UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Ambient Intelligence for Task Integration on Android Devices

Coordonator ştiinţific:
Ş.l. Dr. Andrei Olaru

Absolvent:
Vlad Costin Herescu

BUCUREŞTI

2014

Cuprins

1. Abstract

Aplicaţia Task Scheduler are rolul de asista într-un mod inteligent utilizatorul în programarea propriilor taskuri, oferindu-i informaţii pe baza unor elementele de context precum locaţie, ora de execuţie a taskului, persoanele şi dispozitivele necesare.

Astfel, în afara posibilităţii de a specifica taskuri ce reprezintă programul fix al utilizatorului şi taskuri care nu au un interval fix, repetitive pe baza informaţiilor obţinute din taskurile executate, aplicatia oferă functionalităţi suplimentare precum:

- Indicarea locaţiei , a priorităţii, a dispozitelor necesare în momentul adăugării unui nou task pentru execuţie
- Specificarea taskurilor care pot fi executate în momentul curent, filtrarea fiind facută pe baza informaţiilor introduse de către utilizator şi pe baza contextului actual în care se află utilizatorul :
    - cât ar dura execuţia taskului şi dacă aceasta este în concordanţă cu programul fix al utilizatorului,
    - dacă în preajma utilizatorului se află persoanele necesare pentru execuţie,
    - dacă utilizatorul se află în apropierea locaţiei pentru execuţia taskului
- Calcularea orarului taskurilor, luând în considerare prioritatea acestora, locaţia şi ora de execuţie pentru care durata necesară execuţiei lor ar fi minimă.

2. Platforma şi tehonlogiile utilizate

Aplicaţia a fost dezvoltată pentru sistemul de operare Android si este compatibila cu versiunea 3.2. A fost scrisă în limbajul de programare orientat pe obiecte Java iar pentru dezvoltare a fost folosit mediul de dezvoltare Eclipse care dispune de ADT Plugin pentru comunicarea cu sistemul de operare al dispozitivului. Bibliotecile Android au fost accesate prin intermediul lui Android SDK.

Dezvoltarea aplicaţiei s-a bazat în mare parte pe serviciile Google, pentru care a fost necesară adaugarea bibliotecii google-play-service.

2.1 Platforma utilizată

Aplicaţia a fost realizată pe sistemul de operare Android. Acesta pune la dispoziţie tehnologiile necesare pentru implementarea funcţionalităţilor : retele wifi, GPS, Bluetooth.

Astfel, prin:

- Reţelele Wifi si GPS pot determina aproximativ coordonatele locaţiei utilizatorului
- Bluetooth este utilizată pentru a determina dacă utilizatorul dispune de dispozitivele necesare pentru executia aplicaţiei. De asemenea, fiecare dispozitiv având o adresa mac unică, o persoana ce detine un dispozitiv poate fi asociată cu adresa mac a acestuia. Prin urmare, depistarea unui dispozitiv cu o anumită adresa mac poate fi asociată cu depistarea unei persoane în apropierea utilizatorului şi astfel se poate determina dacă în preajma utilizatorului se află persoanele necesare pentru execuţia taskului.

2.2 Tehnologiile utilizate

Pentru stocarea şi gestiunea datelor, sistemul de operare Android dispune de 2 metode:

- folosirea tehnologiei **Shared Preferences** pentru stocarea datelor sub forma : cheie – valoare asociată. Aceasta se pretează pentru date care nu trebuie să fie structurate, accesarea lor fiind foarte facilă prin această tehnologie spre deosebire de tehnologia **Sqlite**. În cazul aplicaţiei mele, aceasta a fost utilizată pentru a stoca texte introduse de utilizator (de exemplu adrese, titluri). Ulterior, textele sunt extrase din shared preferences pentru câmpurile care folosesc proprietatea autocomplete.
- folosirea tehnologiei **Sqlite** pentru stocarea datelor complexe, care necesită structurare, normalizare. Aceasta optiune a fost preferată pentru stocarea datelor despre dispozitive şi taskuri.

Folosirea acestui sistem de operare permite utilizarea serviciilor Google şi a API-ului Google Map Android V2 a acestora. Google Map pune la dispoziţie o hartă prevazută cu functionalităţi de zoom in, zoom out precum şi posibilitatea de a marca diferite locaţii. Aceste functionalităţi sunt indispensabile pentru a permite utilizatorului să indice locaţia unde se execută un task. De

asemenea, serviciile google au fost folosite pentru a determina locaţia curentă a utilizatorului, pe baza reţelelor Wireless sau a GPS-ului, funcţie de cea care oferă mai multă acurateţe. Cunoaşterea locaţiei curente este de asemenea necesară, în vederea determinării distanţelor între aceasta şi alte locaţii.

Altă bibliotecă specifică Androidului folosită în cadrul aplicaţiei este **android.location** prin intermediul căreia se determină adresa unei locaţii pe baza unor coordonate, respectiv coordonatele unei locaţii pe baza unei adrese.

În ceea ce priveşte interfaţa grafică, Android permite programatorului utilizarea facilă a componentelor grafice în cadrul activităţii prin intermediul XML-ului. Astfel, folosind XML-ul, programatorul poate specifica locaţia relativă a componentelor una faţă de cealalta, organizarea componentelor sub formă ierarhică şi structurarea activităţii în fragmente, a atributelor precum fontul, culoarea textului, mărimea componentei şi id-urile componentelor pentru a fi apoi referite şi utilizate în clasă.

Printre componentele grafice puse la dispoziţie de către Android se numără:

- Date Picker pentru selecţia datei,
- Time Picker pentru selecţia orei,
- liste drop down,
- câmpuri,
- butoane.

3. Noţiuni teoretice

Pentru dezvoltarea aplicaţiei au fost necesare cunoştinţe de:

- Programare orientată pe obiecte
- Sistem operare Android
- Algoritmi inteligenţă artificială

3.1 Programare orientată pe obiecte

Dezvoltarea aplicaţiei a fost realizată în limbajul de programare Java. Java este un limbaj de programare orientat pe obiecte, programul fiind structurat în colecţii de obiecte şi având drept caracteristici urmatoarele principii : abstractizarea, modularizarea, ierarhizarea si încapsularea[1].

Abstractizarea este principiul prin care se încearca identificarea unei similarităţi între situaţii, obiecte concrete din viaţa reală şi contextul problemei de rezolvat, accentul punându-se pe trăsăturile comune, esenţiale în rezolvarea problemei. Astfel, programarea orientată pe obiecte gravitează în jurul definirii claselor şi a caracteristicilor acestora ce reprezintă un set de de variabile şi proceduri denumite membri şi metode.

Modularizarea este principiul prin care se realizează structurarea complexă a programului în componente de o complexitate redusă numite module. Modulele au avantajul de a putea fi compilate separat şi au posibilitatea de a avea conexiuni cu alte module ale programului.

Ierarhizarea este principiul prin care se definesc relaţiile de subordonare între obiectele definite. Relaţiile pot fi :

- de tip , prin care o clasă are acces la structura unei alte clase prin moştenire
- de agregare, prin care o clasă are acces la structura altei clase având drept membru o instanţă a sa.

Încapsularea este principiul prin care vizualizarea metodelor şi valorilor membrilor unei clase poate fi restricţionată de relaţiile între cele două clase şi de modificatorii de acces.

Prin urmare, programarea orientată pe obiecte este un limbaj care preia facilitaţile programării procedurale şi are în plus faţă de aceasta structurarea programului în module uşor de înţeles si găsirea unor şabloane între viaţa reală şi contextul problemei.

3.2 Sistemul de operare Android

Sistemul de operare Android dispune de o serie de tehnologii precum Bluetooth, reţele Wifi şi GPS care permit comunicarea între dispozitive, obţinerea de informaţii de pe Internet, determinarea locaţiei utilizatorului şi, împreună cu suportul senzorilor, a contextului în care se află utilizatorul şi activităţilor sale. Astfel, sistemul de operare oferă mijloacele necesare pentru a implementa o aplicaţie inteligentă şi interactivă cu utilizatorul[2].

Entitatea care stă la baza implementarii unei aplicaţii Android este activitatea. Activitatea reprezintă componenta aplicaţiei prin care este gestionată interfaţa grafică, analog ferestrelor într-o aplicaţie desktop sau a unei pagini dintr-o aplicaţie web. Alte elemente importante utilizate in cadrul unei aplicaţii Android sunt componentele grafice, obiectele Broadcat Receiver şi obiectele Intent.

Precum o aplicaţie web sau desktop, o aplicaţie android conţine mai multe activităţi, comunicarea dintre acestea fiind realizată prin intermediul unui obiect Intent. Un obiect Intent are rolul de a porni şi de a descrie o nouă activitate şi de a-i transmite valori din activitatea actuală. Alte roluri ale obiectului Intent sunt pornirea serviciilor şi a receiver-elor broadcast.

3.2.1 Ciclul de viaţă al unei activităţi Android

Activitatea este caracterizată de un ciclu de viaţă, începând din momentul în care este creată de către un intent până când este distrusă la ieşirea din aceasta. Astfel, stările pe care le deţine o activitate pe perioada acestui interval sunt urmatoarele: stare de creare, pornire, rulare, pauză, oprire, distrugere, pentru setarea acestora apelându-se metodele **onCreate, onStart, onResume, onPaused, onStop, onDestroy** . Stările de rulare, pauză şi oprire sunt statice, restul fiind tranzitive[3].

Starea de creare este una dintre stările tranzitive în care este apelată metoda **onCreate** pentru crearea unei instanţe a activitaţii.Implementarea metodei **onCreate** trebuie sa conţină instanţierea membrilor activităţii, configurarea unor componente grafice şi alte operaţii care trebuie să se realizeze într-un singur ciclu de viaţă al unei activităţi. La sfârşitul metodei **onCreate** sistemul trece rapid la urmatoarele stări : de pornire şi de rulare.

Starea de pornire este starea în care activitatea devine vizibilă şi este urmată rapid de starea de rulare. Starea de rulare este starea în care activitatea este în prim plan iar utilizatorul poate sa interacţioneze cu aceasta.

Starea de pauză este iniţiată în momentul în care activitatea este eclipsată de catre altă activitate aflată în prim plan, dar care nu acoperă tot ecranul (de exemplu un dialog). În această stare, activitatea nu poate primi comenzi de la utilizator şi nu poate să execute operaţiuni, astfel că orice acţiune este oprită până când utilizatorul reia activitatea sau până când iese definitiv din ea. În metoda **onPause**, este indicat să se oprească acţiunile care ar putea ocupa CPU şi memoria, precum animaţiile, obiectele broadcast receiver sau fişiere deschise. Dacă utilizatorul revine la activitate, este reapelată metoda **onResume** iar activitatea devine din nou vizibilă.

Starea de oprire este starea în care activitatea este complet ascunsă şi nu este vizibilă utilizatorului, considerandu-se că se află în fundal. La fel ca metoda **onPause**, şi în metoda **onStop** are loc eliberarea resurselor, cele care implică operaţii CPU intensive, precum scrierea datelor în baza de date. Situaţii posibile în care activitatea ar trece în starea de oprire sunt: când utilizatorul primeşte un apel telefonic în momentul navigării prin aplicaţie sau în momentul în care din activitatea curentă este pornită una nouă. Daca utilizatorul părăseşte activitatea nouă şi se întoarce la cea actuală, se reapelează metoda **onStart**, iar activitatea este repornită.

Ultima stare prin care trece activitatea este cea de distrugere. Activitatea este distrusă în momentul în care este apelată metoda **finalize()**, când utilizatorul apasă pe butonul de back, sau dacă este distrusă de catre sistem din cauza faptului ca foloseşte prea multe resurse.

3.2.2 Fişierul Manifest

Aplicatia are la baza un fisier xml numit implicit AndroidManifest.xml. In acest fisier sunt specificate date generale cu privire functionarea si structurarea aplicatiei. Radacina fisierului are marcajul <manifest>[4].

Astfel, contine informatii cu privire la versiunea minima si ultima versiune de Android pe care este functionala aplicatia. Aceasta informatie este continuta in marcajul uses-sdk, in valorile atributelor android:minSdkVersion si android:targetSdkVersion. Acest lucru este exemplificat prin urmatoarea secventa de cod.

```
<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="18"
/>
```

De asemenea, sunt specificate permisiunile care trebuie oferite aplicatiei pentru ca aceasta sa functioneze in mod correct. Permisiunile sunt oferite de catre utilizator cand aplicatia este instalata. Un exemplu de permisiune, prin care se solcita permisiunea de acces la internet, este urmatoarea:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Alte informatii din fisierul Manifest specifica activitatile aplicatiei, care este activitatea principala a aplicatiei(prin marcajul intent-filter) , metadatele folosite, icon-ul folosit pentru identificarea aplicatiei in interfata dispozitivului, etc. O secventa de cod prin care se specifica aceste informatii este urmatoarea :

```
<application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <meta-data
              android:name="com.google.android.gms.version"
              android:value="@integer/google_play_services_version" />
      <activity
              android:name="com.example.meniu.MainActivity"
              android:label="@string/app_name" >
              <intent-filter>
                      <action android:name="android.intent.action.MAIN" />
                      <category android:name="android.intent.category.LAUNCHER" />
              </intent-filter>
      </activity>
      . . . . . .
 </application>
```

3.2.3 Componentele grafice Android

Interfaţa grafică a unei activităţi este formată dintr-o ierarhie de obiecte **View**, respectiv **View Group**. Instanţele **View** reprezintă frunzele ierarhiei şi sunt componente grafice precum câmpuri text şi butoane. Alte componente grafice utilizate în interfaţă sunt Date Picker şi Time Picker, prin care utilizatorul poate să selecteze data şi ora, slidere, liste spinner[5].

Instanţele **View Group** reprezintă nodurile părinte ale instanţelor **View** şi sunt containerele în care sunt afişate componentele grafice. Rădăcina ierarhiei este un ViewGroup reprezentat de unul din elementele următoare : RelativeLayout (se poate specifica poziţia unei componente grafice relativ faţă de alta), LinearLayout (componentele grafice sunt aranjate într-o singură coloană sau rând) , ScrollView (în cazul în care compontentele grafice nu încap în activitate, este asociat ca parinte obiectului RelativeLayout sau LinearLayout un obiect ScrollView, cu ajutorul căruia utilizatorul poate să baleieze activitatea).

