KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Vytautas Vaičys

# Kuro kolonėlių valdymo sistemos tyrimas

Magistro darbas

Darbo vadovas
doc. dr. K. Motiejūnas

Kaunas, 2006

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Vytautas Vaičys

# Kuro kolonėlių valdymo sistemos tyrimas

Magistro darbas

Kalbos konsultantė
Lietuvių k. katedros doc. dr. J. Mikelionienė
2006-05

Vadovas
doc. dr. K. Motiejūnas
2006-05

Recenzentas
doc. dr. D. Rubliauskas
2006-05

Atliko
IFM-0/2 gr. stud. Vytautas Vaičys
2006-05

Kaunas, 2006

## Summary

This document is a master's thesis analyzing an automated fuel pump management system. In the first chapters we take a general overview of the system and the main problems that we will be facing during the planning and design phases of the project. Later we propose and analyze possible solutions for these problems. Technical system information is revealed in the later chapters. Functional and non functional requirements are discussed along with the main UML diagrams. The research phase of the thesis provides detailed system analysis and software quality reports which later are used to create proposed changes to the system. These changes are analyzed, designed and coded in the final - experimental part of the thesis. The main proposal is to convert the system architecture from flow driven to event driven. This change helps to solve several uncovered architectural problems as well as improve the general quality of the system. These changes are tested and analyzed in the experimental chapter of the thesis and finally the conclusions are made.

The main conclusion is that the proposed architectural changes were chosen correctly. This is also supported by the experimental data.

# Turinys

**1. ĮVADAS ..... 7**

**2. LITERATŪROS APŽVALGA ..... 8**

2.1. KLIENTO-SERVERIO SISTEMOS ..... 8
2.2. PATIKIMŲ SISTEMŲ PROJEKTAVIMAS ..... 8
2.3. PROGRAMINIŲ SISTEMŲ IR INFORMACIJOS SAUGUMAS ..... 9
2.4. PAKARTOTINIO PANAUDOJIMO METODAI ..... 10
2.5. OBJEKTINĖS SISTEMOS ..... 10
2.6. ATMINTIES VALDYMO PROBLEMOS ..... 11
2.7. TIKRALAIKIU DIRBANČIOS SISTEMOS ..... 11
2.8. IŠVADOS ..... 12

**3. ANALITINĖ DALIS ..... 13**

3.1. APIE DARBĄ ..... 13
3.2. PAGRINDINIAI TIKSLAI ..... 13
3.3. SPRENDŽIAMI UŽDAVINIAI ..... 15
3.4. GALIMI UŽDAVINIŲ SPRENDIMO BŪDAI ..... 16
3.5. EGZISTUOJANTYS SPRENDIMAI ..... 17
3.6. ESMINĖS PROBLEMOS ..... 20
3.7. REKOMENDACIJOS PROBLEMŲ SPRENDIMUI ..... 23
3.8. DARBO RIZIKOS ĮVERTINIMAS ..... 28
3.9. PRAKTINIS DARBO REZULTATAS ..... 29
3.10. MOKOMASIS PROJEKTO REZULTATAS ..... 30

**4. PROJEKTINĖ DALIS ..... 31**

4.1. ĮVADAS ..... 31
4.2. APŽVALGA ..... 31
4.3. ARCHITEKTŪROS PATEIKIMAS ..... 31
4.4. PRIIMTI TECHNINIAI SPRENDIMAI ..... 32
4.5. NUMATOMA EKSPLOATACIJOS APLINKA ..... 33
4.6. SPRENDIMŲ ĮTAKA SISTEMOS ARCHITEKTŪRAI ..... 34
4.7. PROJEKTO TECHNOLOGINIAI APRIBOJIMAI ..... 34
4.8. VEIKLOS KONTEKSTAS ..... 35
4.9. VEIKLOS PADALINIMAS ..... 36
4.10. PANAUDOJIMO ATVEJAI ..... 36
4.11. FUNKCINIAI REIKALAVIMAI ..... 37
4.12. NEFUNKCINIAI REIKALAVIMAI ..... 39
4.13. ARCHITEKTŪROS SPECIFIKACIJA ..... 43
4.14. DUOMENŲ MODELIS ..... 49
4.15. PROJEKTO IŠEIGA ..... 50

**5. TIRIAMOJI DALIS ..... 52**

5.1. Įvadas ..... 52
5.2. Priimto sprendimo pagrindimas ..... 52
5.3. Kokybės analizės tikslai ..... 53
5.4. Kokybės vertinimo rezultatai ..... 54
5.5. Projektavimo problemos ..... 56
5.6. Realizacijos ir architektūros trūkumai ..... 57
5.7. Įvykiais valdomos sistemos ..... 59
5.8. Naujų technologijų pasirodymas rinkoje ..... 62
5.9. Rekomendacijos ..... 62

**6. EKSPERIMENTINĖ DALIS ..... 64**

6.1. Pakeitimų apžvalga ..... 64
6.2. Galimos problemos ..... 65
6.3. Pakeitimų realizacijos planas ..... 65
6.4. Pakeitimų įtaka sistemos architektūrai ..... 66
6.5. Pakeitimų eksperimentinis tyrimas ..... 68
6.6. Tyrimo rezultatai ..... 69
6.7. Eksperimentinio tyrimo išvados ..... 74

**7. IŠVADOS ..... 75**

**8. LITERATŪRA ..... 76**

**9. TERMINŲ IR SANTRUMPŲ ŽODYNAS ..... 78**

**10. PRIEDAI ..... 80**

10.1. Ryšio modulio kodo fragmentai ..... 80
10.2. Valdymo modulio kodo fragmentai ..... 81

# Lentelių sąrašas

Lentelė 1. Sistemos rizikų įvertinimas ....28
Lentelė 2. Atsitiktinumų valdymo planas ....28
Lentelė 3. Sistemos veiklos padalinimas ....36
Lentelė 4. Kokybės vertinimo rezultatai ....55
Lentelė 5. Sistemos naudojami resursai ....73
Lentelė 6. Programos versijų priklausomybės ....73

# Paveikslų sąrašas

Pav. 1. Sistemos veiklos kontekstas ....35
Pav. 2. Panaudos atvejų diagrama ....37
Pav. 3. Statinis sistemos vaizdas ....43
Pav. 4. Paketų detalizavimo diagrama ....45
Pav. 5. Pagrindinė programos veiklos diagrama ....46
Pav. 6. Sistemos būsenų diagrama ....47
Pav. 7. Degalų pylimo proceso sekų diagrama ....47
Pav. 8. Degalų pajamavimo proceso sekų diagrama ....48
Pav. 9. Duomenų modelis ....49
Pav. 10. Įprastos programos vykdymo principas ....57
Pav. 11. Tikroviška metodo vykdymo schema ....58
Pav. 12. Veiksmų eiliškumo problema ....58
Pav. 13. Kitos programos vykdymo problemos ....59
Pav. 14. Atnaujinta klasių diagrama ....67
Pav. 15. Programinio kodo eilučių skaičius pradiniame programos variante ....69
Pav. 16. Pirmojo ir antrojo programų variantų palyginimas ....70
Pav. 17. Vidutinis vienos klasės funkcijų skaičius ....70
Pav. 18. Kviečiančių ir kviečiamųjų funkcijų palyginimas ....71
Pav. 19. Vidutinis sąlygos tikrinimo gylis ....71

# 1. Įvadas

Darbo tema – „Kuro kolonėlių valdymo sistemos tyrimas“. Šio darbo vykdymo metu buvo kuriama automatizuota kuro kolonėlių valdymo sistema.

Tokio tipo sistema svarbi ir reikalinga daugumai Lietuvos įmonių, kurių verslas vienaip ar kitaip priklauso nuo transporto priemonių eksploatavimo, transportavimo paslaugų ar kitų su šia veikla susijusių išlaidų. Tokioms įmonėms kiekvieną dieną tenka susidurti su kuro pirkimo problema, kurią tik toliau apsunkina nepaliaujamai augančios kuro kainos. Kiekvienai įmonei nuolat tenka spręsti transporto išlaidų minimizavimo problemą. Dažniausia sudaromos sutartys ar kontraktai su verslo partneriais, kurie užsiima mažmenine kuro prekyba, tačiau toks sprendimas nėra optimalus. Ekonominiu atžvilgiu priimtiniausias sprendimas kiekvienai, pakankamai daug transporto priemonių turinčiai įmonei, yra pasistatyti nuosavą degalinę ir įdiegti joje automatizuota degalinės valdymo programinę įrangą. Šiam tikslui įgyvendinti ir buvo sukurtas šis projektas.

Galutinis produktas tenkina šiuos pagrindinius reikalavimus:

- autonominis sistemos veikimas;
- kuro kolonėlių valdymas ir išduodamo kuro registravimas;
- patikimas sistemos veikimas;
- programinės sistemos ir jos duomenų saugumas;
- nuolatinis sistemos pasiekiamumas.

