Cuprins

Listă de figuri

Listă de tabele

Listă de acronime

ANSI American National Standards Institute. 10

API Application Programming Interface. 28

DLL Dynamic-link library. 6, 17, 29

IP Internet Protocol. 9

TCP Transmission Control Protocol. 9

XML Extensible Markup Language. 8

Capitolul 1
INTRODUCERE

1.1 Tema de proiectare

1.2 Stadiul actual al problemei abordate

În momentul actual există diverse companii precum Google, General Motors sau Volkswagen care dezvoltă propriile sale soluții de navigare predictivă.

Google furnizează alerte predictive de mai mult de 5 ani ca parte a funcționalității Google Now de pe dispozitivele Android.

La un atelier de inovare de la sfârșitul anului 2014, Volkswagen și-a prezentat la sediul său din Wolfsburg, Germania, propria sa soluție in-car de oferire de sugestii de rute alternative chiar și în cazul în care utilizatorii nu folosesc sistemul de navigație pentru o destinație uzuală.

Alți producători de automobile, precum General Motors, au testat soluții pentru autovehiculele electrice de tip plug-in, cum ar fi modelul Chevrolet Volt, care va păstra automat alimentarea electrică pentru ultima porțiune a unei rute dacă destinație se află într-o zonă rezidențială, sau chiar să folosească o parte din bateria de rezervă pentru cazul în care sistemul prezice că autovehiculul va fi alimentat în curând.

Se poate spune deci, că fiecare producător preferă să-și dezvolte propriile soluții ce au un scop limitat și strict aplicat nevoilor lor, din motive clare de marketing și vânzare.

1.3 Scopul şi obiectivele proiectului

Obiectivul principal este să se dezvolte un modul de navigare ce are ca scop estimarea rutelor posibile ale unui autovehicul.

Acest modul va fi împachetat sub forma unei biblioteci cu legare dinamică (DLL) și va fi destinat utilizării de către orice aplicație de navigație.

Având ca scop obiectivul principal, s-au definit și câteva obiective intermediare:

- Deciderea asupra funcționalităților oferite de modul
- Deciderea asupra modului de stocare al datelor
- Împărțirea pe unități software de lucru

- Crearea diagramei de arhitectură
- Finalizarea scrierii codului
- Testarea modulului realizat
- Evindențierea posibilelor erori și oferirea de soluții

1.4 Domeniul de aplicabilitate

Domeniul în care proiectul dezvoltat ar avea cea mai mare aplicabilitate este industria automobilistică, mai specific în cadrul aplicațiilor de navigare. În momentul de față acest domeniu este unul în plină ascensiune, tocmai de aceea se caută constant noi modalități de a satisface nevoile utilizatorului, da a rezolva cerințele noi apărute, de a ușura condusul unui autovehicul și chiar de a face un pas înainte către dezvoltarea vehiculelor autonome.

1.5 Structura pe capitole

1. Introducere
   Primul capitol începe cu o introducere în domeniul automobilisticii, prezentându-se câteva utilizări ale principiului și actualitatea sa în ziua de azi.
2. Baza de date
   În acest capitol sunt prezentați factorii de decizie cu cea mai mare semnificație asupra tipului de date de baze ales. Totodată, capitolul 2 face și o scurtă introducere în cadrul SQLite pentru o mai bună înțelegere a scopului și structurii acesteia în cadrul proiectului.
3. Arhitectura software
   Capitolul 3 descrie structura software folosită în scopul dezvoltării algoritmului de predicție, explicând în detaliu legatura dintre unitățile software și rolul pe care acestea îl îndeplinesc.
4. Descrierea algoritmilor
   Acest capitol se axează pe logica din spatele funcțiilor principale îndeplinite de fiecare unitate software în parte.
5. Gestionarea erorilor
   Scopul capitolului 5 este de a face cunoscute erorile ce sunt predispuse să apară în urma utilizării modulului de predicție, dar și a unor eventuale soluții.
6. Concluzii și dezvoltări ulterioare
   Ultimul capitol al proiectului concluzionează dezvoltarea și rezultatele obținute.
   În urma realizării acestui proiect, a studierii necesităților și așteptărilor utilizatorului comun de la un sistem de navigație, dar și a analizei soluțiilor deja existente se prezintă posibile direcții de dezvoltare.

Capitolul 2
BAZA DE DATE

2.1 Alegerea tipului de bază de date

Nevoia modulului de a stoca şi de a accesa datele stocate anterior a dus la realizarea unui mic studiu în vederea alegerii tipului de bază de date cel mai potrivit.

2.1.1 Factori de decizie

- Consistenţa datelor stocate
- Utilizarea RAM-ului
- Timp de accesare la pornirea aplicaţiei
- Timp de accesare în cadrul aplicaţiei
- Spaţiul ocupat pe disc
- Compatibilitate cu versiunile anterioare

2.1.2 Soluţii propuse

În următorul tabel, se presupune ca pentru fişierele binare, XML şi JSON este necesară încărcarea datelor la pornirea aplicaţiei. SQLite oferă însă soluţii de căutare inteligente, nefiind necesară încărcarea tuturor datelor la pornirea aplicaţiei.

2.1.3 Soluția aleasă

S-a decis folosirea SQLite ca format pentru baza de date deoarece îndeplinea toate criteriile specificate.

TABELA 2.1: Compararea principalelor metode de stocare a datelor pe baza factorilor de influenţare

	Binar	**XML sau JSON**	**SQLite**	**Memorare în Cloud**
Consistenţa datelor stocate	Nu	Nu	Da	Da
Utilizarea RAM-ului	Ridicat	Scăzut	Mediu	Mediu
Timp de accesare la pornirea aplicaţiei	Mediu	Ridicat	Scăzut	Scăzut
Timp de accesare în cadrul aplicaţiei	Scăzut	Scăzut	Mediu	Ridicat
Spaţiul ocupat pe disc	Scăzut	Ridicat	Scăzut	Foarte scăzut
Compatibilitate cu versiunile anterioare	Nu	Da	Da	Nu

2.2 SQLite

SQLite reprezintă o bibliotecă software ce furnizează un sistem de gestiune a bazelor de date relaționale.

Denumirea de Lite (Ușor) este preluată în urma proprietăților sale: minimă necesitate de resurse, respectiv procesul ușor de setare și administrare a bazei de date.

SQLite are următoarele caracteristici notabile: independență, nu necesită un server, zero-configurare, tranzacțională.

2.2.1 Nu necesită server

În mod normal sistemele de gestiune a bazelor de date relaționale precum MySQL, PostgreSQL, ș.a.m.d necesită un server dedicat pentru operare. Aplicații ce doresc să acceseze baza de date de pe server sunt nevoie astfel sa folosească protocolul TCP /IP pentru a trimite cereri și a primi răspunsuri. Acest tip de arhitectură este una de tip client-server.

