    break;

case
TranscriptionProvider.STATUS_FAILED_AUTHE
NTICATION:

    statusText = "Failed: Authentication
error";
    showRetry = true;
    color = Color.RED;
    break;

case
TranscriptionProvider.STATUS_FAILED_INTER
NET:

    statusText = "Failed: No internet
connection";
    showRetry = true;
    color = Color.RED;
    break;
case
TranscriptionProvider.STATUS_FAILED_TRANS
CRPTION:
    statusText = "Failed: Transcription
error";
    showRetry = true;
    color = Color.RED;
    break;

```

```

case
TranscriptionProvider.STATUS_INCOMPLETE_T
RANSRIPTION:

    statusText = "Failed: Incomplete
transcription";
    showRetry = true;
    color = Color.RED;
    break;
}

status.setText("Status: " + statusText);
status.setTextColor(color);

// Read the state of the current
transcription

final long currentId =
cursor.getLong(idIndex);
final String currentAudioFile =
cursor.getString(audioFileIndex);
final String currentRawOutputFile =
cursor.getString(rawOutputIndex);
final String currentTranscriptionFile =
cursor.getString(transcriptionIndex);

// Show a dialog allowing to play back
the current recording
play.setOnClickListener(new
OnClickListener() {

    @Override
    public void onClick(View v) {

        PlayAudioDialogFrament dialog =
PlayAudioDialogFrament

                .newInstance(currentAudioFile);

        FragmentTransaction ft =
getActivity()

                .getSupportFragmentManager().begin
Transaction();
        Fragment previous = getActivity()

                .getSupportFragmentManager().findF
ragmentByTag("dialog");
        if (previous != null) {
            ft.remove(previous);
        }
        ft.addToBackStack(null);
        dialog.show(ft, "dialog");
    }
});

// Show the current raw output in a
Dialog

        showRawOutput.setOnClickListener(n
ew OnClickListener() {
    @Override
    public void onClick(View v) {

        ViewerFragment dialog =
ViewerFragment

```

```

        .newInstance(currentRawOutputFile)
;

        FragmentTransaction ft =
getActivity()

        .getSupportFragmentManager().begin
Transaction();

        Fragment previous = getActivity()

        .getSupportFragmentManager().findF
ragmentByTag("dialog");

        if (previous != null) {

            ft.remove(previous);

        }

        ft.addToBackStack(null);
        dialog.show(ft, "dialog");
    }

});
// If the current transcription failed,
allow it to be retried
if (showRetry) {
    show.setText("Retry");

    show.setOnClickListener(new
OnClickListener() {

@Override
public void onClick(View v) {
    // send an intent to retry this task
    Intent retry = new Intent(
TranscriberService.RETRY_TASK);
    retry.putExtra(TranscriptionProvid
er.KEY_ID,currentId);
    retry.putExtra(
TranscriptionProvider.KEY_AUDIO_FI
LE,

                currentAudioFile);

    retry.putExtra(

TranscriptionProvider.KEY_RAW_OUTP
UT,

        currentRawOutputFile);
    retry.putExtra(
TranscriptionProvider.KEY_TRANSCRI
PTION_FILE,
        currentTranscriptionFile);
    getActivity().sendBroadcast(retry)
;

    Toast.makeText(getActivity(),
        "Retrying transcribe task",
        Toast.LENGTH_SHORT).show();
    }

});
} else if (showTranscription) {
// if the current transcription was
successful, show the output
    show.setText("View
transcription");
    show.setOnClickListener(new
OnClickListener() {

```

```

@Override

        public void onClick(View v) {
            ViewerFragment dialog =
ViewerFragment
            FragmentTransaction ft =
getActivity()
            .getSupportFragmentManager()
            .beginTransaction();
            Fragment previous = getActivity()
            .getSupportFragmentManager()
            .findFragmentByTag("dialog");
            if (previous != null) {
                ft.remove(previous);
            }
            ft.addToBackStack(null);
            dialog.show(ft, "dialog");
        }
    });
} else
    show.setOnClickListener(null);
}}}}

```

### TranscriberService

```

package ro.pub.calltranscriber;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

import
ro.pub.calltranscriber.protocol.Transcrib
erTask;

import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.res.Resources;
import android.database.Cursor;
import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Environment;
import android.os.IBinder;
import android.provider.BaseColumns;
import android.provider.ContactsContract;
import
android.provider.ContactsContract.PhoneLo
okup;
import
android.support.v4.app.NotificationCompat
;
import
android.support.v4.app.TaskStackBuilder;
import
android.telephony.PhoneStateListener;
import
android.telephony.TelephonyManager;
import android.util.Log;

```

```

/**
 * Main component of the Call Transcriber
 application. It's main
 * responsibilities are:
 * 1. record voice calls (incoming and
 outgoing)
 * 2. save recordings to the external
 storage
 * 3. send recordings to the server and
 retrieve transcriptions
 * 4. save transcriptions and server
 communication output to external storage
 *
 * The TranscriberService runs as a
 foreground service so it doesn't get
 killed
 * by the Android runtime when the
 application goes to the background (what
 * usually happens during phone calls).
 Furthermore, it runs in its own separate
 * process (see AndroidManifest.xml,
 'android:process="transcriber"') to have
 * its own separate address space (to be
 able to deal with large amounts of data
 * received from the server).
 */
public class TranscriberService extends
Service {
    private static final int
ONGOING_NOTIFICATION_ID = 0xdeadbeef;
    private static final String TAG =
TranscriberService.class.getSimpleName();

    // Preferences that let the rest
of the application know that this service
// is running/stopped
    public static final String
TRANSCRIBER_PREFERENCES =
"transcriber_prefs";
    public static final String
KEY_TRANSCRIBER_RUNNING =
"KEY_TRANSCRIBER_RUNNING";

    // Intent action thrown by other
components to notify this service that it
// should re-run the transcribing
task again on a given entry from the
// transcription database (to be
used for tasks that have failed due to
// various reasons). Each such
intent must have three extras: task id
// (TranscriptionProvider.KEY_ID),
task audio file path
//
(TranscriptionProvider.KEY_AUDIO_FILE),
task raw output file path
//
(TranscriptionProvider.KEY_RAW_OUTPUT)
and task transcription file path
//
(TranscriptionProvider.KEY_TRANSCRIPTION_
FILE)
    public static final String
RETRY_TASK =
"ro.pub.calltranscriber.RETRY_TASK";