Analitinėje darbo dalyje atsakomi pagrindiniai šio projekto klausimai, pristatomos bendros idėjos ir vėliau apžvelgiama panašių sistemų specifika. Projektinėje dalyje į sistemą pažvelgiama iš architektūros pusės. Sukonkretinami priimti sprendimai ir pateikiamas jų realizacijos aprašymas. Tiriamoje darbo dalyje atliekamas sistemos tyrimas. Įvertinama sistemos kokybė ir pateikiamas apibendrintas sistemos vaizdas kartu su rekomenduojamais patobulinimais. Išskiriamos silpnosios galutinio produkto dalys ir pateikiami pasiūlymai jų tobulinimui. Šių tobulinimų rezultatai apžvelgiami ir įvertinami eksperimentinėje darbo dalyje.

# 2. Literatūros apžvalga

## *2.1. Kliento-serverio sistemos*

Kliento-serverio tipo sistemos pradėtos naudoti jau nuo 1980 metų, kai tik atsirado didžiosios (*mainframe*) sistemos [1]. Šiame darbe taip pat galima pritaikyti kai kurias šio tipo sistemų programavimo metodikas, nes galime rasti visus tokio tipo sistemos elementus: valdantis įrenginys (*master*), valdomas įrenginys (*slave*), dviejų lygių architektūra ir adresuojamas ryšio kanalas.

Šiame projekte galime rasti dvi pagrindines sritis, kuriuose galima pritaikyti tokio tipo projektavimo metodus:

- Kolonėlių valdymas. Kompiuteris atlieka valdančiojo įrenginio rolę, su kolonėlėmis dirba kaip su atskirais klientais. Skirtumas tik tas, kad klientas šiuo atveju dirba pasyviame režime (*slave*).
- Vartotojo sąsaja, kuri gali būti atskiriama nuo fizinės duomenų bazės ar net nuo pačio valdančiojo kompiuterio. Šiuo atveju papildomi projektavimo darbai nėra būtini, nes ryšiu su duomenų baze pasirūpina standartinės duomenų bazės sąsajos, tačiau tai nekeičia pačios sistemos tipo.

Dviejų lygių architektūra realizuota duomenų atvaizdavimo ir ataskaitų spausdinimo modulyje. Šį principą taip pat būtų galima pritaikyti ir kolonėlės valdymo moduliui. Pastaruoju atveju valdančiame kompiuteryje turėtų būti vykdomas tik valdantis procesas, o įrankiai būtų jungiami prie šio proceso per tinklą.

Kliento-serverio sistemos yra naudojamos ir palaikomos beveik visose kompiuterijos srityse, tačiau tai nereiškia, kad sprendimas pritaikyti tokią technologiją neįtakoja galutinės produkto kainos. Priėmus tokį sprendimą atsiranda visa eilė papildomų problemų, tokių kaip: strateginis planavimas, susijęs su sistemos vartotojų augimu, sudėtingesnis kūrimas ir palaikymas, poreikis diegti ir palaikyti daugiau nei vieną sistemą, kurios bendraus per tinklo ar kitas sąsajas. Pagrindinis privalumas, tai puikus sistemos išplečiamumas ir vartotojo bei atvaizdavimo programinės logikos atskyrimas nuo valdančiosios logikos.

## *2.2. Patikimų sistemų projektavimas*

Patikimų sistemų projektavimo principai [2] grindžiami tuo, kad neįmanoma sukurti tobulo gaminio, kuris neturėtų jokių defektų, tačiau galima sukurti pakankamai patikimas sistemas. Imkime kaip pavyzdį liftą. Mes puikiai žinome, kad jame naudojamos detalės nėra tobulos ir žinome, kad teisingai veikti be periodiško patikrinimo ir remonto liftas negalės. Tačiau visi

žmonės kiekvieną dieną naudojasi liftais, nes yra numatyti specialūs saugumo ir patikimumo protokolai, kurie praktiškai garantuoja, jog sugedęs liftas nenukris ant žemės.
Patikimos programinės įrangos projektavimas vykdomas pagal panašias idėjas. Esmėje, tai patikimos sistemos kūrimas iš nepatikimų pavienių dalių. Elektronikos specialistai tokią praktiką vykdo jau nuo labai senų laikų. Pagrindiniai patikimumo principai:

- defektų mažinimas;
- sprendimo paprastumas – padeda išvengti klaidų;
- apsaugų pertekliškumas – leidžia numatyti, kad įvykus gedimui vienoje sistemos dalyje bus kita perteklinė sistema, kuri galės atlikti tas pačias funkcijas;
- klaidų aptikimas ir teisingas apdorojimas.

Šie pagrindiniai principai turi būti taikomi ir šio projekto vykdymo metu, nes patikimumas šioje sistemoje turi sąlyginai aukštus keliamus reikalavimus.

## 2.3. Programinių sistemų ir informacijos saugumas

Kylant kompiuterizacijos lygiui vis daugiau dėmesio skiriama programinės įrangos saugumui. Tai tapo ypač svarbia problema, kai kompiuteriai pradėti jungti į pasaulinį interneto tinklą. Internete šiuo metu galima rasti begales techninių dokumentų ir produktų, kuriuose kalbama apie programinės įrangos saugumą arba siūlomi metodai šiai problemai spręsti.
Bandysime pateikti keletą bendro pobūdžio rekomendacijų saugiai programinei įrangai [3]. Pagrindinės nesaugios programinės įrangos problemos, kurios susijusios su analizuojama problemine sritimi:

- buferio persipildymas;
- teksto formatavimo problemos;
- SQL kodo įterpimas;
- komandų įterpimas;
- iškylančių klaidų neteisingas apdorojimas;
- tinklo informacijos nesaugojimas;
- lengvų slaptažodžių naudojimas;
- informacijos nutekėjimas;
- neteisingas failų naudojimas;
- masyvų ribų tikrinimas;
- sinchronizacijos problemos;

- nepatogus sistemos naudojimas.

Daugumą šių problemų galėtų padėti išspręsti valdoma (*managed*) projektavimo aplinka (pavyzdžiui, Java ar .NET), nes šiose aplinkose atliekama griežta kintamųjų, jų tipų, masyvų ir kitų operacijų kontrolė, o papildomai atliekamas ir automatinis atminties valdymas.

Kiekvienos įmonės kompiuteriniai duomenys yra gyvybiškai svarbūs. Norint apsaugoti savo sistemas ir juose saugomus duomenis pirmiausia reikia pasirūpinti fiziniu sistemų saugumu, o tik po to pradėti taikyti saugaus programinės įrangos projektavimo metodus.

## 2.4. Pakartotinio panaudojimo metodai

Šios sistemos kūrimo metu buvo gauta dalis senos kolonėlių valdymo programos kodo, tačiau nuspręsta toliau nevykdyti šio programinio kodo priežiūros ir atstatydinti pasenusią sistemą. Programinio kodo kokybė buvo pakankamai prasta, todėl nebuvo galima pritaikyti standartinių pakartotinio panaudojimo transformacijos metodų.

Nauja sistema projektuojama atsižvelgiant į galimus pakeitimus sistemos specifikacijoje [4] (*design for change*). Naujoji sistemos architektūra turėtų leisti svarbias sistemos dalis ar jų funkcionalumą keisti arba modifikuoti pritaikant minimalius programinės įrangos palaikymo resursus.

Architektūroje numatyta galimybė prie sistemos prijungti bet kokio tipo kolonėlių valdymo modulį ir vartotojų identifikavimo įrenginį. Taip pat visa logika, susijusi su vartotojo taisyklėmis ir poreikiais yra realizuota atskiroje standartinę sąsają turinčioje klasėje.

Siekiant sukurti lengvai prižiūrimą ir modifikuojamą sistemą taip pat reiktų nepamiršti tvarkingos ir gerai suplanuotos duomenų bazės schemos, kurią ateityje būtų galima prižiūrėti ir keisti be didelės įtakos valdančiai sistemai.

## 2.5. Objektinės sistemos

Dauguma modernių programinių produktų kuriami pritaikant objektiškai orientuotus projektavimo įrankius. Objektinis projektavimas - tai modelis, kuris leidžia į programos objektus žiūrėti kaip į juodas dėžes, kurios tik siunčia ir gauna pranešimus [5]. Tinkamai pritaikius šį modelį galima ne vien tik pagerinti programinės įrangos pakartotinio panaudojimo galimybes, bet tuo pačiu leidžia ir sumažinti palaikymo kainas, bei sumažinti programinės įrangos modifikavimo kaštus.

Pagrindiniai principai, kurie taikomi objektiškai orientuotose sistemose:

- Inkapsuliacija. Objektas – tai juoda dėžė, kuri tik siunčia ir gauna pranešimus. Visi objektai gali dirbti nepriklausomai ir žinoti apie kitus objektus reikia tik tiek, kad būtų galima pasinaudoti jų teikiamomis sąsajomis.

- Paveldimumas. Baziniai objektai realizuoja savo teikiamas funkcijas, kurias toliau galima lanksčiai ir lengvai išplėsti paveldimuose objektuose. Pakeitus bazinį objektą (pavyzdžiui, ištaisius klaidą), pakeitimai persiduoda ir visiems to objekto palikuonims.
- Polimorfizmas. Tai savybė, kuri apibūdina kodo formos keitimą. Vienas aprašas gali būti naudojamas su skirtingais duomenų rinkiniais.