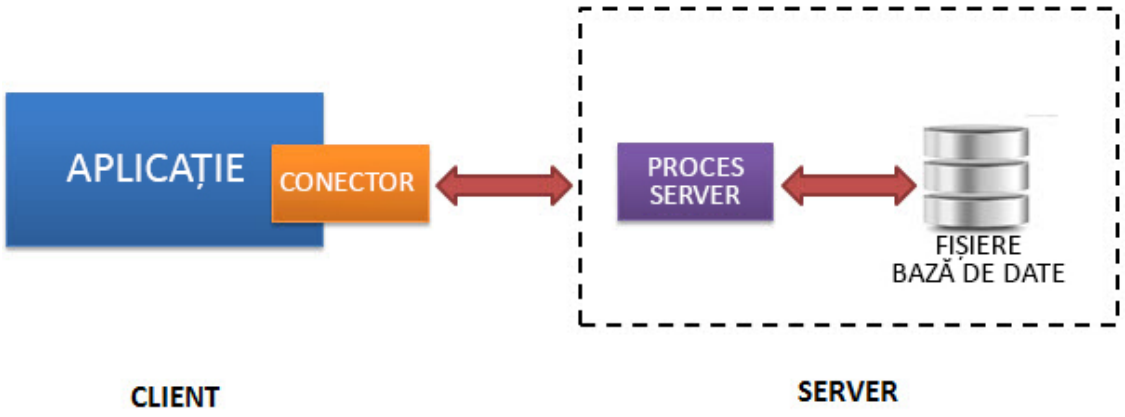

FIGURA 2.1: Arhitectura client-server a sistemelor de gestiune a bazelor de date relaționale

În figura de mai sus se poate observa arhictectura client/server unui sisteme de gestiune a bazelor de date relationale. SQLite nu funcționează respectând aceste principii, acesta neavând necesitatea folosirii unui server pentru a rula. Baza de date SQLite este întegrată în cadrul aplicației ce o accesează. Aplicația interacționează cu direct cu baza de date stocată local.

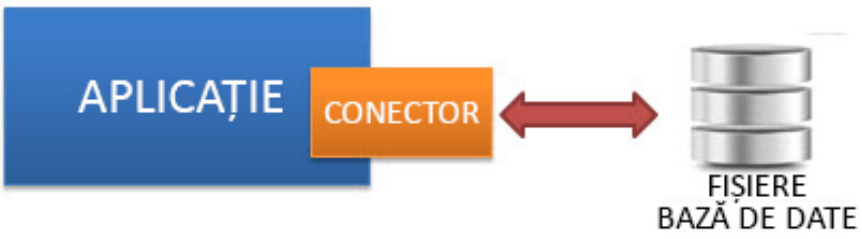

FIGURA 2.2: Arhitectura server-less SQLite

2.2.2 Independența

Proprietatea de independență se definește prin necesitatea minimă de susținere din partea sistemului de operare sau a unei biblioteci externe. Aceasta îi acordă SQLite-ului avantajul de a rula în orice tip de mediu, în mod deosebit dispozitivele embedded precum iPhone, Android, console de jocuri, ș.a.m.d.

SQLite este dezvoltat folosind ANSI-C. Codul sursă este disponibil sub forma unui fișier sqlite3.c și header-ul atribuit acestuia, sqlite3.h. Dacă se dorește dezvoltarea unei aplicații folosind SQLite nu este necesară decât simpla adaugare a acestor fișiere în proiectul respectiv și compilarea codului.

2.2.3 Zero-configurare

Datorită arhitecturii ce nu include server, nu este necesară nici o instalare a SQLite-ului anterior folosirii sale. Nu exită nici un process de server ce necesită configurare, pornire sau oprire.

În plus, SQLite nu folosește nici un fișier de configurare.

2.2.4 Tranzacțională

Toate tranzacțiile din cadrul SQLite sunt conforme ACID. Acest lucru înseamnă că toate interogările și schimbările sunt Atomice, Consistente, Izolate și Durabile.

În alte cuvinte, toate schimbările ce apar în urma unei tranzacțiii sunt fie realizate în totalitate, fie deloc, chiar și în situații neprevăzute precum închidere neașteptată a aplicației, probleme de alimentare, sau probleme datorate sistemului de operare.

2.2.5 Caracteristici specifice ale SQLite

SQLite folosește tipuri dinamice pentru tabele. Acest lucru înseamnă că se pot stoca orice valori în orice coloane, indiferent de tipul de tipul de dată.

De asemenea, SQLite oferă libertatea ca o singură conexiune să acceseze simultan mai multe fișiere ale bazei de date. Acestă caracteristică este foarte facilă mai ales atunci când dorim să unim tabele din baze de date diferite sau să copiem date între baze de date diferite folosind o singură comandă.

2.3 Structura bazei de date

Tabelele din figura de mai sus sunt folosite pentru realiza structura întregii baze de date.

Tabela meta este folosită la identificarea versiunii bazei de date. Acest lucru este necesar pentru a detecta compatibilitatea şi pentru a permite migrarea către o versiune mai recentă.

Datele înregistrate sunt separate în puncte de plecare, destinaţii, rute şi waypoint-uri. O rută este

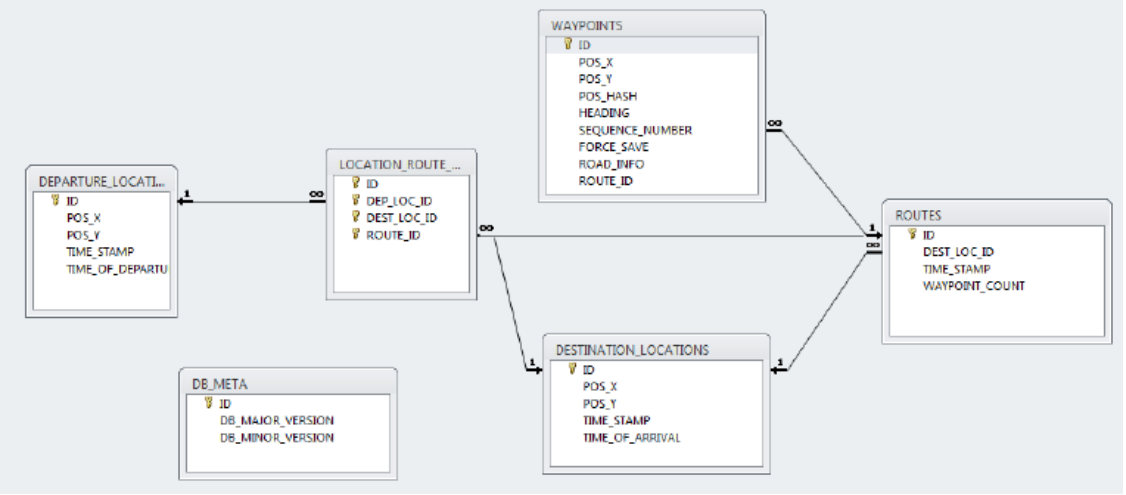

FIGURA 2.3: Structura tabelelor de date şi relaţiile dintre ele

întotdeauna formată din mai multe waypoint-uri, unul sau mai multe puncte de plecare şi una sau mai multe destinaţii. Ruta (waypoint-urile) sunt stocate numai o singură dată, în timp ce toate punctele de plecare şi destinaţiile sunt stocate. În acest fel, numărul de destinaţii poate influenţa probabilitatea rutei.

Accesul la date se face prin SQLite. Toate datele stocate pot fi atât citite cât şi modificate.

Capitolul 3
ARHITECTURA SOFTWARE

În acest capitol este prezentată partea structurală a modulului cât şi unităţile software din care acesta este format.

3.1 Limbajul C++

Limbajul de programare C++ reprezintă de fapt o vastă colecție de comenzi folosite pentru controlul computerelor, numite și cod.

C++ este un limbaj de programare orientat pe obiecte ce a fost dezvoltat ca o extensie a originalului limbaj C în anul 1980. Este considerat a avea un nivel de dificultate intermediar, deoarece acoperă atât funcții de nivel înalt cât și de nivel scăzut.

Potrivit Fundației C++ Standard, limbajul oferă un sitem de memorie sistemică și un calcul structural ce seamănă foarte mult cu programarea celor mai multe computere. Acesta este folosit la o multitudine de sarcini, de la extragerea datelor din bazele de date la afișarea graficii jocurilor video sau chiar la controlarea dispozitivelor electronice atașate computerului.

Deoarece poate fi folosit pe orice sistem de operare, C++ este un limbaj de programare universal ce este găsit pe majoritatea computerelor.

3.1.1 Obiectivele de proiectare

C++ a fost inițial creat de Bjarne Stroustrup în cadrul Laboratoarelor AT&T Bell, ca o cale de a depasi limitările limbajelor de programare deja existente, precum C și Simula.

