UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**PROIECT DE DIPLOMĂ**

Conducător ştiinţific: Conf. dr. ing. Crețulescu Radu

Îndrumător: Conf. dr. ing. Crețulescu Radu

Absolvent: Conțiu Ioan Dumitru

Specializarea: Calculatoare

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**Parcel Tracking**

Conducător ştiinţific: Conf. dr. ing. Crețulescu Radu

Îndrumător: Conf. dr. ing. Crețulescu Radu

Absolvent: Conțiu Ioan Dumitru

Specializarea: Calculatoare

**Cuprins**

[1. Introducere 4](#_Toc453156883)

[2. CONSIDETAȚII TEORETICE 5](#_Toc453156884)

[**2.1. Sistemul de operare Android** 5](#_Toc453156885)

[**2.2 Caracteristici Android** 6](#_Toc453156886)

[**2.3 Arhitectura sistemului Android** 8](#_Toc453156887)

[**2.3.1 Nucleul Linux** 9](#_Toc453156888)

[**2.3.2 Librării de bază** 10](#_Toc453156889)

[**2.3.3 Android RUNTIME** 11](#_Toc453156890)

[**2.3.4 Aplicații Framework** 12](#_Toc453156891)

[**2.3.5 Aplicații** 13](#_Toc453156892)

[**2.3.5.1 Activity (Activitate)** 14](#_Toc453156893)

[**2.3.5.2 Intent (Intentie):** 18](#_Toc453156894)

[**2.3.5.3 Service (Serviciu):** 19](#_Toc453156895)

[**2.3.5.4 Fragment** 19](#_Toc453156896)

[**2.3.4.5 Fisierul Manifest** 20](#_Toc453156897)

[**2.4 Resursele aplicatie android** 21](#_Toc453156898)

[**2.5. Securitatea** 24](#_Toc453156899)

[9.0 Lista acronimelor 25](#_Toc453156900)

[10. Bibliografie 26](#_Toc453156901)

# **Introducere**

Traim intr-o era a vitezei unde dorim sa avem acces la informatii legate de diverse domenii cat mai repede si cu cat mai putin efort posibil. In zilele noastre fiecare aprope persoana detine un telefon mobil inteligent si o conexiune la internet prin GPRS sau WIFI.

In urma studiilor o persoana verifica telefonul de peste 200 de ori pe zi pentru a vedea daca o notificare noua sau daca a fost contactat de catre un prieten. Stiind ca accesul la informatie este foarte rapid in zilele noastre este foarte greu sa fi la curent cu noutatile cotidiene si dorind mai ales sa stim in fiecare clipa unde se afla produsul cumparat de catre noi online tema pe care am ales sa o abordez in aceasta lucrare de diploma este despre implementarea unei aplicatii pentru telefonul mobil cu sistem de operare Android numita Parcel Tracking.

În prezenta lucrare, am decis să implementez o aplicaţie Android datorită avantajelor pe care acest sistem de operare le oferă, printre care se numără următoarele: oportunitatea de dezvoltare (instrumentele folosite pentru elaborarea codului sursă oferite gratuit) și promovare a aplicației (Google Play), costuri financiare reduse şi posibilitatea implementării în limbaj Java (unul foarte răspândit și bine documentat – preferat pentru că este unul dintre cele mai simple limbaje de programare orientate pe obiect), API–ul pentru harti oferit gratuit de catre compania Google si a multitudinilor de exemple si documentatii oferite de catre acestia.

# **CONSIDETAȚII TEORETICE**

## **2.1. Sistemul de operare Android**

Android este un sistem de operare pentru mobil care în prezent este dezvoltat de catre Google, bazat pe un Kernel Linux si este destinat in special pentru dispozitivele Touch Screen cum ar fi telefoanele mobile sau tabletele. Interfața cu utilizatorul este bazată pe manupularea directă, cum ar fi gesturile care fac referire la acțiunile din lumea reală, cum ar fi trecerea peste ecran, apăsări sau ciupituri. În ultimul timp Android a extins acest sistem iar acum se gasește și pe dispositive cum ar fi televizoare, mașini, ceasuri de mână.[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, Qualcom, T-Mobile, Sprint Nextel și Nvidia.[1]

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.

## **2.2 Caracteristici Android**

Printre caracteristicile principale ale sistemului Android se numară 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 in 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 cofigurații 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, dar și un IDE pus la dispoziție de către Google, Android Studio.