Definirea componentelor grafice şi a modului în care sunt afişate poate fi realizată în mod static sau în mod dinamic. În modul static, componentele sunt definite într-un fişier XML şi sunt vizibile la pornirea activităţii fără a fi necesar să fie gestionate în cadrul acesteia. În cadrul fişierului XML, se pot specifica informaţii precum poziţia relativă a componentei grafice faţă de container sau faţă de alte componente grafice din acelaşi container, culoarea componentei grafice, textul continuu, mărimea. Se pot defini de asemenea id-uri prin care componentele pot fi identificate în cadrul activităţii pentru a li se putea asocia ascultători.

Programarea dinamică se realizează prin adaugarea componentelor grafice prin intermediul codului activitatii. Este utilă când existenţa componentelor grafice depinde de aparitia unui eveniment ca de exemplu apasarea unui buton. Se pot specifica aceleasi informatii referitoare la componenta grafica ca in XML.

3.2.4 Broadcast Receiver

Broadcast Receiver reprezintă componenta aplicaţiei Android care poate fi înregistrată într-o activitate cu scopul de a asculta evenimente de sistem sau de aplicaţie. Evenimente sunt instanţe ale clasei Intent si printre acestea se numără: detectarea unei reţele wireless, detectarea unei reţele bluetooth, dacă bateria nu mai are energie sau dacă a fost conectată la încărcător. În momentul în care evenimentul pentru care instanţa receiver a fost înregistrată este sesizat, aceasta este notificată şi îi este apelată metoda **onReceive**[6].

Pentru ca resursele folosite pentru înregistrarea receiver-ului să nu rămână ocupate după distrugerea activităţii, este necesar ca metoda **onStoped** a activităţii să fie suprascrisă pentru dealocarea resurselor prin apelarea metodei **unregisterReceiver()**;

3.2.5 Bluetooth

Platforma Android oferă suport pentru reţele Bluetooth, dând posibilitatea dispozitivului de a face schimb de informaţii cu alte dispozitive prevăzute cu Bluetooth[7].

Accesul la funcţionalitatea Bluetooth se face prin API-ul Android Bluetooth, acest API realizând comunicarea wireless Point to Point între cele două dispozitive. Prin folosirea API-ul, sunt posibile urmatoarele operaţiuni : scanarea pentru alte dispozitve prevăzute cu Bluetooth, conectarea celor două dispozitive, transferul de date de la un dispozitiv la altul, interogarea adaptorului Bluetooth pentru dispozitive împerecheate.

Detectarea de dispozitive este o procedură de scanare a dispozitivelor locale Bluetooth şi care solicită apoi informaţii despre fiecare. Dispozitivele, în cazul în care sunt configurate pentru a fi detectate, vor raspunde oferind informaţii, cum ar fi : numele dispozitivului, clasa şi o adresă MAC unică. Utilizând aceste informaţii, dispozitivul ce a executat cautarea poate iniţia o conexiune cu dispozitivele căutate.

Căutarea de dispozitive Bluetooth se realizează prin apelarea metodei **startDiscovery** de către o instanţă BluetoothAdapter. Este un proces asincron de scanare care dureaza aproximativ 12 secunde, urmat de interogarea în detaliu a fiecărui dispozitiv pentru obţinerea datelor acestuia.

Pentru ca aplicaţia să fie notificată în momentul în care un nou dispozitiv este găsit, este necesară înregistrarea unui Broadcast Receiver cu filtrul asociat **BluetoothDevice.*ACTION_FOUND***.

Pentru ca resursele folosite pentru înregistrarea receiver-ului să nu ramână ocupate după distrugerea activităţii, este necesar ca metoda **onStoped** sa fie suprascrisă pentru dealocarea resurselor prin apelarea metodei **cancelDiscovery()** a instanţei BluetoothAdapter.

Pentru utilizarea API-ului Android Bluetooth, trebuie să fie declarate permisiuni în fişierul Manifest al aplicaţiei :

```
<!-- for Bluetooth discovery devices -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

3.2.6 Stocarea datelor

Stocarea informaţiilor poate fi realizată pe Android folosind două metode:

- Shared Preferences[8]
- Sqlite[9]

Shared Preferences reprezintă o structură de date sub forma unui set de perechi cheie-valoare, fiind o structură Non Relational Database. Este o opţiune optimă în cazul în care datele sunt simple, nu este necesar a fi structurate, numărul intrărilor nu este foarte mare şi are avantajul eliminării latenţei cauzate de join-ul dintre tabele.

După obţinerea handle-ului Shared Preferences asociat activităţii curente, inserarea unei valori se face prin apelarea unei metode care primeşte ca parametri o valoare distincta cu rolul de cheie al perechii şi valoarea asociată acesteia. De asemenenea, obţinerea valorilor inserate se face prin apelarea unor metode ce primesc ca parametru cheia folosită.

În cazul aplicaţiei mele, Shared Preferences a fost o utilă pentru specificarea facilă către utilizator a adresei locaţiei taskului, având drept cheie un string, iar ca valoare totalitatea adreselor specificate de utilizator la adăugarea taskurilor. Astfel, în momentul în care utilizatorul introduce o noua adresă, la salvarea taskului în baza de date, adresa este adaugată la lista de adrese din taskurile precedente. Ulterior, când utilizatorul va dori sa specifice aceeaşi adresă, setul de adrese extras din shared preferences va fi folosit ca "autocomplete" pentru ca utilizatorul să nu fie nevoit să scrie întreaga adresă.

Următorul cod explică modul în care sunt extrase şi adăugate informaţiile din shared preferences:

```
// pentru extragere
SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
valueString = sharedPreferences.getString(valueKey, defaultString);

