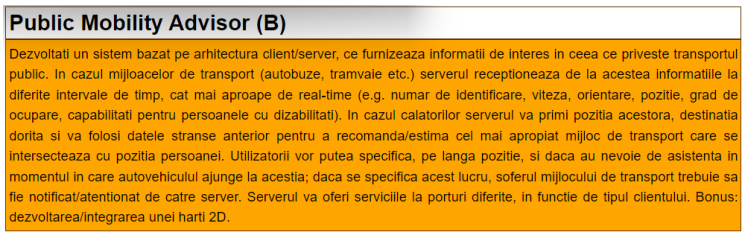
Public Mobility Advisor

Andra Popoiu-Ștefania

Facultatea de Informatică, Universitatea Alexandru Ioan Cuza

1 Introducere



Proiectul Public Mobility Advisor se bazează pe comunicarea client/server, prin intermediul căreia se vor oferi diferite informații(în funcție de tipul clientului - Călător sau Șofer) legate de mijloacele de transport din sistem. Călătorii pot primi informații legate de ruta specificată și pot ocupa un loc în anumite mijloace de transport, iar șoferii sunt notificați dacă un călător are nevoie de asistență. Serverul se va ocupa cu primirea, actualizarea și sincronizarea informațiilor strânse de la clienți.

2 Tehnologii Aplicate

Proiectul a fost implementat folosind la nivelul de transport protocolul de Control al Transmisiei. TCP ul este o alegere potrivită pentru aplicația care trebuie dezvoltată deoarece este un protocol fiabil și orientat conexiune, care asigură transmiterea ordonată și securizată a datelor într-o rețea fără pierdere de informație. TCP asigură faptul că ambele capete (clientul și serverul) sunt pregătite de comunicare, dar pe lângă asta asigură și comunicarea full-duplex, controlul fluxului și verificarea erorilor. Acesta are ca scop să ofere o calitate maximă a serviciului, motiv pentru care a fost ales pentru acest proiect.

3 Structura Aplicației

În implementarea aplicației este folosit un server TCP concurent. La acesta se pot conecta mai mulți clienți de tip diferit. Aceștia vor trimite anumite informații,

la care se va adăuga un prefix specific stadiului în care se află (prefix ce specifică tipul clientului și starea lui de logare), în funcție de care pot avea accesul restricționat sau nu.

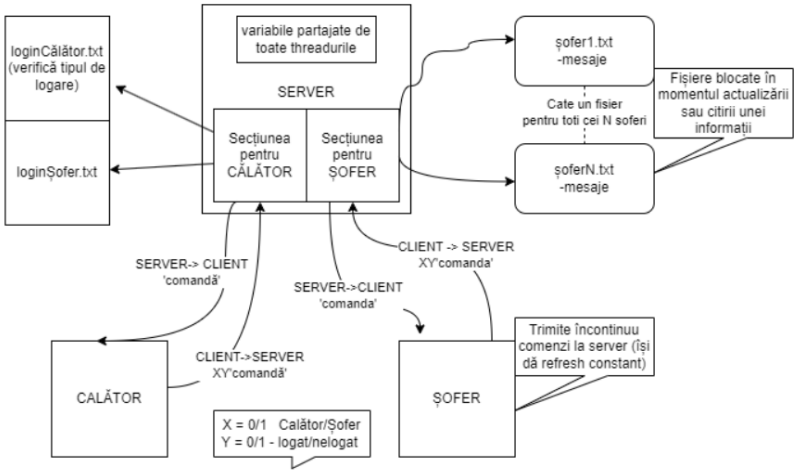


Fig. 1. Diagramă a aplicației

Inițial clientul este neutru, urmând ca după logare să îi fie atribuit un rol.