C++ a fost creat cu scopul de a oferi flexibilitate, eficiență și o organizare structurală. În mod specific, mecanismele de încorporare abstractă au fost proiectate pentru a face față celor mai dificile și solicitante sarcini de programare.

C++ suportă abstractizarea datelor, programarea generică dar și programarea orientată pe obiecte. Deoarece c++ a fost gândit pentru a fi un limbaj de programare cu scop general, este foarte ușor de folosit și de utilizat.

3.1.2 Funcționalitate multiplă

C++ este foarte popular deoarece este extrem de funcțional. Este folosit pentru a crea sisteme de operare, drivere de dispozitiv și protocoale de rețea.

Este de asemenea utilizat pentru a dezvolta aplicații pentru baze de date, foi de calcul și procesare de text. Deoarece este un limbaj cu scop general, este potrivit pentru dezvoltarea oricărui tip de software. Acest lucru este posibil prin intermediul a peste 10 sisteme unice de implementare și sute de biblioteci, manuale și jurnale tehnice.

Deoarece C ++ permite programatorilor să creeze rapid programe în limite urgente de timp și spațiu, C ++ este ideal pentru manipulări hardware directe în constrângeri în timp real. Într-un astfel de cod, previzibilitatea performanței este cel puțin la fel de importantă ca și viteza brută.

Fiabilitatea C ++ a dus la favorizarea organizațiilor de tranzacționare, bancare, de asigurări, militare și de telecomunicații.

3.1.3 Avantaje

C ++ este un limbaj foarte portabil, ceea ce înseamnă că programatorii pot scrie programe indiferent de limitele hardware și de sistemul de operare. Ca rezultat, programatorii pot dezvolta un program inițial care este tradus sistematic pe diferite platforme.

Orice program care a fost dezvoltat în limbajul original C poate fi ușor mutat în C ++ fără modificări majore. Deoarece C ++ oferă flexibilitate, programatorii sunt capabili să creeze construcții puternice și să introducă noi obiecte conceptuale și aplicații abstracte.

Ca rezultat, C ++ permite programatorilor să controleze și să manipuleze resursele hardware pentru a produce programe de funcționare înalte. C ++ vine cu câteva dezavantaje, cum ar fi numeroase erori de securitate și funcționalitate slabă a dezvoltării web.

Limbajul de programare C ++ este astfel unul din limbajele cele mai utilizate și versatile pe care toți începătorii ar trebui să-l învețe.

3.2 Enterprise Architect

Sparx Systems Enterprise Architect este un instrument de modelare vizuală și de proiectare bazat pe OMG UML.

Platforma suportă: proiectarea și construirea de sisteme software, modelarea proceselor de afaceri, și modelarea domeniilor bazate pe industrie.

Este folosită de întreprinderi și de organizații nu numai pentru să modelarea arhitecturii sistemelor lor, ci și pentru a procesa implementarea acestor modele în întregul ciclu de viață al dezvoltării aplicațiilor.

Enterprise Architect este construit pe baza specificațiilor UML 2.

Utilizarea profilelor UML extinde capabilitățile de modelare iar validarea modelelor garantează integritatea.

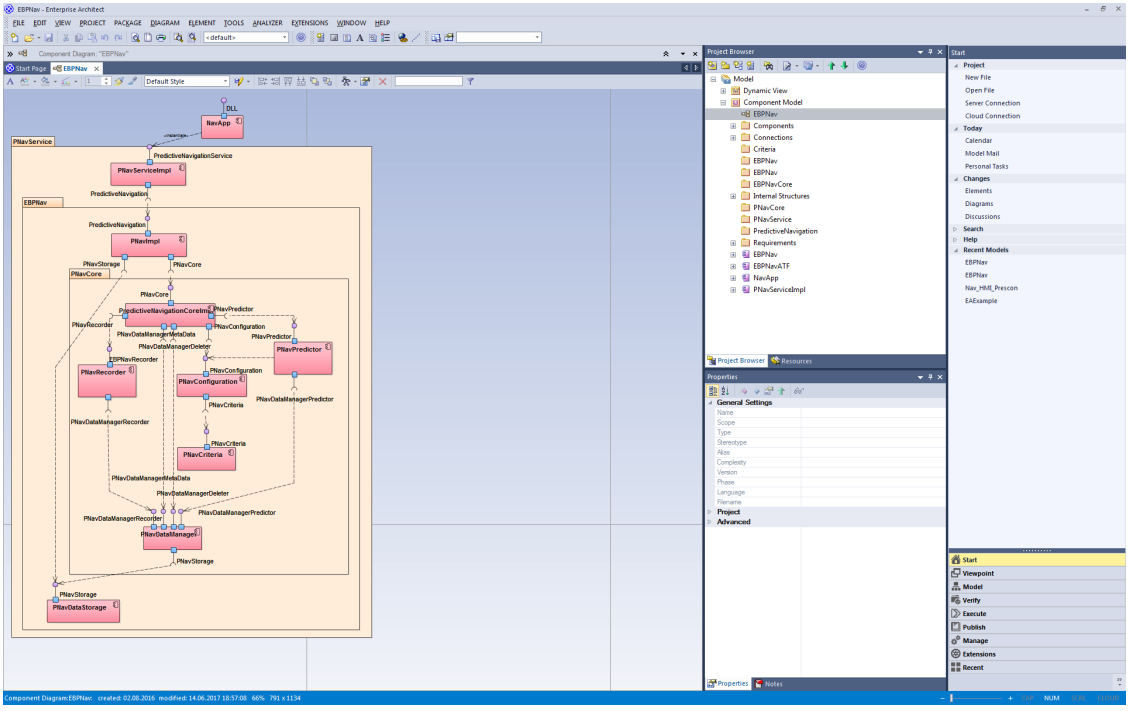

FIGURA 3.4: Captură de ecran din Enterprise Architect

3.2.1 Diagramele UML

UML este o modalitate de vizualizare a unui program software folosind o colecție de diagrame.

Notația a evoluat de la munca lui Grady Booch, James Rumbaugh, Ivar Jacobson și Rational Software Corporation pentru a fi folosită pentru design orientat pe obiecte, dar de atunci a fost extinsă pentru a acoperi o varietate mai largă de proiecte de inginerie software. Astăzi, UML este acceptat de către Grupul de Management al Obiectului(OMG) ca standard pentru modelarea dezvoltării de software.

Termenul de UML vine de la Unified Modeling Language (limbaj unificat de modelare). UML 2.0 a ajutat la extinderea specificației UML originale pentru a acoperi o parte mai mare a eforturilor de dezvoltare software, inclusiv practicile Agile.

Deși este folosit în mod obișnuit în ingineria software, este un limbaj bogat care poate fi folosit pentru a modela structurile de aplicații, comportamentul și chiar procesele de afaceri. Există 14 tipuri de diagrame UML care vă ajută să modelați aceste comportamente, ele putând fi împărțite în două categorii principale; Diagrame structurale și diagrame comportamentale.