// pentru inserare
SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
Editor edit = sharedPreferences.edit();
edit.putString(keyString, valueString);
edit.commit();
```

Sqlite oferă posibilitatea de a crea baze de date relaţionale, utile când exista un număr mic de date de salvat (în comparaţie cu Shared Preferences), iar datele sunt complexe şi pot fi văzute ca un set de de perechi atribut-valoare, între care există relaţii.

În cazul aplicaţiei mele, a fost necesar să stochez informaţii privind dispozitivele detectate de bluetooth, programul fix al utilizatorului şi taskurile introduse. Pentru aceasta, am creat o clasă ce extinde SQLiteOpenHelper şi care are metodele de creare şi updatare a bazei de date suprascrise.

3.3 Algoritmi învăţare automată

Învăţarea automată este o ramură a inteligenţei artificiale care oferă dispozitivelor abilitatea de a învaţa, de a lua o decizie într-un anumit context , fără ca aceasta să fie explicit specificată, dupa realizarea unor observaţii făcute în timpul procesării unui set de date[10].

Învăţarea automată presupune identificarea şi implementarea unei modalităţi cât mai eficiente de a reprezenta informaţii, în sensul facilitării căutării, reorganizarii şi modificării lor. Alegerea modului de a reprezenta datele ţine de conceptia generală asupra modului de rezolvare a problemei şi de caracteristicile datelor cu care se lucrează.

Cei mai mulţi algoritmii de învăţare automată folosesc trei metode care diferă prin feedbackul primit în urma luării unei decizii şi modul în care proceseaza datele: învăţare nesupervizată, învăţare supervizată si învăţare prin recompensă.

În învăţarea nesupervizată, scopul agentului inteligent este de a determina un pattern în inputul primit sau de a lua o decizie, fără să îi fie dat apriori un set de date etichetate pe baza cărora să

determine patternul. Cea mai des întâlnită utilizare a învăţării nesupervizate este clusterizarea: împarţirea setului de date în grupuri de date care pot fi considerate similare.

În învăţarea supervizată, agentul primeşte apriori un set de date deja etichetate numit set de antrenare şi un set de date de testare neetichetate. Fiecărei valori din setul de date de antrenare X_i îi este asociată o valoare Y_i , astfel încât există o funcţie F care, oricare ar fi i, $F(X_i) = Y_i$. Scopul algoritmului este de a calcula o funcţie H numită funcţie ipoteză, care are rolul de a aproxima funcţia F. Astfel, pentru fiecare valoare din setul de antrenare, $H(X_i)$ să ofere aceleaşi valori ca F, şi pentru fiecare valoare din setul de testare, H să ofere un rezultat cât mai apropiat de cel dat de F.

În învăţarea prin recompensă, precum în învăţarea nesupervizată, agentul inteligent nu primeşte un set de date apriori. În timpul derulării algoritmului, după fiecare decizie luată, agentul primeşte o recompensă sau o pedeapsă, în funcţie de calitatea deciziei. Ulterior, în cazul în care agentul se regăseşte intr-o situaţie prin care a mai trecut, va realiza aceeaşi acţiune sau va încerca una diferită, în funcţie de recompensa sau pedeapsa primită anterior.

3.3.1 Algoritmul K-Means

Algoritmul K-Means este un algoritm de clusterizare NP dură, care primeşte la intrare un set de N puncte într-un spaţiu hiperdimensional pe care le partiţioneaza în K clustere în funcţie de apropierea sau depărtarea dintre date, încercând să minimizeze variaţia medie a punctelor. K Means foloseşte ca distanţă între puncte distanţa euclidiană, aceasta fiind folosită în spaţii euclidiene[11].

Paşii implementării algoritmului sunt:

1. Din punctele date ca input se aleg K puncte cât mai depărtate unul faţă de celalalt, numiţi centroizi
2. Fiecare punct din setul de date este asignat celui mai apropiat centroid.
3. După asignarea punctelor, se recalculează coordonatele fiecărui centroid ca media valorilor coordonatelor punctelor asociate în fiecare dimensiune.
4. Consecinţa recalculării coordonatelor centroizilor este că punctele pot deveni ulterior mai apropiate de alţi centroizi faţă de cei cărora au fost asignate iniţial, aşa că se reiau paşii 2 si 3 până când niciun punct nu îşi mai schimbă centroidul asignat.

Scopul acestui algoritm este reducerea numărului de date care să fie folosite într-o problemă ulterioară, astfel ca, în locul folosirii întregului set de date primit ca input se pot utiliza numai centroizii determinaţi în urma clusterizării.

Principala provocare a algoritmului K-Means este estimarea numărului optim de clustere. Metoda cea mai des utilizată pentru a determina K optim se numeşte **metoda elbow**.

Metoda elbow presupune creşterea numărului de clustere începând de la K = 1 şi calcularea pentru fiecare K a variaţiei medie a punctelor în cadrul clusterelor. Variaţia medie scade pe masură ce este crescută valoarea k, astfel că metoda presupune oprirea incrementării valorii lui K şi a recalculării clusterelor după ce valoarea variaţiei medie calculată la un pas nu a scăzut semnificativ faţă de valoarea de la pasul anterior[12][13].

Variaţia medie a punctelor, pentru o valoare K se calculează astfel :

$$Vk = \sum_{r=1}^{k} \frac{Dr}{2Nr}$$

unde N_r este numărul de clustere, iar D_r este suma pătratelor distanţelor euclidiene în cadrul clusterului r între fiecare doua puncte **i** si **j** aparţinând clusterului.

$$Dr = \sum_{Xi} \sum_{Yi} (Xi - Yi)^2$$

Altă formulă care nu presupune calculul distanţei între fiecare doua puncte ale unui cluster, ci se bazeaza numai pe distanţa euclidiană între un punct dintr-un cluster şi centroid, este urmatoarea:

$$Dr = \sum_{Xi \in r} (Xi - \mu r)^2$$

3.3.2. Algoritmi genetici

Algoritmii genetici reprezintă o opţiune euristică de a căuta o soluţie pentru o problemă dată. Reprezintă o alternativă la algoritmii clasici faţă de care are avantajul de a determina mult mai rapid o soluţie într-un spaţiu de căutare foarte mare. Soluţia găsită nu este cea optimă, dar este satisfăcătoare pentru complexitatea problemei[14].

Algoritmii genetici se bazează pe principiile geneticii şi ale selecţiei naturale. Astfel, în contextul algoritmilor genetici, o potentială soluţie a problemei se numeşte **individ**, iar un set de indivizi se numeşte **populaţie**. La o anumita etapă în rezolvarea problemei, populaţia trece printr-un set de schimbări - **selecţie, combinare, mutaţie** -prin care indivizii acesteia devin mai bine adaptaţi mediului (potenţialele soluţii au o valoare mai apropiata de soluţia optimă).

În implementarea celor trei operaţii trebuie să existe un echilibru între exploatare şi explorare. În cazul în care accentul în implementare este pus pe exploatare, este îmbunătăţită valoarea fitnessului celui mai bun individ la un moment dat, fără să se ia în considerare ca viitoare posibile soluţii ceilalţi indivizi, iar algoritmul converge astfel rapid către o soluţie locală. În cazul în care accentul în implementare este pus pe explorare, se iau în considerare ca soluţii finale pentru problemă foarte mulţi indivizi şi se incearcă imbunătăţirea lor, iar algoritmul converge către o soluţie globală, însă foarte lent.

Valoarea individului şi apropierea sa de soluţia optimă se numeşte **fitness** şi este calculată cu ajutorul unei funcţii obiectiv a carei formă depinde de contextul problemei.

Astfel, etapele implementării algoritmului sunt urmatoarele :

1. Stabilirea structurii individului şi generarea aleatoare a unui număr de indivizi care să reprezinte populaţia iniţială.

2. Executarea setului de operaţii asupra populaţiei :
    a. Selecţie pe baza fitnesului
    b. Combinarea
    c. Mutaţie
3. Se reia pasul 2 până când se îndeplineşte criteriul de oprire

În etapa de selecţie, funcţia obiectiv este evaluată pentru fiecare individ pentru determinarea fitness-ului acestora. Ulterior, un număr de indivizi din generaţia curentă este selectat pentru a fi inclus în generatia viitoare. Metodele de selecţie a indivizilor pot fi :

- selecţia ruletă, în care posibilitatea pentru un individ de a fi ales este proporţională cu valoarea fitness-ului acestuia
- selecţia trunchiată, în care este aleasă o fracţie din populaţie reprezentată de cei mai buni indivizi
- selecţia turneu ce presupune generarea aleatorie de un număr de ori a unui subset din populaţie şi selectarea ulterior a celui mai bun individ din fiecare subset.

Combinarea este un operator genetic utilizat pentru diversificarea cromozomilor de la o generaţie la alta şi este analog reproducţiei din genetică pe care se bazează algorimtii genetici. Astfel se selectează aleator un număr de perechi de indivizi, pe baza carora se produc alti indivizi, numiţi descendenţi, până când numărul de indivizi din noua generaţie este egal cu numărul de indivizi din generaţia curentă.

Mutaţia este operatorul genetic utilizat pentru păstrarea diversităţii genetice de la o generaţie la alta şi este analog mutaţiei biologice, astfel că realizează schimbarea unei valori a unui element din structura individului. Mutaţia cauzează creşterea explorării spaţiului problemei pentru găsirea unei soluţii, astfel că după această etapă algoritmul genetic poate obţine soluţii mai satisfăcătoare. Mutaţia se poate petrece la un individ conform unei probabilităţi definită de către programator. În cazul în care valoarea probabilităţii este setată la o valoare mare, căutarea soluţiei va deveni aleatorie.

Criteriile de oprire a algoritmului sunt : găsirea soluţiei ce satisface un criteriu impus, dacă se atinge un număr de generaţii, dacă se consumă bugetul alocat în privinţa resurselor sistemului informatic sau dacă valoarea celui mai bun individ stagnează după un număr de generaţii.

3.4 Inteligenţa ambientală

Inteligenţa ambientală este o ramură a inteligenţei artificiale ce presupune implementarea unor aplicaţii inteligente pe dispozitive senzitive la prezenţa unei persoane, dispozitivele conlucrând pentru a oferi suport utilizatorilor în rezolvarea activităţilor zilnice[15][16].

Implementarea unui sistem prevăzut cu inteligenţă ambientală se bazează pe urmatoarele concepte şi proprietăţi :

- adaptarea comportamentului în funcţie de cererile şi profilul utilizatorului

- caracter proactiv, anticipativ astfel că poate lua decizii proprii fără să fie necesară interacţiunea cu utilizatorul. Acest lucru se poate petrece intr-o situaţie în care sistemul poate să ia o decizie, situatie similară cu o una în care a fost informat de către utilizator cu privire la acţiunea ce trebuia luată.
- Caracter senzitiv cu privire la situaţia curentă în care se află utilizatorul. Determinarea contextului curent se determină cu ajutorul tehnologiilor cu care este echipat dispozitivul. Printre acestea se numară : senzori, tehnologie wireless, bluetooth, GPS, e.t.c.
- Caracter omniprezent, astfel ca tehnologiile cu care este echipat dispozitvul şi pe care se bazează să fie funcţionale indiferent de locaţia în care se află utilizatorul.
- Utilizarea unei tehnologii hardware neobservabilă sau cât mai puţin observabilă, astfel încât interacţiunea cu sistemul ambiental să fie cât mai natural posibilă pentru utilizator.
- Caracter comunicativ, astfel să poată să realizeze schimb de informaţii cu alte dispozitive pentru un suport mai mare oferit utilizatorului.

Pentru ca inteligenţa ambientală sa fie acceptată, scopul implementării acesteia trebuie sa aibă în vedere rezolvarea problemelor/preocupărilor umane. Una dintre acestea este facilitarea interacţiunii dintre oameni într-o comunitate şi a suportului oferit între aceştia. Ca de exemplu, un dispozitiv inteligent ar putea să comunice altor dispozitive că utilizatorul său are nevoie de asistenţă în rezolvarea unei probleme, precum găsirea unei destinaţii (sau asistarea în utilizarea calculatorului)

Un alt aspect al inteligenţei ambientale priveşte siguranţa şi confidenţialitatea utilizatorului, aceasta putând să reacţioneze promt în cazul în care detectează o situaţie în care viaţa utilizatorului este pusă în pericol. Astfel, în cazul unui accident pe autostradă, soldat cu ranirea unor persoane, poate să contacteze serviciile medicale pentru asistarea persoanelor rănite. Mai mult, deoarece se bazează foarte mult pe informaţiile obţinute despre utilizator, este necesar ca informaţiile confidenţiale să fie folosite local de către dispozitiv ţi să nu fie partajate altor dispozitive.

Inteligenţa ambientală poate fi de asemenea folosită pentru motive economice şi ecologice. Ca de exemplu, cu ajutorul tehnologiilor cu care este echipat, dispozitivul poate să informeze utilizatorul care este cel mai scurt şi mai convenabil traseu spre o anumită destinaţie, sau poate regla utilizarea resurselor(precum aerul condiţionat, iluminarea unei încăperi) în funcţie de solicitarea şi necesitatea lor.

4. Algoritmi

4.1 Algoritmi genetici pentru implementarea problemei comis voiajorului

Problema comis voiajorului este definită astfel : Fie G = (V, E) un graf neorientat în care oricare două vârfuri diferite ale grafului sunt unite printr-o latură căreia îi este asociat un cost strict pozitiv. Cerinţa este de a determina un ciclu care începe de la un nod aleatoriu al grafului, care trece exact o dată prin toate celelalte noduri şi care se întoarce la nodul iniţial, cu condiţia ca ciclul sa aibă un cost minim. Costul unui ciclu este definit ca suma tuturor costurilor ataşate laturilor ciclului[17].

În contextul aplicaţiei, calcularea unui program cât mai optim al utilizatorului reprezintă o variantă complexă a problemei comis voiajorului, astfel ca locaţiile unde se execută taskurile reprezintă nodurile grafului şi se doreşte calcularea unui traseu pentru parcurgerea căruia să fie necesar cât mai puţin timp. În plus faţă de aceasta cerinţă, se adaugă urmatoarele:

- nodurile grafului au propriile lor costuri (reprezentate de duratele de execuţie a taskurilor) care pot să difere în funcţie de momentul în care comis voiajorul (utilizatorul) ajunge la ele. Ca exemplu, durata cumpărăturilor poate să difere în cazul în care acest task este primul executat de către utilizator faţă de cazul în care este executat ultimul.
- Costul total al arcelor şi al nodurilor nu trebuie sa depaşeasca o anumita valoare. Pentru acest scop se poate renunţa la trecerea printr-un anumit punct al grafului.
  Astfel, durata execuţiei unui set de taskuri nu trebuie sub nicio formă să se intercaleze cu execuţia programului fix, şi se poate renunţa la unele taskuri pentru aceasta. În cazul în care programul fix nu este respectat, costul este penalizat proporţional cu intercalarea.
- Pentru trecerea prin anumite puncte ale grafului se oferă recompense, în funcţie de importanţa punctului,recompense care scad costul total. Astfel, alegerea taskului pentru execuţie în cazul în care trebuie respectat un program fix ia în considerare prioritatea acestuia.

Pentru calcularea unui program cât mai optim sunt utilizaţi algoritmi genetici. Individul în acest context este o structură formată din setul de taskuri de executat şi ora la care utilizatorul incepe să execute acest set.

În etapa de iniţializare, se creeaza un număr de indivizi cu setul de taskuri de lungime diferită generat aleatoriu şi cu ora de începere aleasă aleatorie, astfel încât să respecte intervalul selectat de utilizator.

În etapa de selecţie, se calculează mai întâi valoarea de fitness a fiecărui individ. Se doreşte obţinerea unui individ cu fitnessul având o valoare cât mai mică, astfel ca : valoarea fitnessului este constituita din durata totală de execuţie a taskurilor, la care se adaugă penalizările nerespectării

programului şi se scad recompensele oferite pentru execuţia fiecărui task. Se ordonează apoi populaţia de indivizi în ordine descrescătoare dupa fitness şi se aleg 40% din numărul total al indivizilor pentru a face parte din generaţia următoare a populaţiei.

Penalizările folosite sunt : fiecare minut care nu este în concordanţă cu intervalul selectat este multiplicat cu 5 pentru primele 15 minute, cu 10 pentru următoarele 15 si cu 15 pentru restul.

Recompensele oferite sunt : 60 pentru un task cu prioritatea MINOR, 120 pentru un task cu prioritatea AVERAGE, 480 pentru un task cu prioritatea MAJOR, 1500 pentru un task cu prioritatea CRITICAL.

În etapa de combinare, se completează restul de 60% din populaţia generaţiei urmatoare astfel:

- se aleg aleator câte 2 indivizi având rolul de parinţi , indiferent de fitness, pentru a fi creaţi doi copii.
- Se calculează ora de începere a fiecărui individ copil după formula recombinării reale liniare :

  offspring = parent1 + Alpha * (parent2 - parent1)

  unde Alpha este un factor de scalare ales aleator în intervalul [-d, 1+d]. Parametrii din formula preiau valorile :

  Ora începere copil = Ora începere Individ1 + Alpha *(Ora de începere Individ2 – Ora de începere Individ1),

  iar Alpha are valorile 0.25 pentru primul copil si -0.25 pentru al doilea copil

- Se calculează setul de taskuri ale indivizor copii astfel :

  - Se alege un task conţinut de ambii părinţi
  - Dintre cele doua taskuri care urmează după acesta dintre cei doi indivizi părinte, se determină care durează mai puţin pentru a fi adăugat la setul de taskuri ai copiilor. Dacă cel care dureaza mai putin există deja în structura copilului, atunci este adăugat celalalt task.
  - Dacă seturile de taskuri ale părinţilor au dimensiuni diferite atunci : primul copil va avea setul de taskuri completat cu taskuri din setul primului părinte, până când vor avea aceeaşi dimensiune, iar al doilea copil va avea setul de taskuri completat cu taskuri din setul celui de-al doilea părinte, până când vor avea aceeaşi dimensiune.

- Se repetă procedeul până când populaţia generaţiei următoare are numărul de indivizi egal cu populaţia curentă.

În etapa de mutaţie, indivizii populaţiei formată din selecţie şi combinare pot suferi următoarele mutaţii, cu probabilitatea de 0.25 :

- Schimbarea aleatorie a orei de începere a execuţiei taskului,
- Interschimbarea poziţiilor a două taskuri în setul de taskuri a individului
- Eliminarea un task din setul de taskuri a individului
- Adăugarea unui task la setul de taskuri a individului
- Înlocuirea unui task din setul de taskuri a individului cu unul ce nu se află în setul de taskuri

Algoritmul se termină când se ajunge la un număr de iteraţii egal cu 10 inmulţit cu numărul de taskuri, dacă este depăşit un interval de timp, sau în cazul în care converge la un individ optim.

4.1.1 Testare algoritm

Pentru testarea algoritmului, au fost introduse în baza de date următoarele taskuri care trebuie să fie executate: cumpărături(cu durata variabilă si prioritate minoră), abonament metrou (cu durata variabilă si prioritate medie), proiect licenţă (cu durata setată de 4 ore şi prioritate majoră). S-a considerat că utilizatorul începe execuţia taskului din poziţia curentă a acestuia şi la sfârşitul execuţiei acesta se intoarce înapoi la poziţia curentă.

În cazul în care intervalul de execuţie a taskurilor rămâne cel implicit, de la ora 7 AM până la 9 PM, algoritmul reuşeşte să sugereze o ordine şi momentele de execuţie a taskurilor astfel încât taskurile “cumpărături” si “abonament metrou” să fie executate în momente ale zilei în care durata execuţiei să fie cea minimă şi astfel durata totală a execuţiei taskurilor să fie minimă.

În cazul în care intervalul de execuţie este restrâns la un numar mai mic de ore, algoritmul sugerează o ordine a acelor taskuri care merită să fie executate, din care reiese că implementarea ţine cont de prioritatea taskurilor, de distanţa lor relativă faţă de utilizator şi de respectarea intervalului stabilit. Astfel, pentru un interval de la 12 PM până la 6 PM, este sugerat pentru execuţie taskul “proiect licenţă”, pentru care durata totală poate să ajungă la 5 ore şi 30 de minute (4 ore execuţia, 50 de minute deplasarea la facultate, 50 de minute înapoi) . În cazul în care intervalul este restrâns şi mai mult, în locul acestui task sunt sugerate celelalte doua taskuri “cumpărături” şi “abonament metrou”, pentru ca intervalul să fie respectat.

Eroarea cu care execuţia taskurilor a depaşit intervalul specificat nu a fost mai mare de 15 minute, iar aceste cazuri implică execuţia taskului “proiect licenţă”, pentru care prioritatea este majoră.

4.2 Algoritmul de clusterizare KMeans

Algoritmul KMeans este un algoritm de clusterizare care partiţionează un set de N puncte în K clustere iar fiecare cluster are un punct reprezentativ,numit centroid, astfel că un punct aparţine clusterului având cel mai similar, apropiat centroid de punct.

Scopul acestui algoritm este de a reduce un număr mare de date la un set de date reprezentative care să fie folosit ulterior în rezolvarea unei probleme, fiind astfel redusă complexitatea problemei.

În aplicaţie, contextul unui task adăugat în baza de date este comparat cu contextele taskurile deja executate de către utilizator. Se va considera că informaţiile contextului cel mai similar vor fi

aceleaşi şi pentru taskul respectiv. Astfel, în cazul în care baza de date conţine un număr foarte mare de taskuri executate, o soluţie avantajoasă comparării contextului taskului ce urmează să fie adăugat cu fiecare context, se bazează pe compararea numai cu un număr mic de contexte reprezentative.

Astfel, pentru determinarea duratei unui task, sunt clusterizate taskurile executate dupa trei proprietăţi ale acestora, analog unor puncte într-un sistem tridimensional: locaţia, titlul si momentul de execuţie a taskului.

Pentru ca noţiuni precum similaritatea, depărtarea să fie inţelese în acest context, se calculează distanţa între doua taskuri folosind urmatoarea formulă :

$$d(T1, T2) = \sqrt{factorDistance^2 * distanceDuration(T1,T2)^2 + factorTitle^2 * distanceTitle(T1,T2)^2 + distanceLocation(T1,T2)^2}$$

unde distanceDuration reprezintă diferenţa între momentele de start ale celor două taskuri exprimată în minute, distanceTitle reprezintă o formă a distanţei Levenshtein dintre titluri, iar distantaLocation reprezintă distanţa euclidiană între locaţiile unde se execută cele două taskuri.

Este necesar ca distanţa dintre durate si distanţa dintre titluri să fie inmulţite cu anumite ponderi pentru ca cele trei distanţe să fie comparabile. Astfel, presupunând că distanţele maxime sunt : cea dintre locaţii de 15 km, cea dintre titluri de 30 de caractere, iar cea dintre momentele de start de 720 de minute, este necesar ca distanţa dintre titluri sa fie amplificată de 500 de ori iar cea dintre momentele de start aproximativ de 14 ori.

Primul pas constă în alegerea a K taskuri cu rol de centroizi cât mai îndepartaţi unul de celalalt.

Al doilea pas constă în determinarea pentru fiecare task executat din baza de date, a celui mai apropiat centroid, iar taskul este asignat clusterului acestuia. Se recalculează coordonatele fiecărui centroid pe baza coordonatelor punctelor ce aparţin clusterelor lor.

Astfel, noul moment de începere a unui centroid se calculează ca media artmetică a momentelor de start ale tuturor punctelor clusterului respectivului centroid :

$$startTime(Centroidj) = \frac{startTime(T1)+startTime(T2)+\cdots+startTime(Ti)}{i}, \text{ pentru } j = 1, K$$

iar coordonatele locaţiei centroidului se calculează ca media aritmetică a coordonatelor tuturor punctelor clusterului respectivului centroid :

$$latitude(Centroid) = \frac{latitude(T1)+latitude(T\ 2)+\cdots+latitude(Ti)}{i}, \text{ pentru } j = 1, K$$

$$longitude(Centroid) = \frac{longitude(T1)+longitude(T2)+\cdots+longitude(Ti)}{i}, \text{ pentru } j = 1,K$$

După acest pas, taskurile pot deveni mai apropiate de alte clustere decât faţă de cele cărora au fost asignate. În acest caz, se repetă algoritmul de la pasul 2, până când taskurile nu işi mai schimbă centroizii cărora au fost asignaţi, iar algoritmul se termină (nu devin mai apropiate de alte clustere).

4.2.1 Determinare K

Provocarea în clusterizarea Kmeans este determinarea lui K pentru care clusterizarea să fie cea corecta. O metodă pentru determinarea lui K este metoda **elbow** care se bazează pe ideea că dacă se repetă clusterizarea pentru o valoare K mai mare cu o unitate, nu se vor obţine rezultate considerabil mai bune.

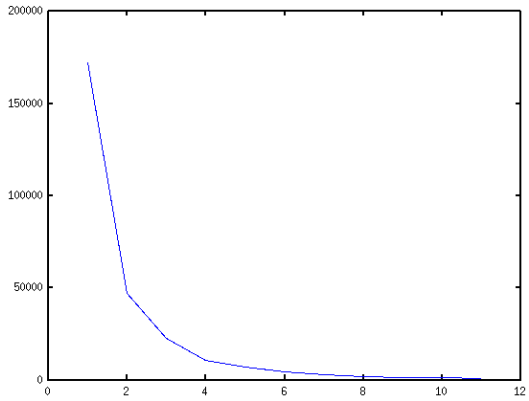
Astfel, metoda elbow presupune etapele:

- Iniţializare K =1
- Clusterizarea setului de taskuri
- Calcularea variaţiei taskurilor într-un cluster
- Incrementarea lui K şi reclusterizare până când diferenţa variaţiilor dintre doua etape consecutive atinge un prag.

Metoda elbow se bazează exclusiv pe distanţa euclidiană într-un sistem Euclidian, astfel că pot aparea erori în determinarea numărului corect de taskuri din moment ce sunt implicate atât distanţa Levenstein cât şi distanţa termporală. Astfel, pentru ca metoda elbow să nu se oprească la un optim local k, condiţia pentru oprirea incrementării numărului de centroizi este :

$\Delta(W(k) - W(k+1)) < \frac{W(k)}{3}$, unde W(k) reprezintă variaţia taskurilor pentru k clustere.

Pentru testare s-a folosit un set de 80 de taskuri. Toate taskurile au acelaşi titlu, se execută câte 40 într-o locaţie şi momentele de execuţie variază de la ora 8:00 pana la 19:00. Graficul din figura “K-means” arată scăderea variaţiei în raport cu mărirea numărului de centroizi. Deşi numărul de centroizi de la care variaţia nu mai scade drastic este 4, numărul de centroizi care verifică condiţia şi care dă rezultate mai satisfăcătoare este 8.



(Fig K-means)

4.2.2 Distanţa între titluri a două taskuri

Distanţa între titluri a două taskuri se bazează pe distanţa Levenshtein dintre două şiruri de caractere. Distanţa Levenshtein reprezintă numărul minim de adăugări, ştergeri, înlocuri de caractere în primul şir pentru a fi obţinut al doilea. Această metodă nu poate fi însă aplicată direct asupra şirurilor celor două titluri, din moment ce pot sa conţină aceleaşi cuvinte însă în altă ordine[18].

Astfel, se ia fiecare cuvânt din primul şir si se determină care este cel mai similar cuvânt din al doilea şir pe baza distanţei Levenshtein. În cazul în care distanţa dintre cuvântul din primul şir şi cel mai similar cuvânt din al doilea şir depaşeşte un anumit prag, se consideră că cele doua cuvinte au sensuri diferite, nu sunt înrudite şi distanţa lor este adăugată la distanţa totală dintre cele doua titluri. Se repetă procedeul pentru fiecare cuvânt din primul titlu.

Ulterior calculării distanţei dintre cele doua titluri, se verifică si dacă aceasta depaşeşte un anumit prag, pe baza ideii că unul dintre şiruri poate avea cuvinte fără importanţă faţă de celălalt, care în mod normal nu ar trebui luate în considerare la calculul distanţei. Astfel, dacă pragul este depăşit, se ia în considerare distanţa totală calculată dintre titluri.

4.2.3 Testare Algoritm

Au fost folosite urmatoarele teste pentru clusterizare:

- O locaţie, 20 de taskuri, durata de începere variabilă de la ora 8 AM la 7 PM
  => 4 clustere
- 2 locaţii, acelaşi titlu, 80 de taskuri, durata de începere variabilă de la ora 8 AM la 7 PM
  => 8 clustere

- 2 locaţii, 2 titluri, 40 de taskuri (câte 20 pentru fiecare locaţie), durata de începere variabilă de la 8 AM la 7 PM => 7 clustere
- 3 locaţii, 3 titluri, 60 de taskuri (câte 30 pentru fiecare locaţie) , durata de începere variabilă de la 8 AM la 7 PM => 11 clustere
- 3 locaţii, 2 titluri, 100 de taskuri, durata de începere variabilă de la 8 Am la 7PM => 11 clustere

5. Detalii implementare

5.1 Funcţionalităţi

Fereastra principală a aplicaţiei pune la dispoziţia utilizatorului mai multe opţiuni:

- setarea programului fix al utilizatorului ce se repetă în fiecare săptamană (activitatea Set Fix Schedule)

- adăugarea unui task şi ce implică acesta : locaţia, durata, prioritatea, e.t.c. (activitatea Add Task)

- vizualizarea taskurilor adăugate şi modificarea proprietăţilor lor (View Tasks)

- vizualizarea taskurilor ce se pot executa în contextul actual (View compatible tasks)

- detectarea dispozitivelor prin Bluetooth şi înregistrarea informaţiilor primate de la acestea (Detect Devices)

- recomandarea orarului pentru un interval al zilei luând în considerare taskurile adăugate şi programul fix al utilizatorului (Suggest Schedule)

- vizualizarea taskului curent de executat (View Current Task)

5.1.1 Setarea programului fix

Utilizatorul poate să specifice în această activitate programul sau săptămânal, fix, format din taskurile pentru care sunt ştiute locaţia şi momentul pentru execuţie.

Pentru fiecare zi a săptămânii, se pot adauga sau şterge taskuri sau li se pot modifica intervalul în care acestea se petrec. Aceste modificări făcute în interfaţa sunt salvate în baza de date.

5.1.2 Adăugarea unui task

Utilizatorul poate să adauge în această activitate taskuri pentru care trebuie să specifice contextul în care acesta poate să fie executat: titlul, locaţia, alte dispozitive ale utilizatorului şi persoanele necesare pentru execuţie, durata estimată de execuţie a taskului şi deadline-ul acestuia plus alte caracteristici precum titlul taskului pentru ca acesta să fie recunoscut şi prioritatea acestuia.

Activitatea dispune de o interfaţă inteligentă astfel că pe masură ce utilizatorul completează unele câmpuri ale interfeţei, altele sunt completate automat de către entitatea inteligentă a aplicaţiei, pe baza istoricului taskurilor excutate de către utilizator si pe baza câmpurilor deja completate.

Câmpurile care sunt completate de catre entitatea inteligenta sunt : locaţia, dispozitivele şi persoanele necesare care se bazează pe titlul taskului introdus de către utilizator şi prioritatea, care se bazează pe titlul introdus, deadline şi durată, în cazul în care acestea sunt completate.

De exemplu, pentru locaţie, activitatea clusterizează apriori taskurile deja executate dupa titlu. În momentul în care utilizatorul introduce titlul taskului, acesta este comparat cu titlurile centroizilor obţinuţi prin clusterizare. Se determină cu care dintre acestea titlul introdus este cel mai similar şi în cazul în care similaritatea depaşeşte un anumit prag, se foloseşte locaţia centroidului respectiv pentru a completa locaţia din interfaţa grafică. Acelaşi procedeu este efectuat şi pentru celelalte câmpuri.

Pentru ca utilizatorul să seteze sau să vizualizeze locaţia recomandată de entitatea inteligentă, acesta dispune de o harta obţinută prin API-ul Google Map. Pentru uşurinţă, în momentul deschiderii activităţii, harta afişează automat locaţia curentă a utilizatorului determinată cu ajutorul clasei LocationClient a bibliotecii Google Play Services.

Următoarea secvenţă de cod descrie determinarea locaţiei curente a utilizatorului:

```
public class AddTask implements com.google.android.gms.location.LocationListener,
                                GooglePlayServicesClient.ConnectionCallbacks{
    /**
     * used to get the user's location
     */
    LocationClient mLocationClient;

    /**
    * used to get updates about the user's location
    */
    LocationRequest locationRequest;
    public void onCreate(Bundle savedInstanceState){
              mLocationClient = new LocationClient(this, this, this);
              locationRequest = new LocationRequest();
              locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
              locationRequest.setInterval(5000);
              locationRequest.setFastestInterval(1000);
              locationRequest.setNumUpdates(1);
    }

    @Override
    public void onConnected(Bundle arg0){
              Location l =  mLocationClient.getLastLocation();
              if(l == null)
                        mLocationClient.requestLocationUpdates(locationRequest, this);
              else
                        currentLocation = l;
    }

    @Override
    public void onLocationChanged(Location arg0){ currentLocation = arg0; }

}
```

Utilizatorul poate de asemenea să caute o locaţie pe hartă specificând în interfaţă cuvinte cheie ce fac parte din adresa acesteia. Procedeul poate fi executat şi în ordine inversă, astfel că în momentul în care utilizatorul selectează o locaţie pe hartă, în interfaţă este afişată adresa acesteia, calculată cu ajutorul instanţei Geocoder din biblioteca android.location

```
List<Address> addresses = null;
try {
        addresses = (new Geocoder(this)).getFromLocationName(searchPattern, Integer.MAX_VALUE);
        if(addresses != null)
        {
                Address closest = determineClosestAddress(addresses);
                LatLng position = new LatLng(closest.getLatitude(), closest.getLongitude());
        }
}
 } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
 }
```

În momentul în care este aleasă locaţia taskului pe hartă (direct de către utilizator, prin căutarea unei adrese sau prin compararea titlului cu cele aflate în baza de date), pe hartă apare un marcaj albastru iar locaţia este afişată în prim plan. Codul pentru această funcţionalitate este :

```
location = Double.toString( position.latitude) + " " +
           Double.toString(position.longitude);
map.clear();
map.moveCamera(CameraUpdateFactory.newLatLngZoom(position, 15));
map.addMarker(new MarkerOptions().position(positionCurrent).
icon(BitmapDescriptorFactory.fromResource(R.drawable.ic_launcher)));
map.addMarker(new MarkerOptions().position(position).
icon(BitmapDescriptorFactory.fromResource(R.drawable.ic_action_location_place)));
```

O ultimă funcţionalitate a interfeţei inteligente este reprezentată de câmpurile titlu şi adresa care au proprietatea de autocomplete. Astfel, adresele căutate de către utilizator pe harta şi titlurile taskurilor executate sunt extrase din baza de date a aplicaţiei pentru a completa automat aceste campuri în cazul în care se determină un şablon comun cu ce a tastat momentan utilizatorul.

5.1.3 Vizualizarea taskurilor introduse si modificarea proprietatilor

În această activitate este posibilă vizualizarea tuturor taskurilor salvate de către utilizator şi încă neexecutate. De asemnea, utilizatorul are ca opţiuni ştergerea taskurilor sau modificarea proprietaţilor acestora, caz în care este pornită activitatea de adaugare taskuri, însă având câmpurile deja completate de catre utilizator. După ce utilizatorul efectueaza modificările dorite, datele câmpurilor sunt folosite pentru updatarea valorilor atributelor înregistrării corespunzatoare din tabelul TASKS .

Datele despre fiecare task sunt afişate în activitate în mod dinamic, prin intermediul codului activităţii şi nu prin intermediul fişierului Xml. O secvenţă de cod care exemplifică această adăugare este următoarea :

```
private void addTaskToInterface(Task task) {

TextView titleView = new textView(this);
RelativeLayout.LayoutParams params_title_value =
new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.MATCH_PARENT,
                                                RelativeLayout.LayoutParams.WRAP_CONTENT);
params_title_value.setMargins(100, 30, 100, 0);
titleValue.setText(task.getNameTask());
titleValue.setTextSize(30);
titleValue.setBackgroundColor(Color.rgb(193, 215, 222));
titleValue.setId( ++ numberOfView);
titleValue.setPadding(0, 5, 0, 5);
```