Șoferii au următoarele comenzi:
- mesaje => în momentul în care clientul accesează această comandă o fereastră nouă va fi deschisă iar în aceasta vor fi afișate și actualizate (la anumite intervale de timp) mesaje referitoare la clienții care au rezervat un loc în mijlocul de transport condus de șoferul actual
- logout => clientul intră înapoi în starea neutră

Călătorii au următoarele comenzi:
- poziție actuală => stația în care se află călatorul.
- destinație=> stația în care călatorul dorește să ajungă.
- determină ruta => în acest caz se vor determina mai multe rute (opțiuni) din care să se aleagă (împreună cu un timp total estimat), iar călătorul va putea alege și daca are sau nu nevoie de asistență în momentul ajungerii mijlocului de transport în stație.
- logout=> clientul intră înapoi în starea neutră

- informații => oferă informații în timp real despre mijloacele de transport

Călătorii și Șoferii vor partaja și modifica fișierele care conțin informații despre mesajele primite de șoferi, dar și variabile legate de capacitatea unui autobuz și gradul de ocupare a acestuia.

Vom codifica traseele sub forma unui graf, ca în imaginea de mai jos, pentru a ne ajuta la determinarea rutelor posibile.

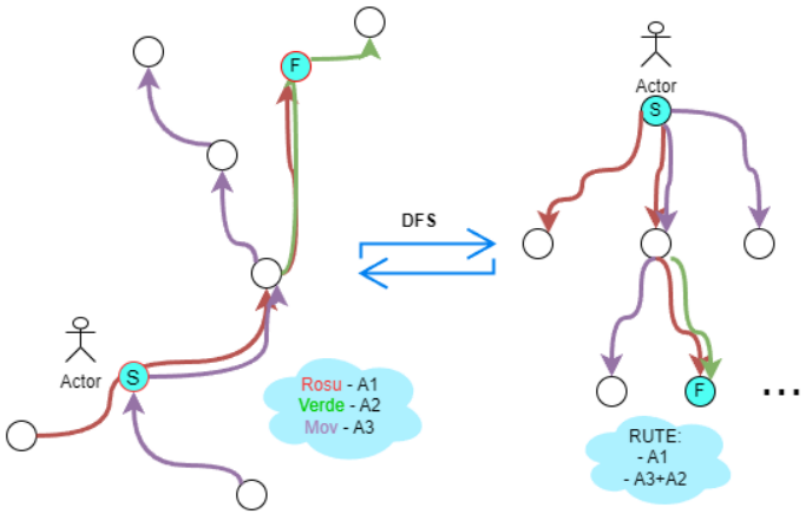


Fig. 2. Diagramă înainte și după parcurgerea în adâncime a traseului

După această codificare se va folosi backtracking pentru a determina pâna la 3 opțiuni de trasee.

4 Aspecte de Implementare

Una din problemele proiectului este calcularea timpului total de a ajunge până într-o stație, problemă rezolvată de codul de mai jos, care se bazeaza pe direcția în timp real a mijlocului de transport.

```
if( directie == 1)
{
```

```
    int ruta_autobuzului_actual = autobuze[x][0];
    int a_cata_statie_e_din_ruta_actuala = minute_ramase / 3 + 1;
    int numele_statiei_de_care_tocmai_a_trecut_atuobuzul = rute[ruta_autobuzului

    int ok = 0;
    int timp_de_ajungere_in_statia_st = 3 - minute_ramase % 3; /* timpul dintre
    for(i = a_cata_statie_e_din_ruta_actuala + 1; i <= rute[ruta_autobuzului_act
        /*...*/
    timp_de_ajungere_in_statia_st -= 3;
    printf("Timp dupa prima parcurgere: %d\nok: %d\n", timp_de_ajungere_in_stati

    if(ok == 0) /* daca nu a gasit statia pana ajunge in capat */
      for(i = 1; i < rute[ruta_autobuzului_actual][0]; ++ i)
        if(rute[ruta_autobuzului_actual][i] == st)
        {
          if(nextst == rute[ruta_autobuzului_actual][i + 1])
            ok = 2; /* trebuie sa ajungs autobuzul in capat, sa se intoarca, sa
          break;
        }

    if(ok == 0) /* nu am trecut prin statia cautata */
    {
      for(i = rute[ruta_autobuzului_actual][0]; i > 0; -- i) /* trec prin toate
        if(rute[ruta_autobuzului_actual][i] == st){
          return timp_de_ajungere_in_statia_st;
          break;
        }
        else timp_de_ajungere_in_statia_st += 3;
    }
    else if(ok == 2)
    {
      timp_de_ajungere_in_statia_st += (rute[ruta_autobuzului_actual][0] - 1) *
      /*...*/
    }
    printf("Ca sa ajung in statia vruta[%d] s au facut : %d minute\n", st, timp
  }
```