- Diagramă tip clasă
- Diagramă tip componentă
- Diagramă de implementare

- Diagramă tip obiect
- Diagramă tip pachet
- Diagramă tip profil
- Diagramă tip structură compozită
- Diagramă tip scenariu
- Diagramă de activitate
- Diagramă tip stare mașină
- Diagramă de secvență
- Diagramă de comunicare
- Diagramă de interacțiune generală
- Diagramă de timp

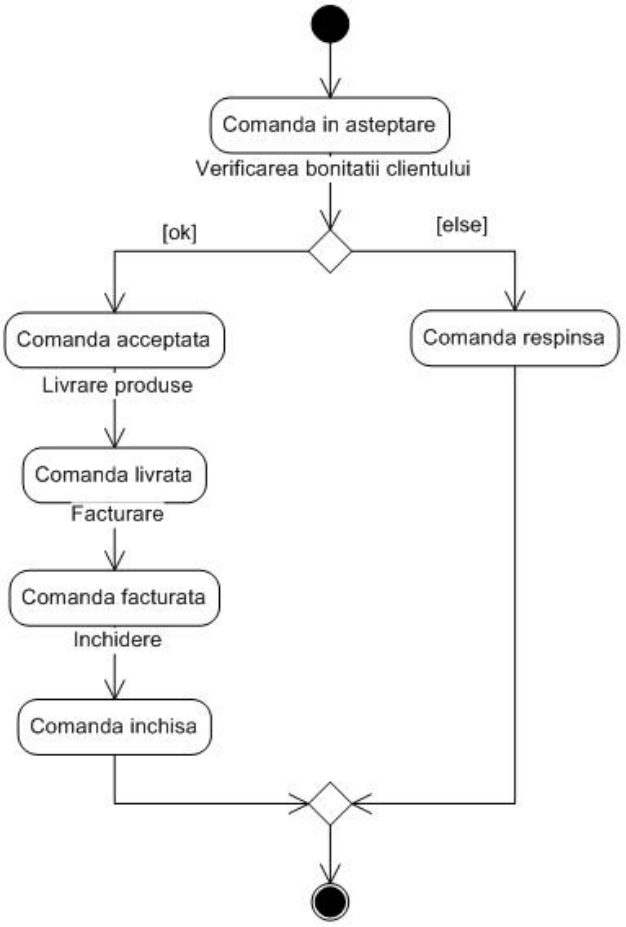

FIGURA 3.5: Diagramă de tip statechart pentru cazul realizării unei comenzi

3.3 Deciderea asupra comportamentului sincron și asincron

Fiecare mod de executare a operațiilor are propriile sale avantaje și dezavantaje. Există operaţii care ar putea necesita mai mult timp pentru a se termina de executat (>100ms) şi nu este direct vizibil faptul dacă acestea au fost declanşate de către o cerere (e.g. o nouă poziţie este trimisă).

O cerere poate declanşată din fire de execuţie diferite. Există posibilitatea ca acest lucru sa fie realizat sincron, în afara modului, între cereri şi răspunsuri, fapt ce poate duce la deadlock.

3.3.1 Factori de decizie

- Timpul în care firul de execuţie este blocat de cerere
- Sincronizarea între operaţii

3.3.2 Soluţii propuse

- Procesul se va executa asincron folosind un fir de execuţie de lucru
- Procesul se va executa sincron, în interiorul cererilor

3.3.3 Decizia

Se vor furniza două interfeţe diferite. Funcţionalitatea va fi oferită printr-o interfaţă sincronă, ce va fi utilizată în cadrul operaţiilor ce au loc pe un singur fir de execuţie. O altă interfaţă va decupla firele de execuţie şi procesele din bucla de lucru. Acest fapt ne oferă libertatea utilizării principiului de multithread-ing (execuţia mai multor thread-uri în acelaşi pipeline, fiecare având propria secţiune de timp în care este menit să lucreze).

3.4 Structura modulului din punctul de vedere al unităților software

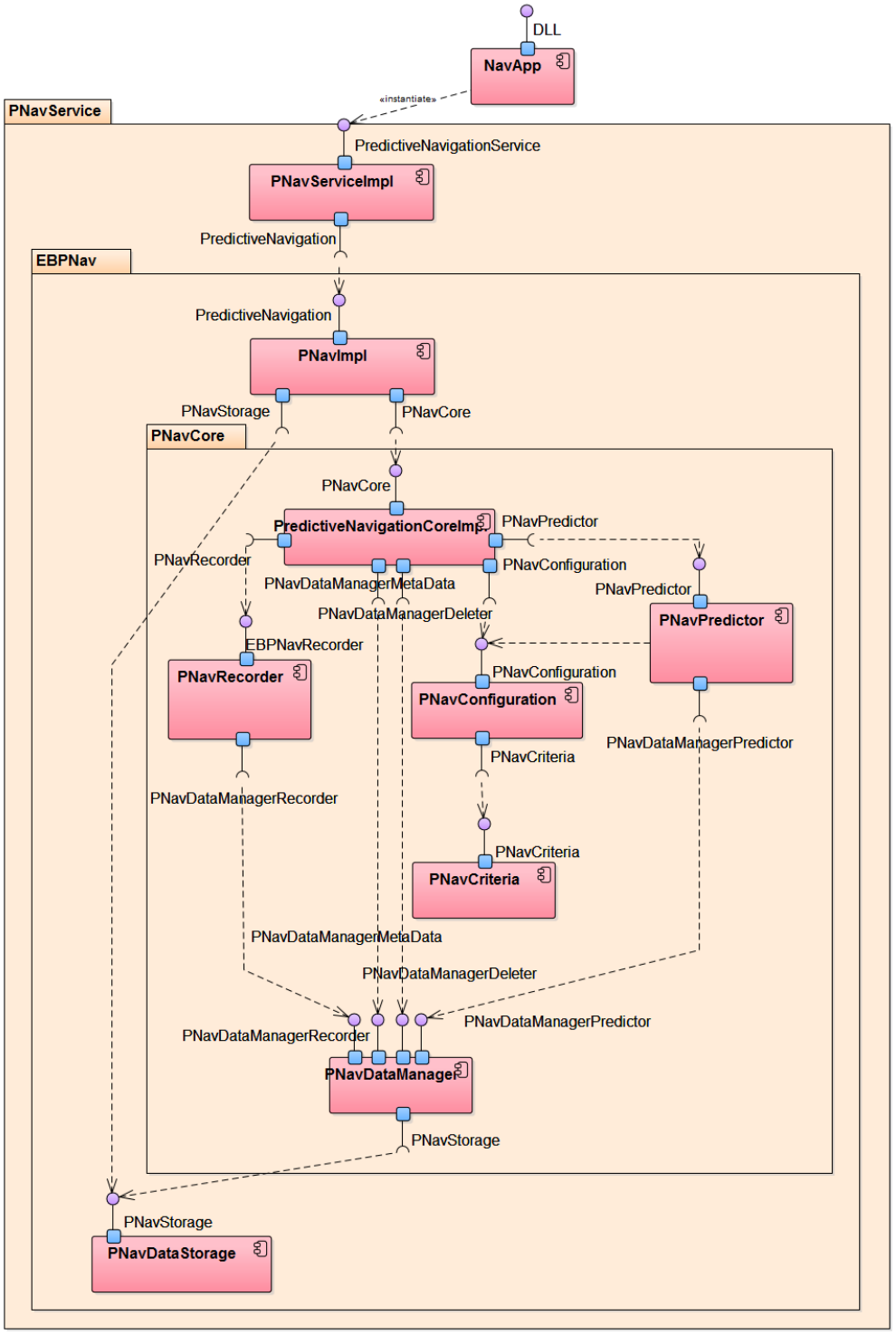

FIGURA 3.6: Diagrama componentelor

3.4.1 Unitatea software PNavDLL

Această unitate realizează "ambalarea" întregului modul sub forma unei biblioteci cu legare dinamică (Dynamic-link library (DLL)).

3.4.2 Unitatea software PNavCoreImpl

Unitatea PNavCoreImpl are rol de dispecer, utilizând restul unităților pentru realizarea funcționalităților de bază.

3.4.3 Unitatea software PNavServiceImpl

Unitatea PNavServiceImpl implementează interfaţa sincronă şi asincronă, şi îi oferă totodată dezvoltatorului posibilitatea de a alege ce interfaţă doreşte să folosească.