Papildomai, pasirinkus objektiškai orientuotą programavimo metodologiją atsiveria visas sąrašas šiai sričiai pritaikytų projektavimo šablonų, kurie leidžia net pačias sunkiausias problemas sumažinti iki valdomų ir prognozuojamų sistemos dalių. Kai kuriose srityse pritaikius projektavimo šabloną galima sukurti paprastesnį ir elegantiškesnį problemos sprendimą.

## 2.6. Atminties valdymo problemos

Atminties valdymas, tai vienas iš kritinių gerą pasiekiamumą ir stabilumą turinčios sistemos atsakomybių. Neteisingai realizavus arba neskyrus pakankamai dėmesio šiai problemai spręsti galiausiai gaunamas nestabilus produktas. Ne visos programos susiduria su šia problema, tačiau kuro kolonėlių valdymo sistema turės dirbti nepertraukiama neribotą laiko tarpą.
Idealiu atveju tokio tipo sistemoje reikėtų pritaikyti (arba realizuoti) specializuotą atminties valdymo programą. Pagrindinės tokios programos funkcijos yra [6]:

- greitas veikimas;
- mažas papildomų resursų sunaudojimas;
- minimalus naudojamų atminties puslapių skaičius;
- atsparumas galimoms problemoms;
- portatyvumas;
- patogumas;
- lankstumas.

Tokio pagalbinio įrankio realizacija reikalautų visiškai atskiro projektinio darbo, todėl šiam projektui būtina surasti egzistuojantį sprendimą ir jį pritaikyti. Lengviausias sprendimas būtų panaudoti vieną iš valdomų (*managed*) projektavimo aplinkų, kurios pagal savo prigimtų jau turi veikiantį atminties valdymo mechanizmą.

## 2.7. Tikralaikiu dirbančios sistemos

Yra daug skirtingų tikralaikiu veikiančių sistemų tipų, kurios pagal apibrėžimą dirba priklausomai nuo laiko apribojimų. Geras tokios sistemos pavyzdys būtų stabdžių blokavimą

reguliuojanti sistema, kai galėtume laikyti, kad sistema neatliko savo darbo jei nesuveikė per tam numatytą trumpą laiko tarpą.

Šiame projekte realizuojama sistema, tai programiškai tikralaikiu (*soft real time*) dirbančios sistemos variantas [7]. Tokiose sistemose reikia atlikti vykstančių vienu metu procesų kontrolę bei atnaujinti visų sistemoje dirbančių sistemų statusą ir informaciją, tačiau jei sistema šiek tiek ilgiau uždelstų, nei buvo numatyta specifikacijoje, negalėtume sakyti, kad sistemoje įvyko gedimas. Projekte valdymo algoritmą reikia realizuoti tokiu būdu, kad visi aptarnaujami įrenginiai būtu apklausiami periodiškai, nepadarant per daug didelės pertraukos tarp atskirų apklausimų.

## 2.8. Išvados

Padarius literatūros analizę buvo išskirtos ir išanalizuotos pagrindinės kuro valdymo sistemų problemos ir apžvelgti literatūroje egzistuojantys būdai šioms problemoms spręsti. Apžvelgti pagrindiniai projektavimo ir inžineriniai sunkumai, apie kuriuos bus plačiau rašoma analitinėje darbo dalyje. Padarytos rekomendacijos galimiems sprendimams.

# 3. Analitinė dalis

## *3.1. Apie darbą*

Darbo idėja gimė tada, kai atsirado poreikis konkrečiai problemai išspręsti. Šiuo atveju problema buvo negalėjimas patenkinti visų užsakovo klientams kylančių pageidavimų ir poreikių. Užsakovui prisireikė sukurti sistemą, kuri užpildytų atsiradusią rinkos nišą, tuo pačiu leistų klientams pateikti lankstesnį produktų ir paslaugų ratą. Taip gimė projektas „Automatizuota kuro kolonėlių valdymo sistema“, kuris buvo vykdomas Kauno technologijos universiteto ir užsakovo suteiktos praktikos darbo vietos dėka visų magistro studijų metu. Šio praktinio darbo vystymo metu buvo iškelta keletas problemų ir klausimų, sudaryta keletas teorinių užduočių, kurios leido projektą transformuoti į magistro darbą „Kuro kolonėlių valdymo sistemos tyrimas“.

Pirmuose skyriuose darbas analizuojamas labiau iš praktinės pusės. Analitinėje dalyje atsakomi bendri klausimai, paaiškinami tikslai, iškilusios problemos ir jų sprendimo metodai. Projektinėje dalyje labiau gilinamasi i sistemos architektūrą. Čia apžvelgiama kaip teoriniai pasiūlymai ir rekomendacijos buvo transformuotos į praktinę veikiančią sistemą. Antroje darbo dalyje atliekamas teorinis darbo tyrimas, sudaromas pasiūlymų ir rekomendacijų sąrašas, kurių realizacijos įtaka tikrinama ir analizuojama darbo eksperimentinėje dalyje.

## *3.2. Pagrindiniai tikslai*

Šiame skyriuje bus aptariami bendri (ekonominiai, vadybos) projekto tikslai. Techniniai sistemos uždaviniai ir jų aprašymai pateikiami sekančiame skyriuje.

Projekto tikslas buvo sukurti automatizuotą kuro kolonėlių valdymo sistemą. Ši sistema turi be papildomo žmonių įsikišimo leisti vartotojams įsipilti degalų. Mėnesio (ar kito laikotarpio) pabaigoje turi būti suformuotos ir atspausdintos įvykdytų pylimo operacijų ataskaitos. Turi būti galimybė duomenis ataskaitose atvaizduoti įvairiais skerspjūviais, pagal užsakovo reikalavimus.

Pagrindiniai produktui keliami reikalavimai yra:

- autonominis veikimas;
- patikimumas;
- saugumas;
- pasiekiamumas.

Daugumai Lietuvos įmonių, kurios išlaiko nors ir mažą lengvųjų ar sunkiųjų automobilių ūkį, kiekvieną dieną tenka susidurti su kuro pirkimo problema. Nemažą kiekvieno transportuojamo produkto (ar paslaugos) savikainos dalį sudaro būtent transporto išlaidos, o prie problemos tik prisideda nuolat augančios dyzelinių ir paprastų degalų kainos. Todėl kiekvienai įmonei neišvengiamai tenka spręsti transporto išlaidų minimizavimo problemą. Dažniausia sudaromos sutartys ar kontraktai su verslo partneriais, kurie užsiima mažmenine kuro prekyba, tačiau toks sprendimas nėra optimalus, nes įmonė praranda:

- pinigus (kuras perkamas su nuolaida, tačiau ne už mažiausią įmanomą kainą);
- kontrolę (priklausomai nuo to, kokia pardavėjo apskaitos sistemos, užsakovas gali papildomos informacijos apie atliekamus pylimus visai negauti, arba gauti daug informacijos, su kuria sunku dirbti, pavyzdžiui, kasos čekiai);
- laiką (kiekvieną kartą prisireikus degalų reikia važiuoti į degalinę; taip švaistomas laikas, o tuo pačiu ir pinigai).

Optimalus sprendimas kiekvienai pakankamai daug transporto priemonių turinčiai įmonei būtų pasistatyti nuosavą degalinę ir įdiegti joje automatizuota degalinės valdymo programinę įrangą. Šiam tikslui įgyvendinti ir sukurtas šis projektas.

Savininkui nusprendus įsigyti nuosavą degalinę iškart atsiveria nemažai naujų perspektyvų:

- kuro pirkimas didmeninėmis kainomis;
- sutaupytas laikas;
- visiška kontrolė kiekvienai pylimo operacijai;
- automatinės ataskaitos, suderinimas su apskaitos programomis;
- galimybė nuomuoti savo degalinę kitoms įmonėms.

Norint pasirinkti tokį sprendimą reikia susipažinti ir su jo trūkumais:

- nemažos pradinės investicijos aparatūrinei ir programinei įrangai;
- papildoma degalų saugyklos eksploatavimo apskaita.

Įnešus pradines investicijas, sistema toliau praktiškai dirbs nemokamai (neskaičiuojant techninės bei programinės įrangos priežiūros ir remonto kainų, kurios sudaro palyginus nedidelę visos sistemos eksploatavimo išlaidų dalį).

Apibendrinus galima pasakyti, kad sistemos pagrindinis tikslas – tai technologiniais sprendimais leisti užsakovui pateikti ekonomiškai patrauklius integruotus sprendimus savo produkcijos naudotojams.