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

De asemenea, sistemul de operare Android conține un kit de dezvoltare software (SDK), care permite crearea de aplicații pentru un anumit pachet de programe software, platforma hardware, sistem informatic sau consolă de jocuri video. Implementarea acestuia poate fi la fel de simplă ca acea a unor interfețe de programare a aplicațiilor (API) sub forma unor biblioteci de interfață pentru un anumit limbaj de programare sau mai complicată incluzând un hardware sofisticat pentru a putea comunica cu un sistem integrat. De cele mai multe ori, SDK-ul include exemple de cod sau documentație tehnică ce ajută mult la clarificarea diferitelor aspecte ale software-ului de bază, îndrumând dezvoltatorii spre aplicații Android de succes.[3]

## **2.3 Arhitectura sistemului Android**

Sistemul Android dispune de o arhitectură alcatuită din 5 niveluri ce comunica intre ele:

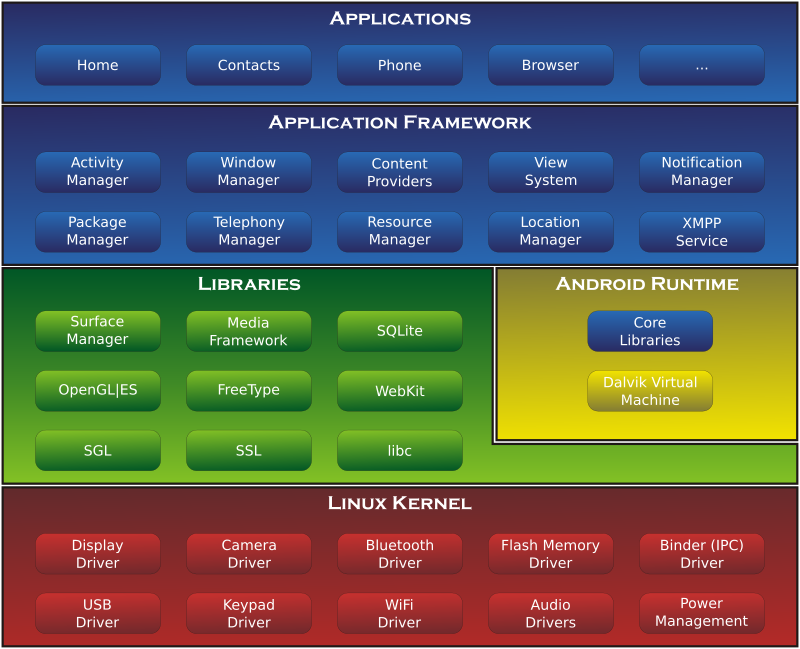
• Nucleu Linux (Linux Kernel)

• Librării de bază

• Android RUNTIME

• Aplicații Framework

• Aplicații



### **2.3.1 Nucleul Linux**

Reprezintă nucleul sistemului de operare deschis care a fost creat de Linus Torvalds în 1991 și care a fost îmbunătățit ulterior cu ajutorul programatorilor din întreaga lume. [4]

Acesta a fost dezvoltat inițial pentru microprocesorul Intel 80386, însă ulterior a fost portat pe multe alte platforme. Este scris aprope în întregime în limbajul de programare C, împreună cu niște extensii GNU C și cu cîteva linii de cod scrise în limbajul de asamblare. Dezvoltat sub licența GNU GPL (GNU General Public License), codul sursă al nucleului Linux este software liber. [4]

Nucleul Linux include multitasking real, memorie virtuală, biblioteci partajate, demand loading, executabile partajate copy-on-write, gestiunea memoriei corectă, și rețele TCP/IP. Astăzi, Linux este un nucleu monolitic cu încărcare de module. Device driver-e și extensii de nucleu rulează tipic în inelul 0, cu acces total la hardware, deși unele rulează în spațiul utilizator. Spre deosebire de nucleele monolitice standard, device driver-ele se configurează ușor ca module, și se încarcă sau se descarcă în timpul rulării sistemului. [4]

### **2.3.2 Librării de bază**

Librăriile și API-urile (Application Programming Interface) sunt de obicei scrise în C și C++ și din acest motiv ele sunt adesea menționate a fi librării native. Bionic libc, librăria dezvoltată de Google special pentru Android, fiind derivată din biblioteca de C standard a Berkeley Software Distribution (BSD) are ca beneficii latenţa mică a runtime-ului şi optimizarea pentru procesoare cu frecvenţa scăzută. [5] Android nu suportă întregul set de biblioteci standard GNU, lucru care face portarea aplicaţiilor sau bibliotecilor deja existente pe Linux dificilă. Asfel, Bionic libc se ocupă de o mulțime de lucruri de bază, dar în special de activitățile sensibile la performanța pe dispozitiv (încărcarea rapidă a paginilor web și actualizarea interfeței).