    // Audio Recoder Constants
    private static final String
RECORDER_FILE_EXT_WAV = ".wav";

```

```

    private static final String
RECORDER_FOLDER = "CallTranscriber";
    private static final int
RECORDER_BPP = 16;
    private static final int
RECORDER_SAMPLERATE = 8000;
    private static final int
RECORDER_CHANNELS =
AudioFormat.CHANNEL_IN_STEREO;
    private static final int
RECORDER_AUDIO_ENCODING =
AudioFormat.ENCODING_PCM_16BIT;

    // The output directory on the external
storage (e.g./sdcard/CallTranscriber)
    private File outputDir;
    // All recording are placed under
recordings/$(CALLER_ID)
    private File recordingsDir;

    // All transcriptions are placed under
transcriptions/$(CALLER_ID)
    private File transcriptionsDir;

    // All raw outputs are placed under
raw/$(CALLER_ID)
    private File rawDir;
    private TelephonyManager
mTelephonyManager;
    private OutgoingCallReceiver
mOutgoingCallReceiver;
    private PhoneStateListener
mPhoneStateListener;
    private ContentResolver
mContentResolver;

    // The current recording task
    private AudioRecordTask
mAudioRecordTask;

    // The current incoming/outgoing contact
(we consider we can have only one call at
the same time)
    private String
mCurrentIncomingNumber;
    private int mAudioBufferSize;

    // Receives Retry requests (intents)
    private RetryReceiver
mRetryReceiver;
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        mContentResolver =
getContentResolver();
        Resources resources =
getResources();
    // The foreground service must show a
notification to the user
        NotificationCompat.Builder
mBuilder = new
NotificationCompat.Builder(
            this)

        .setSmallIcon(R.drawable.ic_launch
er)

```

```

        .setContentTypeTitle(
            resources.getString(R.string.notification_title))

        .setContentTypeText(
            resources.getString(R.string.notification_message));
// An explicit intent for the Main Activity

Intent notificationIntent = new
Intent(this, MainActivity.class);

// The stack builder object will contain
an artificial back stack for the started
Activity. This ensures that navigating
backward from the Activity leads out of
your application to the Home screen.
        TaskStackBuilder
stackBuilder =
TaskStackBuilder.create(this);

// Adds the back stack for the Intent
(but not the Intent itself)

        stackBuilder.addParentStack(MainActivity.class);
// Adds the Intent that starts the
Activity to the top of the stack
        stackBuilder.addNextIntent(notificationIntent);

        PendingIntent pendingIntent
= stackBuilder.getPendingIntent(0,

        PendingIntent.FLAG_CANCEL_CURRENT)
;

        mBuilder.setContentIntent(pendingIntent);

// A foreground service is a service
that's considered to be something the
user is actively aware of and thus not a
candidate for the system to kill when low
on memory

        startForeground(ONGOING_NOTIFICATION_ID, mBuilder.build());

// Setup the retry receiver
        mRetryReceiver = new
RetryReceiver();

        registerReceiver(mRetryReceiver,
new IntentFilter(RETRY_TASK));

// Setup directory structure
        outputDir = new
File(Environment.getExternalStorageDirectory()

        .getPath(),
RECORDER_FOLDER);
if (!outputDir.exists())
        outputDir.mkdirs();

        recordingsDir = new
File(outputDir, "recordings");
if (!recordingsDir.exists())

```

```

        recordingsDir.mkdirs();
transcriptionsDir = new File(outputDir,
"transcriptions");
        if (!transcriptionsDir.exists())

transcriptionsDir.mkdirs();

        rawDir = new File(outputDir,
"raw");

if (!rawDir.exists())
        rawDir.mkdirs();
// Setup audio recording
        mAudioBufferSize =
AudioRecord.getMinBufferSize(RECORDER_SAMPLERATE, RECORDER_CHANNELS,
RECORDER_AUDIO_ENCODING);
// Register for calls
        mPhoneStateListener = new
PhoneStateListener() {
@Override
public void onCallStateChanged(int state,
String incomingNumber) {
switch (state) {
case TelephonyManager.CALL_STATE_OFFHOOK:
startRecordingCall();
break;

case TelephonyManager.CALL_STATE_IDLE:
stopRecordingCall();
break;

case TelephonyManager.CALL_STATE_RINGING:

// For incoming calls, only the ringing
state tells us about the current incoming
number

setupIncomingCall(incomingNumber);
break;

}

}

};

// For outgoing calls we need to register
a receiver in order to find out the
outgoing number
        mOutgoingCallReceiver = new
OutgoingCallReceiver();

        registerReceiver(mOutgoingCallReceiver, new IntentFilter(
Intent.ACTION_NEW_OUTGOING_CALL));
mTelephonyManager = (TelephonyManager)
getSystemService(TELEPHONY_SERVICE);
mTelephonyManager.listen(mPhoneStateListener,

        PhoneStateListener.LISTEN_CALL_STATE);
// Let others know i'm alive
getSharedPreferences(TRANSCRIBER_PREFERENCES,
MODE_MULTI_PROCESS | MODE_PRIVATE).edit()

        .putBoolean(KEY_TRANSCRIBER_RUNNING, true).commit();
}
@Override

```

```

public void onDestroy() {
    // release all used resources
    unregisterReceiver(mRetryReceiver);
    unregisterReceiver(mOutgoingCallReceiver);
    ;

    mTelephonyManager.listen(null,
        PhoneStateListener.LISTEN_CALL_STATE);

    // Let others know i'm going away

    getSharedPreferences(TRANSCRIBER_PREFEREN
        CES,
        MODE_MULTI_PROCESS | MODE_PRIVATE).edit()

        .putBoolean(KEY_TRANSCRIBER_RUNNIN
            G, false).commit();

    // Don't forget to cancel the
    notification
    stopForeground(true);
    super.onDestroy();
    }
    /**
     * The RetryReceiver is
     responsible for re-launching
     transcription tasks that
     * have previously failed.
     */
    private class RetryReceiver
    extends BroadcastReceiver {

    @Override
    public void onReceive(Context context,
        Intent intent) {

    if
    (intent.getAction().equals(RETRY_TASK)) {
        long id =
        intent.getLongExtra(TranscriptionProvider
            .KEY_ID, -1);
        String audioFile = intent