## 3.3. Sprendžiami uždaviniai

Šiame skyriuje apžvelgiami projekto tikslai iš programų inžinerijos pusės. Toliau pateikiamas sistemos sprendžiamų uždavinių sąrašas su trumpais paaiškinimais:

- Sistema turi atlikti Dresser Wayne tipo kuro kolonėlių valdymo darbą. Tai vienas iš pagrindinių užsakovo reikalavimų. Kuro kolonėlės dažniausia dirba dvejopai: autonomiškai arba su valdančiu įrenginiu. Šio projekto tikslas buvo sukurti sistemą, kuri valdytų kuro kolonėles.
- Sistema turi dirbti autonomiškai, be žmogaus įsikišimo. Žmogaus darbo užmokestis yra santykinai didelė sistemos eksploatavimo išlaidų dalis. Dėl šios priežasties reikia sukurti sistemą, kuri, atlikus instaliacijos darbus, toliau galėtų dirbti nepriklausomai. Toks uždavinys įneša ne vien tik papildomus reikalavimus sistemos vykdymo charakteristikoms bei nefunkciniams reikalavimams, bet tuo pačiu ir priverčia ieškoti betarpiško vartotojo ir sistemos bendravimo variantų.
- Pilna atliekamų operacijų registracija. Visi sistemos atliekami veiksmai turi būti pilnai dokumentuoti. Kadangi galutinė programa turės tiesiogiai dirbti su materialiniu turtu, visi vykdymo eigoje atliekami veiksmai turi būti registruojami ir vėliau lengvai peržiūrimi ar tikrinami.
- Reikia realizuoti vartotojo sąsają su sistema be papildomo tarpininko. Vartotojas šioje sistemoje nedirbs tiesiogiai su kompiuteriu ar kompiuterine programa. Vartotojas dirbs su kuro kolonėlėmis (pils kurą). Programa turi žinoti, kuris vartotojas pasinaudojo konkrečia kuro kolonėle, kad galėtų atlikti pilną išduodamo kuro proceso dokumentavimą.
- Reikia atlikti minimalias kuro apskaitos funkcijas. Sistema turi žinoti informaciją apie kuro likučius, kad galėtų atlikti kuro išdavimo funkcijas. Tai reiškia, kad reikės sekti kuro pajamavimus, likučius ir pylimus. Tokia informacija bus reikalingo ir atliekant sistemos patikrinimą ar auditą.
- Turi būti realizuotas lankstus mechanizmas sukauptiems duomenims atvaizduoti. Tai apima duomenų atvaizdavimų ekrane, ataskaitų spausdinimo, duomenų eksporto ir kitas funkcijas.

Čia pateikti tik bendri sistemos sprendžiami uždaviniai, kurie realizacijos metu gali būti toliau smulkinami ir koreguojami. Pagal šiuos pateiktus uždavinius taip pat galima susidaryti bendrą sistemos panaudos atvejų schemą.

Dėl šių uždavinių iškylantys papildomi funkciniai ir nefunkciniai sistemos reikalavimai bei kitos problemos bus aptariamos projektinėje ir tiriamoje darbo dalyse.

## *3.4. Galimi uždavinių sprendimo būdai*

Pažvelgus į sistemą iš sprendžiamų uždavinių pusės galima pasiūlyti keletą šiame darbe keliamoms problemoms tinkamų spręsti būdų.

### 3.4.1. Integruota sistema

Pirmas būtų techniškai sudėtingiausias, kainos atžvilgiu nelabai palankus, tačiau panašių produktų realizacijose plačiai naudojamas sprendimas – sukurti integruotą kuro valdymo sistemą. Tokioje sistemoje būtų naudojama fiksuota techninė bazė, firminis (*proprietary*) procesorius ir operacinė sistema, kuriai programuoti dažniausia būna pateikiamas visas specializuotų įrankių rinkinys.

Pagrindinis tokio sprendimo privalumas – tai sąlyginis sistemos vykdymo stabilumas. Sistemoje nereikia rūpintis kitais vykdomais procesais, atminties trūkumais, procesoriaus užimtumu ar kitais pašaliniais trukdžiais. Integruoti įrenginiai kuriami tik tam, kad atliktų vieną specifinį darbą. Procesorius vykdo tik vieną programą. Pagrindinės tokio sprendimo problemos, tai aukštas technologinis sudėtingumas, būtini papildomi apmokymai, sudėtingas klaidų taisymas bei išaugusi produkto kaina. Pasirinkus šį sprendimą taip pat atsirastų ir duomenų saugojimo bei perdavimo klausimas. Integruotuose įrenginiuose atmintis dažniausia būna ribota, o šioje sistemoje planuojama naudoti vidutinio dydžio duomenų rinkinius, kurie gali tiesiog netilpti į egzistuojančią darbinę atmintį, jei bus pasirinkta integruotos sistemos realizacija.

### 3.4.2. Paprasta operacinė sistema

Kitas galimas sprendimas būtų atsisakyti jau minėtos integruotos sistemos techninės įrangos, bet pasilikti paprastą operacinę sistemą. Tokios sistemos pavyzdys būtų programos realizacija MS-DOS operacinėje sistemoje. Iš pirmo žvilgsnio gali atrodyti, kad sistemos realizacija tokioje sistemoje reiškia pasilikti visus integruotos sistemos privalumus ir atsisakyti visų trūkumų, tačiau tai nėra taip paprasta.

Visų pirma, tai reikštų pasenusios technologijos pritaikymą naujoje sistemoje. Problemiškas būtų ir naujausių technologijų palaikymas (pavyzdžiui, naujos tinklo plokštės įdiegimas). Pasirinkus tokį kelią, neišvengiamai bus susiduriama su senos aparatūrinės įrangos palaikymo klausimais, programavimo sunkumais ar net pačių projektavimo įrankių trūkumu. Taip pat atsirastų griežti vartotojo sąsajos ir spausdinamų ataskaitų galimybių apribojimai.

### 3.4.3. Moderni nemokama operacinė sistema

Antrojo sprendimo alternatyva būtų kurti sistemą naujesnėje operacinėje sistemoje (pavyzdžiui, Linux). Šioje sistemoje galima rasti visus naujausius programavimo paketus ir pasirinkus tokį sprendimo kelią neliktų problemų dėl senos ar naujos aparatūrinės įrangos palaikymo. Tačiau didžiausios šio sprendimo problemos būtų vartotojų priešiškumas ir sistemos kūrėjų kompetencijos trūkumas. Galiausiai nukentėtų pačios sistemos kūrimas (pasikeistų terminai). Įvertinta, kad sumažėjusi operacinės sistemos kaina, neatitiks padidėjusių sistemos kūrimo ir eksploatavimo išlaidų.

### 3.4.4. Galutinis sprendimas

Atsižvelgus į visus aukščiau išvardintus sprendimus, jų trūkumus ir pranašumus, buvo nuspręsta sistemą kurti standartinėje Microsoft Windows operacinės sistemos bazėje. Visų pirma, tai (sistemos kūrimo metu) pati populiariausia operacinė sistema. Didžiausia vartotojų ir sistemos kūrėjų patirtis. Sukaupta plati palaikymo ir problemų sprendimo informacinė bazė. Microsoft Windows – tai puiki aplinka komerciniams produktams kurti, dėl savo populiarumo susilaukusi daugelio programinės ir aparatūrinės įrangos gamintojų palaikymo. Šioje aplinkoje taip pat galima rasti plačiausią pagalbinių įrankių asortimentą (nuo duomenų bazių iki ataskaitų formavimo įrankių).

Pagrindiniai faktoriai, kurie leido priimti tokį sprendimą, tai užsakovo klientų pasiruošimas ir noras naudoti šią operacinę sistemą bei joje realizuotą programinę įrangą ir didžiausia projekto vykdytojų patirtis šioje srityje. Taip pat, šis pasirinkimas leido patenkinti ir kitus užsakovo reikalavimus: maža sistemos (aparatūros) kaina ir sudėtingumas. Personalinių kompiuterių įranga ir komponentai šiuo metu lengviausiai prieinami rinkoje, tuo pačiu ir pateikiami geriausiomis kainomis.

## *3.5. Egzistuojantys sprendimai*

Šiuo metu rinkoje visiškai analogiško komercinio produkto nėra, todėl sunku kuriamą sprendimą lyginti su konkrečia komercine realizacija. Panašios kuro kolonėlių valdymo programos yra diegiamos visose degalinėse, tačiau tokiose programose nėra numatytas autonominio sistemos veikimo darbo režimas, todėl tokios sistemos negali visiškai užpildyti šio rinkos segmento. Šį produktą geriausiai atitiktų automatinės degalinės, tačiau jos skiriasi vienu pagrindiniu aspektu – jose naudojami pinigai, dėl ko turi būti vedama buhalterija, samdoma apsauga, inkasacija ir t.t.

Dėl analogiško produkto rinkoje nebuvimo projektą lyginti galime tik pagal konkrečius aspektus, bendras funkcijas ir pagrindinius veikimo principus. Kuro kolonėlės praktiškai gali dirbti kelėtoje skirtingų darbo režimų:

- autonominis režimas (kuro kolonėlė atlieka grynai tik pompos ir dozatoriaus vaidmenį, jokia apskaita neatliekama);
- neautonominis režimas (kuro kolonėlę valdo išorinė programa, kuri gali atlikti ir pylimų apskaitą).

Pastarąjį sprendimą galime suskirstyti į tris pagrindines grupes:

- nutolęs valdymo pultas;
- autonominė valdymo programa;
- degalinės valdymo programa.

Šios grupės bus aptariamos sekančiuose skyriuose.

### 3.5.1. Automatinės degalinės

Tai degalinės, kurias specialiais tikslais šiuo metu eksploatuoja didieji Lietuvos degalų tiekėjai (pavyzdžiui, Neste arba Statoil Lietuva). Šiose degalinėse kuras išduodamas automatiškai, be žmogaus įsikišimo, tačiau šioje degalinėse galima pastebėti vieną svarbų skirtumą – jose naudojami gryni pinigai. Grynų pinigų naudojimo atveju atsiranda visa eilė įvairių degalinės darbo komplikacijų. Reikia vesti buhalteriją, atsakyti už mokesčių apskaitą, samdyti apsaugą ir inkasaciją. Planuojamoje kurti sistemoje gryni pinigai nenaudojami. Kiekvienas vartotojas kolonėlėje identifikuojamas specialios priemonės pagalba, o jo atliktus pylimus toliau galima apmokestinti pagal įmonės poreikius (pavyzdžiui, pavedimu).