Spre exemplu, Android are propriul sistem de biblioteci în limbaj C, care implementează și răspund la apelurile de sistem, ce crează procesee, fire de execuție, execută calcule matematice, alocă memorie și multe altele. Există, de asemenea, un media framework pentru redare fișierelor audio și video, un webkit pentru redarea și afișarea paginilor web, librăria SQLite pentru gestionarea în memorie a bazelor de date relaționale, și nu în ultimul rând librăria Open GL pentru performanța grafică.

### **2.3.3 Android RUNTIME**

Pe lângă bibliotecile de sistem, este inclus și sistemul de rulare în timp real Android, compus din două mari componente: librăriile de bază Java bazate pe Apache Harmony și mașina virtuală Dalvik (Dalvik Virtual Machine – DVM), care susțin scrierea și buna funcționare a aplicațiilor Android.

În primul rând, aplicațiile Android sunt tipic scrise în limbajul de programare Java, și pentru o implementare cât mai ușoară, sistemul de operare Android pune la dispoziție un set de Java “block - codes” reutilizabil (spre exemplu: Java și Java X - extentions packages - includ software de bază pentru lucruri, cum ar fi structurile comune de date, mecanisme de concurență sau fișiere de Input - Output). Prin urmare, pachetele Android conțin software care e specific ciclului de viață al aplicațiilor mobile, pachetele org sprijină diverse operații web (Internet) și Junit sunt folosite pentru testarea aplicațiilor.

În al doilea rând, DVM este software-ul care execută efectiv aplicațiile Android. S-ar presupune că datorită faptului că aplicațiile Android sunt scrise în Java, acestea pot rula în mod firesc pe mașina virtuală Java (Java Virtual Machine - JVM). Dar, lucrurile nu sunt atât de simple și ceea ce se întâmplă de fapt după scrierea aplicațiilor în limbaj Java sunt prezentate în cele ce urmează. În primul pas, compilatorul Java va compila fișierele codului sursă în mai multe fișiere de tip bytecode.

Mai departe, un instrument numit DX transformă Java bytecode într-un singur fișier numit dex. (un format de tip bytecode diferint, care de obicei se numește classes.dex). În următorul pas, fișierul .dex este împachetat cu alte resurse ale aplicației și instalat într-un fișier .apk pe dispozitivul mobil. În sfârșit, când utilizatorul lansează în execuție aplicația, DVM va executa atunci fișierul classes.dex. Motivul pentru tot acest proces îl reprezintă faptul că DVM, spre deosebire de JVM, a fost proiectat să ruleze într-un mediul constâns de resurse (memorie minimă), ce caracterizeză în fapt dispozitivele mobile. În comparație cu un dispozitiv de tip desktop, cel mobil este mai mod evident mai puțin performant și limitat din mai multe puncte de vedere (are, probabil, CPU mai lent, memorie mai puțină, o baterie limitată).

Fiecare aplicație Android rulează în propriul proces, cu propria instanță a mașinii virtuale Dalvik, ce a fost scris în așa fel încât un dispozitiv să poată rula mai multe mașini virtuale eficient. În concluzie, Android foloseşte DVM cu compilare „just-in-time” pentru a execută codul Dalvik, care este de obicei tradus în bytecode Java. Android 4.4 suportă şi o nouă maşină virtuală experimentală ART (Android RunTime), care tinde să îmbunătățeacă performanțele aplicației per ansamblu, deși unele tehnici ce se executau cu Dalvik, cu ART nu funcționeză.[5]

### **2.3.4 Aplicații Framework**

Nivelul de aplicații Framework este cel cu care lucrează direct programatorul, oferind dezvoltatorilor toate funcționalitătile și resursele oferite de sistem. O aplicație framework este package manager, care este în mod esențial o bază de date ce păstreză evidența tuturor aplicațiilor actuale instalate pe un dispozitiv. Acest nivel este preinstalat în Android și este organizat pe componente pentru a putea extinde și crea noi componente. Cele mai importante componente sunt:

**Managerul de activități**. La un nivel înalt, activitățiile Android-ului, de cele mai multe ori, corespund unui singur ecran (interfață cu utilizatorul). Aplicațiile Android sunt create prin cuplarea mai multor activități, ce permit utilizatorului să navigheze – să treacă dintr-o interfață în alta – această acțiune fiind coordonată de manager-ul de activități. Presupunând că doresc să ascult muzică, parcurg următorii pași: intru în aplicația de Music Media Player, unde va aparea o anumită interfață ce conține toate albumele momentan disponibile, selectez albumul dorit și o a doua interfață e afișată, conținând melodiile corespunzătoare, în cele din urmă un simplu click pe melodie va afișa ultima interfață cu posilitățiile de start / stop / pause / replay.