            .getStringExtra(TranscriptionProvi
                der.KEY_AUDIO_FILE);
        String rawOutputFile =
        intent

            .getStringExtra(TranscriptionProvi
                der.KEY_RAW_OUTPUT);
        String transcriptionFile = intent

            .getStringExtra(TranscriptionProvi
                der.KEY_TRANSCRIPTION_FILE);
        Log.d(TAG, "Retrying task: " +
            id);
        new
        TranscriberTask(mContentResolver,
            id).execute(new File(

            audioFile), new
            File(rawOutputFile), new
            File(transcriptionFile));
        }
    }

    private void startRecordingCall() {

```

```

        Log.d(TAG, "Ongoing call " +
            mCurrentIncomingNumber);
        if (mCurrentIncomingNumber
            == null) { // Should not happen
            Log.e(TAG, "Don't know caller id.
                Aborting record!");
            return;
        }
        synchronized (this) {
            if (mAudioRecordTask != null) {
                Log.e(TAG, "Already recording.
                    ...");
                return;
            }
        }
        // When a call is under going start a
        recorder task
        mAudioRecordTask = new
        AudioRecordTask(mCurrentIncomingNumber);

        mAudioRecordTask.execute();
        }
    }
    private void stopRecordingCall() {
        Log.d(TAG, "Call ended " +
            mCurrentIncomingNumber);

        if (mCurrentIncomingNumber != null)
            mCurrentIncomingNumber = null; // reset
            state

        else {
            Log.e(TAG, "Don't know caller id.
                Must've been aborted!");
            return;
        }
        synchronized (this) { // Stop the
            recording task when the call is over

            mAudioRecordTask.stopRecording();
            mAudioRecordTask = null;
        }
    }
    private void setupIncomingCall(String
        incomingNumber) {

        mCurrentIncomingNumber = incomingNumber;

        // Check to see if the current incoming
        number is actually a contact
        Uri uri =
        Uri.withAppendedPath(PhoneLookup.CONTENT_
            FILTER_URI,

            Uri.encode(incomingNumber));
        Cursor cursor =
        mContentResolver.query(uri, new String[]
            {

            BaseColumns._ID,
            ContactsContract.PhoneLookup.DISPLAY_NAME
            },

            null, null, null);

        try {
            if (cursor != null && cursor.getCount() >
                0) {

                cursor.moveToNext();

```



```

        // Use contact name instead of
        number

        mCurrentIncomingNumber =
        cursor.getString(cursor

        .getColumnIndex(ContactsContract.D
        ata.DISPLAY_NAME));
    }
    } finally {
        if (cursor != null)

        cursor.close();
    }

    Log.d(TAG, "Ringing: " +
    mCurrentIncomingNumber);
}

/**
 * The OutgoingCallReceiver is
    responsible for retrieving the callee
    phone
    * number as the PhoneStateListener can
    only retrieve it from incoming calls
    */
    class OutgoingCallReceiver extends
    BroadcastReceiver {

    @Override
    public void onReceive(Context context,
    Intent intent) {
    if
    (Intent.ACTION_NEW_OUTGOING_CALL.equals
    (intent.getAction())) {
    String number = intent
    .getStringExtra(Intent.EXTRA_PHONE_NUMBER
    );
    if (number != null && !"".equals(number))
    {
    // to setup similar to incoming calls

        setupIncomingCall(number);
    }
    }
    }

    }

/**
 * The AudioRecordTask is responsible for
    recording all voice calls.
    *
    * It attempts to set up a voice call
    recorder, if this fails it then sets
    * up a microphone recorder. In the off
    chance that mic recordings are
    * unimplemented, it uses a pre-defined
    WAV file and copies it instead of
    * the recording.
    *
    * When this task finished, it copies the
    recording as a WAV file in the
    * contacts output directory and start
    the TranscriberTask.
    *
    * The implementation uses an AsyncTask
    as it provides the means to both run

```

```

    * on the main UI Thread (e.g.
    onPreExecute) and on a separate thread
    (e.g.
    * doInBackground).
    */

    class AudioRecordTask extends
    AsyncTask<Void, Void, File> {

    private boolean mIsRecording = false;
    private AudioRecord mAudioRecorder;
    private String mCaller; // Caller ID
    private long mTime; // Time of call start
    private File mAudioRecordFile; // the
    output file for recording

    private File mRawOutputFile; // the
    output file for transcribing output

    // (used by the TranscriberTask)
    private File mTranscriptionFile;
    // the output file for transcription

    public AudioRecordTask(String
    contact) {
    // Create the appropriate recording and
    trascription directories

    // For each call:
    // 1. the recording is named
    <timestamp>.wav
    // 2. the recoding is placed under
    //
    /sdcard/CallTranscriber/recordings/<CALLER_ID>/
    // 3. the transcribing output file is
    named <timestamp>.raw
    // 4. the transcribing output file is
    placed under
    //
    /sdcard/CallTranscriber/raw/<CALLER_ID>/
    // 5. The transcription output file is
    named <timestamp>.txt
    // 6. The transcription output file is
    placed under
    //
    /sdcard/CallTranscriber/transcriptions/<CALLER_ID>/
    File recordingContactDir = new
    File(recordingsDir, contact);

    if (!recordingContactDir.exists())

        recordingContactDir.mkdirs();

    File transcriptionContactDir = new
    File(transcriptionsDir, contact);

    if (!transcriptionContactDir.exists())

        transcriptionContactDir.mkdirs();
    File rawContactDir = new File(rawDir,
    contact);
    if (!rawContactDir.exists())

        rawContactDir.mkdirs();
    mCaller = contact;
    mTime = System.currentTimeMillis();
    String timestamp = mTime + "";

```

```

mAudioRecordFile = new
File(recordingContactDir, timestamp
    + RECORDER_FILE_EXT_WAV);
mRawOutputFile = new File(rawContactDir,
timestamp + ".raw");
mTranscriptionFile = new
File(transcriptionContactDir, timestamp
    + ".txt");
}