### 3.5.2. Dresser Wayne iCON

Šiame produkte realizuota nutolusio kolonėlės valdymo pulto idėja. Tai paprastas konsolinis įrenginys, kuris gali būti per atstumą jungiamas prie kuro kolonėlės ir po to būti naudojamas kaip kolonėlės valdymo pultas. Šis produktas išsprendžia tik vieną problemą – suteikia galimybę valdyti pylimo procesą. Be šio įrenginio kuro kolonėlė veiktų kaip paprasčiausias kuro siurblys. iCON produktas leidžia:

- palaikyti ryšį su kuro kolonėle;
- valdyti pylimo operacijas;
- ekrane matyti kolonėlės displėjaus parodymus.

Atliekant pylimus iCON konsolė nekaupia jokių duomenų. Norint dirbti su produktu, reikia kad nuolat dalyvautų žmogus, kuris galėtų atlikti operatoriaus rolę. Pagrindiniai produkto trūkumai:

- nėra informacijos kaupimo ir formavimo įrankių;
- negeneruojamos jokios ataskaitos;
- būtinai turi dalyvauti žmogus;
- teikiamos tik bazinės kolonėlės valdymo funkcijos.

Šis produktas naudojamas mažiems rinkos segmentams užpildyti, kuriuose būtų galima sėkmingiau pritaikyti automatizuotą degalinės valdymo programinę įrangą.

### 3.5.3. UVS POSFuel

Tai visavertė degalinės valdymo programa, kuri jau ilgą laiką yra diegiama ir naudojama įvairiuose Lietuvos degalinėse. Pagrindiniai sistemos privalumai:

- platus funkcijų pasirinkimas;
- atlieka ne vien degalų valdymo, bet ir parduotuvės funkcijas;
- galimybė dirbti su kortelėmis;
- galimybė dirbti su nuolaidomis, kuro rezervuarais ir t.t.

Kadangi ši sistema turi beveik visas degalinės valdymo sritis apimančias funkcijas, ji tuo pačiu yra ir per daug didelė nuosavai degalinei valdyti. Pagrindiniai produkto trūkumai:

- funkcijų pertekliškumas;
- per daug didelė kaina;
- pasenusi, nelanksti ir neintuityvi vartotojo sąsaja;
- darbuotojo, kuris turės atlikti operatoriaus rolę poreikis;
- tai fiskalizuotas kasos aparatas, su kuriuo dirbant reikia laikytis visų prekybos vietose galiojančių įstatymų.

Ši sistema populiari Lietuvoje. Programa naudojama daugumoje naujai statomų degalinių, o taip pat ir senesnėse degalinėse (apribota programos versija). Tokio tipo produktas nėra naudojamas nė vienoje privačioje degalinėje dėl aukščiau paminėtų produkto trūkumų.

### 3.5.4. IPS MasterPOS

Tai panašus degalinės kasos aparatas, kaip ir anksčiau minėtas UVS POSFuel, tačiau realizuotas naujesnėje techninėje bazėje. Produktas turi panašius trūkumus, kaip ir visos kasos, tačiau turi šiuos papildomus pranašumus:

- galimybė dirbti be grynų pinigų;
- galimybė atpažinti vartotojus per specializuotus identifikavimo įrenginius.

Pasak gamintojo, jų produktas yra naudojamas kelėtoje specializuotų privačių degalinių, tačiau išlieka viena didelė problema – tai produkto kaina. Kasos aparate realizuota labai daug funkcijų, kurių niekada nepavyks panaudoti privačioje degalinėje, tačiau už kurias vis vien reikia sumokėti gamintojui.

## *3.6. Esminės problemos*

### 3.6.1. Bendro pobūdžio problemos

Prieš pradedant analizuoti technologines projekto įgyvendinimo problemas, reikia atkreipti dėmesį į problemas, kylančias dėl kitų priežasčių. Dauguma šių priežasčių atsiranda tiesiogiai dėl projekto nefunkcinių reikalavimų, dėl to ir pradėsime problemų analizę būtent nuo jų.

#### 3.6.1.1. Saugumo klausimai

Pirmi pagal svarbumą yra saugumo reikalavimai. Sistemos saugumą galima suskirstyti į tris pagrindines dalis:

- saugus sistemos darbas (nekenkia aplinkai ar darbuotojams);
- fizinis sistemos saugumas;
- programinės įrangos saugumas.

Pirmas punktas susijęs su aplinka. Sistema turi dirbti taip, kad aplinkai ir su sistema dirbantiems darbuotojams nebūtų iškeltas joks pavojus ir nepadaryta žala. Jei to padaryti neįmanoma, tada reikia projektą kurti taip, kad šie pavojai būtų minimizuoti.

Dirbant su kuro kolonėlėmis dažniausia tenka susidurti su aplinkos apsaugos klausimais. Pačios kuro pylimo kolonėlės suprojektuotos tokiu būdu, kad avarijos atveju žala aplinkai būtų minimizuota, tačiau valdymo programinėje įrangoje reikia įdėti papildomas apsaugas, kurios, pavyzdžiui, neleistų įsipilti daugiau kaip 100 litrų degalų, jei degalinė aptarnauja tik lengvojo tipo automobilius.

Fizinis sistemos saugumas labiau priklauso nuo sistemą eksploatuojančios įmonės politikos ir ten dirbančių kompiuterių administratorių. Jei fizinis sistemos saugumas nebus užtikrintas, tai jokios programinės priemonės šią problemą išspręsti nepadės.

Programinė įranga turi būti kuriama atsižvelgiant į projekto analizės metu nustatytus saugumo reikalavimus. Jei su sistema dirbs daugelis žmonių, tai atitinkamai reikia numatyti galimybę skirtingiems vartotojams suteikti skirtingas priėjimo prie sistemos teises.

#### 3.6.1.2. Pasiekiamumo problemos

Viena svarbiausių problemų, kuri turi būti išspręsta projektuojant kuro kolonėlių valdymo programą, tai visiškas sistemos pasiekiamumas. Jei techninėje įrangoje nėra gedimų, tai kolonėlė turi pilti degalus. Problema yra ta, kad dauguma degalinių dirba ištisą parą, o net ir mažesnėse nuosavose degalinėse daugiau kaip šešių valandų prastovos gali atnešti milžiniškus nuostolius. Todėl programinės įrangos projekto vykdymo metu buvo svarbu į tai tinkamai atsižvelgti.

#### 3.6.1.3. Patikimumo ir tikslumo reikalavimai

Pats svarbiausias reikalavimas šioje sistemoje yra patikimas ir tikslus sistemos darbas. Niekam ne paslaptis, kad kuro kainos šiais laikais yra milžiniškos, todėl netgi padarius keleto litrų paklaidą gali atsirasti dideli trūkumai buhalterijoje. Prie šios problemos tik prisideda faktai, kad programa turi veikti visą parą septynias dienas per savaitę, o į sistemą pakliūnantys duomenys ateina iš kartais sunkiai ar nepatikimai veikiančios kuro pylimo kolonėlės. Taip pat gali pasitaikyti įvairios ryšio problemos, kurios dalinai yra išspręstos protokolo specifikacijoje, bet niekada nebus galima įvertinti visų išorinių sąlygų (pavyzdžiui, ugnis, nuplėštas kolonėlės pistoletas, žaibas ir pan.).

### 3.6.2. Techninės problemos

Technines projekto įgyvendinimo problemas beveik tiesiogiai atspindi funkciniai sistemos reikalavimai, tačiau kūrimo metu būtinai reikia neužmiršti ir nefunkcinių reikalavimų, dėl kurių reikia priimti ne ką mažiau svarbius architektūros ir programinės įrangos projektavimo sprendimus.

#### 3.6.2.1. Pagrindinės kliūtys

Pagrindinės problemos, su kuriomis susiduriama kuriant panašaus pobūdžio sistemas, tai:

- programos greito veikimo užtikrinimas;
- atminties valdymas (ilgalaikio programos vykdymo metu);
- informacijos valdymas, saugumo ir vientisumo užtikrinimas.

Šiame projekte taip pat papildomai yra visa eilė nefunkcinių reikalavimų, kurie savo ruožtu taip pat labai apsunkina projektavimo ir diegimo darbus.

Greito veikimo reikalavimas pats trivialiausias. Akivaizdu, kad sistema negali vėlinti svarbių veiksmų. Nuo sistemos sugebėjimo laiku atlikti savo funkcijas priklauso jos veikimo teisingumas.