**Content provider** reprezintă, în esență, o bază de date care permite aplicațiilor să stocheze și să partajeze informații structurate. De exemplu, aplicația de telefon poate accesa detaliile stocate despre fiecare contact în parte și folosește această bază de date pentru a apela numărul respectiv. Content provider este conceput pentru a funcționa între aplicații și folosește aceste informații nu doar pentru a apela contacte, ci și pentru a trimite mesaje de tip text, multimedia sau e-mail și pentru a contacta prieteni pe diferite rețele de socilizare.

Următoarea aplicație prezentată succint este manage-ul de locație, folosit pentru a primi informații despre locație și mișcare, cum ar fi cele generate de sistemul de navigare GPS. Acest lucru permite aplicațiilor să efectueze sarcini specifice contextului: găsirea direcțiilor din locația curentă. Ultima componentă abordată din cadrul arhitecturii sistemului de operare Android este manager-ul de notificări, ce plasează informații în bara de notificări (o componentă a ferestrei). Acesta înștiințează utilizatorii despre anumite evenimente ce au avut loc. Un exemplu concret ar fi atunci când dorim să trimitem un mesaj de tip text unui contact care tocmai redacteză un e-mail și pentru a nu deranja există un program software ce rulează mereu și activează doar o notificare în bara de sus a ecranului, astfel fiind informat de primirea mesajului si doar ulterior să îl citeacă în întregime.

### **2.3.5 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.

În parte cea mai de sus, apare ultimul nivel din arhitectura sistemului de operare Android ce este reprezentat de nivelul de aplicație. E deja cunoscut faptul că Android oferă unele aplicațiile standard, încorporate în sistem: Phone Dialer, Web browser, E-mail Reader, Calculator, Camera, Contacts, Clock, Music Player, Voice Recorder, dar în componența acestui nivel de aplicație se regăsesc și cele instalate de pe Play Store sau cele dezvoltate de programatori.

Cele mai importante componente ale unei aplicatii Android sunt:

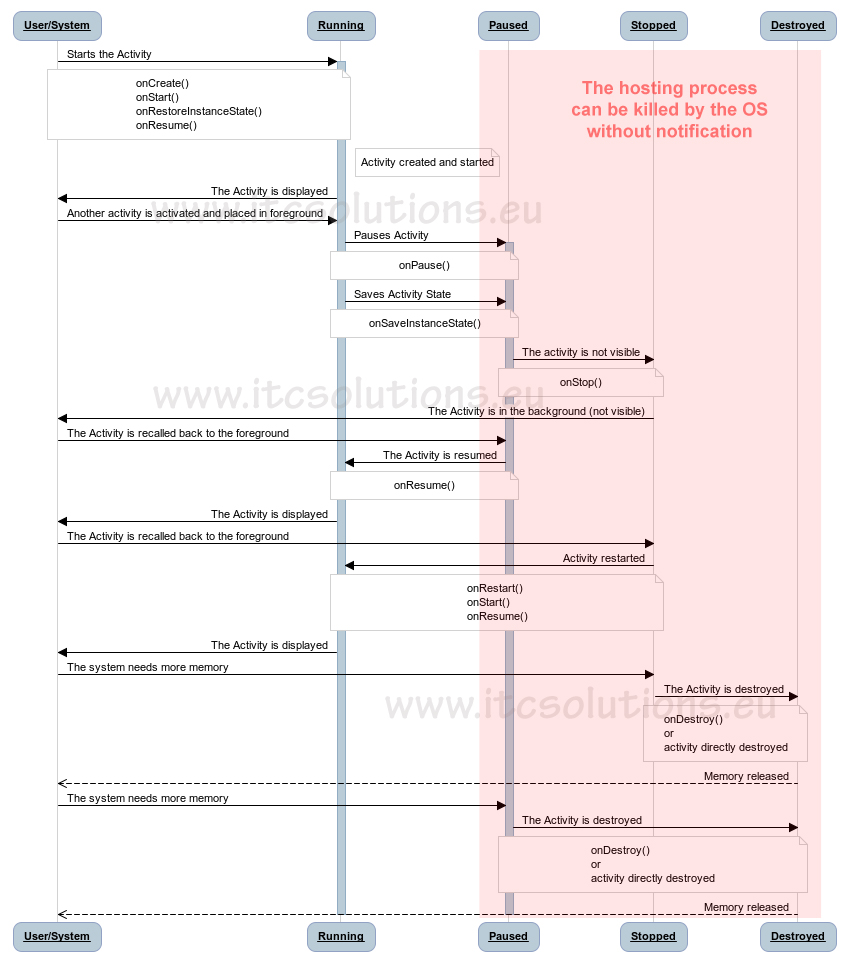
#### **2.3.5.1 Activity (Activitate)**

Reprezinta o interfata cu utilizatorul,fereastra sau formular. O aplicatie Android poate avea una sau mai multe activitati: de exemplu o aplicatie de tip Agenda poate avea o activitate pentru a gestiona contactele, o activitate pentru a gestiona intalniri si una pentru a edita o intrare in agenda.