Cea asincronă are avantajul de a decupla firele de execuţie şi de a permite rularea activităţilor în paralel, însă are şi dezavantajul necesităţii de implementare unui mecanism de sincronizare în codul aplicaţiei în care va fi folosit modulul.

3.4.4 Unitatea software PNavImpl

Deşi funcţionalităţile de bază precum precum învăţarea şi predicţia sunt realizare de către unitatea PNavRecorder respectiv PNavPredictor, unitatea PNavImpl realizează funcţionalităţi suplimentare cum ar fi multiple profile de utilizatori, ştergerea bazelor de date.

Funcţionalitatea multiplelor profile de utilizatori permite gestionarea mai multor baze de date, ce pot fi selectate pe baza unui ID de profil. Acest ID poate cuprinde valori în intervalul 0-255. Pentru fiecare profil este creat un nou fişier în care vor fi stocate datele de utilizator. Unitatea PNavImpl implementează de asemenea şi funcţionalităţi de întreţinere a profilelor de utilizator precum ştergerea individuală, ştergerea totală, copierea, schimbarea între profile.

3.4.5 Unitatea software PNavRecorder

NavRecorder-ul este unitate în care întreg procesul de învăţare are loc.

Unitatea primeşte datele de geolocație şi de timp (oră - zi/lună/an) şi învaţă rutele parcurse de către dezvoltator într-un mod inteligent. Acest lucru înseamnă că waypoint-urile (punctele prin care a trecut utilizatorul în timpul rutei sale) sunt stocate numai când autovehiculul şi-a schimbat orientarea semnificativ (valoare standard: $> 15°$) sau distanţa dintre waypoint-uri nu este prea scurtă (valoare standard: > 200m). Valori pot fi configurate înaintea procesului de compilare.

Când sesiunea de înregistrare este finalizată, waypoint-urile sunt trimise către unitatea software PNavDataManager pentru a fi scrise în baza de date. Totodată, unitatea are implementate funcţionalităţi de oprire-pornire, lucru ce-i acordă dezvoltatorului dreptul de opri şi porni oricând sesiunea de înregistrare.

3.4.6 Unitatea software PNavPredictor

Rolul unităţii PNavPredictor este acela de a calcula predicţiile.

Primul pas constă în încărcarea datelor prin unitatea PNavDataManager, care sunt mai departe

prioritizate în funcţie de criterii specifice (descrise în tabela 4.2, "Criterii de prioritizare pentru predicţia bazată pe rute"). Datele pot fi de asemenea filtrate pe baza aceloraşi criterii, rezultând astfel o cantitate mai mică de date şi un timp mai scurt de încărcare a acestora.

Prioritizarea datelor este bazată atât pe datele de geolocație cât şi cele de timp, astfel încât o rută va avea o probabilitate mult mai mare de utilizare într-o anumită zi din săptămână sau la o anumită oră din zi.

Ca şi unitatea PNavRecorder, unitatea PNavPredictor are implementate funcţionalităţi de oprire-pornire.

3.4.7 Unitatea software PNavConfiguration

Unitatea PNavConfiguration configurează unitatea NavPredictor, prin intermediul unor funcţii ce folosesc criteriile definite în unitatea PNavCriteria.

3.4.8 Unitatea software PNavCriteria

Unitatea PNavCriteria conţine toate tipurile de criterii ce pot fi folosite la filtrarea sau prioritizarea datelor.

Fiecare criteriu în parte este folosit la procesarea datelor de către unitatea PNavPredictor. După procesarea tuturor criteriilor cea mai probabilă rută este creată.

3.4.9 Unitatea software PNavDataManager

Unitatea PNavDataManager implementează logica necesară pentru a realiza comunicarea între unitatea PNavDataStorage şi restul unităţilor.

În general, obiectele sunt stocate separat (e.g. rutele sunt stocate separat faţă de destinaţiilor lor). Cum însă pentru predicţia unei rute este nevoie de toate informaţiile, unitatea PNavDataManager le comasează. Aceasta oferă de asemenea şi alte funcţionalităţi precum adăugarea, gruparea, căutarea sau ştergerea de obiecte.

3.4.10 Unitatea software PNavDataStorage

Unitatea PNavDataStorage este dezvoltată pe baza structurii bazei de date.

În afară de funcţionalitatea principală de a stoca sau încărca date, aceasta asigură şi accesarea selectivă a obiectelor. Pentru realizarea acestor funcţionalităţi se execută interogări prin intermediul SQLite.

Capitolul 4
DESCRIEREA ALGORITMILOR

4.1 Învăţarea rutelor

Rutele sunt învăţate printr-un mecanism inteligent. Acest lucru are avantajul de a reduce semnificativ baza de date şi de a accelera încărcarea datelor.

4.1.1 Reducerea waypoint-urilor

Criteriile pentru excluderea waypoint-urilor sunt:

1. Vehiculul nu şi-a schimbat orientarea semnificativ (valoare standard: $> 15°$)
2. Poziţiile sunt apropiate una de cealaltă (valoare standard: > 200m)

Valorile standard pot fi configurate înaintea procesului de compilare.

4.1.2 Verificarea rutelor duble

În cazul în care ruta curentă are o secţiune similară cu o rută deja învăţată, unitatea software PNavDataManager va detecta secţiune respectivă şi va stoca numai waypoint-urile situate după aceasta. În schimb, toate destinaţiile sunt memorate. Criteriile pentru a detecta o astfel de rută sunt:

1. Destinaţia noii rute trebuie să fie situată într-o rază de 1km faţă de locaţia vechii destinaţii
2. Punctul de start al noii rute trebuie să fie situată într-o rază de 1km faţă de punctul de plecare vechii destinaţii
3. Toate waypoint-urile noii rute trebuie:
   (a) Să fie situat într-o anumită rază faţă de punctul de start al vechii rute
   (b) Sau să fie situat într-o anumită rază faţă de destinaţia vechii rute
   (c) Sau să fie situat într-o anumită rază faţă de un waypoint al vechii rute

Pragurile pot fi configurate înaintea procesului de compilare.