Atminties valdymas tai aktuali problema tokio tipo sistemose, kuriuose daug dirbama su dideliais duomenų kiekiais. Nenaudojant specializuotos atminties valdymo programinės įrangos atsiranda poreikis išsamiai testuoti galutinę programinę įrangą, kad įsitikinti, jog joje nėra likę sistemos veikimui kritinių resursų valdymo klaidų. Atminties nuotėkis ne tik pačios sistemos problema, bet tai tuo pačiu stabdo ir visus operacinės sistemos procesus, mažina stabilumą, o galiausiai sistema apskritai pasidaro nebepasiekiama. Ši problema yra ne pačios operacinės sistemos trūkumas, bet grynai projektavimo metodologijos arba technologijos pasirinkimo problema. Augant sistemų sudėtingumui ir vis trumpėjant darbų terminams programuotojai neišvengiamai susiduria su kodo klaidomis. Reikia įvertinti šį niuansą ir surasti būdų kaip jį įveikti.

Su informacijos kaupimo problemomis susiduriama tada, kai reikia saugoti ne vien rezultatus, bet ir tarpinius bei pradinius duomenis, programų būsenas, veiksmų eigas ir kt. Kadangi tikralaikės sistemos dirba su aplinka, jų renkama informacija dažniausia būna tokio tipo, kurios neįmanoma suskaičiuoti ar atkurti pagal apibendrintus rezultatus. Kuriant projektą būtinai reikia į tai atsižvelgti, nes įvykus avarinei situacijai sistemos naudotojai dėl duomenų praradimo gali atsidurti aklagatvyje.

#### 3.6.2.2. Autonominis veikimas

Tai apribojimas, nurodantis, kad sistema (po sėkmingų diegimo darbų) turi dirbti automatizuotai, be žmogaus įsikišimo. Tokia galimybė šio tipo sistemoje yra viena iš pagrindinių sistemos kūrimo priežasčių. Veikimo apribojimas nurodo, kad normalaus darbo metu sistema turi priimti sprendimus be žmogaus įsikišimo, pagal iš anksto numatytas taisykles. Toks apribojimas atsirado, nes privačių įmonių degalinėse dažniausia būna susiklosčiusi tokia darbo aplinka, kurioje degalinės darbuotojas (operatorius) turi atlikti grynai mechanišką darbą (leidžia pilti degalus, į kelionės lapą užrašo, kiek degalų užpilta). Tokias funkcijas vietoj operatoriaus būtinai turės atlikti sistema.

#### 3.6.2.3. Sistemos vartotojų atpažinimo mechanizmas

Degalų valdymo sistema panaši į materialiai atsakingo asmens valdymo sistemą tuo požiūriu, kad išpiltas kuras visąlaik būtinai turi priklausyti konkrečiam objektui (įmonės darbuotojui, vairuotojui, automobiliui ar pan.). Dėl to sistemoje turi būti įdiegta paprasta, saugi ir patikima vartotojų atpažinimo sistema, kurios pagrindu būtų galima sekti degalų judėjimą įmonėje, skaičiuoti automobilių degalų sąnaudas, spausdinti degalų sunaudojimo ataskaitas ir kt. Tai svarbi problema, nes neįdiegus tinkamo jos sprendimo sistema negalės normaliai funkcionuoti.

#### 3.6.2.4. Dresser Wayne tipo kolonėlių ryšio protokolo palaikymas

Sukurti patikimą ir stabilų ryšį su išoriniu (ne personalinio kompiuterio bazės) įrenginiu visada bus savotiška problema. Sistema turi valdyti Dresser Wayne tipo kolonėles, kurios palaiko DART protokolą. Tai specifinis užsakovo apribojimas, kuris užtikrina, kad sukurta programinė įranga dirbs teisingai su užsakovo diegiama technine įranga (šiuo atveju – degalų pilstymo kolonėlėmis). Sistema turi programiškai realizuoti DART protokolą ir veikimo metu atlikti valdytojo (*master*) rolę. Poreikis realizuoti ryšio protokolą Microsoft Windows sistemoje tik dar labiau apsunkiną šią problemą, nes papildomai atsiranda sinchronizavimo, resursų valdymo ir stabilumo išlaikymo klausimai.

#### 3.6.2.5. Diegimo aplinka

Prie techninių problemų reikia paminėti ir pačios diegimo aplinkos stabilumą. Sistema bus instaliuojama degalinėje esančiame personaliniame kompiuteryje. Ne paslaptis, kad personaliniai kompiuteriai nėra labai patikimi. Taip pat atsiranda ir papildomos diegimo aplinkos problemos: kaip sistema dirbs dingus elektrai, trenkus žaibui, kaip bus daromos duomenų kopijos ir kaip bus galima išvengti arba atstatyti sistemą po kritinių kompiuterio techninės įrangos gedimų. Visa tai reikia įvertinti iš anksto arba vėliau gali tekti susidurti su nenumatytomis sunkiai išsprendžiamomis problemomis.

Gali iškilti problemų dėl ryšio kanalo. Kompiuteris dažniausia statomas pastate greta aikštelės, kuriose eksploatuojamos kuro kolonėlės, o kartais kompiuterį reikia iškelti dar toliau. Standartinė RS-232 sąsaja, kuria bus perduodami duomenys tarp kompiuterio ir kolonėlės gali veikti tik maždaug 30-60 metrų atstumu. Reikia sugalvoti, kaip šį atstumą padidinti iki 1000 metrų. Tai labiau techninė problema, kuri tiesiogiai įtakos programinei sistemai neturės, tačiau dėl šių apribojimų gali atsirasti problemos ryšio protokolo programavimo metu.

## *3.7. Rekomendacijos problemų sprendimui*

Šiame skyriuje bus iš eilės apžvelgiamos praeitame skyriuje išvardintos problemos ir pateikiami sprendimai ir pasiūlymai šioms problemoms spręsti.

### 3.7.1. Rekomendacijos ir reikalavimai aplinkai

Pirmiausia reikia aptarti bazinius dalykus. Šiame skyriuje apžvelgsime operacinės sistemos ar projektavimo aplinkos funkcijas, kurios yra būtinos šios sistemos kūrimui. Galima išsiversti ir ne su visomis išvardintomis funkcijomis, tačiau atsisakant jų tuo pačiu reikia atsisakyti ir

sistemos paprastumo, suprantamumo ir greito veikimo. Toliau pateikiamas apibendrintų rekomendacijų sąrašas.

- Atminties valdymas. Automatinis atminties priskyrimo ir atlaisvinimo mechanizmas. Leidžia efektyviau panaudoti kompiuterio atmintį, o svarbiausia – išvengti sudėtingiausių programavimo klaidų – atminties resursų valdymo problemų (*memory leak*). Papildomas funkcionalumas gaunamas paaukojant programos vykdymo greitį. Dėl šios priežasties atminties valdymo programose nuolat ieškoma naujų optimalesnių ir greičiau veikiančių algoritmų, kurie leistų greičio sąnaudas sumažinti iki minimumo.
- Sinchronizacija ir resursų pasidalijimas. Neįmanoma kurti sudėtingos programos, jei nėra metodų tos programos veiksmams sinchronizuoti (t. y. sudėlioti ir vykdyti pagal numatytą tvarką). Lygiagrečiai vykdomos programos taip pat turi turėti galimybę dalintis tarpusavio resursais. Be šios funkcijos lygiagrečiai vykdomos programos negalėtų papildyti viena kitos skaičiavimus ir rezultatus, o svarbiausia – negalėtų būti efektyviai vykdomos sistemose su keletu procesorių.
- Asinchroninis įvykių valdymas. Tikralaikiu vykdomos sistemos beveik visada palaiko ryšį su aplinkoje vykstančiais įvykiais. Lyginant su programos vykdymo logika, aplinkoje vykstantys dariniai dažniausia yra asinchroniniai, tai reiškia, kad vyksta be iš anksto numatytos tvarkos. Tokiems įvykiams įvertinti ir išmatuoti operacinėje sistemoje turi būti numatytos asinchroninių įvykiu apdorojimo funkcijos, kurios nenutraukdamos programos vykdymo galėtų leisti programai bendrauti su aplinka. Ši tema bus plačiai analizuojama darbo tiriamoje dalyje.
- Asinchroninis valdymo perdavimas. Kartais aplinkos įvykiai keičiasi taip greitai ir nenumatomai, kad naudojant net ir labai gerai išvystytą programos logikos branduolį vis tiek nepavyktų sukurti optimalaus įvykių apdorojimo mechanizmo. Tokioms situacijoms įveikti naudojamas asinchroninis programos valdymo perdavimas, kur programos vykdymo tvarką nustato ne jos algoritmas, bet išoriniai įvykiai. Panašiai, kaip ir su asinchroniniu įvykiu valdymu, šis klausimas bus plačiau analizuojamas darbo tiriamoje dalyje.
- Tiesioginis atminties valdymas. Vertinant visus sistemai iškeltus reikalavimus, tai nėra kritinis būtinai reikalingas funkcionalumas, tačiau dažnai yra pageidaujamų įrankių sąraše, nes leidžia kūrėjams kurti greitesnius, paprastesnius ir labiau integruotus sprendimus, nei naudojant standartines operacinės sistemos funkcijas ir konstrukcijas.

### 3.7.2. Sistemos saugumas ir pasiekiamumas

Saugumo ir patikimumo problemas gana efektyviai galima išspręsti pritaikius standartines programavimo [15] ir operacinės sistemos [16] aplinkos funkcijas. Standartinės operacinės sistemos saugumo priemonės šiuo atveju yra pakankamai geros.

