**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

****

LUCRARE DE LICENȚĂ

**Mobile Control**

**propusă de**

***Samuel Timofte***

**Sesiunea:** *Iulie, 2018*

**Coordonator științific**

Lect. dr. Vlad Radulescu

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

Mobile Control

*Samuel Timofte*

**Sesiunea:** *Iulie, 2018*

**Coordonator științific**

Lect. dr. Vlad Radulescu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele Lect. dr. Vlad Radulescu

Data \_\_\_\_\_\_\_\_\_\_\_\_ Semnătura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

[Introducere 4](#_Toc518026821)

[Contributii 5](#_Toc518026822)

[Informatii ajutatoare 5](#_Toc518026823)

[Industria auto 5](#_Toc518026824)

[Embedded in automotive 8](#_Toc518026825)

[De ce Android ? 11](#_Toc518026826)

[Concept mobile control 15](#_Toc518026827)

[Implementare concept 18](#_Toc518026828)

[1.Meniu andoid 21](#_Toc518026829)

[2.Controlul. 23](#_Toc518026830)

[3.Interfata statica 24](#_Toc518026831)

[4.Interfata dinamica. 25](#_Toc518026832)

[Activitati android. 26](#_Toc518026833)

[5.Json 29](#_Toc518026834)

[6.Websocket. 31](#_Toc518026835)

[Mesaje CAN 33](#_Toc518026836)

[1.Creare mesaj CAN 33](#_Toc518026837)

[2.Mesaj CAN in Json 35](#_Toc518026838)

[3.Transmitere mesaj CAN – trimitere ciclica mesaj can explica 35](#_Toc518026839)

[4.Receptare mesaj CAN (json) 37](#_Toc518026840)

[5.Implementarea functiei “SendMessageHandler”: 38](#_Toc518026841)

# Introducere

Evolutia, inteligenta si goana dupa confort au fost unele dintre cele mai importante motive pentru motivul translatiei de la mersul pe jos, la calarit, la caruta si apoi masini cu abur, urmate de cele pe ardere interna. De cand omul a “descoperit” masina, tot timpul s-au gasit lucruri de imbunatatit. De la o banca din lemn la un scaun ergonomic, de la o roata din lemn, la un pneu sustinut de telescoape si un intreg chit de suspensie, de la un volen greu de rotit la sisteme servo, de la un felinar pe ulei la faruri led si laser. Evolutia acestor sisteme nu prea pare sa ajunga la o limita cunoscuta.

Industria auto, fiind o industrie foarte mare, inca de la inceput a fost domintata de anumiti giganti. Asa incat inca de la aparitia motorului cu ardere interna, un numar mic de corporatii au impartit piata auto intre ei. Si pentru ca a existat din totdeauna aceasta lupta pentru a castiga piata, au investit in cercetare. Asa au inceput sa apara tehnologii noi, scopurile principale fiind siguranta si confortul utilizatorilor.

Odata cu aparitia calculatorului, au inceput sa aleaga controlul electronic al componentelor, peste cel mecanic, fiind mult mai sigur si fiabil. Asa a luat nastere in industria automotiv utilizarea componentelor electronice. Odata cu aceste componente, a aparut nevoie da protocoale automotive, concepte si programatori specializati in domeniu.

In ultimii ani, prin conceptul de „internet of things” a inceput sa se digitalizeze si mai mult aceasta industrie. Licenta mea vine ca o continuare logica a evolutiei. Mobile Control, prezinta o posibila varianta de comunicare intre om si masina, via Wi-fi, in care constructorul de masina poate sa ofere utilizatorului control si observatie asupra anumitelor componente integrate in masina.

Ideea este simpla. ECU-urile integrate pe masina comunica printr-un protocol numit CAN, si pentru comunicare folosesc mesaje de tip CAN. Mobile Control se comporta ca un ECU (trimite si el mesaje CAN), doar ca de pe o tableta Android, si pentru a avea acces la reteaua in care sunt conectate ECU-urile, se foloseste de un server montat pe un Raspberry Py.

Conceptul nu este nou, dar ca si contributie personala, am venit cu ideea unei interfete generate dinamic, creata specific pe ce „suporta” componentele integrate in automobil.

# Contributii

Pentru dezvoltarea conceptului Mobile Control a trebuit sa invat multe protocoale, standarde, tehnologii si tool-uri.

Vom prezenta pe scurt contributiile aduse pentru lucrarea prezentata:

Dezvoltarea concetului Mobile Control, diagnoza wi-fi, unde aportul principal legat de concept a fost ideea de a avea o interfata generata dinamic, care la conectarea cu un dispozitiv cu capacitate de server, sa primeasca informatii despre automobilul in cauza, si sa genereze interfata utilizabila specific pentru el.

Pentru a demonstra functionalitatea conceptului a fost dezvoltata o aplicatie mobile, (contributie integral personala), unde au fost implementate functionalitatile de baza pentru a sustine conceptul Mobile Control cum ar fi, conexiune cu server-ul (websocket), creare/prelucrare/stocare/trimitere de mesaj CAN, interfata statica, interfata dinamica, si doar pentru a prezenta concepte mobile au fost create si 2 meniuri mobile.

O contributie majora este si configurarea mediului de lucru. Fiind o lucrare care este distribuita pe mia multe platforme (Windows, Andoid, Raspian) si avand una dintre functionalitati transmiterea de mesaj de tip CAN de pe aplicatie Mobile pana la un controler electronic din industria automotive (ECU), o provocare a fost sa creezi o cale de mijloc intre Andoid si ECU. Aici intervine un dispozititv cu capacitatile unui computer, unde au fost montate 2 routere wireless, si a fost configurat si comportamentul lor. Unul a fost configurat ca la pornirea Rasperry-ului sa se conecteze cu un router conectat la PC-ul pe care lucram, pentru capabilitati de cross-compiler (dezvoltare pe platforma Windows – compilare directa, prin wi-fi, pe Raspian), iar celalalt router era folosit pentru conexiunea cu tableta pe care rula aplicatia Mobile Control.

La nivelul serverului pe langa partea de server din websocket-ul descris la nivel de aplicatie a fost nevoie de utilizarea interfetei CanInterface, interfata capabila sa foloseasca capabilitatile modulului de CAN montat pe Raspberry Py.

De mentionat (tehnologii si protocoale folosite) :

* tehnologie JSON
* CanInterface (interfata dezvoltata pentru functionalitati ale protocolului CAN)
* Raspian (sistem de operare pentru Raspberry Py)
* Sincronizare de biblioteci (cygwin)
* Mesaje CAN
* Standard OBD
* Simulare CANoe (unde simulam comportamnetul Bus-ului de CAN)
* Hardware
  + Raspberry Py
  + Sursa de curent
  + ECU (Electronic Control Unit)
  + CANCase
  + Fire de cupru ( CAN hight/low )

# Informatii ajutatoare

## Industria auto

Industria automotive a avut debutul in anii 1890 cu sute de vehicule care erau devoltate

avand tractiune mecanica.

1896 Henry Ford contruieste un “quadricycle” (foto).

 (1)<https://corporate.ford.com/history.html>

Primul vehicul contruit de Henry Ford, contruit pe 4 roti de bicicleta, avand un mototr estimate la 4 cai putere. Cutia de viteze avea 2 viteze pentru mersul inainte si 0 pentru mersul cu spatele.

In anul 1929, inainte de “Great Depression”, erau aproximativ 32 de milioane de automobile, dintre care 90% erau produse de industria Americana.

In anul 1980 Japonia a preluat titlu de cel mai mare exportator auto din lume, titlu revendicat de China in 2009, un titlu greu de catigat de alta natiune deoarece China produce anual aproximativ 28 de milioane de unitati (2(2) <http://www.oica.net/category/production-statistics/>

), echivalentul urmatoarelor 3 state din clasament(USA 12 mil, Japonia 9 mil, Germania 6 mil). Alte tari care s-au remarcat prin exportul de masini : India 4,4 mil, Coreea de Sud 4,2 mil, Mexic 3,5 mil, Canada 2,3 mil. (statistica 2016)

Pentru a evidentia si mai mult dimensiunile acestei industrii, in 2007 erau inregistrate 806 milioane de masini si camioane care cunsumau aproximativ 980 de miliarde de litri de combustibil annual (3 (3)[*"Automobile Industry Introduction"*](http://www.plunkettresearch.com/Industries/AutomobilesTrucks/AutomobileTrends/tabid/89/Default.aspx). Plunkett Research. 2008*. Retrieved 25 March 2014*.

).

Pe parcursul evolutiei masinilor, au aparut ECU-uri specifice automotive, Electronic Control Unit, este un termen aproape generic care se refera la partea de embedded system care controleaza unul sau mai multe sisteme sau subsiteme ale vihiculului. Foarte curand au “acaparat” piata automotive, fiecare masina produsa in zilele noastre are un numar impresionant de ECU-uri. LCU Light Control Unit:

Modul de control al luminilor montate pe un automobil. Controlat electronic, poate sa aiba controleze aprinderea si stingerea becuril/led/laser, dar si controlul oglinzilor pentru a regla fanta de lumina.