@Override
protected void onPreExecute() {

// Create the Audio Recorder

synchronized (this) {
// try voice call
    mAudioRecorder = new AudioRecord(

        MediaRecorder.AudioSource.VOICE_CA
LL,

        RECORDER_SAMPLERATE,
RECORDER_CHANNELS,

        RECORDER_AUDIO_ENCODING,
mAudioBufferSize);

    if (mAudioRecorder.getState() ==
AudioRecord.STATE_UNINITIALIZED) {
        // If voice call recording is not
supported, than just open the mic

        mAudioRecorder = new AudioRecord(

            MediaRecorder.AudioSource.MIC,
RECORDER_SAMPLERATE,

            RECORDER_CHANNELS,
RECORDER_AUDIO_ENCODING,

            mAudioBufferSize);
    }
}

@Override
protected File doInBackground(Void...
params) {
String timestamp =
System.currentTimeMillis() + "";
byte data[] = new byte[mAudioBufferSize];
File temp = null;
FileInputStream fis = null;
FileOutputStream fos = null;

// Start Recording
if (mAudioRecorder.getState() ==
AudioRecord.STATE_UNINITIALIZED) {

// Mock implementation returning a pre-
cached WAV file. To be used in case both
voice call and mic recording are
unimplemented
    Log.d(TAG, "AudioRecord
unintialized. Entering mock mode");
    InputStream ais = null;

try {
ais = getAssets().open("mock.wav");

```

```

fos = new
FileOutputStream(mAudioRecordFile);
byte buffer[] = new byte[1024];
int read = 0;

while ((read = ais.read(buffer)) != -1) {

fos.write(buffer, 0, read);
fos.flush();
}
} catch (Exception e) {

    e.printStackTrace();
} finally {
    try {

        ais.close();
    } catch (Exception e) {
    }

try {

        fos.close();
    } catch (Exception e) {
    }

return mAudioRecordFile;
}

// Start the recording process

mAudioRecorder.startRecording();

synchronized (this) {
    mIsRecording = true;
}

// In the mean time, flush the audio
record to a temp file
try {
temp = File.createTempFile(timestamp,
".tmp");
fos = new FileOutputStream(temp);
int read = 0;
while (true) {
synchronized (this) {

    if (!mIsRecording)

        break;

        read = mAudioRecorder.read(data,
0, mAudioBufferSize);
    }
    if (read !=
AudioRecord.ERROR_INVALID_OPERATION) {

        fos.write(data);

        fos.flush();
    }
}
} catch (IOException e) {

    e.printStackTrace();
} finally {
    try {

        fos.close();
    } catch (Exception e) {

```

```

    }
}
// Copy audio file as a WAV file
long totalAudioLen = 0;
long totalDataLen = totalAudioLen + 36;
long longSampleRate =
RECORDER_SAMPLERATE;
int channels = 2;
long byteRate = RECORDER_BPP *
RECORDER_SAMPLERATE * channels / 8;

data = new byte[mAudioBufferSize];

try {
    fis = new FileInputStream(temp);
    fos = new
FileOutputStream(mAudioRecordFile);
    totalAudioLen = fis.getChannel().size();
    totalDataLen = totalAudioLen + 36;

    // Write WAVE file header
    byte[] header = new byte[44];

    header[0] = 'R'; // RIFF/WAVE header
    header[1] = 'I';
    header[2] = 'F';
    header[3] = 'F';
    header[4] = (byte) (totalDataLen & 0xff);
    header[5] = (byte) ((totalDataLen >> 8) &
0xff);
    header[6] = (byte) ((totalDataLen >> 16)
& 0xff);
    header[7] = (byte) ((totalDataLen >> 24)
& 0xff);
    header[8] = 'W';
    header[9] = 'A';
    header[10] = 'V';
    header[11] = 'E';
    header[12] = 'f'; // 'fmt ' chunk
    header[13] = 'm';
    header[14] = 't';
    header[15] = ' ';
    header[16] = 16; // 4 bytes: size of 'fmt
' chunk
    header[17] = 0;
    header[18] = 0;
    header[19] = 0;
    header[20] = 1; // format = 1
    header[21] = 0;
    header[22] = (byte) channels;
    header[23] = 0;
    header[24] = (byte) (longSampleRate &
0xff);
    header[25] = (byte) ((longSampleRate >>
8) & 0xff);
    header[26] = (byte) ((longSampleRate >>
16) & 0xff);
    header[27] = (byte) ((longSampleRate >>
24) & 0xff);
    header[28] = (byte) (byteRate & 0xff);
    header[29] = (byte) ((byteRate >> 8) &
0xff);
    header[30] = (byte) ((byteRate >> 16) &
0xff);
    header[31] = (byte) ((byteRate >> 24) &
0xff);
    header[32] = (byte) (2 * 16 / 8); //
block align
    header[33] = 0;

```

```

    header[34] = RECORDER_BPP; // bits per
sample
    header[35] = 0;
    header[36] = 'd';
    header[37] = 'a';
    header[38] = 't';
    header[39] = 'a';
    header[40] = (byte) (totalAudioLen &
0xff);
    header[41] = (byte) ((totalAudioLen >> 8)
& 0xff);
    header[42] = (byte) ((totalAudioLen >>
16) & 0xff);
    header[43] = (byte) ((totalAudioLen >>
24) & 0xff);

    fos.write(header, 0, 44);
    fos.flush();

    // Copy the audio record afterwards
    while (fis.read(data) != -1) {

        fos.write(data);
    }
    fos.flush();
} catch (IOException e) {

    e.printStackTrace();
} finally {
    try {

        fis.close();
    } catch (Exception e) {

    }

    try {

        fos.close();
    } catch (Exception e) {

    }

}

// Delete the temp file
temp.delete();

return mAudioRecordFile;
}

public void stopRecording() {
    // Call has ended
    synchronized (this) {
        if (mAudioRecorder != null) {
            if
(mAudioRecorder.getState() ==
AudioRecord.STATE_UNINITIALIZED) {

                Log.d(TAG,

                    "AudioRecord uninitialized.
Exiting mock mode");

                return;
            }
        }

        mIsRecording = false;

        mAudioRecorder.stop();

        mAudioRecorder.release();

```

```

// Need to cancel the task as we need to
interrupt the AudioRecord.read(). This
means the result will be sent to
onCancelled

        cancel(true);
    }
}

private void startTranscribing(File
recording) {
    ContentValues values = new
    ContentValues();