Fiecare Activitate are propriul sau ciclu de viata, independent de ciclul de viata al procesului asociat aplicatiei. Fiecare activitate are propria stare si datele acesteia pot fi salvate sau restaurate. Activitatile pot fi pornite de aplicatii diferite (daca e permis). Are un ciclu de viata complex deoarece aplicatiile pot avea activitati multiple si doar una este in prim-plan; utilizand managerul de activitati, sistemul Android gestioneaza o stiva de activitati care se gasesc in diferite stari (pornire, in executie, intrerupta, oprita, distrusa).

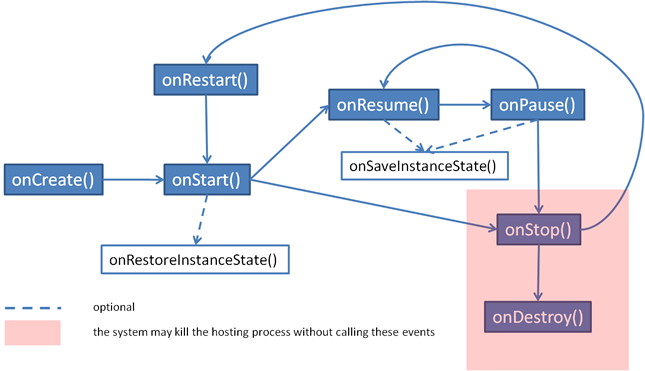
In SDK, activitatea este implementata folosind o subclasa a clasei Activity care extinde clasa Context. Ciclul de viata al unei activitati Ciclul de viata al unei activitati descrie starea in care o activitate poate fi la un moment dat:[6]

* **Running** - Activitatea a fost creata (*onCreate()*), pornita (*onStart()*) si este afisata pe ecranul aparatului; in cazul in care activitatea a mai fost utilizata si aplicatia a salvat starea acesteia (*onSaveInstanceState()*), activitatea este reluata din acel punct (*onRestoreInstanceState()* si*onResume()*); in aceasta stare utilizatorul interactioneaza cu activitatea prin intermediul interfetei dispozitivului (tastatura, touchscreen, display);
* **Paused** - Activitatea pierde prim-planul (*onPause()*), deoarece o alta activitate este executata, cum ar fi o fereastra de dialog; de asemenea, in cazul in care aparatul intra in modul *sleep*, activitatea este oprita temporar; activitatea isi poate relua executia (*onResume()*) si este plasata inapoi in prim-plan;
* **Stopped** – Activitatea nu este mai in uz si pentru ca este oprita (*onStop()*) nu este vizibila; pentru a fi reactivata (ea deja exista), activitatea trebuie sa fi repornita (*onRestart()* si *onStart()*) si reluata (*onResume()*);
* **Destroyed** – Activitatea este distrusa (*onDestroy()*) si memoria sa eliberat, deoarece nu mai este necesara sau sistemul are nevoie de memorie suplimentara pentru rutinele proprii sau pentru alte activitati; deoarece managementul memoriei este un aspect important pentru sistemul de operare Linux al dispozitivului mobil, procesul care gazduieste o activitate intrerupta, oprita sau distrusa, poate fi terminat pentru a elibera memorie pentru noi activitati; doar procesele ce gestioneaza activitati ce ruleaza sunt protejate; [6]



Dupa cum se poate observa in imaginea anterioara, Activitatea are mai multe stari intre care exista tranzitii clare. In ciuda faptului ca lucrurile pot arata complicat, in realitate ele sunt mult mai simple daca ne concentram pe urmatoarele elemente:

* o singura Activitatea poate fi in prim-plan la un moment dat;
* doar sistemul gestioneaza starile si tranzitiile unei Activitati si nu programatorul (nu in mod direct, deoarece atunci când se lanseaza o activitate noua se modifica implicit starea activitatii curente). [6]