```
titleValue.setTextColor(Color.BLUE);
titleValue.setGravity(Gravity.CENTER);
params_title_value.addRule(RelativeLayout.BELOW, numberOfView - 2);
titleValue.setLayoutParams(params_title_value);
layout.add(titleView);
. . . . . .
}
```

5.1.4 Vizualizarea taskurilor ce se pot executa în contextul actual

Utilizatorul poate vizualiza în această activitate taskurile a caror condiţii sunt îndeplinite de către contextul curent pentru a fi executate şi care au asociată starea “TO DO” în baza de date. Pentru ca taskul să fie afişat, toate condiţiile trebuie să fie respectate : deţinerea dispozitivelor şi existenţa în apropierea utilizatorului a persoanelor necesare, proximitatea faţă de locaţia aleasă, nesuprapunerea execuţiei taskului cu programul fix stabilit de utilizator.

O primă condiţie a taskului priveşte proximitatea locaţiei curente a utilizatorului faţă de locaţia aleasă. Astfel, este calculată distanţa dintre cele 2 locaţii care nu trebuie sa depăşească un prag pentru a fi indeplinită condiţia.

A doua condiţie de respectat este nesuprapunerea execuţiei taskului cu programul fix al utilizatorului. Pentru a verifica aceasta condiţie, se calculează durata deplasării din locaţia curentă până la locaţia unde trebuie să fie executat taskul, durata propriu-zisă de execuţie şi durata deplasării până în locaţia unde utilizatorul trebuie să respecte programul fix. Astfel, sunt adunate cele trei valori determinate la timpul curent, pentru a fi ulterior comparate cu programul fix. Dacă este depăşită ora de începere a programului fix, taskul nu respectă condiţia pentru a fi executat.

În cazul în care pentru realizarea taskului sunt necesare anumite persoane, aplicaţia detectează cu ajutorul tehnologiei Bluetooth dacă acestea se află în preajmă. Aplicaţia detectează toate dispozitivele aflate în proximitatea utilizatorului, le interoghează pentru a afla adresele lor mac, pe care le compară apoi cu adresele mac ale dispozitivelor persoanelor necesare taskului. Acelaşi procedeu este aplicat şi pentru detectarea dispozitivelor utilizatorului şi în plus, cazul în care dispozitivul este momentan închis.

Taskurile sunt afişate în această activitate în ordinea descrescătoare a priorităţii.

Utilizatorul poate selecta un task pentru a fi executat. În cazul acesta, starea taskului din baza de date se schimbă din “AMONG TO DO” în “CURRENT TASK”. Interfaţa este apoi actualizată pentru a nu mai lua în considerare taskul respectiv.

Codul următor explică modul în care sunt verificate taskurile: clasa conţine un set de checkere pentru fiecare condiţie, astfel că pentru fiecare task din setul de taskuri se parcurge setul de checkere. Un checker primeşte ca parametri elementul de context din contextul curent si elementul de context din contextul taskului verificat.

```
for(Task task : tasks)
{
        isTaskCompatible = true;
        for(ContextElementType elementType: checkers.keySet())
        {
                Compatibility  checker = checkers.get(elementType);
                if(checker.check(task.getScheduledContext().getContextElementsCollection
```

```
                    ().get(elementType),
                    currentContext.getContextElementsCollection
                    ().get(elementType), task) == false)
                {
                        isTaskCompatible = false;
                        break;
                }
        }
        if(isTaskCompatible == true)
                addTaskToInterface(task) ;
}
```

5.1.5 Detectarea dispozitivelor prin Bluetooth şi înregistrarea informaţiilor primite de la acestea

Această funcţionalitate este utilizată pentru a stoca informaţii care sunt ulterior necesare pentru a verifica dacă utilizatorul are dispozitivele necesare pentru execuţia taskului sau dacă persoanele necesare sunt în preajma acestuia.

Astfel, activitatea detectează prin Bluetooth dispozitivele din proximitatea utilizatorului. Le interoghează ulterior pentru a afla adresele MAC şi numele dispozitivelor. Datele sunt afişate în interfaţă, iar utilizatorul le poate salva în baza de date, specificând dacă dispozitivul este al lui sau al altei persoane. La urmatoarea scanare a dispozitivelor, nu vor mai fi afişate dispozitivele ale căror informaţii au fost salvate anterior în baza de date.

Tehnologia Bluetooth ascultă asincron notificările privind dispozitivele detectate timp de 12 secunde, astfel că detectarea tuturor dispozitivelor nu se va face instantaneu.

Pentru ca aplicaţia să fie notificată în momentul în care este detectat cu Bluetooth un dispozitiv, este necesară înregistrarea unui Broadcast Receiver cu filtrul asociat **BluetoothDevice.*ACTION_FOUND***.

Codul metodei onReceive este urmatorul :

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

        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                    // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                  if(checkIfExists(device.getAddress()) == false)
             {
                    myActivy.getDeviceInfo().put(device.getAddress(), device.getName());
                    myActivy.addInfoMethod();
            }
        }
}
```

5.1.6 Recomandarea orarului pentru un interval al zilei luând în considerare taskurile adăugate şi programul fix al utilizatorului

Utilizatorului îi sunt recomandate în aceasta activitate un set de taskuri într-o anumită ordine astfel încât programul fix al utilizatorului să fie respectat. Taskurile luate în considerare şi ordinea lor se bazează pe obţinerea unei valori cât mai optime ca rezultat al unei funcţii obiectiv din domeniul algoritmilor genetici. Această funcţie obiectiv ia în considerare durata totală a execuţiei taskurilor în ordinea respectivă, prioritătile lor şi nerespectarea intervalului selectat de către utilizator.

În interfaţa grafică, utilizatorul poate selecta chiar el un anumit interval oarecare în care sa îi fie calculat programul zilei respective sau poate selecta un interval între perioadele în care are programul fix. În interfaţă îi sunt afişate ordinea execuţiei taskurilor, durata de execuţie a fiecăruia şi durata de deplasare dintr-un loc precedent.

Pentru durata de deplasare, se consideră că dacă distanţa nu depaşeşte un anumit prag, deplasarea se poate face fără mijloc de transport, altfel, este necesară folosirea unuia. În cazul în care este folosit un mijloc de transport, se considera că utilizatorul se deplasează un metru in 0.6 secunde, iar în carzul în care nu este folosit, atunci utilizatorul se deplasează un metru în 1.2 secunde.

5.1.7 Vizualizarea taskului curent de executat

Utilizatorul poate vizualiza în această activitate taskul selectat pentru a fi executat din activitatea Show Compatible Tasks.

În interfaţă sunt afişate proprietăţile taskului selectate de către utilizator, un timer care indică cât timp a mai rămas până la executarea taskului şi butoane pentru a se putea adăuga minute sau ore la durata estimată şi pentru a informa aplicaţia când a finalizat taskul.

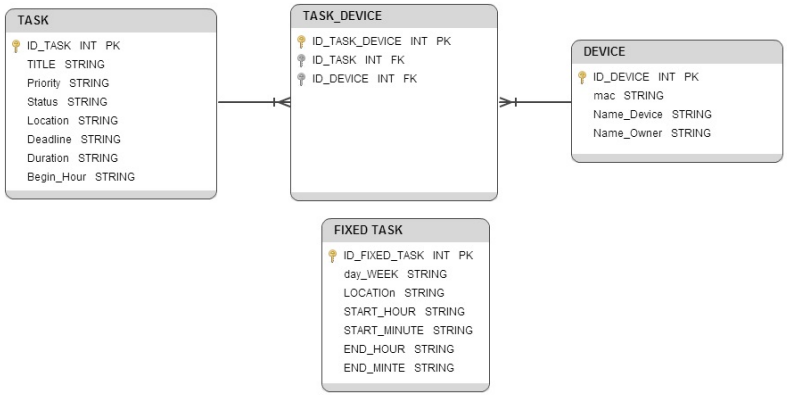
Durata afişată de către timer la inceputul execuţiei taskului este valoarea selectată de catre utilizator în interfaţa “**Add task**”, sau, în cazul în care utilizatorul nu a ştiut durata la momentul respectiv, alegând opţiunea “**Unknown duration**”, este valoarea calculată de entitatea inteligentă. În ultimul caz, locaţia , titlul taskului şi ora curentă sunt date folosite pentru a determina care este cel mai apropiat centroid din cei calculaţi prin clusterizare. Durata centroidului, calculată ca medie a tuturor punctelor din clusterul său, este folosită apoi ca valoare a taskului curent de executat.

Durata ramasă din execuţia taskului este calculată ca diferenţă între durata estimată în momentul alegerii acesteia pentru execuţie şi timpul trecut din acel moment.

Taskul este considerat a fi finalizat în momentul în care utilizatorul apasă butonul “**Finalize**”. În cazul în care timpul a expirat fără ca utilizatorul să facă acest lucru, la o ulterioară intrare în activitate sunt afişate 10 minute până la sfârşitul taskului. În momentul apăsării butonului “**Finalize**” , timpul rămas din execuţie este scăzut din durata taskului salvată în baza de date.

5.2 Baza de date

Baza de date care stochează informaţiile necesare funcţionalităţilor aplicaţiei este formată din urmatoarele tabele : **TASKS**, **DEVICES**, **SCHEDULE**, **DEVICES_TASKS.** Structura acesteia este urmatoarea



5.2.1 Tabela Tasks

Această tabelă conţine informaţii privind taskurile adăugate de către utilizator prin activitatea "**Add Tasks**". Conţine urmatoarele câmpuri:

- ID_TASK : reprezintă cheia primară a tabelului şi valoarea este id-ul taskului asociat unei intrări în tabelă.

- TITLE : şir ce reprezintă titlul taskului.

- LOCATION : şir ce conţine coordonatele locaţiei alese de către utilizator, delimitate de caracterul ",".
- DEADLINE : şir ce reprezintă deadline-ul taskului asociat de către utilizator. În cazul în care nu se ştie deadline-ul, are valaorea UNKNOWN.

- PRIORITY : şir ce reprezintă prioritatea taskului, putând lua valorile : MINOR, AVERAGE, MAJOR,

CRITICAL. În cazul în care nu se ştie prioritatea, are valoarea UNKNOWN.

- DURATION : şir ce reprezintă durata taskului. În cazul în care nu este ştiută durata, are valoarea -1. Valoarea îi este schimbată în momentul în care taskul este ales pentru a fi executat, cu durata unuia dintre centroizii calculaţi de entitatea inteligentă prin clusterizare.

- STATUS: şir ce reprezintă statusul taskului. Poate avea valorile : AMONG_TO_DO, dacă a fost adăugat în lista de taskuri, CURRENT_TASK, dacă este taskul ce este executat la momentul actual, EXECUTED, dacă a fost finalizat.

- BEGIN_TIME : şir ce reprezintă momentul în care taskul a fost selectat pentru a fi executat. Este de forma dd/MM/yyyy/hh/mm.

5.2.2 Tabela SCHEDULE

Această tabelă conţine informaţii privind programul setat de către utilizator prin activitatea "**Set fixed Schedule**". Conţine următoarele câmpuri:

- ID_TASK : reprezintă cheia primară a tabelului şi valoarea este id-ul taskului asociat unei intrări în tabelă.
- DAY_WEEK : şir ce specifică în ce zi a săptămânii trebuie executat programul. Poate avea valorile MONDAY, TUESDAY, WEDNESDAY, THURSADAY, FRIDAY.
    - LOCATION : şir ce contine coordonatele locaţiei alese de către utilizator, delimitate de caracterul "," .
- START_HOUR : şir ce indică ora la care începe taskul.
- START_MINUTE : şir ce indică minutul la care începe taskul.
- END_HOUR : şir ce indică ora la care se termină taskul.
- END_MINUTE : şir ce indică minutul la care se termina taskul.

5.2.3 Tabela DEVICES

Această tabelă conţine informaţii privind dispozitivele detectate cu Bluetooth în activitatea "**Detect devices**". Conţine următoarele câmpuri:

- ID_DEVICE : reprezintă cheia primară a tabelului şi valoarea este id-ul device-ului asociat unei intrari în tabelă.
- MAC : şir ce reprezintă adresa mac a dispozitivului detectat.
- NAME_DEVICE : şir ce reprezintă numele dispozitivului detectat.
- NAME_OWNER : şir ce reprezintă numele persoanei care deţine dispozitivul. În cazul în care este vorba despre utilizator, NAME_OWNER are valoarea "**My Device**".

5.2.4 Tabela Task_Devices

Această tabelă are rolul de a rezolva relaţia de "many to many" între tabela Devices şi tabela Tasks: contextul unui task poate implica utilizarea mai multor dispozitive, iar un dispozitiv poate fi necesar în mai multe taskuri. Conţine următoarele câmpuri:

- ID_TASK_DEVICE : reprezintă cheia primară a tabelei
- ID_TASK : reprezintă cheia straină, corelaţia cu cheia primară a tabelei Tasks
- ID_DEVICE : reprezintă cheie straină, corelaţie cu cheia primară a tabelei Devices

5.2.5 Operaţii DML Sqlite

Pentru a realiza operaţii DML în cadrul unui tabel în sqlite, se utilizeaza clasa SQLiteOpenHelper. Pentru inserare sau updatare de înregistrări, valorile sunt salvate într-o instanţă ContentValues, care are structura unei liste de perechi cheie-valoare, unde cheia este reprezentată de numele atributului înregistrării.