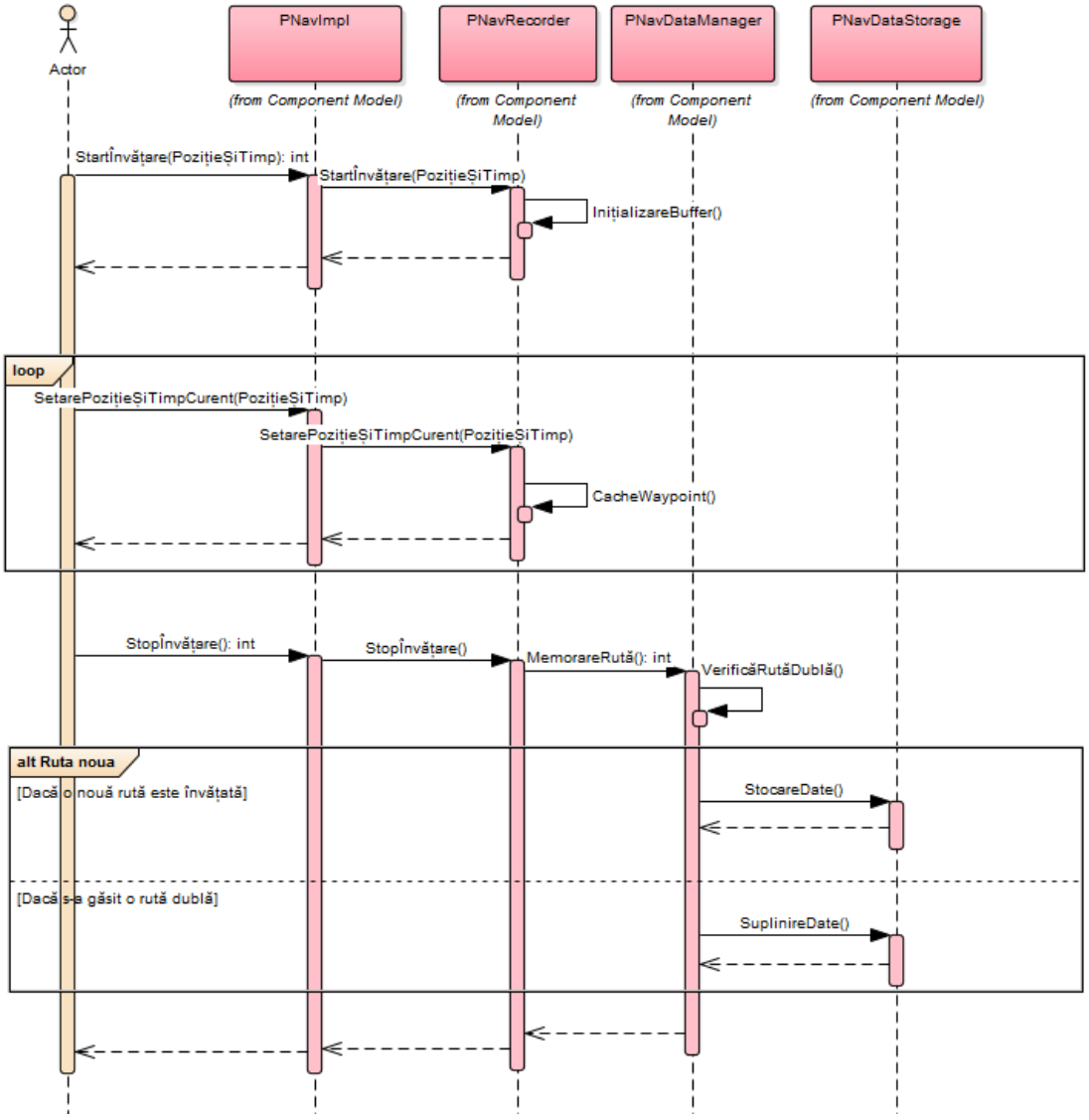

FIGURA 4.7: Procesul de verificarea al rutelor duble (sincron)

4.1.3 Filtrarea rutelor inutile

În timpul învăţării, rutele prea scurte (< 2km) nu vor fi stocate deoarece acest lucru ar însemna faptul că utilizatorul este destul de aproape de destinaţie.

Totodată, rutele foarte lungi (> 200km) vor fi de asemenea exclude din stocare deoarece ele nu reprezintă rute uzuale.

Pragurile pot fi configurate înaintea procesului de compilare.

4.2 Furnizarea predicţiilor

Pentru ca predicţiile să fie disponibile sunt necesari doi paşi.

Primul pas reprezintă încărcarea datelor relevante, în timp de al doilea constă în prioritizarea lor.

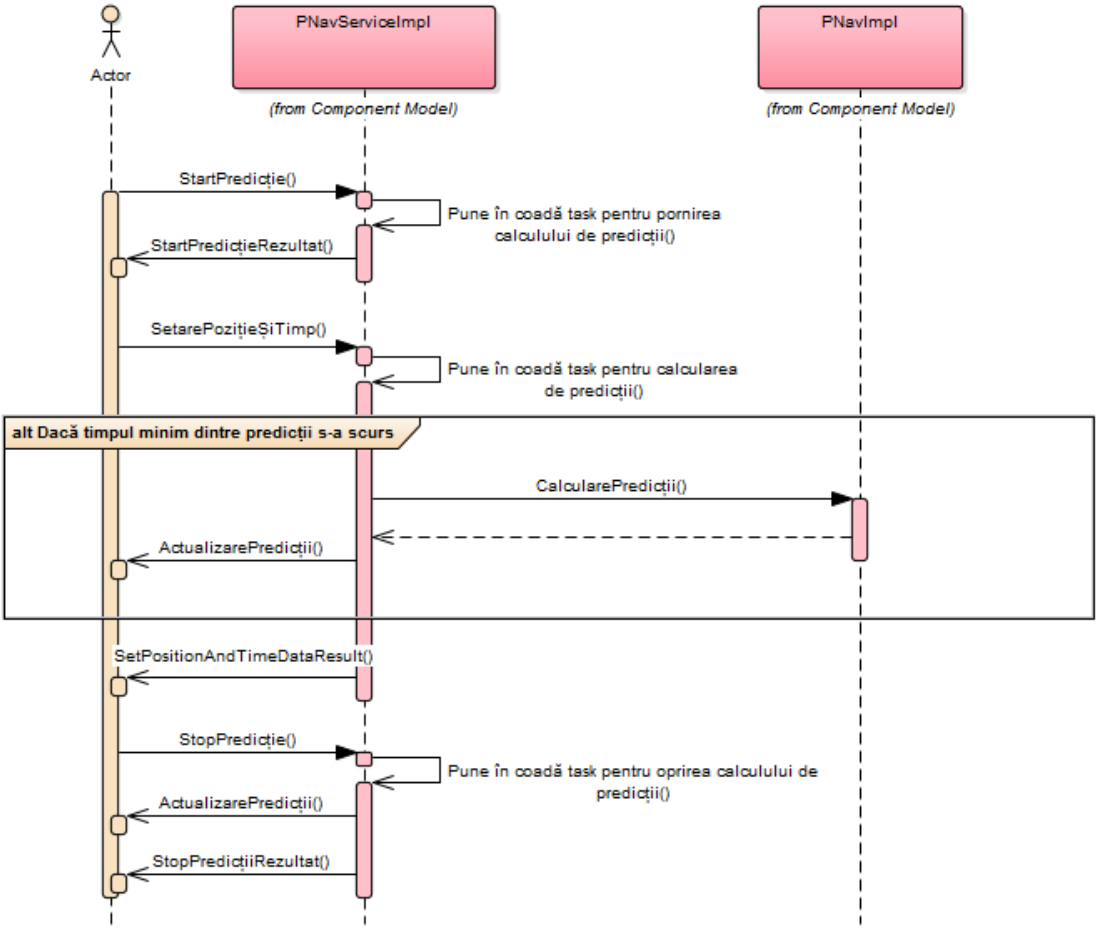

FIGURA 4.8: Procesul de calculare al predicțiilor(asincron)

4.2.1 Filtrarea datelor

Pentru o încărcare selectivă şi mai rapidă a datelor din unitatea PNavDataStorage, sunt create interogări. Sunt definiţi trei paşi în filtrare, unde pasul următor se excută doar în cazul în care cel curent nu a returnat destule rezultate:

1. Filtrarea rutei în funcţie de distanţa de la poziţia curentă la waypoint-urile rutelor
2. Filtrarea rutei în funcţie de distanţa de la poziţia curentă la punctele de start ale rutelor
3. Filtrarea rutelor în funcţie de timpul scurs până la ajungerea la destinaţie

Toate rutele ce duc la o destinaţie situată la o distanţă mai mică de 1km faţă de poziţia actuală sunt ignorate deoarece utilizatorul aproape a ajuns la eventuala destinaţie.

În cazul predicţiilor baza pe timp, modulul furnizează o predicţie fără a cunoaşte ruta, ci doar destinaţia sa. Un astfel de caz ar fi cel în care utilizatorul a condus pe o rută în intervalul luni-miercuri, însă in ziua de joi a pornit de la o altă locaţie. Astfel, bazat pe timp, modulul prezice destinaţia fără a şti ruta corespunzătoare acesteia.

4.2.2 Frecvenţa predicţiei de rute

De obicei, waypoint-urile furnizate de modul de predicţie sunt transformate într-o rută ce foloseşte drumuri din hartă, fapt ce durează câteva secunde. Pentru a nu supraîncărca sistemul de navigaţie cu prea multe predicţii, dezvoltatorul poate seta timpul minim dintre două predicţii. Acestă setare se poate efectua chiar în timpul rulării.