**Ciclul de viață al unei activități**

* **onCreate(Bundle)** – apelata cand activitatea este creata; folosind argumentul metodei de tip *Bundle*exista posibilitatea sa restabiliti starea activitatii, care a fost salvata intr-o sesiune anterioara; dupa ce activitatea a fost creata, va fi pornita (*onStart()*);
* **onStart()** – apelata in cazul in care activitatea urmeaza sa fie afisata; din acest punct, activitatea poate veni in prim-plan (*onResume()*) sau ramane ascunsa in fundal (*onStop()*);
* **onRestoreInstanceState(Bundle)** – apelata in cazul in care activitatea este initializata cu datele dintr-o stare anterioara, ce a fost salvata; in mod implicit, sistemul restaureaza starea interfetei cu utilizatorul (starea controalelor vizuale, pozitia cursorului, etc);
* **onResume()** – apelata cand activitatea este vizibila iar utilizatorul poate interactiona cu aceasta; din aceasta stare, activitatea poate fi plasata in fundal, devenind intrerupta (*onPause()*);
* **onRestart()** – apelata in cazul in care activitatea revine in prim-plan dintr-o stare oprita (stopped); dupa aceasta, activitate este pornita (*onStart()*) din nou;
* **onPause()** – apelata atunci când sistemul aduce in prim-plan o alta activitate; activitatea curenta este mutata in fundal si mai târziu poate fi oprita (*onStop()*) sau repornita si afisata (*onResume()*); acesta este un moment bun pentru a salva datele aplicatiei intr-un mediu de stocare persistent (fisiere, baze de date) deoarece dupa aceasta faza activitatea poate fi terminata si distrusa fara a se anunta acest lucru.
* **onSaveInstanceState(Bundle)** – apelata pentru a salva starea curenta a activitatii; in mod implicit, sistemul salveaza starea interfetei cu utilizatorul;
* **onStop()** – apelata in cazul in care activitatea nu mai este utilizata si nu mai este vizibila deoarece o alta activitate interactioneaza cu utilizatorul; din acest punct, activitatea poate fi repornita (*onRestart()*) sau distrusa (*onDestroy()*);
* **onDestroy()** – apelata in cazul in care activitatea este distrusa, iar memoria sa eliberata; acest lucru se poate intâmpla in cazul in care sistemul necesita mai multa memorie sau daca programatorul termina explicit activitatea apeland metoda *finish()* din clasa Activity; [6]

Pentru a supradefini metodele anterioare, trebuie sa se acorde atentie semnaturii acestora (numele metodei si lista de parametri). Este mai sigur sa folosesti adnotarea @Override pentru a solicita o validare din partea compilatorului (in cazul in care metoda definita nu supradefineste o metoda din clasa de baza, atunci veti obtine o eroare de compilare).

Pentru a supradefini onCreate() sau onPause(), solutia este:

public class SomeActivity extends Activity {

// The activity is being created.

@Override

public void onCreate(Bundle savedInstanceState) {

// DO NOT forget to call the base method

super.onCreate(savedInstanceState);

}

@Override

protected void onPause() {

// DO NOT forget to call the base method

super.onPause();

}

}

Atunci când se supradefineste una din metodele *onXXX()*, trebuie sa apelati (prima instructiune din metoda) forma metodei din clasa de baza (Activity), deoarece scopul acestei abordari este de a adauga propriul cod la evenimentele activitatii si NU de a le rescrie.

Dupa cum se poate observa in ultimele doua imagini, activitatea se poate gasi in stari, cum ar fi Stopped sau Destroyed, in care sistemul de operare poate termina procesul care gazduieste activitatea. Sistemul de operare va elibera memoria acupata de activitate, fara a apela metodele *onStop()* si *onDestroy()*. In concluzie, evenimentul anuntat de *onPause()* este momentul recomandat pentru a salva starea unei activitati inainte ca aceasta sa fie distrusa.[6]

#### **2.3.5.2 Intent (Intentie):**

* Reprezinta o entitate folosita pentru a descrie o operatie care urmeaza sa fie executata;
* Oarecum similar cu conceptual de event-handler din NET sau Java;
* Un mesaj asincron utilizat pentru a activa activitati sau servicii; Gestionata de o instant a clasei Intent.