Pentru ca industria IT a deschis un nou drum in industria automotive, au aparut noi companii, adevarate corporatii care au inceput sa se ocupe de dezvoltarea sistemelor de control electronic, precum Continental, Sony sau Bosch.Acest drum a produs schimbari masive in ce intelegem confort in automobil, siguranta si eficenta. De la scaune electrice care se ajusteaza automat dupa nevoile pasagerului, la efincetizarea folosirii electricitatii intre sistemele electrice montate pe automobil, la Glare Free Lighting care ajusteaza fanta de lumina produsa de sistemul de lumini pentru un confort maxim al pasagerului si al celorlalti participanti la trafic.



Corporatia Continental, cunoscuta pentru amvelope, a preluat in 2007 Siemens AG, si in 2012 ajungea pe locul 3 in lume la vanzari in industria componentelor automotive(4

(4) https://www.autonews.com/assets/PDF/CA89220617.PDF

). Cu vanzari de peste 34,5 miliarde de euro in 2014, cu peste 200 000 de angajati in 53 de tari.Pe plan local, Continental este unul dintre cei mai mari angajatori, cu peste 20 000 de angajati in mai multe domenii de activitate, printre principale fiind si industria software unde chiar in Iasi sunt aproape 2000 de angajati in 5 departamente de R&D(Research and development). (<http://www.zf.ro/companii/continental-a-depasit-grupul-renault-ca-numar-de-angajati-in-romania-si-se-pregateste-sa-treaca-pragul-de-20-000-de-salariati-16254308>)

https://www.ziaruldeiasi.ro/stiri/continental-pregateste-noi-angajari-la-iasi--91818.html

Pentru a intelege mai bine dimensiunea activitatii corporatiei Continental in Romania, de notat este faptul ca in anul 2017 s-a produs la fabrica din Timisoara amvelopa cu numarul 200 000 000(https://www.continental-corporation.com/ro-ro/presa/comunicate-de-presa/continental-anvelope-timisoara-a-produs-anvelopa-cu-numarul-200-de-milioane--95570). Un numar care pune tara noastra in topul tarilor cu export de amvelope notabil.

## Embedded in automotive

Dupa cum am spus, in industria auto, pe partea electronica, ECU, Electronic Control Unit, este un termen generic care se refera in general la partea de Embedded System, care controleaza una sau mai multe sisteme electrice sau subsisteme intrun automobil.

Tipuri de systeme embedded :

* ECM (Electronic/engine Control Module)
* PCM (Powertrain Control Module)
* TCM (Transmission Control Module)
* BCM (Body Control Module)
* LCU (Lights Control Unit)

Aceste module comunica intre ele printr-un protocol automotive, numit CAN Protocol,

folosind o retea numita CAN Bus.

CAN, Controller Area Network, protocol dezvoltat de Robert Bosch in anul 1983. Lansat in anul 1986 la Society of Automotive Engineers(SAE) in Detroit Michigan. Primele cipuri de tip CAN controller au fost produse de Intel si Philips si au iesit pe piata in anul 1987. In 1988 BMW seria 8 a fost prima masina cu un system bazat pe CAN.

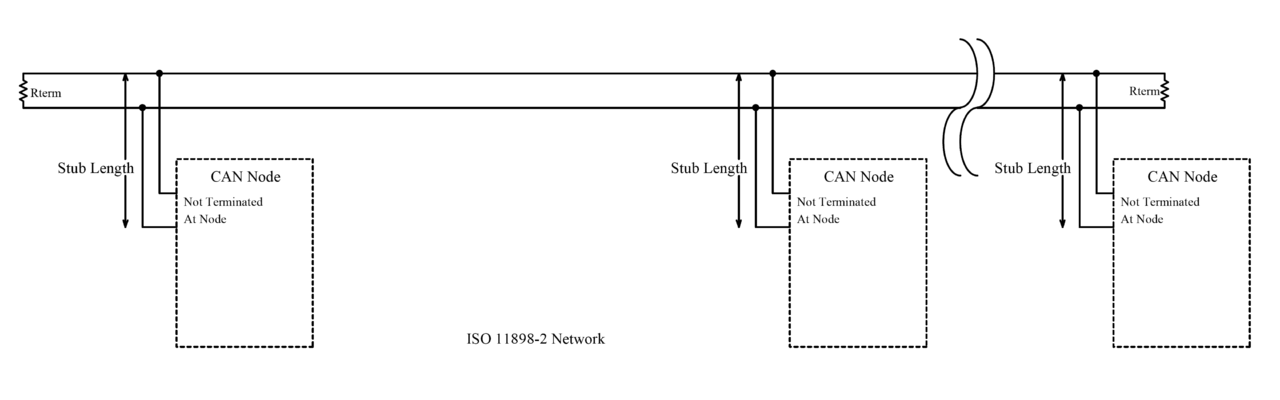
Bosch a publicat mai multe variante de CAN, dar CAN 2.0 a fost publicat in 1991. Care are 2 variante, una cu identificator pe 11 biti (A), si variant B cu identificator pe 29 de biti, usual numit CAN 2.0B.

CAN bus este un standard care ajuta microcontrolerele montate pe masina sa comunice intre ele. Este un protocol bazat pe mesaje (CAN messages).

Un automobile modern are in medie 70 de componente electronice(ECU), pentru sistemele sale si subsisteme. Deobicei cel mai mare processor este montat pe unitatea care gestioneaza motorul, (Engine Control Unit).

(can si lin)

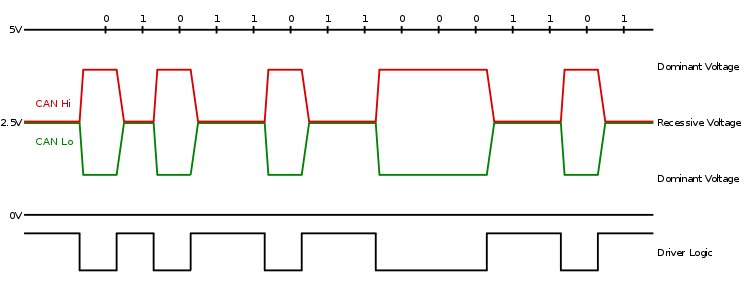
3.1Arhitectura CAN este un multi-master serial bus standard pentru conexiunea componetelor electronice. Doua sau mai multe ECU-uri au nevoie de o retea CAN pentru a comunica. Toate nodurile sunt conectate intre ele printr-un “bus” de 2 fire. ISO 11898-2, usual : High Speed CAN, utilizeaza un bus linear care la fiecare capat are un resistor de 120 de ohmi.(1)



https://en.wikipedia.org/wiki/File:CAN\_ISO11898-2\_Network.png

(1)High Speed CAN Network. ISO 11898-2

In continuare vom discuta despre CAN high speed. Semnalul High Speed CAN este transmis pe 2 fire. Este foarte diferit de semnaul digital normal. In semnalul digital normal “1” semnifica tensiune pe fir, pe cand “0” semnifica absenta tensiunii. La High Speed CAN cele doua fire sunt denumite “high” si”low”. Cand este transmis un semnal High Speed CAN, firul “high” trage tensiune de 5 V, si firul “low” coboara la 0 V (dup ace intial amandoua erau la 2,5 , toate astea pentru a transmite un (0) dominant. Iar cand amandoua sunt cu o tensine de 2.5 V pe bus este transmis un bit (1) dominant. Ca in imagine ace urmeaza : (2).



(2)https://upload.wikimedia.org/wikipedia/commons/1/1a/ISO11898-2.svg

Bus-ul de CAN este unul dintre cele 5 protocoale utilizate in “on-board diagnostics (OBD)-II vehicle diagnostics standard”. Standardul OBD-II este obligatoriu pentru toate masinile de uz personal cat si pentru camioanele usoare vandute in Statele Unite inca din 1996, iar standardul EOBD este obligatoriu pentru toate masinile pe benzina vandude in Uniunea Europeana inca din anul 2001, iar pentru masinile pe diesel din 2004.

Pentru a intelege concepte si utilizari ale standardului ne vom folosii de o mica descriere preluata de pe site-ul : “http://www.e-automobile.ro/categorie-diagnoza/149-diagnoza-auto-obd-citire-parametrii-motor.html” .

“ Producatorii auto sunt obligați să transmită un număr minim de parametrii ai motorului către un dispozitiv de diagnosticare OBD 2. Acești parametrii sunt:

* Turația motorului
* Temperatura lichidului de racire
* Viteza automobilului
* Presiunea combustibilului din rampă (diesel)
* Sarcina calculată a motorului

    Pe lângă acești parametrii fiecare constructor poate decide dacă mai adaugă informații adiționale sau nu.

    Acești parametrii mai sunt utilizați pentru a memora informații despre starea motorului (freeze frame) în cazul aparitiei unui defect (DTC). În plus pot furniza informații obiective pentru a putea diagnostica o eventuala funcționare defectuoasă a motorului sau pentru a citi istoricul motorului.

    De exemplu parametrul cu identificatorul (0x31) ne informează cați km au fost parcurși de la ultima ștergere a unui cod de defect (DTC). Informația poate fi utilă la achiziționarea unui automobil utilizat, pentru a verifica veridicitatea rulajului afișat în bordul automobilului.

    Parametrii fizici ai motorului (presiune, debit de aer, presiune combustibil) pot fi inregistrați și apoi reprezentați grafic pentru o mai bună înțelegerea a funcționării motorului, atât în regim de funcționare normală dar mai ales în cazul funcționării defectuoase.”