Norint užtikrinti sistemos pasiekiamumą reikia peržiūrėti kritinius sistemos modulius ir numatyti kaip, ir kokioje aplinkoje jie bus realizuoti. Microsoft Windows sistemose šiuo metu sparčiai plinta ir populiarėja .NET programavimo aplinka, kurioje įdiegtos visos pagrindinės anksčiau išvardintos savybės, kurių reikia tokio tipo sistemos vykdymui palaikyti. Šioje sistemoje taip pat yra puikios galimybės valdyti programos saugumą iki žemiausio kodo lygmens.

Esminė problema, su kuria galima susidurti šio projekto metu – tai atminties nuotėkis, kuris gali atsirasti dėl nepilnai ištestuotų sistemos modulių darbo. Pasirinkus naują, stabilią ir lanksčią platformą (pavyzdžiui, Microsoft .NET ar Java) dauguma šių problemų praktiškai išsisprendžia savaime. Kritinius sistemos modulius, kurie tiesiai bendrauja su aparatūrine įranga galima realizuoti kitose projektavimo aplinkose, o modulius, kurie atlieka daug duomenų mainų operacijų, dirba su duomenų bazėmis ir ataskaitų failais galima realizuoti valdomoje aplinkoje.

### 3.7.3. Patikimumas ir tikslumas

Tai labiau programavimo kultūros ir naudojamų algoritmų klausimas, tačiau šiek tiek siejasi ir su kitais programos reikalavimais. Norint užtikrinti tikslų ir patikimą programos darbą, rekomenduojama naudoti standartinius programavimo metodus, atlikinėti duomenų validavimą, perteklinį duomenų tikrinimą ir kitus programinius sprendimus. Rekomenduojama visus programos atliekamus veiksmus su duomenimis aprašyti ir saugoti pagalbinėse duomenų saugyklose.

### 3.7.4. Autonominis veikimas

Šiam tikslui realizuoti reikia sukurti gerai apibrėžtą sistemos atliekamų veiksmų modelį. Reikia numatyti visus standartinius sistemos atliekamus veiksmus, tuo pačiu ir įvertinti visus numatomus trukdžius ar nesklandumus. Šiems veiksmams turi būti sukurti įvairių galimų vykdymo kelių planai. Reikia, kad būtų apibrėžti žingsniai, kuriuos reikia atlikti prie įvairių sistemos būsenų. Turi būti numatyta, kaip sistema gali pasielgti vykdymo metu atsiradus klaidoms.

Reikia suplanuoti ir klaidų prevencijos arba prisitaikymo prie klaidų planą. Nespecializuotų sistemų vykdymo metu įvykus klaidai, stabdoma visa sistema, prarandami duomenys ir

vartotojas savo darbą turi pradėti iš naujo. Šioje sistemoje tokia veiksmų seka negalima. Rekomenduojama nepalikti neapdorotų klaidų pranešimų, o ir pačias klaidas sutvarkyti taip, kad jos neturėtų ilgalaikės įtakos programos vykdymo charakteristikoms.

Išsamią klaidų apdorojimo metodiką apžvelgia D. Katz savo straipsnyje „Klaidų kodai ar išimtys. Kodėl taip sunku sukurti patikimą programinę įrangą“ [8].

Projektuojant autonomiškai veikiančią sistemą svarbu ne vien tik atlikti visus anksčiau išvardintus darbus, bet ir numatyti kiek galima daugiau variantų, pagal kuriuos sistema galėtų priimti numatytą ir įvertintą sprendimą be žmogaus įsikišimo.

### 3.7.5. Sąsajos su vartotoju sprendimai

Vartotojui teks bendrauti su sistema per dvi pagrindines sąsajas. Pirmoji sąsaja – tai kompiuterio ekrane matomas vykdomos programos vaizdas. Čia jokių papildomų problemų ir neaiškumų neiškyla. Antroji sąsaja yra ta, kurią vartotojas mato prie kuro kolonėlės. Tai yra kolonėlės ekranas.

Praktiškai neįmanoma parodyti vartotojui visų reikalingų pranešimų pritaikius vien tik tokį išvedimo įrenginį, todėl vartotojo sąsaja bus balansuojama tarp kompiuterio ir kolonėlės ekrano. Kolonėlė gali vartotojui parodyti, ar yra užmegztas ryšys su kompiuteriu ir leisti arba neleisti pradėti pylimą, bei pateikti informaciją apie pylimo vykdymą. Tuo atveju, kai nepavyksta sėkmingai pradėti pylimo išsamesnės informacijos reikia ieškoti kompiuteryje esančioje kuro valdymo sistemoje. Normaliu programos darbo atveju tai turėtų būti ne sistemos gedimo ar trukdžio, o išorinė priežastis (pavyzdžiui, pasibaigęs kuras).

Kompiuteryje esančioje vartotojo sąsajoje reikės įdiegti duomenims apdoroti reikalingas funkcijas, o taip pat ir vartotojo sąsają, kuri galėtų pranešti apie kuro kolonėlių būsenas.

Rekomenduojama kompiuteryje esančioje sąsajoje pateikti kiek galima detalesnius, tuo pačiu metu ir visiems lengvai suprantamus pranešimus.

### 3.7.6. Kuro valdymas ir vartotojų autorizavimas

Norint leisti vartotojui piltis kurą, sistemai pirmiausia reikia jį atpažinti. Kartais gali būti dar svarbiau žinoti, kokią transporto priemonę jis vairuoja. Lengviausia būtų suteikti galimybę vartotojams patiems prisistatyti sistemai, pavyzdžiui, įvedant unikalų savo numerį ir slaptažodį. Tačiau tai būtų ne vien tik nepraktiška (pavyzdžiui, lauko sąlygomis), bet ir labai apsunkintų pačius vartotojus ir kiekvienai pylimo operacijai atlikti reikėtų daug daugiau laiko. Kiekvieną kartą, kai žmogui suteikiama galimybė tiesiogiai įvesti informaciją į sistemą, tuo pačiu atsiranda ir klaidos padarymo tikimybė. Įmanoma dirbti ir su tokiais trūkumais, tačiau žymiai paprasčiau būtų pasirinkti kitokią vartotojų autorizavimo metodiką.

Šiame darbe efektyviausia būtų pritaikyti kortelių sistemą, kuri unikaliai leistų identifikuoti kiekvieną transporto priemonę ir vairuotoją. Šiuo metu rinkoje yra naudojamos trys pagrindinės technologijos:

- magnetinės kortelės;
- lustinės kortelės;
- nuotolinės kortelės.

Kiekviena kortelių rūšis turi savo privalumus ir trūkumus, tačiau šiam projektui nuspręsta naudoti nuotolinio veikimo korteles. Jos taip pat turi savų saugumo problemų, tačiau leidžia išvengti visų nepageidaujamų mechaninių, susidėvėjimo ar kitokių problemų.
Sistemos autorizavimo ciklą siūloma daryti keturių skirtingų lygmenų:

1) už kurą atsako žmogus, kuras leidžiamas automatiškai;
2) kuras išduodamas identifikuotam automobiliui;
3) kuras išduodamas identifikuotam vairuotojui;
4) kuras išduodamas identifikuotai porai: vairuotojui ir automobiliui.

Šie keturi pagrindiniai atvejai leidžia patenkinti visus iškeltus vartotojų reikalavimus.

### 3.7.7. Programinė realizacija personalinio kompiuterio bazėje

Techninę darbo realizaciją rekomenduojama atlikti personalinio kompiuterio techninės įrangos bazėje, panaudojant šiuo metu dažniausiai naudojamą ir labiausiai paplitusią programinę įrangą (Microsoft Windows ir/arba Microsoft Office). Lanksčias ir lengvai modifikuojamas sukauptų duomenų ataskaitas galima realizuoti Microsoft Access duomenų bazių valdymo sistemos aplinkoje. Šios rekomendacijos pateiktos tam, kad galutinės sistemos kaina ir palaikymo darbai reikalautų kiek galima mažesnių išlaidų ir resursų. Darbo metu siūloma panaudoti kiek galima daugiau egzistuojančių patikrintų ir praktikoje pasitvirtinusių modulių. Tai ne vien tik pagreitintų sistemos kūrimo procesą ir sumažintų galimų klaidų skaičių, bet tuo pačiu leistų sumažinti ir kuriamos sistemos kainą.

### 3.7.8. Ryšio ypatumai

Kuro kolonėlės neįmanoma pastatyti šalia valdančio kompiuterio, o ir kompiuterį nepraktiška eksploatuoti lauko sąlygomis, todėl reikia numatyti galimus nutolusio ryšio sprendimus.
Kompiuteryje ryšį galima užmegzti per USB, vietinį tinklą ir nuosekliąją (RS-232) jungtis, o kolonėlė palaiko tik RS-485 jungtį. Maksimalus atstumas tarp USB jungtimi sujungtų įrenginių yra nuo 3 iki 5 metrų, tai akivaizdu, kad ši jungtis šiam darbui netiks. RS-232 sąsaja palaiko atstumus nuo 30 iki 60 metrų, kas tam tikrais atvejais būtų pakankama, bet ne visada.