Un exemplu de secvenţă de cod pentru inserarea unei inregistrari în tabela Devices este următorul :

```
ContentValues values = new ContentValues();
values.put(DeviceData.KEY_MAC, macAddress);
values.put(DeviceData.KEY_DEVICE, nameDevice);
values.put(DeviceData.KEY_OWNER, owner);
SQLiteDatabase db = this.getWritableDatabase();
db.insert(nameTable, null, content);
db.close();
```

5.3 Modularizare

Aplicaţia este modularizată fiind prevăzută pe lângă activităţile proprii cu următoarele module: **DatabaseOperation , ContextElements, Salesman, Check_Compatibility , Clustering.**

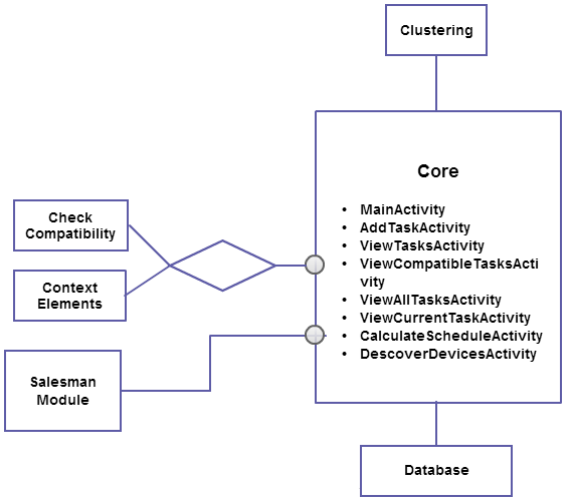


Fig. – Schema modularizare Aplicatie

5.3.1 DatabaseOperation

Acest modul conţine implementarea bazei de date a aplicaţiei . Principala clasă este **DataBaseHandler** ce conţine metodele ce implementează operaţiile DDL şi DML ale tabelelor:

- crearea tabelelor,
- adăugarea, updatarea, ştergerea unui task, a unui dispozitiv, a unui task din programul fix al utilizatorului,
- extragerea tuturor informaţiilor din cele trei tabele sau doar a unui set de înregistrări pe baza unor criterii primite ca parametri.

Conţine de asemenea câte o interfaţă pentru fiecare tabel, în care sunt specificate atributele acestora : **deviceData, fixedTasksData, tasksData**.

Alte componente ale acestui modul sunt clase ce implementeaza ascultătorii asociaţi butoanelor din activităţi care apelează operaţiile DDL si DML ale bazei de date.

5.3.2 Clustering

Acest modul conţine clasele ce clusterizează taskurile cu statusul Executed. Taskurile sunt nişte puncte în sisteme având mai multe dimensiuni.

- KMeansDuration : sistemul are trei dimensiuni : în prima dimensiune este utilizată ca distanţă diferenţa orelor de începe a taskurilor, în a doua dimensiune este utilizată distanţa euclidiană între locaţiile taskurilor, iar în a treia dimensiune este utilizată distanţa Levenshtein dintre titlurile taskurilor.
- KMeansLocation , KMeansDevices, KMeansPeople : sistemul are o dimensiune în care este utilizată distanţa Levenshtein dintre titlurile taskurilor.

Aceste clase implementează metodele interfeţei KMeans:

- void calculateKlusters() : apelată de către activităţi pentru calcularea centroizilor.
- Task detectCentroid(Task currentTask) : primeşte ca parametru un task şi returnează centroidul cel mai apropiat de acesta.
- void chooseCentroid() : selectează taskul cel mai îndepărtat de centroizii curenţi pentru a fi considerat el însuşi centroid.
- calculateDistance(Task t1, Task t2): calculează distanţa dintre două taskuri.
- void calculateNewCentroizi() : calculează noile poziţii ale centroizilor în funcţie de valorile punctelor aflate în clusterele lor.
- boolean checkCentroiziNotChanged() : verifică dacă există taskuri care şi-au schimbat centroizii.
- float calculateError() : calculează eroarea clusterizării ca suma pătratelor distanţelor dintre puncte şi centroizi.

Modulul conţine în plus şi o clasă în care sunt implementate metodele de calcul a distanţelor Levenshtein, distanţa euclidană şi diferenţa dintre orele de început a taskurilor.

5.3.3 Travelling Salesman

Modulul conţine clasele utilizate pentru implementarea algoritmului genetic care concepe programul utilizatorului pentru o zi :

- PopulationEvolution: conţine implementarea propriu-zisă a algoritmului, având câte o metodă pentru fiecare operaţie a acestuia : iniţializare, calcularea fitnessului, selecţie, crossover, mutaţie. Acestea sunt apelate dintr-o metodă principală până când se îndeplineşte criteriul de finalizare.
- Individual : reprezintă structura individului utilizat în algoritm şi care reprezintă Soluţia problemei.
- ComputationalMethods : calculează durata deplasării între două locaţii şi distanţa dintre acestea. Conţine de asemenea recompensele asociate fiecărui tip de prioritate.
- ComparatorFitness : Implementează funcţia **compare** a interfeţei Comparator, pentru sortarea indivizilor în etapa de selecţie.
- ConstantsPopulation : interfaţă ce conţine constantele utilizate în implementarea algoritmului precum pragul pentru mutaţie, procentul ales pentru selecţie,penalizările pentru întârizere e.t.c.

5.3.4 ContextElements

Taskul este conceput ca fiind format dintr-un set de elemente de context. Modulul ContextElements conţine aceste elemente de context. Pentru ca valorile acestora să fie obţinute într-un mod facil şi clar din clasa task, aceasta are ca membru o listă de perechi cheie-valoare, având drept chei valorile din enumeraţia ContextElementType şi ca valori următoarele clase :

- DeadlineContext : are ca membru un şir ce memorează data setată de utilizator ca fiind deadline-ului taskului.
- DeviceContext : are ca membru o listă unde sunt memorate dispozitivele necesare pentru execuţia taskului.
- LocationContext : are doi membri cu valoarea Double unde sunt memorate coordonatele locaţiei şi anume latitudinea şi longitudinea.
- PeopleContext : are ca membru o listă unde sunt memorate dispozitivele necesare pentru execuţia taskului.
- DurationContext : are ca membri durata propriu-zisa a taskului şi durata de deplasare din locaţia curentă până în locaţia unde trebuie să fie executat taskul.

- TemporalContext : are ca membru o listă având ca elemente intervale ale zilei. În cazul contextului curent, acestea sunt intervalele în care se desfaşoară programul fix, iar în cazul contextului taskului, reprezintă intervalul în care este executat taskul, calculat pe baza timpului curent, a duratei propriu-zise a execuţiei taskului împreună cu durata de deplasare până la locaţia taskului şi durata de deplasare până la locaţiile unde se desfasoară programul fix al utilizatorului.

5.3.5 CheckCompatibility

Acest modul este folosit pentru afişarea taskurilor în activitatea View Compatible Tasks ale căror contexte sunt în compatibilitate cu cel curent. Astfel, acesta conţine următoarele clase ce implementează metoda checkCompatibility a interfeţei Checker. Metoda checkCompatibility primeşte ca parametri două elementele de context de acelasi tip, unul aparţinând contextului curent, iar celălalt aparţinând contextului taskului de verificat. Rezultatul funcţiei este **true** dacă taskul poate fi executat sau **false** în caz negativ.

- DeviceCompatibility : folosit pentru a verifica dacă dispozitivele contextului taskului sunt valabile şi în contextul curent. Compară elementele de context de tip DeviceContext.
- PeopleCompatibility : folosit pentru a verifica dacă persoanele contextului taskului sunt valabile şi în contextul curent. Compară elementele de context de tip PeopleContext.
- LocationCompatibility : folosit pentru a verifica dacă locaţia taskului este în proximitatea locaţiei curente. Compară elementele de context de tip LocationContext.
- TemporalCompatibility : folosit pentru a verifica dacă durata taskului şi momentul de execuţie sunt în compatibilitate cu programul fix al utilizatorului, dacă se intercalează cu acesta. Compară elementele de context de tip TemporalContext.

6. Descrierea aplicaţiei

6.1 Meniu principal

La startarea aplicaţiei, interfaţa afişează funcţiile disponibile, care apar în meniul principal. Aceste funcţii sunt următoarele:

- Adăugarea unui nou task
- Vizualizarea şi modificarea taskurilor încă neexcutate
- Vizualizarea taskurilor compatibile cu taskul curent
- Vizualizarea taskului curent de executat
- Setarea programului fix al utilizatorului
- Vizualizarea unui program pentru o zi sugerat de aplicaţie
- Detectarea dispozitivelor prin Bluetooth .

Interfaţa aplicaţiei este prietenoasă iar meniul principal este prezentat în figura 6.1

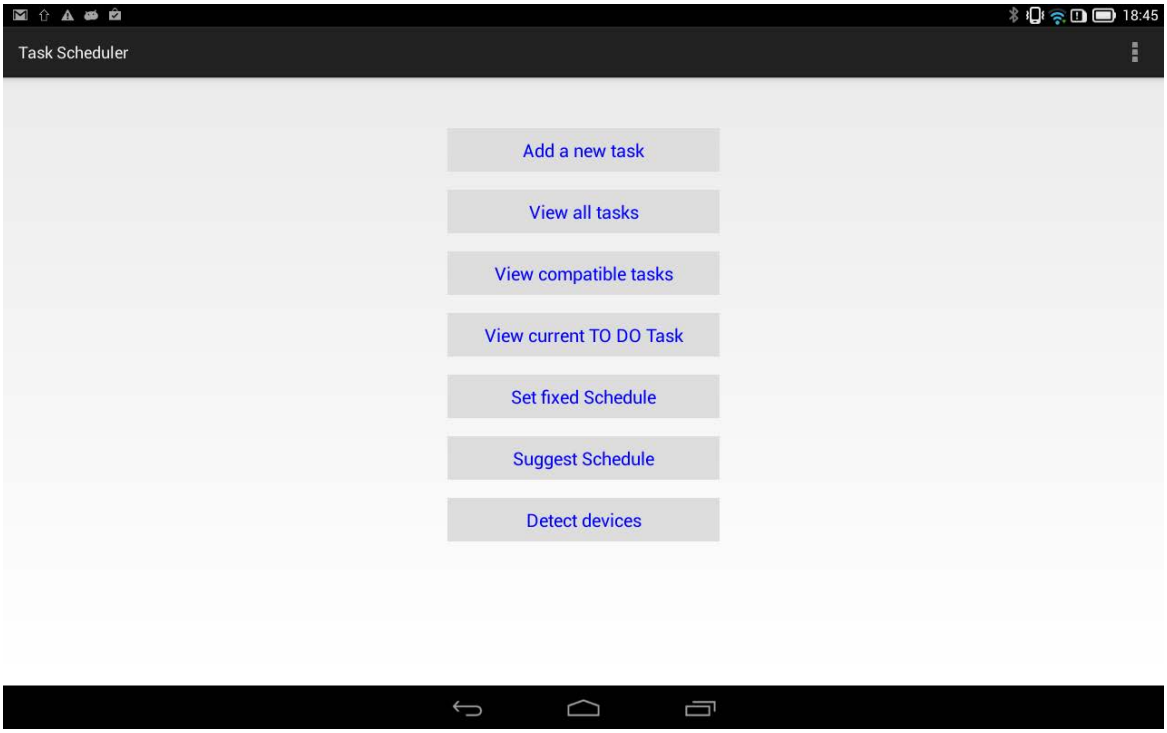


Fig. 6.1 – Meniul principal al aplicaţiei

În continuare sunt descrise detaliat funcţiile oferite de aplicaţie.

6.2 Adăugarea unui task

Pentru adăugarea unui task, utilizatorul trebuie să apese pe butonul "**Add a new task**" din meniul principal. În urma accesării acestei funcţii, pe ecran apare fereastra următoare :

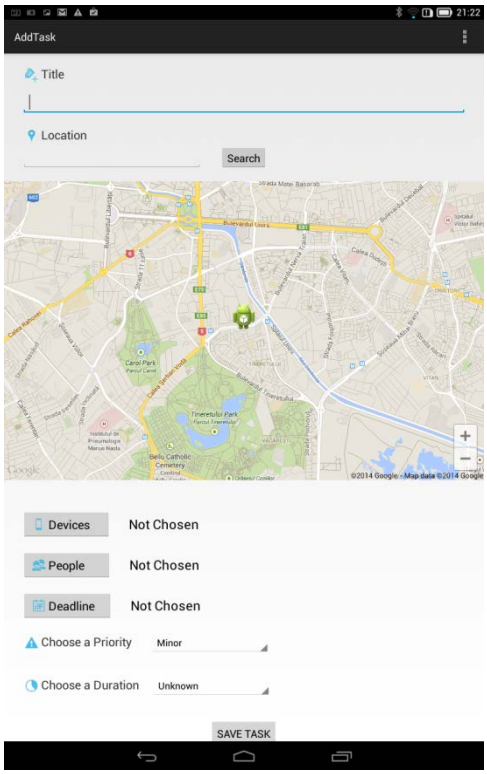
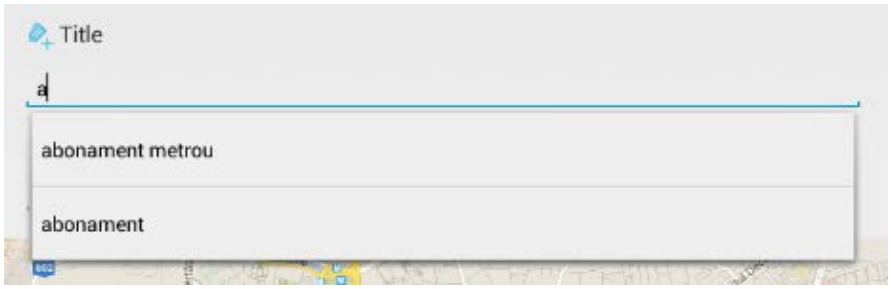


Fig. 6.2 – Interfaţă adăugare task

Utilizatorul poate să specifice titlul, locaţia, dispozitivele şi persoanele necesare, deadline-ul priorităţii şi durata taskului. Pentru acestea, sunt oferite următoarele facilităţi :

- Asigurarea suportului pentru completarea titlului sau a locaţiei cu ajutorul funcţionalităţii autocomplete (vezi fig. 6.3)



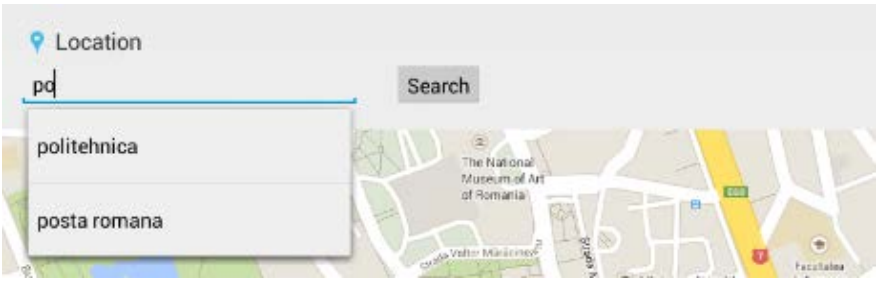


Fig. 6.3 – Adăugarea titlului şi locaţiei

- Pentru selecţia locaţiei, utilizatorul are următoarele alternative :
  - poate selecta de pe hartă (prin navigare pe aceasta) locaţia dorită , sau
  - poate căuta adresa locaţiei prin completarea câmpului corespunzator locaţiei (vezi figura 6.3) şi apăsarea butonlui “Search”.