Am explicat standardul OBD deoarece conceptul Mobile Control este un concept definit in mod special pentru a citii diagnoza .

## De ce Android ?

Dupa mii de ani de evolutie a umanitatii, in ultimul secol oamenii de stiinta au facut un salt mare in tot ce inseamna umanitate. Chiar daca la inceput nu se asteptau ca o conglomeratie de cabluri, electricitate si porti logice sa atinga, sa modeleze si chiar sa controleze atat de mult din tot cea ce ne reprezinta. De la agricultura mecanizata si controlata prin internet la operatii effectuate pe oameni de roboti medicinali, pana la “banalul” smartphone. Ultimul din urma fiind atat de raspandit pe Pamant, incat in zonele dezvoltate ale planetei, aproape fiecare fiinta umana mai mare de 5 ani detine unul. Iar dreptul la internet a ajuns : “**Potrivit unei rezoluţii a Consiliului pentru Drepturile Omului din cadrul ONU, accesul la internet este un drept uman de bază.”(** [**http://www.capital.ro/onu-accesul-la-internet-este-un-drept-al-omului.html**](http://www.capital.ro/onu-accesul-la-internet-este-un-drept-al-omului.html)**)**

La bazele evolutiei rapide ale informaticii, sta si un roman care este considerat parintele Ciberneticii, ramura a informaticii. Meritele lui fiind recunoscute, ca in majoritatea cercetatorilor romani, post-mortem. A ajuns atat de apreciat incat in 1990 in Elvetia a fost deschisa in onoarea lui “Academia de Cibernetică din Elveția”, iar in Romania sunt 2 licee si un spital care ii poarta numele.( <https://ro.wikipedia.org/wiki/%C8%98tefan_Odobleja>)

Dupa ce calculatoarele au renuntat sa mai ocupe camere intregi si prin initiativele Dell si Microsoft in care liderii aveau in plan (inimaginabil la vremea lor, si fantazmagoric), ca si concept “personal computer”, calculator personal.Concept in care toata lumea sa aiba acces la un calculator propriu. Desi nu avea sorti de izbanda acest concept a ajuns realitate, si mult mai mult de atat. Acum intr-o masina sunt mai multe calculatoare decat numarul de calculatoare folosit de omenire sa trimita primul om pe luna, Apolo 11. Si pe langa PC-uri, “personal computer”, aproape fiecare om, cum am mentionat mai sus, detine si un smartphone, care fara doar si poate este un calculator, poate putem sa ii spunem un PC miniaturizat.

Desi primul telefon “smart”, este considerat de omaneii de stiinta ca fiind prototipul “Angel” dezvoltat de Frank Canova in 1992 (https://en.wikipedia.org/wiki/Smartphone#History ), dar primul model a fost lansat de IBM(compania unde lucra Frank), “IBM Simon” pe numele lui



(https://en.wikipedia.org/wiki/File:IBM\_Simon\_Personal\_Communicator.png )  touchscreen, [email](https://en.wikipedia.org/wiki/Email), and PDA capabilitati. Nasterea smartphon-ului modern, in perspectiva pietei si a publicului larg, a fost introducerea pe piata mobile a modelelor “iPhone”. Inca de la primul model compania Apple a dat tonul in inovatie si design pentru multe din modelele aparute pe piata. Avand baza software diferita de toate lelelate modele existente, dar la aceasta vom revenii mai tarziu.De mentionat este ca smartphon-urile actuale ajung sa aiba specificatii exosrbitante, cum sunt 12-14 Gb memorie Ram sau 256 Gb memorie Rom, baterii de 12000 mA, camere foto de peste 20 mP, si chiar cu vedere infra-rosu (model de telefonprezentat anul trecut de compania

Caterpilar) (http://www.mycoolbin.com/2016/02/20/caterpillar-unveils-worlds-first-smartphone-with-built-in-thermal-camera/).

Avand toate functiile de baza, conexiune la internet, creare si editare de fisiere, capacitati de conectare la monitoare, imprimante si tot felul de periferice, capabile de calcule numerice si doatate cu putere de procesare care in trecutul apropiat erau inimaginabile si pentru statiile PC ultramoderne, putem trage concluzia clara ca acest concept de “Personal computer” care dupa cum am mentionat parea utopic, a fost depasit considerabil, si inca nu a ajuns la o bariera sau o limitare fizica in dezvoltare si inovatie, deci, e foarte probabil ca in timpul vietii nostre sa observam modificari substantiale ale smartphon-ului si modului in care acesta ne afecteaza vietile.

Avand complexitate hardware, de la inceputurile calculatorului a aparut necesitatea unui sistem de operare, “OS – Operatin Sistem”, un sistem care sa faca usoara manipularea resurselor si a puterii de procesare oferita de hardware.

Primul OS a fost facut de General Motors, GM Operatin sistem, pentru IBM 701 in 1955.(<http://www.csee.wvu.edu/~jdm/classes/cs258/OScat/early.html)>. Dar cel care a venit cu primul model de OS usor de folosit de publicul larg a fost cel care a ramas 2 decenii in topul celor mai bogati oameni din lume, mult timp fiind pe primul loc. Bill Gates, CEO Microsoft, a venit cu ideea unui Operatin Sistem “user frendly”, Acesta idee si tot software-ul produs de Microsoft a propulsat compania pe primele locuri in IT inca de la infiintare, pana in zilele noastre.

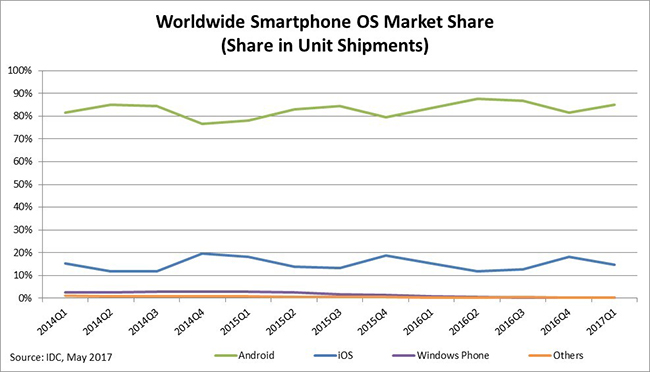
Este timpul sa impartim OS-urile in 2 categorii. OS-uri pentru PC-uri(traditionale) si OS-uri pentru aria “mobile, arie larg dezvoltata. Pentru PC-uri sunt 3 mari OS-uri folosite in ziua de azi, Linux, Windows, si macOs. Linux a fost gandit sa fie gratis, open source, si este folosit mai mult in dezvoltare. Windows, OS-ul dezvoltat de Microsoft, a ramas favoritul pietei, datorita suportului oferit de producator, si varietatii tool-urilor dezvoltate pentru acest OS, este preferat si de companii. Iar macOs, sistem de operare oferit de Apple doar pentru PC-urile fabricate de ei, este un sistem de operare fiabil si gratuit, bazat pe Linux.

In aria mobile, au fost multe OS-uri la inceput, aproape ca fiecare producator incerca un OS propriu. OS-uri ca :  [Symbian Ltd](https://en.wikipedia.org/wiki/Symbian_Ltd.), Nokia S40,Palm OS, Meamo OS, au fost printre primele sisteme de operare pentru mobile. BlackBerry OS a castigat mult teren in trecut pe partea de securitate si era considerat clasa business. Dar in trecutl apropiat sau modelat, definitivat ca si concurenti principali pe piata OS-urilor mobile, 2 OS-uri. Android si iOS. Aproape ca este un razboi taboo, ceva ce defineste generatia tanara, aceasta alegere intre Android si iOS. iOS este sistemul de operare bazat pe C dezvoltat si oferit gratis de Apple pentru toate device-urile produse de ei, in aria mobile. Fiind bazat pe C, un limbaj foarte puternic si rapid, OS-ul este unul fiabil, si foarte puternic. Neavand nevoie de prea multe resurse, Apple a intarziat dotarea smartphon-urilor produse de ei cu specificatii echivalente pietei mobile. Dupa cum am specificat, era doar o chestiune de necesitate, OS-ul fiind bazat pe un limbaj care nu necesita resurse multe, nu au adaugat resurse degeaba, ex: iPhone 4s are 512 mb memorie Ram cand majoritatea companiilor din anul in care a fost lasnat iPhone 4s, care au lansat smarphon-uri high-end, au folosit 1 Gb memorie Ram.Ce mai e de remarcat la acest OS este securitatea foarte buna oferita. Odata furat un smartphone dotat cu iOS, daca avea un cont de iCloud(cont de autentificare pe platforma Apple), acel telefon este inutil fara parola de autentificare. Pentru a putea fi folosit, trebuie, ori autentificat, si modificat din setari un nou proprietar, ori schimbata placa de baza complet.

Inca o mentiune, aplicatiile pentru acest OS, sunt gasite in Apple Store, un magazin foarte bine gestionat, iar aplicatiile stocate, sunt mai sigure decat pe alte magazine, datorita testelor prin care trebuie sa treaca fiecare aplicatie inainte sa poate fi urcata in magazin.