    // Now that recording is done, create a
    new entry in the TranscriptionProvider
    database for this new transcription

    values.put(TranscriptionProvider.KEY_CALL
    ER, mCaller);

    values.put(TranscriptionProvider.KEY_TIME
    , mTime);

    values.put(
    TranscriptionProvider.KEY_AUDIO_FILE,

        mAudioRecordFile.getAbsolutePath()
    );

    values.put(
    TranscriptionProvider.KEY_TRANSCRIPTION,
    "-");

    values.put(
    TranscriptionProvider.KEY_TRANSCRIPTION_F
    ILE,

        mTranscriptionFile.getAbsolutePath
    ());

    values.put(
    TranscriptionProvider.KEY_RAW_OUTPUT,

        mRawOutputFile.getAbsolutePath());

    values.put(
    TranscriptionProvider.KEY_STATUS,

        TranscriptionProvider.STATUS_START
    ING);

    long id = Long.parseLong(mContentResolver

        .insert(TranscriptionProvider.CONT
    ENT_URI, values)

        .getPathSegments().get(1));

    // Start the TranscriberTask for the
    current transcription
        new
    TranscriberTask(mContentResolver,
    id).execute(recording,

        mRawOutputFile,
    mTranscriptionFile);
}

```

```

@Override
public void onCancelled(File recording) {
    // Called when the AudioRecordTask has
    finished
    Log.d(TAG, "Processing " +
    recording.getAbsolutePath());

        startTranscribing(recording);
    }

@Override

public void onPostExecute(File recording)
{
    // Only called when the AudioRecordTask
    has finished and is running
    // in mock mode (no voice, no mic, just
    preloaded WAV file)
    Log.d(TAG, "Processing in mock mode " +
    recording.getAbsolutePath());

        startTranscribing(recording);
    }
}

PlayAudioDialogFrament
package ro.pub.calltranscriber;

import java.text.SimpleDateFormat;
import java.util.Date;

import android.annotation.SuppressLint;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import
android.support.v4.app.DialogFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.ImageButton;
import android.widget.Toast;

/**
 * The PlayAudioDialogFragment is used to
    play back recorded conversations. The
 * file path for the audio recording must
    be provided as an argument to the
 * constructor method.
 *
 * It displayed the duration of the
    recording (in s) and allows playing and
 * pausing the audio recording.
 */
@SuppressLint("SimpleDateFormat")
public class PlayAudioDialogFrament
    extends DialogFragment {
    private String mAudioFilePath;
    private MediaPlayer mMediaPlayer;

    /**
 * Create a new instance of
    PlayAudioDialogFragment, providing file
    path as
 * an argument.
 */
}

```

```

        static PlayAudioDialogFrament
newInstance(String audioFilePath) {
    PlayAudioDialogFrament f =
new PlayAudioDialogFrament();

    // Supply audio file path as
an argument
    Bundle args = new Bundle();
args.putString("path",
audioFilePath);
    f.setArguments(args);

    return f;
}

@Override
public void onCreate(Bundle
savedInstanceState) {

    super.onCreate(savedInstanceState)
;

    mAudioFilePath =
getArguments().getString("path");

    // prepare the media player
(might take some time)
    mMediaPlayer = new
MediaPlayer();

    mMediaPlayer.setAudioStreamType(Au
dioManager.STREAM_MUSIC);

    try { // an also may fail

        mMediaPlayer.setDataSource(getActi
vity().getApplicationContext(),

        Uri.parse("file:/// " +
mAudioFilePath));

        mMediaPlayer.prepare();
    } catch (Exception e) {

        Toast.makeText(getActivity(),
"Cannot play audio file!",

        Toast.LENGTH_SHORT).show();

        getDialog().dismiss();
        return;
    }

    @Override
    public View
onCreateView(LayoutInflater inflater,
ViewGroup container,
Bundle
savedInstanceState) {
        View v =
inflater.inflate(R.layout.audio_play,
container, false);

        getDialog().setTitle("Duration: "
+ (new SimpleDateFormat("ss").format(new
Date( mMediaPlayer.getDuration())))) + "
s");

        ImageButton play =
(ImageButton) v.findViewById(R.id.play);

```

```

        ImageButton pause =
(ImageButton) v.findViewById(R.id.pause);

        // play audio
        play.setOnClickListener(new
OnClickListener() {
            @Override
            public void
onClick(View v) {

                mMediaPlayer.start();
            }
        });

        // pause audio
        pause.setOnClickListener(new
OnClickListener() {
            @Override
            public void
onClick(View v) {

                mMediaPlayer.pause();
            }
        });

        return v;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();

        // Clean up after ourselves
        mMediaPlayer.stop();
        mMediaPlayer.release();
    }
}

```

### TranscriptionProvider

```

package ro.pub.calltranscriber;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import
android.database.sqlite.SQLiteDatabase;
import
android.database.sqlite.SQLiteOpenHelper;
import
android.database.sqlite.SQLiteDatabase.Cu
rsorFactory;
import
android.database.sqlite.SQLiteQueryBuilde
r;
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

/**
 * The database containing a table for
storing transcription information
 * 1. unique transcription id
 * 2. caller id
 * 3. time of call start
 * 4. audio file recording path
 * 5. raw output file path (for the
TranscriberTask)

```

```

* 6. the transcription (empty if not
there yet)
* 7. the transcription file path (as
outputed by the TranscriberTask)
*
* This database allows us to maintain
coherency between the TranscriberService
* and the UI (MainActivity). It can also
be used by external applications if
* needed.
*/
public class TranscriptionProvider
extends ContentProvider {
    // The URI used by other
components/applications that want access
to transcriptions
    public static final Uri
CONTENT_URI =
Uri.parse("content://ro.pub.calltranscriber.provider.transcription/transcriptions");

    private static final int
TRANSCRIPTIONS = 0x01;
    private static final int
TRANSCRIPTION_ID = 0x02;

    private static final UriMatcher
matcher;
    private static SQLiteDatabase
transcriptionsDB;

    static {
        matcher = new
UriMatcher(UriMatcher.NO_MATCH);

        matcher.addURI("ro.pub.calltranscriber.provider.transcription",
"transcriptions", TRANSCRIPTIONS);

        matcher.addURI("ro.pub.calltranscriber.provider.transcription",
"transcriptions/#", TRANSCRIPTION_ID);
    }

    private static final String TAG =
TranscriptionProvider.class.getSimpleName();
    private static final String
DB_NAME = "transcriptions.db";
    private static final String
TRANSCRIPTIONS_TABLE = "transcriptions";
    private static final int
DB_VERSION = 1;