Dacă selectează locaţia direct pe hartă, în câmpul unde ar introduce locaţia căutată este scrisă adresa acesteia.

- Locaţia curentă este marcată pe hartă cu un roboţel, iar cea cautată de utilizator cu un pointer albastru (vezi figura 6.4)

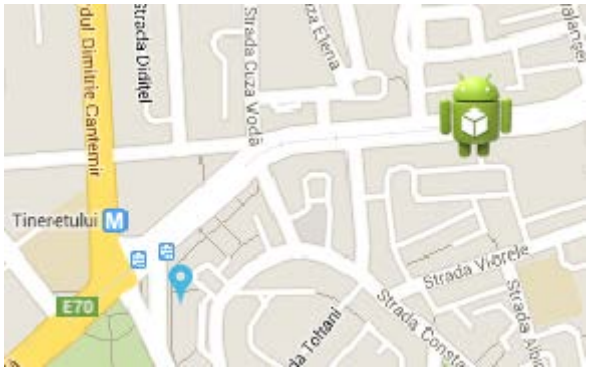


Fig. 6.4 – Locaţie curentă şi locaţii taskuri

- Pentru menţionarea dispozitivelor, utilizatorul trebuie să apese butonul “**Devices**” şi să selecteze din lista afişată ulterior dispozitivele necesare :

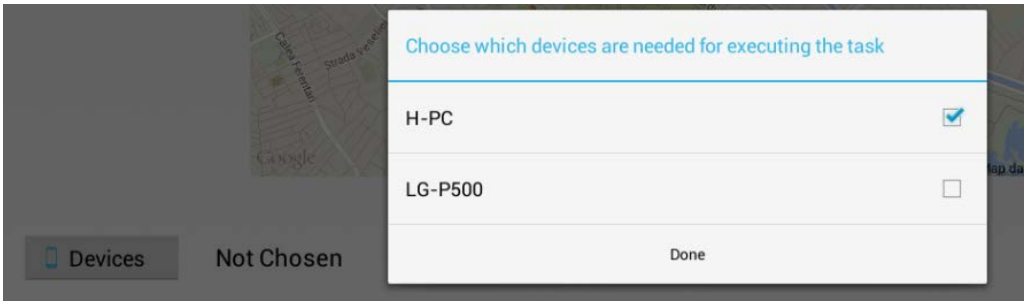


Fig. 6.5 – Alegere dispozitive

- Pentru menţionarea persoanelor necesare execuţiei taskului, utilizatorul trebuie să apese pe butonul “**People**” şi să selecteze din lista afişată persoanele necesare :

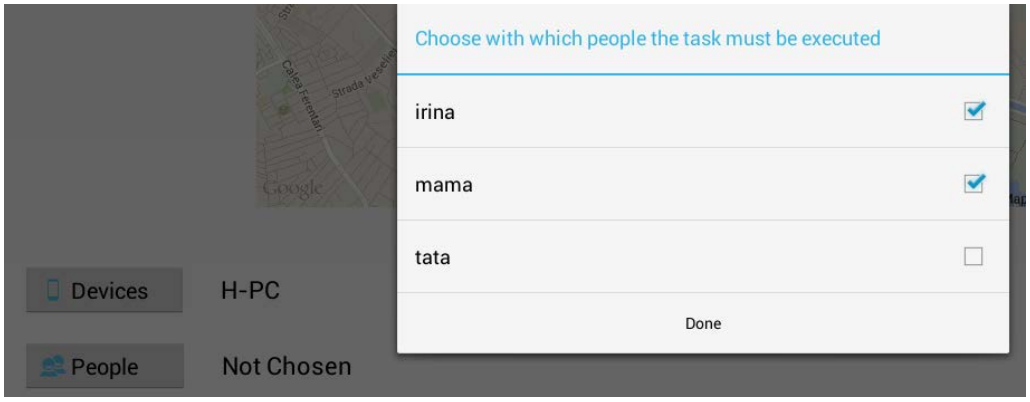


Fig. 6.6 – Alegere persoane

- Pentru specificarea priorităţii taskului, utilizatorul trebuie să apese pe câmpul din dreapta a textului “Priority”. Efectul acestei acţiuni este afişarea unei liste cu valori (Minor, Average, Major, Critical, Unknown) din care utilizatorul trebuie să aleagă nivelul priorităţii. Valoarea implicită a priorităţii este **Minor** (vezi figura 6.7)

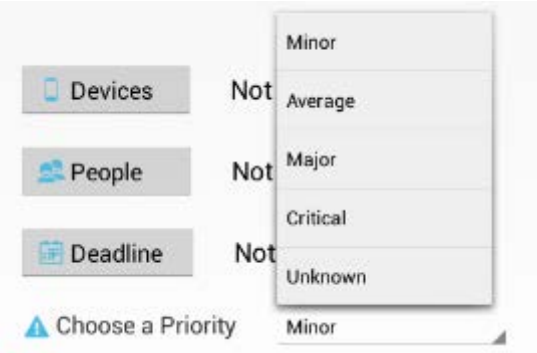


Fig. 6.7 – Specificarea priorităţii taskului

- Pentru specificarea duratei, utilizatorul trebuie să apese pe câmpul din dreapta textului “Choose a duration”. Efectul acestei acţiuni este afişarea unei liste cu valori din care utilizatorul trebuie sa aleagă durata dorită (vezi figura 6.8).Valoarea implicită a duratei este **Unknown**.

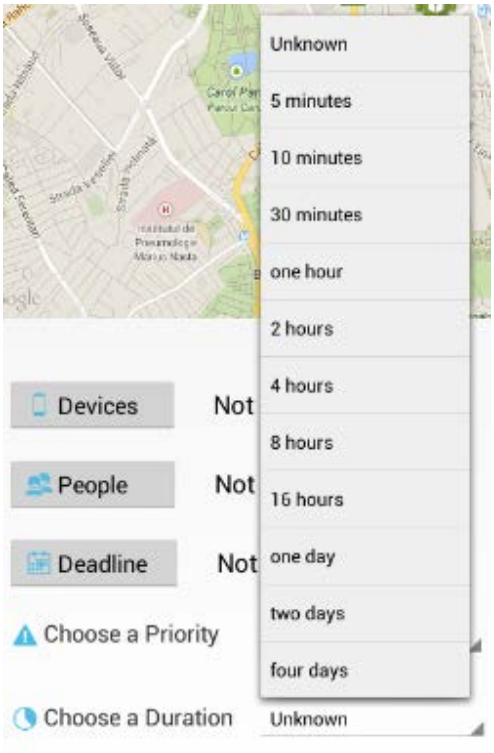


Fig. 5.8 – Specificarea duratei taskului

Pentru salvarea datelor, utilizatorul trebuie să apese butonul “Save Task” . Dacă informaţiile introduse sunt corecte, este afişat mesajul “**The data has been inserted**”. Dacă datele completate sunt insuficiente utilizatorul este avertizat în momentul încercarii salvării datelor.

Fig. 6.9 – Salvare task şi mesaje aferente

6.3 Vizualizarea si modificarea taskurilor adaugate

Pentru modificarea sau ştergerea unui task, utilizatorul trebuie să apese pe butonul “**View all tasks**” din meniul principal. În fereastra apărută, utilizatorul poate vizualiza toate taskurile adăugate şi încă neexecutate cât şi datele completate pentru fiecare în fereastra “**Add Task**” .Informaţiile taskurilor sunt delimitate de o linie albastră orizontală (vezi figura 6.10)

Fig. 6.10 – Detalii taskuri care nu au fost încă executate

- Titlul este scris cu culoarea albastră şi încadrat într-un dreptunghi de culoare deschisă.
- Informaţiile privind contextul în care poate fi executat taskul sunt afişate imediat sub titlu :

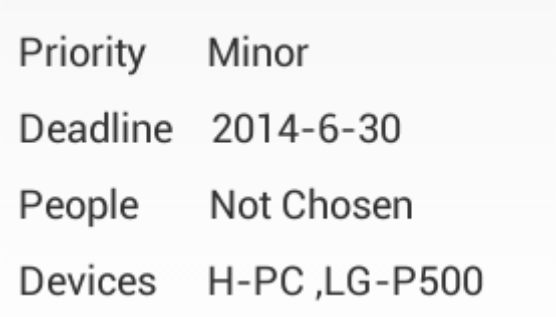


Fig. 6.11 – Detalii taskuri

- Utilizatorul poate şterge sau modifica taskul folosind butoanele “**ERASE TASK**” sau “**Modify Task**“ (vezi figura 6.10)
  - Apăsarea butonului “**Erase task**” are ca efect ştergerea datelor aferente taskului respectiv din interfaţa grafică.
  - Apăsarea butonului “**Modify task**” va conduce la deschiderea ferestrei “**AddTask**” cu câmpurile deja completate ale taskului, în vederea modificării acestora.

- Utilizatorul este informat în cazul în care nu sunt taskuri de executat, în fereastră fiind afişat următorul mesaj:

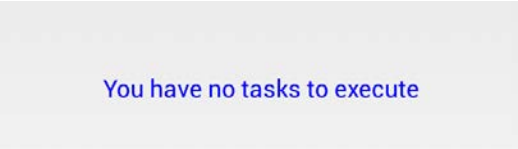


Fig. 6.12 – Mesaj lipsă taskuri de executat

6.4 Setarea programului fix

Pentru setarea programului fix al utilizatorului, utilizatorul trebuie să apese pe butonul " **Set fixed Schedule**" din meniul principal. În fereastra apărută, utilizatorul poate vizualiza programul fix pentru o zi. (vezi figura 6.13)

Fig. 6.13 – Interfaţă program fix

- Trecerea de la programul unei zile la programul zilei următoare se realizează prin utilizarea sagetilor care încadrează numele zilei curente.
- Intervalul fiecărui task dintr-o zi este reprezentat de ora de începere a taskului şi ora de terminare a taskului.

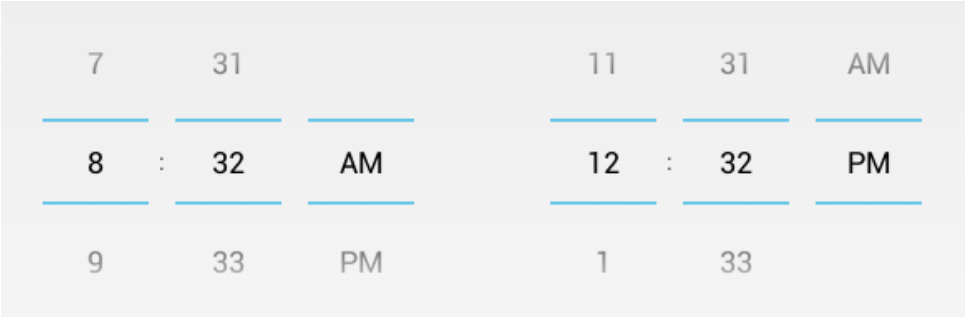


Fig. 6.14 – Interval execuţie task

- Pentru adăugarea / ştergerea unui task din baza de date, sunt utilizate butoanele "**Add Task**" respectiv "**ERASE TASK**" din dreptul intervalului taskului respective {vezi figura 6.13)

6.5 Vizualizarea taskurilor ce pot fi executate în contextul curent

Pentru vizualizarea taskurilor care pot fi executate în contextul curent, utilizatorul trebuie sa apese butonul "**View compatible tasks**" din meniul principal. Modul de afişare a taskurilor compatibile este similar cu cel al tuturor taskurilor (vezi figura 6.15).

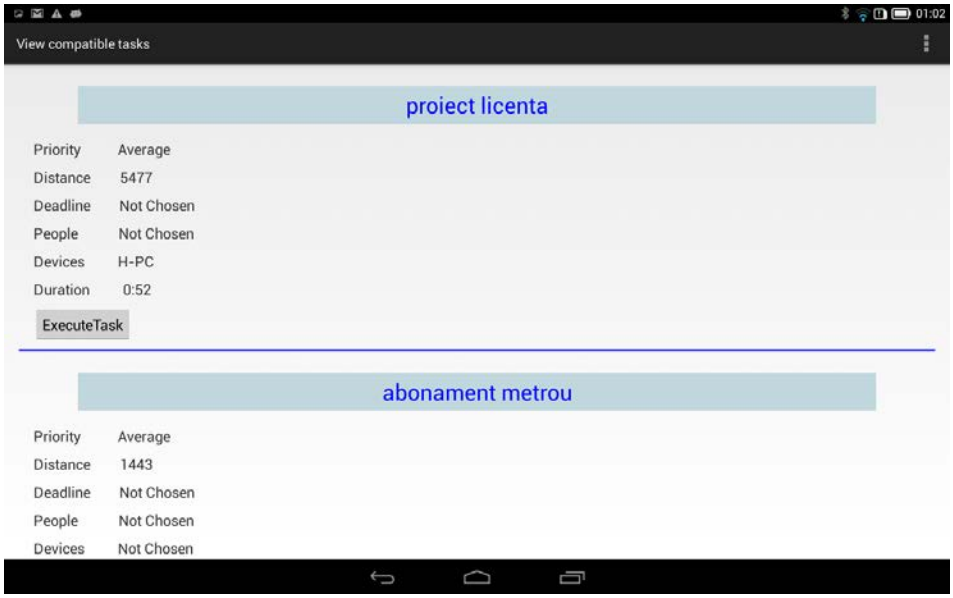


Fig. 6.15 – Interfaţă taskuri posibil de executat

- pentru a informa aplicaţia ce task este ales pentru a fi executat în momentul actual, utilizatorul trebuie sa apese butonul "**Execute task**"

6.6 Detectarea dispozitivelor

Pentru vizualizarea dispozitivelor detectate cu tehnologia Bluetooth, utilizatorul trebuie să apese butonul "**Detect devices**" din meniul principal. Informaţiile despre dispozitivele detectate sunt separate de o linie albastră (vezi figura 6.16).