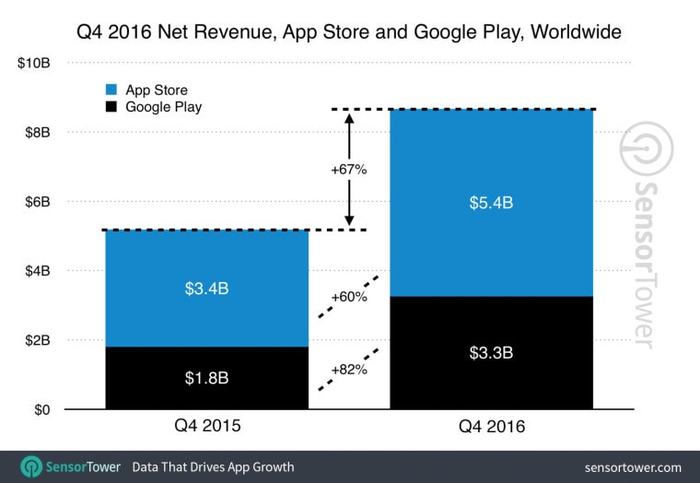
Android.

In 2003, in California, SUA, avea sa se nasca cel mai folosit OS din toate timpurile. Android. Produl de compania Android Inc. in Octombrie,  [Andy Rubin](https://en.wikipedia.org/wiki/Andy_Rubin), [Rich Miner](https://en.wikipedia.org/wiki/Rich_Miner), Nick Sears, and Chris White, lansau prima versiune. Doar 2 ani mai tarziu, in 2005 Google, actulau cel mai mare gigant IT din lume, a cumparat Andorid pentru doar 50 de milioane de dolari. Ajungand sum marca Google sa valoreze peste 2 miliarde de dolari. Bazat pe java, este un OS relativ greoi, care consuma resurse si are nevoie de hardware, cum am mentionat mai sus. Principalul magazin pentru aplicatii android este Google Play, unde poti urca aplicatii relativ usor, fiind necesar un cont de developer, care necesita la randul lui o taxa de 20 de dolari.Lansand versiuni noi de android aproape anual, de mentionat este ca numele versiunilor sunt luate din dulciuri celebre si urmeaza ordinea alfabetica pentru aliteratie. Ex Marshmello, Nughat, Oreo.. etc.

Pentru a vedea o distributie reala a OS-urilor in aria mobile am luat de pe : <https://www.idc.com/promo/smartphone-market-share/os>, urmatoarea statistica :

(<https://www.idc.com/promo/smartphone-market-share/os>)

Se vede clar ca Android-ul acapareaza o mare parte din piata, dar un lucru interesant si demn de montionat este ca desi detine 80-90% din piata mobile, bani din aplicatii mobile majoritatea sunt in App Store, de la Apple, pentru iOS.

(<https://www.upwork.com/hiring/for-clients/android-vs-ios-development/>)

# Concept mobile control

Conceptul de Mobile Control la nivel de idee este foarte puternic: control si verificare de la distanta prin intermediul internetului. Avand 2 componente principale: aplicatie mobile si server c++. Aplicatia este conceputa cu o interfata generic, in sensul in care primeste de la server un json care cuprinde informatii legate de ce functionalitati sunt implementate in ECU-ul care urmeaza sa fie verificat/controlat, si genenreaza o interfata care cuprinde tot ce poate fi afectat la nivel de ECU. Avand interfata generic nu este nevoie de o aplicatie specifica pentru fiecare ECU. Serverul este montat pe o component electronica care poate fi conectata cu masina (protocol OBD), sau direct pe canalul de comunicatie la care sunt conectate ECU-urile masinii(protocol CAN).

* Mufa OBD
* Aplicatie de diagnoza wi-fi

Avand in minte conceptul prezentat mai sus, urmeaza sa intram in detalii legate de implementarea conceptului, tool-uri folosite si echipament.



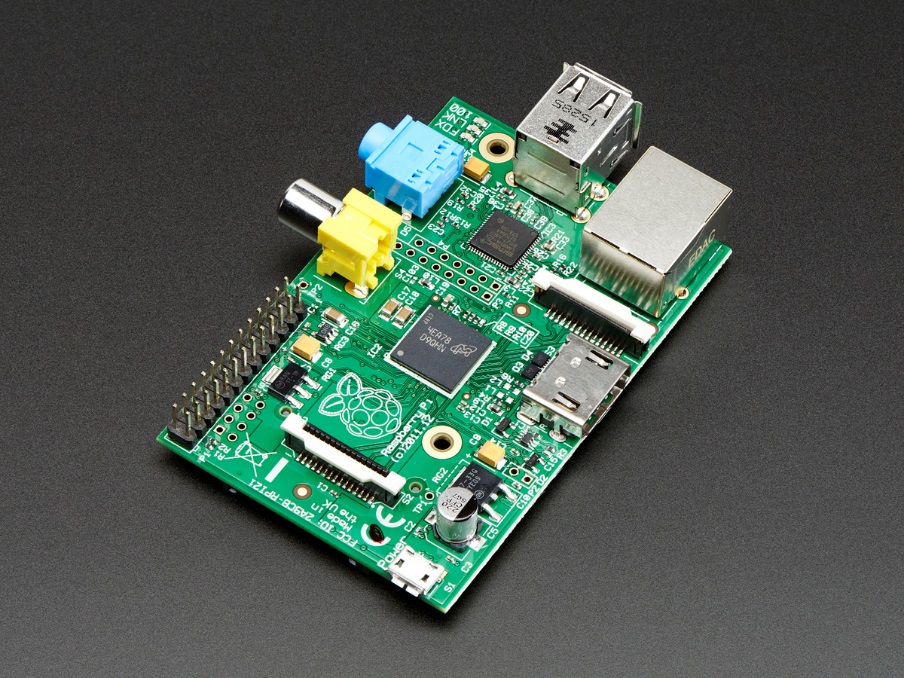
(https://www.ebay.co.uk/itm/Samsung-Galaxy-Tab-3-7-0-SM-T211-2G-3G-Tablet-Phone-Android-8GB-White-Unlocked-/162751555044)

Am folosit pentru aplicatia mobile o tablet de la Samsung (Samsung Tab3), capabila de conexiune wi-fi, si cu o versiune de android ….. Am folosit pentru dezvoltarea aplicatiei ide-ul Andorid Studio….Eclipse… pentru stocarea serverului de C++ am folosit un Raspberry py 1, iar pentru testare efectiva un ECU care a fost conceput pentru controlul unui far.

Seria Raspberry Pi reprezinta o serie de “small single-board computers” dezvoltate in Regatul Unit de catre Raspberry Pi Fundation, pentru a promova predarea informaticii de baza in scoli si in tarile in curs de dezvoltare. Modelul principal a devenit mult mai popular decat se anticipase, cu vanzari massive in afara pietei tinta, cum s-a intamplat in cazul roboticii. Nu include periferice (tastaturi, carcase ..etc).Potrivit Fundatiei Raspberry Pi, peste 5 milioane de placute Raspberry Pi au fost vandute pana in Februarie 2015, devenind cel mai bine vandut calculator britanic. Pana in Noiembrie 2016 au vandut 11 milioane de unitati, devenind al trei-lea best-selling "general purpose computer".In Martie 2018 vanzarile au atins 19 milioane.

(<https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/>)

(<https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/>)



(https://www.adafruit.com/product/998)

Modelul folosit in dezvoltarea conceptului Mobile Control face parte din seria 1. Modelul Raspberry Pi model B.Specificatile acestui model fiind : 512 MB RAM, 2porturi USB2, 40 de pini GPIO(Genereal Purpose Input/Output), si port de Eternet, slot pentru SD, si port HDMI.



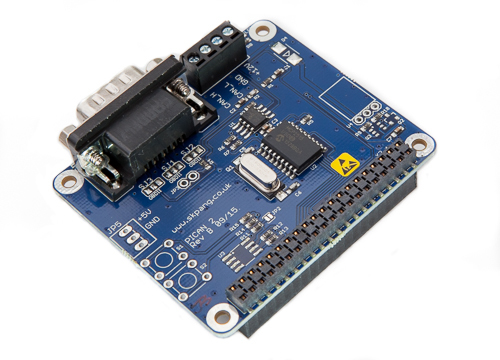
(<https://www.amazon.com/D-Link-DWL-G122-Compact-Wireless-Adapter/dp/B0002DQUHC>)



(<https://www.tp-link.com/us/products/details/cat-5520_TL-WN725N.html>)

Pentru comunicarea cu tableta am montat pe Raspberry Pi un modul extern de wi-fi. Am configurat OS-ul (Rasberian, versiune de linux speciala pentru Raspberry) ca la pornire sa porneasca un hotspot wi-fi la care te poti conecta cu orice dispozitit capail wi-fi, in cazul nostru tableta. Pentru aceasta am urmat pasii descrisi in mini-tutorialul : <https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md>. Comenzi clasice de linux.