    // Table keys
    public static final String KEY_ID
= "_id";
    public static final String
KEY_CALLER = "_caller";
    public static final String
KEY_TIME = "_time";
    public static final String
KEY_AUDIO_FILE = "_audio";
    public static final String
KEY_TRANSCRIPTION = "_transcription";
    public static final String
KEY_TRANSCRIPTION_FILE =
"_transcription_file";

```

```

    public static final String
KEY_RAW_OUTPUT = "_raw_output";
    public static final String
KEY_STATUS = "_status";

    // Table column indexes
    public static final int ID_COLUMN
= 0;
    public static final int
CALLER_COLUMN = 1;
    public static final int
TIME_COLUMN = 2;
    public static final int
AUDIO_FILE_COLUMN = 3;
    public static final int
TRANSCRIPTION_COLUMN = 4;
    public static final int
TRANSCRIPTION_FILE_COLUMN = 5;
    public static final int
RAW_OUTPUT_COLUMN = 6;
    public static final int
STATUS_COLUMN = 7;

    public static final int
STATUS_STARTING = 0;
    public static final int
STATUS_PENDING = 1;
    public static final int
STATUS_FAILED_AUTHENTICATION = 2;
    public static final int
STATUS_FAILED_TRANSCRIPTION = 3;
    public static final int
STATUS_FAILED_INTERNET = 4;
    public static final int
STATUS_INCOMPLETE_TRANSCRIPTION = 5;
    public static final int
STATUS_DONE = 6;

    @Override
    public boolean onCreate() {
        transcriptionsDB = new
TranscriptionDatabaseHelper(getContext(),
DB_NAME, null,
DB_VERSION).getWritableDatabase();

        return (transcriptionsDB !=
null);
    }

    // SQLite CRUD
(Create, Read, Update, Delete) -----
-----

    @Override
    public Cursor query(Uri uri,
String[] projection, String selection,
String[]
selectionArgs, String sortOrder) {
        SQLiteQueryBuilder builder =
new SQLiteQueryBuilder();

        builder.setTables(TRANSCRIPTIONS_T
ABLE);

        if (matcher.match(uri) ==
TRANSCRIPTION_ID) {

            builder.appendWhere(KEY_ID + "=" +
uri.getPathSegments().get(1));
        }
    }

```

```

        Cursor cursor =
builder.query(transcriptionsDB,
projection, selection, selectionArgs,
null, null, sortOrder);

        cursor.setNotificationUri(getConte
xt().getContentResolver(), uri);

        return cursor;
    }

    @Override
    public String getType(Uri uri) {
        switch (matcher.match(uri))
        {
            case TRANSCRIPTIONS:
                return
"vnd.android.cursor.dir/vnd.pub.calltrans
criber.transcription";
            case TRANSCRIPTION_ID:
                return
"vnd.android.cursor.item/vnd.pub.calltran
scriber.transcription";
            default:
                throw new
IllegalArgumentException("Bad URI: " +
uri);
        }
    }

    @Override
    public Uri insert(Uri uri,
ContentValues values) {
        long row =
transcriptionsDB.insert(TRANSCRIPTIONS_TA
BLE, "", values);

        if (row > 0) {
            Uri changedUri =
ContentUris.withAppendedId(CONTENT_URI,
row);

            getContext().getContentResolver().
notifyChange(changedUri, null);

            return changedUri;
        }

        throw new
IllegalArgumentException("Bad URI: " +
uri);
    }

    @Override
    public int delete(Uri uri, String
selection, String[] selectionArgs) {
        switch (matcher.match(uri))
        {
            case TRANSCRIPTIONS:
                return
transcriptionsDB.delete(TRANSCRIPTIONS_TA
BLE, selection, selectionArgs);
            case TRANSCRIPTION_ID:
                String segment =
uri.getPathSegments().get(1);
                String
selectionClause = KEY_ID + "=" + segment;

```

```

                if
(!TextUtils.isEmpty(selection))

                    selectionClause += " AND (" +
selection + ")";

                return
transcriptionsDB.delete(TRANSCRIPTIONS_TA
BLE, selectionClause, selectionArgs);
            default:
                throw new
IllegalArgumentException("Bad URI: " +
uri);
        }
    }

    @Override
    public int update(Uri uri,
ContentValues values, String selection,
String[]
selectionArgs) {
        int count = 0;

        switch (matcher.match(uri))
        {
            case TRANSCRIPTIONS:
                count =
transcriptionsDB.update(TRANSCRIPTIONS_TA
BLE, values, selection, selectionArgs);
                break;
            case TRANSCRIPTION_ID:
                String segment =
uri.getPathSegments().get(1);
                String
selectionClause = KEY_ID + "=" + segment;

                if
(!TextUtils.isEmpty(selection))

                    selectionClause += " AND (" +
selection + ")";

                count =
transcriptionsDB.update(TRANSCRIPTIONS_TA
BLE, values, selectionClause,
selectionArgs);
                break;
            default:
                throw new
IllegalArgumentException("Bad URI: " +
uri);
        }

        getContext().getContentResolver().
notifyChange(uri, null);
        return count;
    }
    // -----
    -----

    private static class
TranscriptionDatabaseHelper extends
SQLiteOpenHelper {
        private static final String
CREATE_DB;

        static {

```

```

        StringBuilder builder
= new StringBuilder();

        builder.append("create table
").append(TRANSCRIPTIONS_TABLE).append("
");

        builder.append(KEY_ID).append("
integer primary key autoincrement, ");

        builder.append(KEY_CALLER).append(
" text not null, ");

        builder.append(KEY_TIME).append("
integer, ");

        builder.append(KEY_AUDIO_FILE).app
end(" text not null, ");

        builder.append(KEY_TRANSCRIPTION).
append(" text not null, ");

        builder.append(KEY_TRANSCRIPTION_F
ILE).append(" text not null, ");

        builder.append(KEY_RAW_OUTPUT).app
end(" text not null, ");

        builder.append(KEY_STATUS).append(
" integer);");