4.2.3 Numărul maxim de predicţii

Pentru a putea suporta multiple platforme, modulul permite setarea numărului maxim de rute prezise ce apar deodată. Acestă setare se poate efectua chiar în timpul rulării.

4.2.4 Prioritizarea predicţiilor

Prioritatea unei rute reprezintă de fapt de probabilitatea ca această rută să fie ce pe care utilizatorul ar fi ales-o în mod uzual. Aceasta este exprimată sub formă de procente și se încadrează în intervalul 0%-100%.

Există mai multe criterii pentru a stabili prioritatea unei rute, fiecare dintre acestea fiind raportat la intervalul 0 (minimul) - 100 (maximul).

Pentru calcularea probabilității unei rute sunt necesari trei pași:

1. Pentru fiecare criteriu, se înmulțește ponderea cu probabilitatea sa

$$SC = \sum_{i=1}^{n} p_i * c_i \tag{4.1}$$

2. Se însumează toate criteriile

$$SP = \sum_{i=1}^{n} p_i \tag{4.2}$$

3. În urma raportului dintre suma de la pasul 1 și suma de la pasul 2, se obține probabilitatea rutei

$$P = \frac{SC}{SP} \tag{4.3}$$

unde, *P* = probabilitatea rutei, *SC* = suma criteriilor, *SP* = suma ponderilor, *p* = ponderea criteriului, *c* = probabilitatea criteriului (0%-100%), *n* = numărul total de criterii

Metoda de definire a fiecărui criteriu depinde de criteriile însăși, după cum se observă în tabelele următoare.

TABELA 4.2: Criterii de prioritizare pentru predicţia bazată pe rute

Criteriu	**Descriere**	**Min (0% probabilitate)**	**Max (100% probabilitate)**	**Pondere**
Frecvenţa rutei	Numărul de utilizări al unei rute	Niciodată	Folosită de mai mult de 10 ori	4
Frecvenţa rutei într-un anumit interval de timp	Numărul de utilizări al unei rute într-un anumit interval de timp (de la -1 oră la +2 ore) faţă de ora curentă	Niciodată	Folosită de mai mult de 10 ori	1
Frecvenţa rutei într-o anumită zi	Numărul de utilizări al unei rute în ziua curentă din săptămână	Niciodată	Folosită de mai mult de 10 ori	1
Frecvenţa rutei într-un anumit grup de zile	Numărul de utilizări al unei rute într-un anumit grup de zile (e.g. luni-vineri)	Niciodată	Folosită de mai mult de 10 ori	1
Ultima utilizare a unei rute	Diferenţa dintre ultima utilizare a rutei şi ora/data curentă	>4 săptămâni	<2zile	3
Distanţa până la destinaţie	Distanţa dintre poziţia actuală şi destinaţie	>100km	0km	1

4.2.5 Predicţii bazate pe filtrarea rutei în funcţie de distanţa de la poziţia curentă la waypoint-urile rutelor

Criteriile definite în tabela 4.2, “Criterii de prioritizare pentru predicţia bazată pe rute” sunt extinse prin adăugarea următoarelor criterii:

TABELA 4.3: Criterii de prioritizare pentru predicţia bazată pe rutele din jurul unei poziţii

Criteriu	**Descriere**	**Min (0% probabilitate)**	**Max (100% probabilitate)**	**Pondere**
Distanţa până la rută	Distanţa dintre poziţia actuală şi waypoint-urile rutei	> 1km	0km	1
Direcţia către rută	Diferenţa dintre orientarea waypoint-urilor şi direcţia de navigare	180°	0°	1

4.2.6 Predicţii bazate pe filtrarea rutei în funcţie de distanţa de la poziţia curentă la punctele de start ale rutelor

Criteriile definite în tabela 4.2, "Criterii de prioritizare pentru predicţia bazată pe rute" sunt extinse prin adăugarea următoarelor criterii:

TABELA 4.4: Criterii de prioritizare pentru predicţia bazată pe filtrarea rutele în funcţie de distanţa până la punctul de start al rutei

Criteriu	**Descriere**	**Min (0% probabilitate)**	**Max (100% probabilitate)**	**Pondere**
Distanţa până la punctul de start al rutei	Distanţa până la cel mai apropiat punct de start al rutei	> 3km	0km	1

4.2.7 Predicţii bazate pe filtrarea rutei în funcţie de timpul scurs până la ajungerea la destinaţie

Pentru acest caz sunt folosite criteriile definite în tabela 4.2, "Criterii de prioritizare pentru predicţia bazată pe rute".

4.3 Eliberarea de spaţiu

Dacă pragul setat iniţial pentru dimensiunea maximă a bazei de date este atins, se va activa funcţia de eliberare a spaţiului. Aceasta va şterge obiectele cele mai vechi pentru a crea loc pentru obiectele noi.

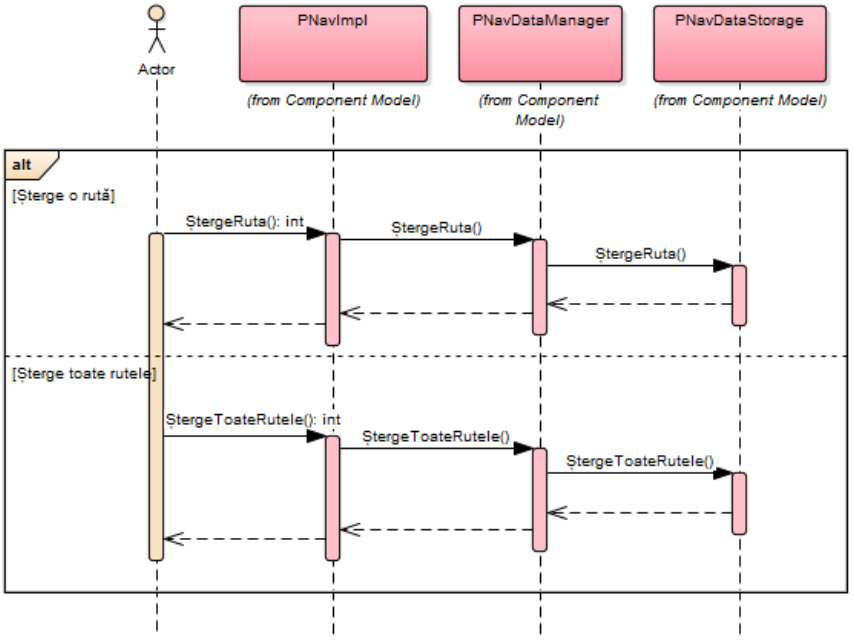

FIGURA 4.9: Procesul de ștergere a uneia sau mai multor rute (sincron)

Capitolul 5
MANAGEMENTUL ERORILOR

5.1 Tipuri de erori

- Erori de secvenţă
- Erori apărute la accesarea bazei de date
- Bază de date coruptă

5.2 Detectarea erorilor

Erorile sunt raportate pentru fiecare apel către şi dinspre interfaţa modului de predicţie. Toate funcţiile returnează un număr ce corespunde unui tip de eroare.

5.2.1 Erori de secvenţă

O maşină de stare va verifica încălcarea ordinii secvenţelor.

5.2.2 Erori la accesarea bazei de date

Trebuie evaluate rezultatele funcţiilor native de accesare ale bazei de date.

5.2.3 Bază de date coruptă

Trebuie evaluate rezultatele funcţiilor native de accesare ale bazei de date.