Modul PiCan (Modul CAN pentru Raspberry Pi)



(http://copperhilltech.com/pican-2-can-interface-for-raspberry-pi-2-3/)

Pentru cominucarea cu Bus-ul de CAN, am folosit un modul de can PiCAN – CAN Interface for Raspberry Pi. PiCAN ofera capabilitati Controller Area Network(CAN) pentru Raspberry Pi. Foloseste microcipul MCP2515 CAN controller cu MCP2551 CAN transceiver. Conexiunea este facuta via DB9 sau 3-way screw terminal. Este usor de montat pe Rasberry si programarea lui poate fi facuta in Python sau C. Features : CAN v2.0 A/B la 1 MB/s, 120 ohm resistor terminal + multe altele.( <http://copperhilltech.com/pican-2-can-interface-for-raspberry-pi-2-3/>)

Pentru a usura procesul de dezvoltare a conceptului, am folosit o metoda de compilare mai complexa. Cross-compile intre platformele Windows si Linux. Desi a durat foarte mult sincronizarea bibliotecilor de C, compilarea cross-compile a fost un success. Am instalat pe statia doatata cu Windows un modul extern de wi-fi. Cum am notat mai sus, Raspberry Pi-ul cand porneste este configurat sa genereze hotspot. Statia dotata cu Windows capabila acum la conexiune Wi-fi, s-a conectat cu Raspberry Pi, am folosit tool-ul Cygwin pentru sincronizarea bibliotecilor, iar pe scurt, din Eclipse(Windows) puteam sa fac direct debug pe Raspberry Pi, unde a fost instalat un simplu server C, dotat cu intervata : canInterface.

# Implementare concept

Pana acum am prezentat concepul din exterior, componente si tehnologii.

Mobile Control Implementare:

Avand in minte istoricul si detaliile prezentate mai sus urmeaza sa intram in implementarea efectiva a conceptului.

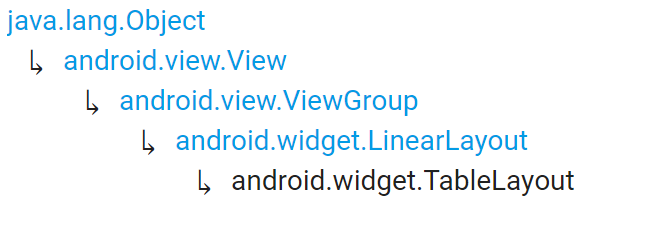
Aplicatiile android sunt compuse din 2 componente prnicipale : user interface si control.

User interface pentru o aplicatie android este imptartit la randul lui in componenete denumite activitati. Grafica putand fi ajustata direct in tool-ul folosit, in cazul nostru Android Studio, unde componentele integrate in interfata pot fi adaugate si miscate in activitate folosind un simplu mouce. Dar pentru lucru de finete, Andoid Studio ofera fisiere .xml, asemanatoare celor de CSS din programamrea web. Aceste fisiere .xml sunt generate din motorul care gestioneaza crearea obiectelor din interfata despre care am vorbit mai sus, dar sunt editabile si prin modificarea valorilor generate sau adaugarea de tag-uri si valori, poti gestiona interfata exact cum doresti.Pentru a intelege descrierea, vom explica o bucata de cod xml dintr-un fisier de tip .xml care se ocupa de gestionarea interfetei legate activitatii de “static\_menu”, activitate la care vom revenii in descriere ulterior.



Vom explica exemple de tag-uri, parametri si valori folosite in interfata “static\_menu.xml”.

Inainte sa incepem putem mentiona un concept de java, fiecare obiect este mostenit la randul lui din alt obiect recursiv, iar la radacina se afla clasa Object. Pentru primul tag pe care il vom explica ierarhia arata in felul urmator:



(<https://developer.android.com/reference/android/widget/TableLayout>)

Tagu-uri:

* TableLayout – layout care aranjeaza “copii”(cildrens) - descendentii in randuri si coloane.
* TableRow – layout care aseaza descendentii la nivel orizontal.
* TextView – element din interfata care afiseaza text static, pentru text editabil se va folosii EditText.
* Button – element din interfata pe care utilizatorul il poate apasa (“click or tap”) pentru a genera o actiune.
* ImageView – afiseaza imagini stocate in directorul de resurse al aplicatii

Parametri:

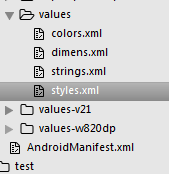
* layout\_width – clasa care exprima “inaltimea” tag-ului in care este setat.
* layout\_height – clasa care explima “latimea” tag-uli in care este setat.
* Gravity – folosit pentru plasarea unui obiect intr-un conteiner posibil mai mare, containerul fiind ajustabil, obiectul care are “gravitatea” setata va lua dinamic o pozitite in functie de valoarea specificata acestei clase.
* stretchColumns – index pentru numarul de coloane.
* srcCompat – metoda de a integra imagini, sau resurse din categoria “drawables” in interfata.
* Text – folosit pentur specificarea textului stocat de obiect.

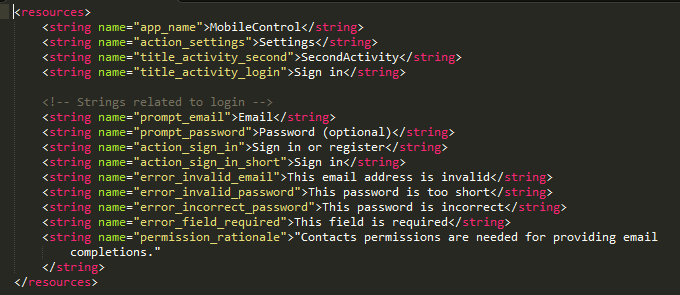
Valori:

* Pentru “layout\_width” si “layout\_height” sau orice alt layout mostenit din ViewGroup.LayoutParams sunt 2 metode principale prin care putem specifica pozitia, una este mai relativa : “fill\_parent” si valori asemanatoare care specifica valori relative tag-ului parinte, iar acesta specifica ca obiectul va umple spatiul liber din tag-ul parinte. Iar o prezentare mai exacta a pozitiei sau a dimensiunilor se descrie in dp.
* Pentru “gravity”, valorile sunt valori care exprima pozititonare relativa in obiectul parinte: “left”, “right”, “top”, “botom”
* Pentru “srcCompat” se va folosi numele resursei de tip “drowable” din directorul de resurse, in exemplul nostru primeste valoarea : “conti\_logo”.

In dezvoltarea aplicatiei am acumulat 9 fisiere de tip .xml pentru gestioanrea interfetei aplciatiei Mobile Control, 7 pentru activitatile aplicatiei si 2 pentru test si pentru implementarea unui meniu.

Resurse de tip values:





In aceste fisiere de tip .xml avem stocate resurse cu valori predefinite. Putem definii aici tot felul de resurse, fiecare resursa are un nume iar functional putem afla valoarea resursei/ folosii valoarea resursei, doar prin numele dat. Mai sus sunt prezentate valorile predefinite in .xml-ul “strings.xml”, unde pentru a explica functionalitatea prezentam prima resursa, numita :”app\_name” caruia i-am atribuit valoarea “MobileControl”, acum putem folosi “app\_name” ca si resursa oridecateori avem nevoie de numele aplicatiei care este :”MobileControl”.

Pentru a face tranzitia de la interfata la partea de control al aplicatiei, vom definii conceptul de meniu, si vom explica implementarea lui in aplicatia Mobile Control.

### 1.Meniu andoid

Pentru meniu, a fost create doua fisiere de tip .xml separata care se ocupa de interfata meniurilor. Un fisier “static\_menu”, altul “menu\_main”. Iar imprementarea functionalitatilor meniului “static\_menu.xml” a fost facuta in fisier-ul de tip java care este asignat fiecarui activitati care utilizeaza meniul descris de “static\_menu.xml”. Iar fiserul java care gestioneaza activitatea principala, contine implementarea functionalitatilor meniului descris de “menu\_main.xml”.



Pentru meniul definit in “menu\_main.xml” au fost facute urmatoarele setari :



Meniul este unul generic conceput mai mult pentru a prezenta acest concept tipic aplicatiilor mobile. Contine 3 mari obiecte, unul dintre ele, primul, fiind la randul lui un sub-meniu care contine la randul lui 3 obiecte.

Primul obiect prezent in meniu este un sub – meniu care a fost gandit ca si concept (neimplementat) pentru setarile aplicatiei. Si prezinta doar scheletul pe care se paote construi functionalitati de tipul : selectare font, selectare marime, selectare culori …etc.

Al doilea obiect este o imagine, preluata dupa cum am descris mai sus din directorul cu obiecte de tip “drowable”, este un obiect de tip PNG, si prezinta sigra companiei unde am dezvoltat conceptul Mobile Control.

Al treilea si al patrulea obiect din lista prezinta 2 butoane simple, text, care sunt gandite pentru a directiona utilizatorul care apasa pe buton spre activitatea notata in textul afisat.

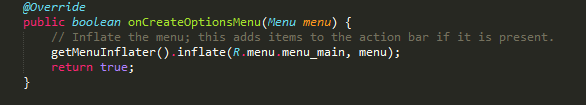
Sau folosit taguri de tip “item” si de tip “menu”. Cum este intuitiv un tag de tip “menu” este gandit sa contina unul sau mai multe tag-uri de tip “item”.

Parametri notabili din acest fiser ar fi:

* android:id, folosit pentru a identifica unic obiectul .
* icon, care specifica numele obiectului de tip”drowable”.
* showAsAction, ca specifica in context daca obiectul sa fie sau nu vizibil tot timpul.
* title, care reprezinta textul afisat de interfata, vizibil pentru utilizator.