        CREATE_DB = builder.toString();
    }

public
TranscriptionDatabaseHelper(Context
context, String name, CursorFactory
factory, int version) {
    super(context, name, factory,
version);
}

@Override
public void
onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_DB);
}

@Override
public void onUpgrade(SQLiteDatabase db,
int oldVersion, int newVersion) {
    Log.w(TAG, "Upgrading
(" + oldVersion + " -> " + newVersion +
")");

    db.execSQL("DROP TABLE IF
EXISTS " + TRANSCRIPTIONS_TABLE);
    onCreate(db);
}

}

ViewerFragment
package ro.pub.calltranscriber;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

```

```

import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
import
android.support.v4.app.DialogFragment;
import android.widget.FrameLayout;
import android.widget.ScrollView;
import android.widget.TextView;

/**
 * The ViewerFragment is used to display
either raw output files or
 * transcription files. The file path
must be provided as a parameter for the
 * constructor method.
 */
public class ViewerFragment extends
DialogFragment {
    private String mFilePath;

    /**
 * Create a new instance of
ViewerFragment, providing file path as
 * an argument.
 */
    static ViewerFragment
newInstance(String filePath) {
        ViewerFragment f = new
ViewerFragment();

        // Supply file path as an
argument
        Bundle args = new Bundle();
        args.putString("path", filePath);
        f.setArguments(args);

        return f;
    }

    @Override
    public void onCreate(Bundle
savedInstanceState) {

        super.onCreate(savedInstanceState)
;

        mFilePath =
getArguments().getString("path");
    }

    @Override
    public Dialog
onCreateDialog(Bundle savedInstanceState)
{
        StringBuilder message = new
StringBuilder();

        // Read the entire file into
the string builder
        File file = new
File(mFilePath);

        if (file.exists()) {
            BufferedReader reader
= null;

            try {
                reader = new
BufferedReader(new FileReader(file));

```



```

        String line =
null;

        while ((line =
reader.readLine()) != null) {

            message.append(line).append("\n");
            }
        } catch (Exception e)
{

            message.append("Exception reading
" + mFilePath + ": " + e);
            } finally {
                try {

                    reader.close();
                } catch
(Exception e) {

                }

            } else // or show errors
                message.append("No
such file: " + mFilePath);

            // Create a Scroll View
            which will hold the main TextView
            displaying
            // the contents of the file
            (the file may be large and might require
            // scrolling)
            ScrollView scroller = new
            ScrollView(getActivity());
            scroller.setLayoutParams(new
            FrameLayout.LayoutParams(

                FrameLayout.LayoutParams.MATCH_PAR
            ENT,

                FrameLayout.LayoutParams.MATCH_PAR
            ENT));

            TextView text = new
            TextView(getActivity());

            text.setText(message.toString());
            scroller.addView(text, new
            FrameLayout.LayoutParams(

                FrameLayout.LayoutParams.MATCH_PAR
            ENT,

                FrameLayout.LayoutParams.MATCH_PAR
            ENT));

            // Create the dialog
            return new
            AlertDialog.Builder(getActivity()).setVie
            w(scroller).create();
        }
    }
}

```

### TranscriberTask

```

package ro.pub.calltranscriber.protocol;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;

```

```

import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

import javax.xml.parsers.DocumentBuilder;
import
javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import
javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import
javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

import
ro.pub.calltranscriber.TranscriptionProvi
der;

import android.content.ContentResolver;
import android.content.ContentValues;
import android.os.AsyncTask;
import android.util.Log;

/**
 * The TranscriberTask is responsible for
the communication with the
 * transcribing server:
 * 1. open socket to the server
 * 2. authenticate
 * 3. request transcription
 * 4. request audio data port
 * 5. send audio recording
 * 6. receive transcription
 *
 * The TranscriberTask updates the
TranscriptionProvider entry
 * respective to the current
transcription (the status of the
transcription:
 * started, pending, finished, error
reporting).
 *
 * The input for this task is: the unique
transcription id, the audio file,
 * the raw output file and the
transcription file provided by the
AudioRecordTask
 * (or the RetryReceiver).
 */
public class TranscriberTask extends
AsyncTask<File, String, Boolean> {
    private ContentResolver
mContentResolver;
    private long mTaskId;

    private static final String TAG =
TranscriberTask.class.getSimpleName();

    // Server constants
    private static final String
SERVER_ADDRESS = "dev.speed.pub.ro";
    private static final int
SERVER_PORT = 5004;

```

```

        public
TranscriberTask(ContentResolver resolver,
long id) {
    mContentResolver = resolver;
    mTaskId = id;
}

@Override
protected Boolean
doInBackground(File... recordings) {
    String audioFileName =
recordings[0].getAbsolutePath();
    File rawOutput =
recordings[1];
    File transcription =
recordings[2];
    boolean success = true;
    Socket socket = null;
    XMLOutputStream outputStream
= null;
    XMLInputStream inputStream =
null;
    FileOutputStream
rawOutputStream = null;

    try {
        socket = new
Socket(SERVER_ADDRESS, SERVER_PORT);
        outputStream = new
XMLOutputStream(socket.getOutputStream())
;
        inputStream = new
XMLInputStream(socket.getInputStream());
        rawOutputStream = new
FileOutputStream(rawOutput);

        publishProgress("Connected to
server");

        DocumentBuilder
documentBuilder = DocumentBuilderFactory
.newInstance().newDocumentBuilder(
);

        Transformer
transformer =
TransformerFactory.newInstance()
.newInstance();

        transformer.setOutputProperty(Outp
utKeys.INDENT, "yes");

        transformer.setOutputProperty(

"{http://xml.apache.org/xslt}inden
t-amount", "4");

// Send authentication request

        Document requestDocument =
XMLBuilder.createAuthenticateRequest(

"diana.enescu", "plmmde*00");

        transformer.transform(new
DOMSource(requestDocument),
new StreamResult(outputStream));
        outputStream.send();

```

```

// Receive authentication reponse
inputStream.receive();
Document responseDocument =
documentBuilder.parse(inputStream);
        Element responseElement =
responseDocument.getDocumentElement();
        boolean authenticated
= responseElement.getAttribute(