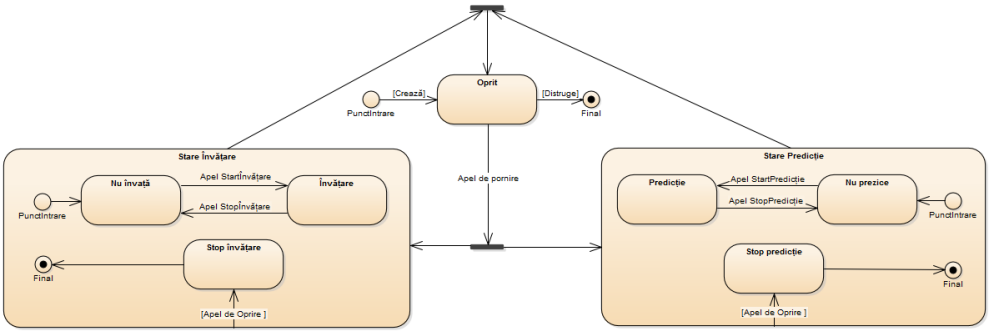

FIGURA 5.10: Mașină de stare pentru mecanismul învățare-predicție

5.3 Tratarea erorilor

În funcţie de tipul de eroare, aceasta poate fi prevenită pe viitor sau nu de către dezvoltator.

Există erori de secvenţă precum "Înainte de apelarea funcţiei X este necesară pornirea unităţii software PNavPredictor". Se poate întâmpla de asemenea ca atunci când dezvoltatorul porneşte procesul de predicţie de două ori consecutiv sa fie întâmpinat de eroarea "Procesul de predicţie se află deja în curs de rulare". În astfel de cazuri este clar cum se pot preveni erorile.

Mai sunt însă şi cazuri ce nu pot fi tratate de către dezvoltator. Astfel de erori sunt cele precum "Baza de date este coruptă", ce pot să apară în cazul în care fişierul de sistem folosit pentru stocarea datelor este corupt. În aceste situaţii datele stocate anterior nu mai pot fi recuperate, toate informaţiile referitoare la rute fiind definitiv pierdute.

Capitolul 6
CONCLUZII

6.1 Concluzii generale

Scopul acestei lucrări a fost acela de a dezvolta un modul de navigare pentru estimarea rutelor posibile ale unui autovehicul. Fiind într-o lume în continuă mișcare și în care orice autovehicul modern dispune de o aplicație de navigare, s-a dorit oferirea utilizatorului de servicii care să-i ușureze acestuia munca și să transforme experiența utilizării unei astfel de produs într-o plăcere.

În cadrul lucrării au fost prezentate deciziile cu cea mai mare influență asupra proiectului, definirea arhitecturii modului, logica din spatele algoritmilor dar bineînțeles și eventualele erori ce pot să apară și soluții aferente prevenirii lor.

Pentru realizarea proiectului s-a folosit limbajul de programare C++. S-a făcut această alegere deoarece este unul dintre limbajele cel mai bine stăpânite, dar și datorită faptului se dispunea deja de o aplicație de navigare scrisă în C++, lucrul ce a făcut ca dezvoltarea, integrarea și testarea modulului să fie mult mai facilă.

Primul pas a fost bineînțeles stabilirea clară a funcționalităților ce se doresc a fi obținute de la modulul în cauză. Odată decișii asupra acestori factori s-a creat diagrama de design software pe baza căreia, mai târziu, s-a început definirea unităților software și scrierea codului în sine. Până la scrierea codului însă au fost necesare și luarea anumitor decizii, precum alegerea tipului de bază de date. În urma alegerii făcute a fost realizată și o scurtă documentare asupra modului cum poate fi folosit API-ul acesteia pentru crearea legăturii și comunicarea efectivă dintre partea de cod și baza de date în sine.

Din punct de vedere al funcționalității, modulul a fost împărțit în două, prima parte fiind învățarea rutelor iar a doua predicția acestora.

Prin învățarea se înțelege stocarea datelor colectate pe parcursul rutelor frecventate de utilizator. Pentru ca baza de date rezultată să fie însă atât compactă cât și încărcată cât mai rapid, s-a optat pentru un mod de învățare inteligentă, prin aplicarea unor algoritmi de constrângere ce au ca scop filtrarea acestor date în funcții de anumite criterii.

Pe partea de predicție a rutelor pot exista diverse abordări și rezultate dorite din partea celui ce folosește modulul în cadrul aplicație sale, așa că pe lângă valorile standard s-a oferit și posibilitatea de configurare a criteriilor ce stau la baza mecanismului de predicție.

Ca și concluzie finală putem spune că s-a dezvoltat un modul ambalat sub forma unei biblioteci cu legare dinamică (DLL), configurabil și utilizabil în cadrul oricărei aplicații de navigare, ce colectează date despre rute și oferă la rândul său predicții pe baza acestora.

6.2 Posibilități de dezvoltare ulterioară

În urma realizării acestui proiect, a studierii necesităților și așteptărilor utilizatorului comun de la un sistem de navigație, dar și a analizei soluțiilor deja existente se pot indica mai multe direcții de dezvoltare.

O posibilă direcție de dezvoltare ar putea fi folosirea acestui modul împreună cu un altul de furnizare al datelor de trafic.

Presupunem faptul că utlizatorul este pe ruta folosită de acesta în mod uzual însă la un alt moment de timp față de obicei, moment în care traficul este mult mai intens pe tronsonul respectiv de drum. Folosind datele furnizate de cele două module, utilizatorului i se va putea oferi posibilitatea de a alege o rută alternativă, mult mai rapidă.

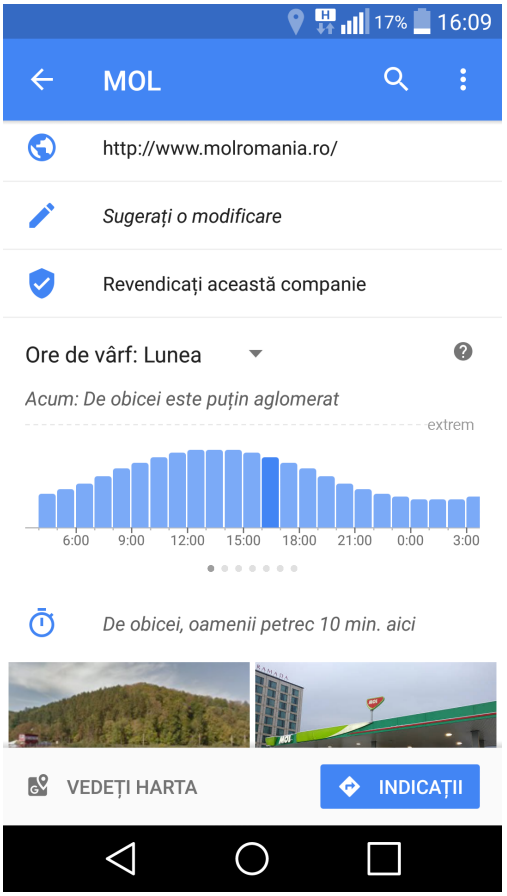

FIGURA 6.11: Captură de ecran din aplicația Google Maps

Același principiu s-ar putea aplica și pentru alte evenimente întâlnite în trafic, precum accidente, lucrări în desfășurare, și așa mai departe.

O altă direcție de dezvoltare ar fi crearea unei extensii pentru modul, cu scopul de a se sincroniza

cu dispozitivul utilizatorului (e.g. telefon înteligent) și de a prelua evenimentele planificate în calendarul acestuia. Fiind astfel extinsă funcționalitatea modulului, utilizatorului i se vor putea oferi predicții și pe baza planificărilor făcute de acesta anterior.

FIGURA 6.12: Sincronizarea între un sistem de navigare de pe Ford XL 2014 și un telefon inteligent

Bineînțeles, cum industria automobilistică este una în continuă evoluție vor apărea întotdeauna noi oportunități de dezvoltare a aplicațiilor de navigare și implicit a modulului în sine.