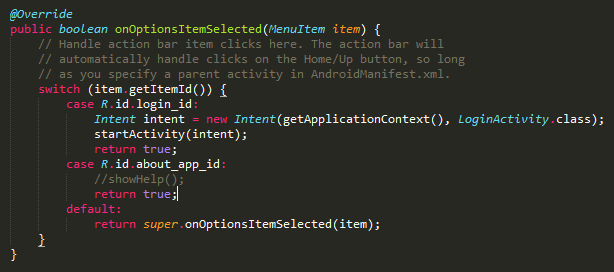
Dupa cum am spus, controlul pentru meniu este implementat in fiecare activitate care afiseaza acel meniu, pentru examplificare vom folosii in contrinuare meniul definit in “menu\_main.xml”, definit in ce tine de functionalitate in fisierul “MainActivity.java”, fisier care intuitiv detine implementarea functionalitatilor definite pentru activitatea principala (activitatea care se deschide la deschiderea aplicatiei in OS-ul Android).

In continuare vom prezenta cele mai improtante 2 functii implementate pentru functionalitatile meniului:



MenuInflater este clasa care este folosita pentru intantierea meniurilor socate in fisiere de tip .xml(in cazul nostru “menu\_main”), in obiecte de tip Menu(in cazul nostru obiectul a fost denumit generic ”menu”, si se poate vedea in antetul functiei “onCreateOptionMenu(Menu menu)”).

Iar functionalitatea principala a meniului este definita in functia :



Pe scurt codul spune ca la selectia item-ului cu id-ul “login\_id” se va folosi un obiect de tip Intent care va afisa user-ului activitatea LoginActivity, “pornind” activitatea prin intermediul functiei “startActivity(inteint)”.

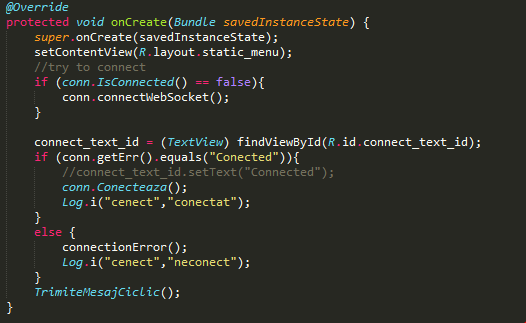
In cazul in care userul apasa pe item-ul care are id-ul “about\_app\_id”, nu se va intampla nimic.

## 2.Controlul.

Inainte de a intra in detaliile legate de orice functionalitate specifica Mobile Control vom explica cateva din functiile specifice activitatilor android:

* void onCreate()
* void onDestroy()
* void onPause()
* void onStop()

Functionalitati specifice fiecarei activitati. In care putem sa definim actiuni in contextul activitatii. Cand e creata, cand e distrusa, cand e in pauza…etc. Pentru demonstratie vom folosii codul functiei onCreate() din fisierul StaticMenuActivity.java, in care la crearea activitatii se incearca o conectare la server, printr-un websocket (tehnologie la care vom revenii in continuarea acestei lucrari).



Controlul activitatilor si implementarea functionalitatilor aplciatiei Mobile Control au dezvoltate in fisiere java, deobicei cate un java file pentru fiecare activitate. In plus au fost create alte 5 fisiere .java care stocheaza implementarea functionalitatilor de manipulare a datelor (json & CAN message) si de conexiune cu serverul(websocket).

Dupa cum am specificat mai sus, aplicatia Mobile Control a fost dezvoltata pentru prezentarea unui concept de wi-fi diagnosis.

Aplicatia este dezvoltata cu 2 “meniuri”, nu meniuri cum am explicat mai sus, meniuri native, ci meniuri care stocheaza functionalitati, sau schelet de functionalitate pentru concept. Unul dintre meniuri este static, ceea ce inseamna ca fiecare imagine si fiecare buton, si tot ce tine de interfata grafica a fost adaugat din engine-ul care ajuta developerul la creat interfata grafica, si functionalitatile trebuie implementate separat in fisierul .java asignat activitatii in care au fost adaugate obiectele din interfata.

Dar meniul cu numarul 2, meniul dinamic prezinta conceptul de meniu generic. Avand in fisierul de configurare a interfetei, fisier de tip .xml doar 2 tag-uri-obiect un “ScrollView” si un “TableLayout”, primeste in cod informatii despre ce obiecte vrea serverul(in concept, in implementare json-ul de configurare a fost stocat pe local in aplicatia android) sa afiseze aplicatia.

In continuare vom prezenta caracteristicile de implementare ale celor 2 “meniuri” de functionalitati.

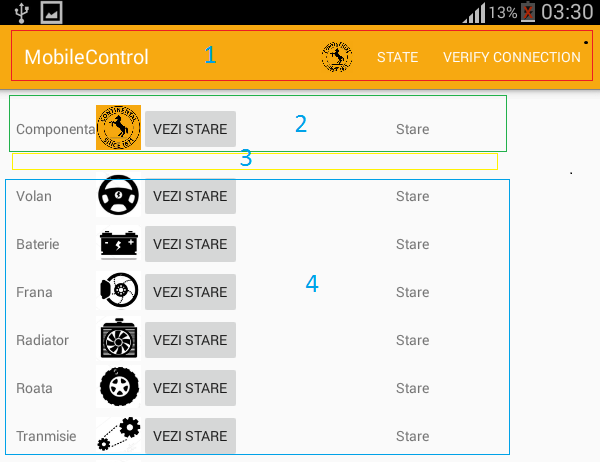
## 3.Interfata statica

* are in fisierul de configurare a interfetei 137 de linii de cod xml.
  + 1 obiect de tip “ScrollView”
  + 1 obiect de tip “TableLayout”
  + 9 obiecte de tip “TableRow” fiecare avand cate
    - 2 obiecte de tip “TextView”
    - 1 obiect de tip “ImageView”
    - 1 obiect de tip “Button”
* Deci, estimativ 47 de obiecte, hardcodate in interfata.



## 4.Interfata dinamica.

* Are in fisierul de configurare doar 35 de linii de cod xml.
  + 1 obiect de tip “ScrollView”
  + 1 obiect de tip “TableLayout”
* Deci, 2 obiecte.



Putem observa ca diferentele nu sunt mari, si conceptul de interfata generica, generata din cod functioneaza. Ideea care face puternic acest concept este aceea de a avea un dispozititv care se paote conceta la componentele electronice ale masinii (mufa OBD poza OBD) cauta sa sa vada ce functionalitati poate accesa, ce informatii poate cere componentelor, aduna toata aceasta informatie si o trimite la client(aplicatie Mobile Control), iar aplicatia genereaza interfata in functie de ce resurse ofera, informatii poate accesa. Dupa cum am mentinat mai sus, nu am trimis json-ul cu interfata ce se vrea generata de la server, ci l-am stocat intr-un fisier de tip .java intre fiserele care gestioneaza functionalitatea aplicatiei.

Pentru interfata generata dinamic, pe langa functiile primare din fiecare activitate, definite default pentru fiecare activitate, au fost definite functionalitati care iau informatii dintr-o structura si incearca definirea interfetei.

Paranteza din prezentarea conceptului MobileControl pentru a prezenta concepte de android, pentru intelegerea functionalitatilor implementate.

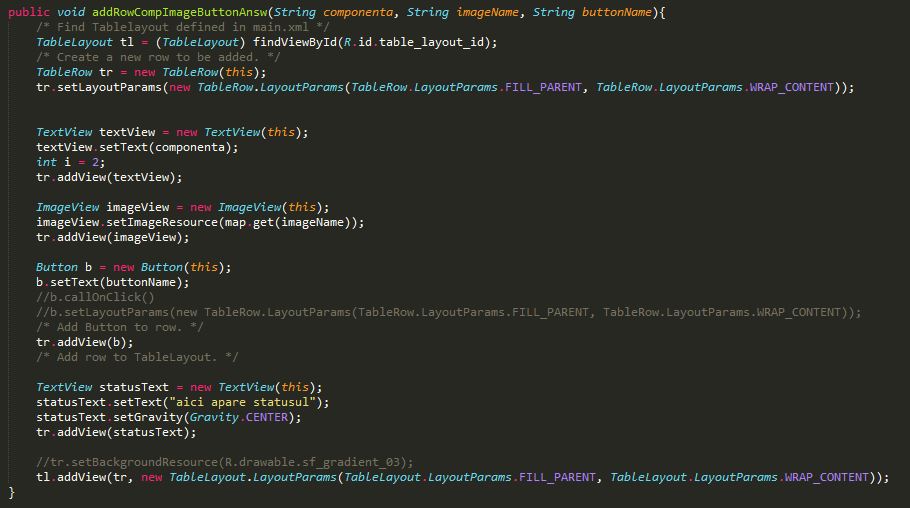
### Activitati android.

Pentru a intelege cum functioneaza o Activitate Android am copiat o imagine de pe site-ul : “https://developer.android.com/reference/android/app/Activity [“ unde este explicat tot ciclul activitatii.](mailto:)

[Functionalitatile sunt deductibile, iar implementarile lor reprezinta comportamentul aplicatiei (mai ales al activitatii) in timpul si functionalitatea respectiva. Pentru intelegere prezentam doar functia onPause().In aceasta functie cel care dezvolta aplicatia poate scrie tot ce vrea ca aplicatia (in mod special aceasta activitate pentru care este definita functia) sa faca in timpul in care activitatea prezenta este in starea de pauza. Functia vine predefinita de Android, dar cu un simplu “@Override](mailto:)” dezvoltatorul aplicatiei isi poate pune amprenta lui in comportamentul activitatii, si poate suprascrie comportamentul normal, sau dezvolta cel existent.