ProtocolConfig.ATTRIBUTE_RESULT).e
quals("OK");

        publishProgress("Authenticated by
server");

        if (authenticated) {
            // Send a
getAudioDataPortRequest

            requestDocument =
XMLBuilder.createGetAudioDataPortRequest(
);

            transformer.transform(new
DOMSource(requestDocument),

            new StreamResult(outputStream));
            outputStream.send();

            transformer.transform(new
DOMSource(requestDocument),

            new
StreamResult(rawOutputStream));

            // Receive the
getAudioDataResponse XML, get the port
and print

            // XML
            // document on
the screen

            inputStream.receive();

            responseDocument =
documentBuilder.parse(inputStream);

            responseElement =
responseDocument.getDocumentElement();
            int
audioDataPort =
Integer.parseInt(responseElement

.getAttribute(ProtocolConfig.ATTRI
BUTE_PORT));

            transformer.transform(new
DOMSource(responseDocument),

            new
StreamResult(rawOutputStream));

            publishProgress("Using audio port
" + audioDataPort);

```

```

// Connect to
server audio socket
Socket
audioDataSocket = new
Socket (SERVER_ADDRESS,

audioDataPort);
OutputStream
audioDataOutputStream = audioDataSocket

.getOutputStream();

publishProgress("Opened audio data
socket");

// Request a transcriber
boolean receivedStartTranscriptionAck =
false;
int numTries = 0;
// Allow only 5 re-tries in case the
server is busy
int maxNumTries = 5;

while
(!receivedStartTranscriptionAck) {
// Send
a transcription request

requestDocument =
XMLBuilder.createGetTranscriptionRequest(

0, "PCM_SIGNED", "narrow",

new TranscriptionOptions(true,
true, true, true));

transformer.transform(new
DOMSource(requestDocument),

new StreamResult(outputStream));

outputStream.send();

transformer.transform(new
DOMSource(requestDocument),

new
StreamResult(rawOutputStream));

//
Receive a startTranscriptionAck or a
//
transcriberTemporarilyUnavailableError

inputStream.receive();

responseDocument =
documentBuilder.parse(inputStream);

responseElement =
responseDocument.getDocumentElement();

receivedStartTranscriptionAck =
responseElement

.getNodeName().equals(

```

```

ProtocolConfig.ACK_START_TRANSCRIP
TION);

transformer.transform(new
DOMSource(responseDocument),

new
StreamResult(rawOutputStream));

// Wait
5 seconds before sending another
//
getTranscriptionRequest

Thread.sleep(5000);

if
(++numTries > maxNumTries)

break;
}

if (numTries
<= maxNumTries) {

publishProgress("Acknowledged
transcription request");

// Start
sending audio data
File
audioFile = new File(audioFileName);

InputStream audioFileStream = new
FileInputStream(audioFile);
byte[] buffer = new
byte[8192];
int length;

while
((length = audioFileStream.read(buffer))
!= -1) {

audioDataOutputStream.write(buffer
, 0, length);

}

audioDataOutputStream.close();

audioDataSocket.close();

audioFileStream.close();

publishProgress("Sent audio
data");

//
Receive several
getTranscriptionResponses.
boolean
receivedDoneTranscriptionAck = false;

StringBuilder builder = new
StringBuilder();

while
(!receivedDoneTranscriptionAck) {

```

```

try {
    inputStream.receive();

    responseDocument = documentBuilder
        .parse(inputStream);

    responseElement = responseDocument
        .getDocumentElement();

    receivedDoneTranscriptionAck =
responseElement
        .getNodeName()

        .equals(ProtocolConfig.ACK_DONE_TR
ANSRIPTION);

    transformer.transform(new
DOMSource(

        responseDocument), new
StreamResult(

        rawOutputStream));

    if (responseElement.getNodeName().equals(

        ProtocolConfig.RESPONSE_GET_TRANSC
RIPTION)) {

        builder.append(" ")

        .append(responseElement

            .getAttribute(ProtocolConfig.ATTRI
BUTE_BEST_PROCESSED_TEXT));

        }

    catch (SAXException e) {

        Log.e(TAG, "Dubious input. Transcription
will be incomplete");

        e.printStackTrace();

        updateProvider(TranscriptionProvid
er.STATUS_INCOMPLETE_TRANSCRIPTION);

        success = false;

        break;

    }

    // write the final transcription here

    PrintWriter writer = new
PrintWriter(transcription);

```

```

        writer.println(builder.toString())
;

        writer.flush();

        writer.close();

        publishProgress("Wrote
transcription to file");
    } else {
        success
= false;

        updateProvider(TranscriptionProvid
er.STATUS_FAILED_TRANSCRIPTION);
    }
    } else {
        success = false;

        updateProvider(TranscriptionProvid
er.STATUS_FAILED_AUTHENTICATION);
    }
    } catch (UnknownHostException e) {

        updateProvider(TranscriptionProvid
er.STATUS_FAILED_INTERNET);
        success = false;
    } catch (Exception e) {

        updateProvider(TranscriptionProvid
er.STATUS_FAILED_TRANSCRIPTION);
        success = false;

    } finally {
        // Clean up after ourselves
        try {
            inputStream.close();
        } catch (Exception e) {

        }

        try {
            outputStream.close();
        } catch (Exception e) {

        }

        try {

            rawOutputStream.close();
        } catch (Exception e) {

        }

        try {
            socket.close();
        } catch (Exception e) {

        }

        return success;
    }

    @Override
    protected void
onProgressUpdate(String... progress) {
        // Log all important events
in the transcribing lifecycle
        Log.e(TAG, "Progress: " +
progress[0]);
    }

    @Override

```

```

        protected void onPreExecute() {
            super.onPreExecute();

            updateProvider(TranscriptionProvid
er.STATUS_PENDING);
        }
        @Override
        protected void onPostExecute(Boolean
result) {
            Log.e(TAG, "Result: " + result);
            if (result) // mark the
success of the transcription request

            updateProvider(TranscriptionProvid
er.STATUS_DONE);
        }
        private void updateProvider(int
status) {
            Log.e(TAG, "Updating status for "
+ mTaskId + " to " + status);
            // Update the transcription entry
accordingly
            ContentValues values = new
ContentValues();

            values.put(TranscriptionProvider.K
EY_STATUS, status);

            mContentResolver.update(Transcript
ionProvider.CONTENT_URI, values,

            TranscriptionProvider.KEY_ID + "="
+ mTaskId, new String[] {});

        }
    }
}

```