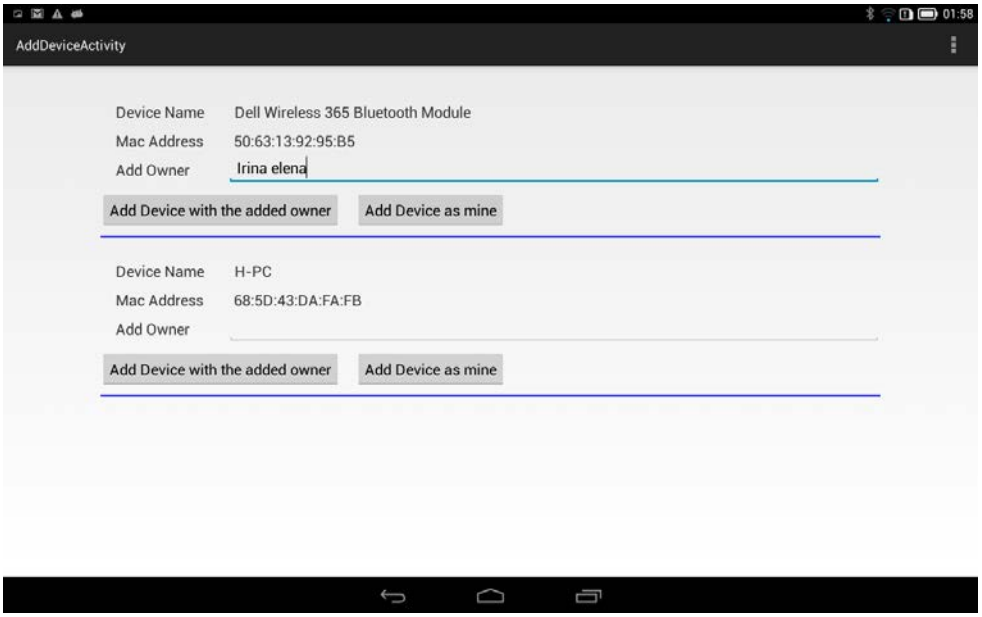


Fig. 6.16 – Interfaţă detectare dispozitive

- Datele afişate despre fiecare dispozitiv sunt : numele setat de proprietatul acestuia şi adresa Mac
- Pentru adăugarea dispozitivului în baza de date ca aparţinând utilizatorului, acesta trebuie sa apese butonul “**Add Device as mine**”. Orice este scris în câmpul din dreptul textului “**Add owner**” este omis.
- Pentru adăugarea dispozitivului in baza de date ca aparţinând altei persoane, utilizatorul trebuie să specifice în câmpul din dreptul textului “Add owner” numele acestuia si să apese pe butonul “**Add Device with the added owner**”

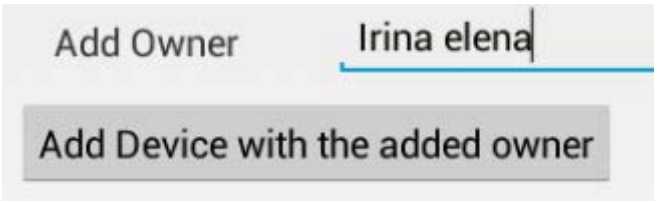


Fig. 6.17 – Adăugare înregistrare task în baza de date

- După ce este adăugat în baza de date, interfaţa este reactualizată şi nu mai conţine dispozitivul respectiv.

6.7 **Vizualizarea unui program pentru o zi sugerat de aplicaţie**

Pentru vizualizarea programului recomandat de către entitatea inteligentă, utilizatorul trebuie să apese pe butonul "**Suggest Schedule**" din meniul principal.

- Pentru recomandarea programului, utilizatorul trebuie să seteze intervalul orar dorit şi să apese butonul "**Calculate Schedule**" (vezi figura 6.18).

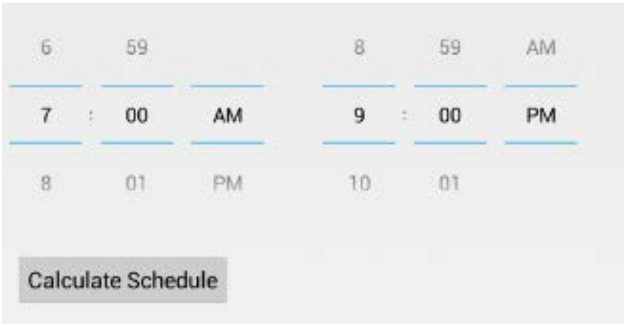


Fig. 6.18 - stabilire interval

- În cazul în care utilizatorul are în aceeasi zi un program fix, interfaţa este schimbata având si opţiunea de a alege un interval prin butonul "**Show next Interval**", care setează automat următorul interval care nu se intercalează cu programul fix. (vezi figura 6.19).

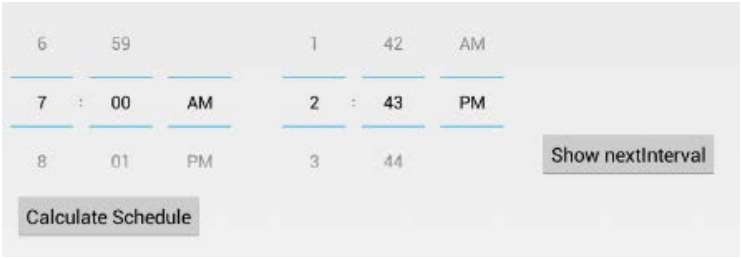


Fig. 6.19 – stabilire interval între programe

- După apăsarea butonului "**Calculate Schedule**", se afişează programul sugerat(vezi figura 6.20):

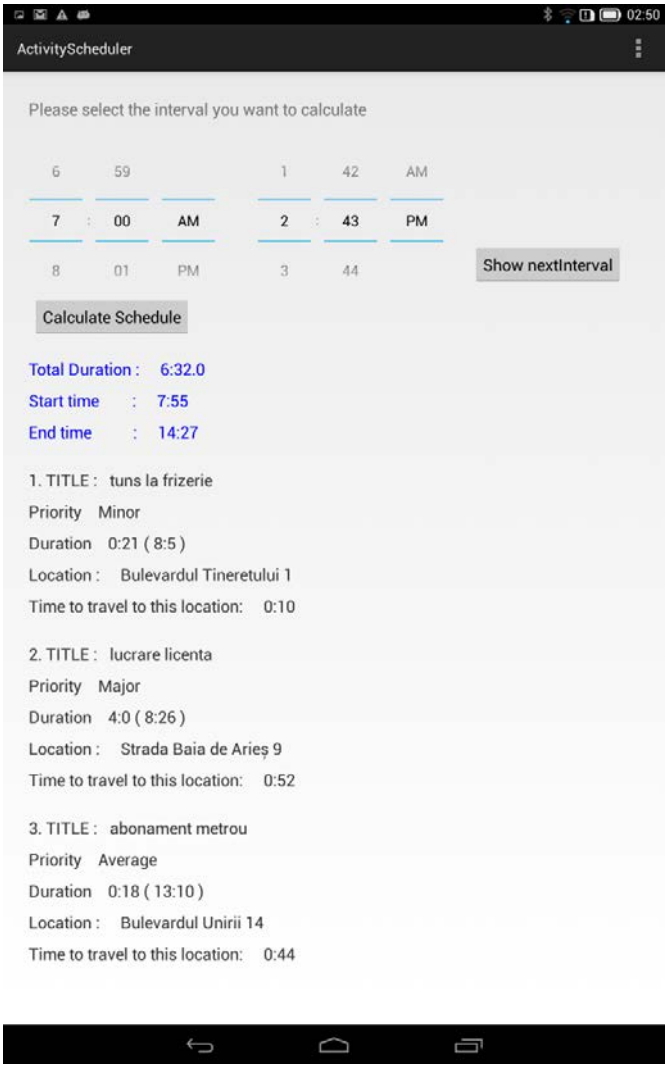


Fig. 6.20 – interfaţă program sugerat

- Utilizatorul este informat cu privire la momentul în care trebuie să plece pentru executarea taskurilor din poziţia curenta, ora estimată la care s-ar intoarce în acelasi loc şi durata totală

6.8 Afişarea taskului current

Pentru vizualizarea taskului curent pe care trebuie să îl execute, utilizatorul trebuie să apese butonul "**View current task**" din meniul principal.

- este afişat titlul taskului, timpul rămas până la execuţia lui si date privind contextul în care se desfăşoară taskul (vezi figura 6.21):

Fig. 6.21 – interfaţă task curent de executat

- sunt puse la dispoziţia utilizatorului butoane pentru gestionarea timpului. În cazul in care durata a fost estimată greşit, utilizatorul poate apasa pe butonul “**Add 10 minutes**” pentru adăugarea a 10 minute, sau “**Add one hour**” pentru adaugarea unei ore la durata estimată.
- dacă utilizatorul apasă butonul “**Task finished**”, taskul este considerat a fi terminat, iar timpul rămas este scăzut din durata totală.

7. Concluzii

Task Scheduler este o o aplicaţie implementată pe Android, care dispune de multe functionalităţi inteligente şi care oferă o interfaţă prietenoasă utilizatorului.

Interfaţa prietenoasă facilitează comunicarea dintre dispozitiv şi utilizator, fiind redus considerabil numărul de paşi prin care utilizatorul obţine sau introduce informaţiile necesare dispozitivului. De asemenea, interfaţa grafică a unei activităţi are componentele grafice dispuse într-un mod în care utilizatorul percepe cât mai rapid rolul activităţii şi informaţia afişată.

În ceea ce priveşte funcţionalităţile, acestea fac ca aplicaţia să asiste utilizatorul în a-şi organiza programul, prin recomandarea unei ordini în care să execute taskurile zilnice astfel încât să economisească cât mai mult timp posibil. Mai mult decât atât, vizualizarea taskurilor posibile de executat în contextul curent facilitează sarcina utilizatorului cu atât mai mult cu cât programul acestuia este foarte incărcat şi prin urmare îi este dificil să memoreze toate taskurile sau să decidă care taskuri pot fi executate.

Modul în care sunt utilizate tehnologiile şi API-urile oferite de sistemul de operare Android defineşte aplicaţia ca fiind un sistem inteligent ambiental. Implementarea aplicaţiei nu ar fi fost posibilă fără tehnologia Bluetooth pentru determinarea dispozitivelor din proximitatea utilizatorului şi fără GPS pentru determinarea poziţiei curente a utilizatorului.

Una dintre viitoarele funcţionalităţi ale aplicaţiei este partajarea informaţiilor pe un server. În acest fel, aplicaţia ar dispune de informaţiile mai multor utilizatori pentru clusterizare şi nu s-ar mai baza doar pe informaţiile locale ale utilizatorului, grăbind astfel procesul de obţinere a datelor necesare algoritmului K-Means. Un alt aspect pozitiv al acestui lucru este clusterizarea pe server, astfel că aplicaţia nu ar mai suferi de latenţa acestui proces care momentan este implementat local. Provocarea în partajarea datelor pe un server ar fi păstrarea confidenţialităţii datelor utilizatorilor.

Altă funcţionalitate priveşte calcularea programului utilizatorului. Momentan, acest calcul ia în considerare ca execuţie a unui task doar o posibilă locaţie a acestuia. În cazul în care taskul s-ar putea executa în mai multe locaţii, durata taskului poate sa difere considerabil şi astfel aplicaţia ar putea recomanda un program mai eficient decât cel pe care îl recomandă în acest moment. Mai mult decât atât, aplicaţia va trebui sa ia în considerare granularitatea duratei taskului. Acesta nu va trebui să fie executat într-o etapa dacă este foarte complex ci va fi executat în mai multe etape pe parcursul mai multor zile.

Nu în ultimul rând, aplicaţia va dispune de un sistem de notificare prin care sa înştiinţeze utilizatorul când sunt îndeplinite condiţiile pentru a putea fi executat un task. De asemenea, utilizatorul ar putea să transmită într-un mod mai facil aplicaţiei cererile sale prin intermediul microfonului.

Drept concluzie, aplicaţia oferă funcţionalitaţile necesare pentru a putea fi utilizată şi pentru a asista utilizatorul în viaţa de zi cu zi. Viitoare functionalităţi vor optimiza această asistare.

B I B L I O G R A F I E

[1]http://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte

[2] Mark L. Murphy , The busy Coder's Guide to Android Development – versiunea 4.2, 2012, editura CommonsWare, pg 3

[3] http://developer.android.com/training/basics/activity-lifecycle

[4]http://developer.android.com/guide/topics/manifest/manifest-intro.html

[5] http://developer.android.com/training/basics/firstapp/building-ui.html

[6]http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html

[7] http://developer.android.com/guide/topics/connectivity/bluetooth.html

[8] http://examples.javacodegeeks.com/android/core/content/android-sharedpreferences-example/

[9] http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/

[10] Stuart Russell, Peter Norvig ,Artificial Intelligence A Modern Approach (Third Edition) – capitolul 18

[11] http://en.wikipedia.org/wiki/K-means_clustering

[12] http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set

[13] http://web.stanford.edu/~hastie/Papers/gap.pdf : Estimating the number of clusters in a data set via the gap statistic – Robert Tibshirani, Guenther Walter, Trevor Hastie

[14] http://en.wikipedia.org/wiki/Genetic_algorithm

[15] Ducatel, Ken and Bogdanowicz, Marc and Scapolo, Fabiana and Leijten, Jos and Burgelman, Jean-Claude, Scenarios for ambient intelligence in 2010, 2001
ftp://ftp.cordis.europa.eu/pub/ist/docs/istagscenarios2010.pdf

[16]http://en.wikipedia.org/wiki/Ambient_intelligence

[17]http://software.ucv.ro/~cmihaescu/ro/teaching/ACA/docs/PCV.pdf

[18]http://en.wikipedia.org/wiki/Levenshtein_distance