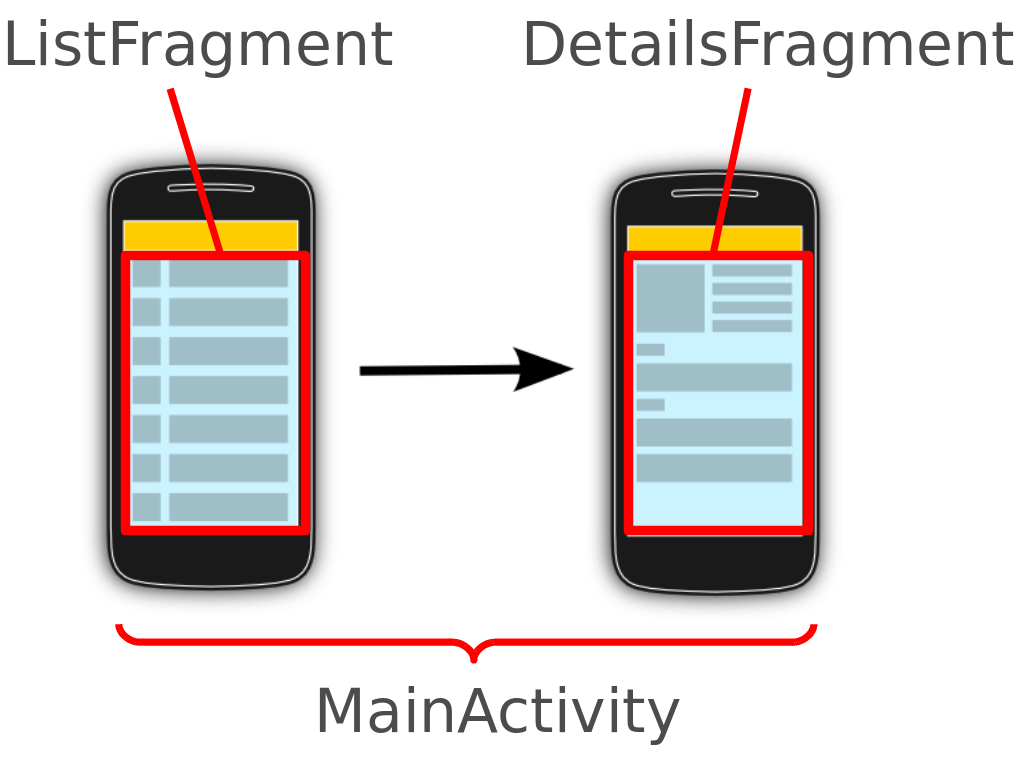
#### **2.3.5.3 Service (Serviciu):**

* Un task care se executa in fundal, fara interactiunea directa cu utilizatorul;
* Gestionata de o instant a clasei Service;
* Content provider(Furnizor sau manager de continut);
* Un API folosit pentru a gestiona datele private ale aplicatiei o Un sistem de management de date ce descrie o alternativa la sistemul de fisiere, baze de date SQLite sau orice alta solutie de stocare persistenta;
* Implementata de o subclasa a clasei ContentProvider; o solutie pentru a partaja si controla (pe baza de pemisiuni) transferul de date intre aplicatii (de exemplu , sistemul Android ofera un furnizor de continut pentru datele de contact);
* Broadcast receiver;
* Oarecum similar cu conceptual de handler global (sau evenimente de sistem);
* Implementata de o subclasa a clasei BroadcastReceiver

#### **2.3.5.4 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.[7]



Obiectele de tip fragment au propiul 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 cand 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.

#### **2.3.4.5 Fisierul Manifest**

Fiecare aplicație trebuie să aibă un fișier AndroidManifest.xml în directorul principal. Acest fișier conține informații referitoare la toate componentele, permisiile, seviciile și librăriile utilizate în aplicație. 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.

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



## **2.4 Resursele aplicatie android**

Aplicatiile utilizeaza siruri de caractere, bitmap-uri, fisiere video si audio pentru a satisface exigentele utilizatorilor. Toate aceste reprezinta resurse ale aplicatiei. In ciuda faptului ca sunt controlate din codul sursa, acestea reprezinta resurse externe si recomandarea este sa fie externalizate (nu le definitm in codul aplicatiei; de exemplu un string constant), pentru a facilita:

* internationalizarea interfetei cu utilizatorul prin furnizarea acesteia in diferite limbi si formate;
* portabilitatea aplicatiei pe dispozitive diferite.

Resursele sunt stocate in directorul *res* din cadrul proiectului Android. Pentru a gestiona diferite tipuri de resurse, directorul *res* al proiectului contine subdirectoare specifice:

|  |  |
| --- | --- |
| Director | Resursa |
| res/animator/ | fisiere XML ce definesc proprietatile animatiilor |
| res/anim/ | fisiere XML pentru animatii bazate pe transformari |
| res/color/ | fisiere XML ce definesc culori |
| res/drawable/ | Fisiere bitmap (.png, .jpg, .gif) sau fisiere XML ce definesc resurse ce pot fi desenate |
| res/layout/ | fisiere XML ce definesc design-ul interfetei |
| res/menu/ | fisiere XML ce definesc meniurile aplicatiei |
| res/raw/ | fisiere diferite in format binar sau text |
| res/values/ | fisiere XML care contin valori simple, cum ar fi siruri de caractere, numere intregi si culori. Este recomandat sa utilizati conventiile de nume de fisiere pentru anumite tipuri de resurse: arrays.xml, colors.xml, dimens.xml, strings.xml, styles.xml |
| res/xml/ | fisiere XML care pot fi citite |

Toate resursele din subdirectoarele directorului *res/* sunt incluse in pachetul aplicatiei de catre compilator. Pentru a facilita accesul si controlul resurselor din codul aplicatiei, compilatorul genereaza o clasa, numita **R**, care contine identificatori statici utilizati pentru referirea fiecarei resurse.