Functionalitati folosite in generarea interfetei dinamice:



Functia : “addRowCompImageButtonAnsw()” adauga un rand in lista configurata pentru interfata.

Pentru inceput se cauta contextul pentru care trebuie adaugat un rand, iar pentru aceasta este creat local un obiect de tipul “TableLayout”, “tl”, unde se cauta dupa id, obiectul din interfata (.xml), in cazul nostru id-ul = “table\_layout\_id”.

* Se instantiaza un obiect de tipul “TableRow”, “tr”, pentru care setam parametri de layout, FILL\_PARENT si WRAP\_CONTENT, parametri care spun ca acest nou rand va lua dimensiunea obiectului parinte, ca latime, iar ca lungime va incorpora continutul.
* Urmeaza crearea unui obiect de tip “TextView”, obiect care va afisa numele componentei integrate in masina.
* Se adauga acest TextView in obiectul de tip TableRow.
* Se creaza un obiect imagine, “ImageView”, care primeste o imagine si este adaugat la randul lui in TableRow-ul “tr”.
* Se adauga in acelasi mod un buton. Ce e de mentionat e ca nu au functionalitate atasata, este comentat codul care poate atasa o functionalitate fiecarui buton, “b.callOnClick()”. Am lasat comentat codul pentru a observa posibilitatea adaugarii de functionalitate interfetei generate.
* Se mai adauga un “TextView”
* Dupa care se adauga acest “tr” in “tl”-ul parinte, cu parametri ce seteaza layout-ul descris initial.

Mai sunt definite 2 functii:

* addStaticFirstRow() – functie care adauga un rand mai special, este randul care contine interfata:



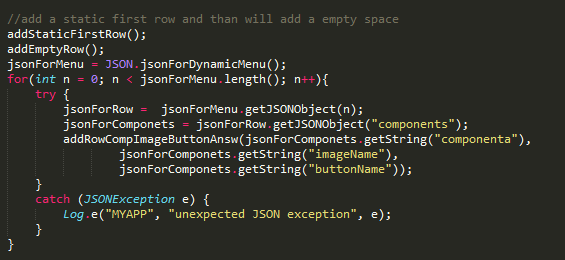
* addEmptyRow()- functie care adauga un rand gol, avand aceleasi specificatii ca celelalte, dar cum continutul e gol, dimensiunea lui relativa la continut “WRAP\_CONTENT”, e mai mica



Nu vom mai descrie codul pentru aceste functionalitati pentru ca este foarte asemnataor functiei “addRowCompImageButtonAnsw()”.

### 5.Json

Pentru a intelege complet aceste functii generice, vom analiza textul care adauga concret informatia stocata in JSON in interfata dinamica:



(Aici este parsata structura care contine informatia necesara generarii interfetei.)

* Pentru inceput se adauga primul rand care este mai special, si definit in functie separata.
* Se adauga un rand gol (estetic – separata randul care arata modelul fiecarui rand urmator de randurile efective)
* Pentru toata lungimea structurii se parseaza JSON-ul si se apeleaza functia : “addRowCompImageButtonAnsw()” cu valorile preluate din JSON.

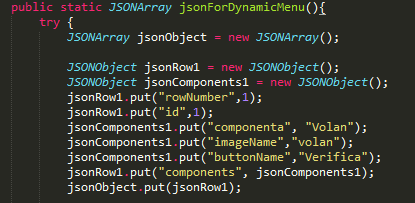
In continuare vom trata aceasta parte de Json.

Json tehnologie de stocat informatie ierarhizat, dezvoltat si folosit mai mult in dezvoltarea aplicatiilor web. Este deobicei alegerea celor mai multi dezvoltatori in ceea ce se vrea comunicatie intre server si client, castigand mult terent in favoarea .xml-ului, tehnologie folosita initial.

In aplicatia Mobile Ocntrol tehnologia Json a fost folosita pentru a stoca doua structuri interne, una folosita pentru a stoca informatii legate de mesajul de CAN care urmeaza sa fie trimis la server, pentru a fi trimis pe bus-ul de CAN, si una folosita pentru a stoca informatia necesara generarii interfetei generice. Pentur ca pana acum am explicat conceptul de interfata dinamica, vom explica pentru inceput Json-ul in care stocam informatiile necesare interfetei.

Json pentru interfata dinamica.

Pentru a simplifica intelegerea conceptelor din spatele structurii de tip JSON, vom gandi toata structura stocata in obiectul de tip JSON ca o structura bine definita cu sub-structuri care stocheaza informatia, usor de creat, usor de folosit, logic si concis, care in final dupa ce a fost creat logic si concret, este trimis ca un simplu String, sir de caractere, care de capatul celalalt al comunicarii poate fi foarte usor transpus iara intr-un obiect de tip JSON si folosit ca structura logica de date.



Dupa cum putem vedea in imaginea de mai sus a fost creat un obiect principal, parinte, “jsonObiect”, in care vom stoca toata informatia. Este creat de tipul “JSONArray”, clasa conceputa pentru a stoca obiecte asemanatoare de tip JSON. Au mai fost create doua obiecte de tip “JSONObiect”, “jsonRow1” si “jsonComponents1”.

Ierarhia structurii este simpla:

JSONArray - jsonObject:

JSONObject – jsonRow1:

text: “rowNumber”

text: “id”

JSONObject – jsonComponents1:

text: “componenta”

text: “imageName”

text: “buttonName”

JSONObject – jsonRow2:

text: “rowNumber”

text: “id”

JSONObject – jsonComponents2:

text: “componenta”

text: “imageName”

text: “buttonName”

…..

Structura contine sase substructuri care descriu fiecare cate un rand din interfata ce urmeaza sa fie generata. Interfata generata dupa cum am explicat anterior.

Despre Json-ul in care stocam detaliile legate de CAN message vom explica cand vom aborda capitolul legat de mesajele de CAN.

## 6.Websocket.

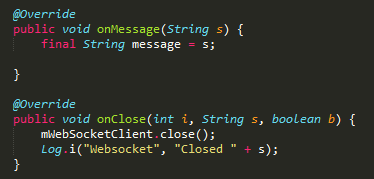
Pentru conexiunea dintre tableta si raspberry py am folosit o tehnologie larg utilizata : websocket.

Codul de baza l-am luat de pe pagina <https://www.varvet.com/blog/using-websockets-in-native-ios-and-android-apps/> unde prezinta secventele de cod minimale pentru partea de server si pentru partea de client. In aplicatia Mobile Control am folosit evindet partea de client, iar pe serverul montat pe Raspberry am folosit secventa pentru server.

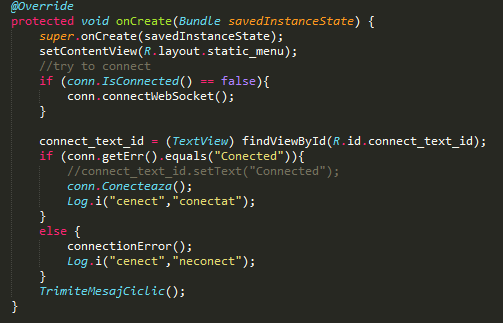
Nu mai explic codul pentru ca sunt pur functii de conectare server – client, am folosit portul 8080 si un IP static, si clasa “WebSocketClient” din java, clasa care are functiile :

* void onOpen()
* void onMessage()
* void onClose()
* void onError()

Pentru exemplificare codul pentru onMessage() si onClose() folosit este :

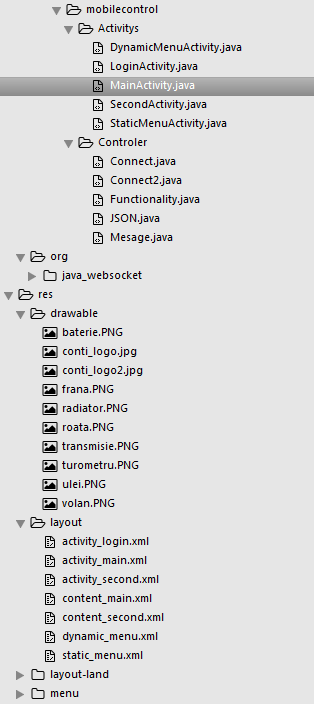


Iar o secventa de cod care apeleaza conectarea la server este prezentat un urmeatoare imagine. Unde se verifica daca nu exista deja o conexiune la server, si dac anu este se incearca o conexiune. Iar in cazul prezentat se finalizeaza cu trimitere de mesaj ciclic, la care o sa revenim cu prezentare cand ajungem la capitolul legat de mesajele de CAN.



onCreate() din StaticMenuActivity.java

Per general o parte din ierarhia directoarelor si fisierelor din cadrul workspecu-lui dedicat aplicatiei mobile arata in felul urmator.