Pentru ca proiectul să fie posibil, trebuie găsită o modalitate de a determina toate rutele posibile între poziția actuală și destinația dorită. Pentru determinarea acestor opțiuni dar și pentru a le îmbunătăți am folosit o funcție bazată pe backtracking. Această funcție merge din stație în stație și verifică dacă mijlocul de transport luat până în stația actuală își va continua ruta. În caz afirmativ nu se mai tratează alte cazuri (pentru că nu sunt optime) și se continuă cu ruta actuală. Dacă mijlocul de transport nu își poate continua ruta atunci se vor căuta alte autobuze care trec prin stația respectivă. Călătorul va fi informat de toate opțiunile și de timpul de așteptare din fiecare stație în care

are loc o schimbare de automobil.

```
void Backtracking(int x, int drum[], int n , int ruta_din_care_vin)
{
  if(x == n) {
    m ++;
    vector_rute[m][0] = ruta_posibila[0];
    for(int i = 1; i <= n; ++ i)
    {
      vector_rute[m][i] = ruta_posibila[i];
    }
    return;
  }

  if( rute_accesibile_din_statii[drum[x+1]][ruta_din_care_vin] == 1 ) /* incerc
  {
    ruta_posibila[x + 1] = ruta_din_care_vin;
    Backtracking(x + 1, drum, n, ruta_din_care_vin);
  }
  else
  for(int i = 1; i <= 3; ++ i) /* incerc si alte rute */
      if( i != ruta_din_care_vin && rute_accesibile_din_statii[drum[x+1]][i] ==
      {
        ruta_posibila[x + 1] = i;
        ruta_posibila[0] ++;
        Backtracking(x + 1, drum, n, i);
        ruta_posibila[0] --;
      }

}
```

Un atribut important al acestui proiect este faptul că clienții sunt tratați diferit în funcție de tipul lor. Pentru a determina tipul fiecărui client (în main, unde clientul este neutru) vom avea o parte de testare în care vom diferenția clienții în funcție de 2 variabile. Prima variabila este "logat". Aceasta verifică dacă vom continua în clientul neutru sau dacă i-a fost atribuit un tip. A doua variabilă este "tip client". Aceasta verifică tipul clientului și îl trimite în funcția specifică acestuia. În cazul în care vreun client dorește să se delogeze, pentru a continua aplicația, funcțiile specifice se vor termina, iar clientul se va întoarce în main (devine iar client neutru).

```
/*in main, practic in clientul neutru*/
if(logat == 0) printf("\n[client]-Comenzi-permise:-login:name\n");
    else if(logat == 1)
            if(tip_client == 0) { Client(); printf("\n[client]-Comenzi-permise:-
```

```
else { Sofer(); printf("\n[client]-Comenzi-permise:-login:name\n");
```

5 Concluzii

Cu toate că abordatea propusă acoperă cele mai importante aspecte legate de proiectul dat, unele îmbunătățiri ar putea fi aduse:
- introducerea posibilității de a rezerva mai multe locuri deodată
- posibilitatea de a vizualiza istoricul călătoriilor
- posibilitatea de a anula comanda

6 Referințe Bibliografice

https://profs.info.uaic.ro/ computernetworks/
https://sites.google.com/view/fii-lab-retele-de-calculatoare/