Deoarece fisierele din resurse sunt compilate, NU salvati fisiere direct in interiorul directorului *res/*, deoarece aceasta va cauza o eroare de compilare.

De exemplu, este recomandat sa se defineasca si sa se salveze valori constante de tip String in interiorul fisierului **res/values/strings.xml** si sa nu se defineasca direct in cod. Fiind definite in afara codului sursa inseamna ca le puteti modifica fara a afecta codul sursa Java. Fisierul **strings.xml** poate arata ca acesta:

**<?xml** version="1.0" encoding="utf-8"**?>**

**<resources>**

**<string** name="hello"**>**Hello World, Activity!**</string>**

**<string** name="app\_name"**>**Android**</string>**

**</resources>**

Asa cum am mai spus, pentru fiecare resursa compilatorul genereaza un identificator static si unic in clasa **R**. Identificatorul static are acelasi nume ca si elementul din strings.xml.

**public final class** R {

*//...*

**public static final class** string {

*//unique ID for app\_name string in strings.xml*

**public static final int** app\_name=0x7f040001;

*//unique ID for hello string in strings.xml*

**public static final int** hello=0x7f040000;

}

}

Deci, daca se doreste preluarea valorii elementului *hello* (din strings.xml) intr-o variabila de tip String, se va folosi referinta statica din clasa **R**:

String stringHello = **this**.getString(R.string.hello);

## **2.5. Securitatea**

Aplicațile Android fac uz de hardware și software avansat, atât de date locale, cât și pe server, expuse prin intermediul platformei de a aduce inovație și valoare pentru utilizatori. Pentru a proteja această valoare, platforma trebuie să ofere un mediu aplicație care asigură securitatea utilizatorilor, date, aplicații, dispozitivul mobil, și rețeaua.

Din punct de vedere al securităţii şi al asigurării intimităţii utilizatorilor sistemul Android este unul sigur. Atunci când o aplicație este instalată se acordă în mod explicit permisiuni de acces la resursele sistemului; în general fiecare aplicaţie rulează izolat față de toate celelalte. Magazinul oficial de aplicaţii oferit de Google, numit Play Store, afişează la începutul instalării toate permisiunile necesare unei aplicaţii pentru a funcționa fără întreruperi. Dacă utilizatorul nu este de acord cu aceste permisiuni poate alege să nu instaleze aplicaţia. Această proiectare a fost efectuată împotriva “atacurilor” comune, cum ar fi cele sociale prin convingerea utilizatorilor să instaleze malware (viruși).

Astfel, sistemul reduce semnificativ impactul vulnerabilităţilor însă din cauza unei documentaţii superficiale s-a creat o confuzie pentru dezvoltatori, ajungându-se ca aplicaţiile să ceară mai multe drepturi de acces decât e necesar, fapt ce slăbeşte securitatea sistemului. Din acest motiv Google a creat un sistem, Android Verify Apps [1], care rulează în background pentru a detecta procese maliţioase. Datorită faptului că Android-ul e open source, unele companii li s-a permis să ia versiuni deja existente şi să le modifice pentru a putea fi utilizate într-un mod mai sigur. Un astfel de exemplu poate fi compania Samsung care pe baza Android Jelly Bean a creat proiectul „Knox” care permite folosirea în siguranță a telefonului.

# **9.0 Lista acronimelor**

AAC – Advanced Audio Coding

ADT – Android Developer Tools

API - Application Programming Interface

BSD - Berkeley Software Distribution

CDMA – Code Division Multiple Access

GIF – Graphics Interchange Format

GPL – General Public License

GPRS- General Packet Radio Service

GPS - Global Positioning System

JPEG – Joint Photographic Experts Group

MPEG – Moving Picture Experts Groug

PNG – Portable Network Graphics

SDK – Software development kit

XML – Extensible Markup Language

# **10. Bibliografie**

[1] <https://en.wikipedia.org/wiki/Android_(operating_system)>

[2] <https://ro.wikipedia.org/wiki/Android_(sistem_de_operare)>

[3] <http://www.androidauthority.com/?q=ARM+vs+x86>

[4] https://ro.wikipedia.org/wiki/Linux\_(nucleul)

[5] <https://class.coursera.org/android-001/lecture>

[6] <http://www.itcsolutions.eu/2011/09/08/android-tutorial-concepte-activitati-si-resurse-ale-unei-aplicatii-android/>

[7] <http://www.vogella.com/tutorials/Android/article.html>