# Mesaje CAN

## 1.Creare mesaj CAN

Una dintre componentele majore ale conceptului Mobile Control este trimiterea mesajelor de tip CAN de pe aplicatie mobile pe Bus-ul de CAN. Aceasta componenta a fost implementata pe 2 platforme, pe Android si pe Raspbian. La nivel de aplicatie a fost creata o structura de tipul mesajului CAN.( Pentru informare amintim ca mesajul de can are 8 bytes dedicati informatiei transmise, un Id – identificator unic si un DLC – dimensiunea datelor ce urmeaza a fi transmise.) Dupa care a fost implementata o functie care transforma structura locala intr-un Json care are sa fie folosit ca structura pentru comunicarea cu serverul.Urmat fiind momentul in care serverul care asculta, primeste comanda “send”, atunci se despacheteaza informatia stocata in Jsonul trimis de client(aplicatia android) si se adauga intr-un obiect de tip “CanInterface::Message” si trimis pe Bus-ul de CAN ca si mesaj, cu Id-ul si DLC-urile setate.

Pentru inceput putem privi structura in care stocam mesajele de CAN:



Putem observa un obiect de tip Message (clasa definita in cadrul aplciatiei, dar asupra detaliilor de implementare a clasei nu vom intra deoarece sunt simple concepte de POO, definire containere, definire functii de “set” si “get” pentru informatia stocata in fiecare container.

Prezentam detalii specifice acestui mesaj inainte sa continuam cu explicarea implementarii :

* DLC – dimensiunea mesajului care urmeaza sa fie transmis e prezentata ca fiind 8(bytes)
* Id – e setat ca fiind 193 (in hexazecimal).
* Urmatorii bytes sunt 0 inafara de Byte2 care este setat pe 5(dupa cum se vede in comentariu, deoarece aveam acces la informatii interne despre ECU pe care testam stiam ca in implementarea codului stocat pe ECU care controla un far. Si am stiu ca pentru mesajul cu Id-ul 193 daca setam Byte-ul 2 cu valoare 5 aplicatia montata pe ECU trebuie sa permita sistemului de lumini aprinderea blinker-ului “semnalizare”)
* Ca si informatie, deobicei o parte din acesti bytes transmisi pe Bus-ul de CAN, in mesajele normale ai si functie de CRC – checsum, folosita pentru verificarea integritatii datelor.

In continuare putem observa mutarea intr-un Json a informatiei stocate in obiectul “mes1”, obiect

de tipul clasei “Message”

## 2.Mesaj CAN in Json

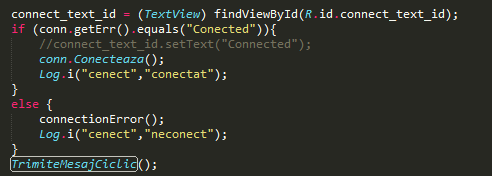


In aceasta functie “toJson” se incearca crearea unui string de tip Json in care sa fie stocata informatia stocata momentan intr obiectul “mesage” obiect din clasa “Message”:

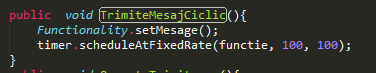
* Pentru inceput se creaza un obiect de tip Json denumit generic “json”, se seteaza “targhet” cu valoarea “main\_handler” (informatie care ajuta serverul sa directioneze mesajul catre “main\_handler”)
* Urmatoarea setare este serarea comenzii (“cmd”) ca fiind “send” (informatie care ajuta “main\_handler”-ul sa stie pentru ce operatie este informatia care transmisa de client), intr-un nou obiect Json, “jsonMes”.
* Urmeaza crearea unui nou obiect de tip Json, “jsonData”, si setarea valorilor pentru “Id”, “DLC” si urmatorii 8 Bytes, valori preluate direct din obiectul mesage.
* Acum in obiectul “jsonMes” la tagul “data” se seteaza obiectul de tip Json, “jsonData”.
* Urmatorul pas este adaugarea in obiectul Json parinte, la tag-ul “message”, obiectul de tip Json, “jsonMes”.
* La final se returneaza un string in care este cuprins toata informatia descrisa mai sus.

3.Transmitere mesaj CAN – trimitere ciclica mesaj can explica

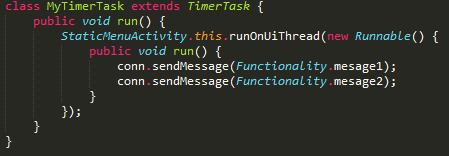
Intr-o activitate, in cazul nostru, “StaticMenuActivity”, in cadrul functiei “onCreate” am testat transmiterea informatiei (mesajului de CAN) spre server, folosind tehnologii de tip websocket si Json, dupa conectarea cu serverul.



Pentru a analiza functia “TrimiteMesajCiclic()” ne vom folosii de urmatoarea imagine:



* Unde in Clasa java numita Functionality stocam functionalitatea “setMesage” explicata mai sus.
* Timer – unde se seteaza ca la cu delay(al doilea parametru din antet) de 100 de milisecunde, si ciclic la 100 de milisecunde (ciclicitate data de al doilea parametru) sa se execute task-ul – “functie” definit mai jos de clasa “TimerTask”:



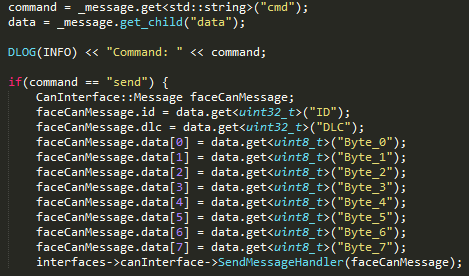
Ca un rezumat: codul de mai sus, trimite ciclic 2 mesaje (de tip CAN stocate in string-uri care contin obiecte Json) spre server.

Receptionarea:

Receptionarea la nivelul serverului se face in functia “main\_handler” dupa cum am specificat la crearea Json-ului. Unde se va face desfacerea Json-ului in informatia ce ne intereseaza sa o transmitem pe Bus-ul de CAN. La nivelul serverului este prezenta o interfata numita “can\_interface.c” care pe langa multele functii implementate implementeaza si functia SendMessageHandler”, functie la care o sa revenim in viitor. Pentru inceput sa explicam cum se desface informatia primita de la client (aplicatia MobileControl).

## 4.Receptare mesaj CAN (json)

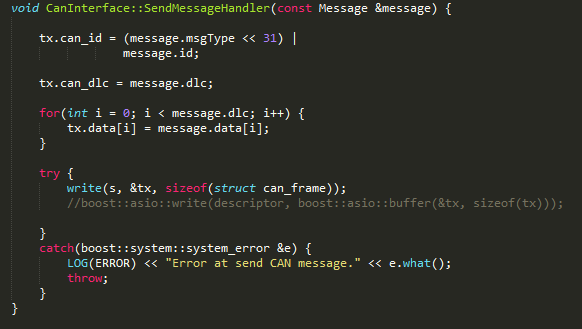
Dupa cum am explicat pana acum in contextul in care am povestit despre Json, la server a ajuns un string, in care am stocat toata informatia trimisa de client. La nivelul serverului desfacem exact dupa structura dupa care am alcatuit string-ul si obtinem informatia trimisa, ce urmeaza sa fie trimisa iara, dar de data asta direct pe Bus-ul de CAN.



(Implementarea preluarii informatiei de la client si trimiterii pe BUS)

* In \_message este tot Json-ul contruit de noi la nivel de client stocat la nivel de client in “jsonMes”(pentru detalii putem sa ne intoarcem la capitolul unde am descris acest lucru)
* In string-ul “command” este stocata informatia setat de noi pe client in tag-ul “cmd” (care era “send”).
* In “data” stocam informatia stocata in Json la tag-ul “data” unde la nivel de client se numea “jsonData”.
* If(command == “send”) ne intreaba daca comanda e “send”, cea ce lasa posibilitatea exitentei multor altor comenzi.
* Si daca comanda este “send”(cum am setat-o la nivel de client) atunci se creaza un obiect ce mosteneste interfata CanInterface, in care adaugam toata informatia stocata in “data” si folosind functia “SendMessageHandler” se trimite direct pe BUS.

## 5.Implementarea functiei “SendMessageHandler”:



(“SendMessageHandler” implementare)

CanInterface este interfata care gestioneaza functionalitatile implemetnate pentru modulul de CAN montat pe Raspberry Py, despre care am discutat in capitolul legat de componentele fizice ale conceptului.

In frame-ul “tx” se seteaza pe rand , id-ul, dlc-ul si informatia ce trebuie stocata in cei 8 bytes, dupa care se incearca un “write” operatie ce incearca scrierea directa pe Bus-ul de CAN a mesajul tx(“tx” petnru ca e transmis, daca era receptionat era “rx”)

Observatie :

Pentru partea de server nu am explicat toate detaliile deoarece codul este preluat din surse de tip opensurce. Dar ca rezumat este un server cu capacitati de conectare cu clietul in tehnologie websocket, ce are in implemmentare si CanInterface, interfata folosita dupa cum am spus pentru gestionarea functionalitatilor oferite de modulul de CAN de pe Raspbery Py, iar ce era esential pentru intelegerea conceptului si demonstrarea functionalitatii a fost descris mai sus.

Deci, inafara de anumite functionalitati si modificari asupra codului existent, codul de pe partea de server nu este scris de mine, ci provine din usrse de tip OpenSurce sau documentatii publice.