Lieka tik vietinio tinklo ir RS-485 jungtys, tačiau norint naudoti vietinio tinklo jungtį reikėtų gaminti ir diegti technologiškai sudėtingus ir brangius valdymo įrenginius, o ir maksimalus atstumas idealiomis sąlygomis būtų tik apie 100 metrų. Tai ne visada yra pakankama.

Dėl šių ribojančių parametrų ryšio linijose nuspręsta naudoti RS-485 sąsają, kurios maksimalus ryšio numatytas standartuose atstumas yra 1200 metrų [17]. Kompiuterio gale galima įdiegti pakankamai nesudėtingus ir pigius signalo keitiklius (RS-232 į/iš RS-485).

Duomenų perdavimas bus užtikrinamas keliais skirtingais lygiais: fiziniame, sąsajos ir protokolo. Sistemos programavimo metu reikia tinkamai realizuoti protokolo lygio duomenų perdavimo kontrolę. Šį darbą palengvins užsakovo pateikiama oficiali DART protokolo specifikacija.

## *3.8. Darbo rizikos įvertinimas*

Prieš pradedant vykdyti projektą reikėjo įvertinti, kokios bus pasekmės neįvykdžius tam tikrų sistemai keliamų užduočių. Šiuo tikslu buvo sukurta projekto rizikos analizė ir atsitiktinumų valdymo planas.

### 3.8.1. Galimos sistemos kūrimo rizikos

Rizikos žala šioje lentelėse vertinama penkiabalėje sistemoje. 5 reiškia didžiausia žala. Rizikos tikimybė vertinama penkiabalėje sistemoje. 5 reiškia absoliuti pasitaikymo tikimybė.

**Lentelė 1. Sistemos rizikų įvertinimas**

| Rizika | Tikimybė | Žalos įvertinimas |
|---|---|---|
| Didelis spaudimas dėl darbų grafiko | 1 | 4 |
| Kintantys užsakovo reikalavimai | 2 | 3 |
| Prasta kokybė | 3 | 3 |
| Projekto atšaukimas | 1 | 4 |
| Neteisingai įvertintos kainos | 2 | 2 |
| Neteisingai įvertintas panaudojimo atvejų skaičius | 2 | 2 |
| Neteisingai įvertinti funkciniai reikalavimai | 3 | 2 |
| Nefunkcinių reikalavimų skaičius | 1 | 4 |

### 3.8.2. Atsitiktinumų valdymo planas

**Lentelė 2. Atsitiktinumų valdymo planas**

| Rizika | Valdymo planas |
|---|---|
| Neteisingai numatytas veiklos | Formalizuotas reikalavimų rinkimas. Papildomi susitikimai reikalavimams derinti. Sistemos |

| | |
|---|---|
| įvykių skaičius | prototipų kūrimas. |
| Didelis spaudimas dėl darbų grafiko | Funkcinių reikalavimų mažinimas arba sistemos kokybės mažinimas. |
| Besikeičiantys užsakovo reikalavimai | Formalizuotas reikalavimų rinkimus su individualiu suderinimu ir parašais. |
| Prasta kokybė | Papildomi testavimo darbai. Užsakovo įtraukimas į testavimą. Papildomas sistemos kūrimo laikas. |
| Projekto atšaukimas | Surinktos informacijos ir kodo panaudojimas ateities projektams. |
| Neteisingai įvertintos kainos | Derybos su užsakovu. Funkcinių reikalavimų mažinimas arba kokybės mažinimas. |
| Neteisingai įvertinti funkciniai reikalavimai | Formalizuotas funkcinių reikalavimų rinkimo procesas su detaliais panaudojimo atvejais. |
| Nefunkcinių reikalavimų skaičius | Daugmaž stabilus skaičius šiame projekte. Perspėti užsakovą, kad šie pasikeitimai sistemos kūrimo metu gali smarkiai padidinti viso projekto kainą. |

Šiuo metu Lietuvos rinkoje visiškai analogiško komercinio produkto nėra, todėl jam kurti atsiveria šiek tiek palankesnės sąlygos. Klientai neturi su kuo palyginti produkto, o ir pats užsakovas dėl šios priežasties nekelia per daug didelių reikalavimų sistemai.

## *3.9. Praktinis darbo rezultatas*

Užbaigus darbą gautas galutinis programinės įrangos produktas, kuris pristatytas užsakovui ir savo ruožtu toliau sėkmingai parduodamas ir eksploatuojamas užsakovo klientų įmonėse.
Taip pat, darbo vykdymo metu buvo sukurta programinės įrangos techninė dokumentacija. Visą dokumentų sąrašą sudaro:

- projekto paraiškos dokumentas;
- projektavimo metodologijos ir technologijos analizė;
- detalus projekto planas;
- reikalavimų specifikacija;
- architektūros specifikaciją;
- detalios architektūros specifikacija;
- programinės įrangos licencija;
- testavimo planas;
- kokybės analizės dokumentas;
- vartotojo dokumentacija ir kiti pagalbiniai dokumentai.

Dalis šių dokumentų medžiagos panaudota ir šiame darbe. Papildomai buvo sukurta ir pristatyta keletas projekto skaidrių bei pateikta atspausdinta produkto reklama. Taip pat darbų pradžioje buvo sukurta projekto informacinė sistema, kurioje yra saugoma visa su projektu susijusi dokumentacija.

## *3.10. Mokomasis projekto rezultatas*

Sunku pateikti konkrečius projekto mokomosios naudos rezultatus. Galime įvertinti tik įgytų žinių ir praktinės patirties naudą, kuri buvo įsisavinta projekto vykdymo metu.

Magistrinio darbo vykdymas leido geriau suprasti programinės inžinerijos srityje vykstančius veiksmus:

- įgytas praktinis supratimas apie bendrą programinės įrangos kūrimo proceso eigą;
- apžvelgti visi svarbiausi programinės įrangos kūrimo aspektai;
- susipažinta su programinės įrangos proceso valdymu ir planavimu;
- apžvelgta visa formali proceso dokumentacija nuo projekto paraiškos pateikimo iki vartotojo dokumentacijos ar instaliavimo instrukcijų;
- išmokta dirbti su svarbiais sistemos kūrimui žmonėmis (rinkti reikalavimus, derinti terminus, skirstyti darbus ir pan.);
- įgytos bazinės programinės įrangos projekto vadybai reikalingos žinios.

Galutinis šios mokomosios veiklos rezultatas yra šis magistro darbas.

# 4. Projektinė dalis

## 4.1. Įvadas

Šiame skyriuje nagrinėjamos techninės projekto detalės. Konkretizuojami priimti sprendimai, pateikiamos architektūros ir modulių diagramos, aptariami priimtų sprendimų rezultatai.
Pirmuose skyriuose trumpai aptariami priimti techniniai sprendimai. Vėliau apžvelgiamas bendras sistemos architektūros vaizdas, tuo pačiu pateikiamos esminės sistemos architektūrą paaiškinančios schemos. Galiausiai pateikiamas sistemos duomenų modelis ir aptariama projekto išeiga.

## 4.2. Apžvalga

Dokumente aiškinama, kaip sistemoje nuspręsta realizuoti ryšį su degalų kolonėlėmis ir kaip veikia pagrindinė kolonėlių valdymo programa.
Architektūros specifikacijoje bus kalbama apie:

- Ryšio modulio architektūrą ir priimtus sprendimus. Tai fizinio kolonėlės valdymo lygmuo. Čia realizuotas DART linijos protokolas, kuris per specialią techninę įrangą turi palaikyti ryšį su degalų kolonėlėmis.
- Kortelių valdymo modulio architektūrą ir kaip sistemoje dirba vartotojų identifikavimo mechanizmas.
- Valdymo programos architektūrą. Čia kalbama apie tai, kaip sistemoje susiejamos visos pagrindinės dalys į vientisą nuoseklų mechanizmą.

## 4.3. Architektūros pateikimas

Sistemos architektūra dokumentacijoje pateikiama pritaikius tokius pagrindinius dokumentacijos būdus:

- Panaudojimo atvejų vaizdas. Čia pateikiami pagrindiniai sistemos panaudojimo atvejai, kaip jie siejasi su programiniais sistemos komponentais. Kokias funkcijas konkretus komponentas turės atlikti ir kokią rolę kiekvienoje funkcijoje turės sistemos aktoriai.
- Statinis sistemos vaizdas. Tai bendras nekintantis sistemos architektūros vaizdas. Pateikiama UML klasių diagramų pagalba.
- Dinaminis sistemos vaizdas arba kaip sistema reaguoja į kintančius aplinkos poveikius. Pateikiamos pagrindinių sistemos veiksmų veiklos diagramos. Nurodomos

pagrindinių dalių būsenų diagramos ir sudėtingesnėms sistemoms operacijoms pateikiamos sekų diagramos bei jų struktūrogramos.

- Sistemos išdėstymo vaizdas. Šiose diagramose bus parodoma, kokią vietą programa užima tarp kitų sistemų, o taip pat kokie ryšiai sudaromi su jomis. Nurodomi informacijos perdavimo srautai tarp vidinių sistemos komponentų ir išorinių sistemų.
- Duomenų struktūrų vaizdas. Čia pateikiami pagrindinių sistemos duomenų struktūrų vaizdai ir jų aprašymai.

## *4.4. Priimti techniniai sprendimai*

### 4.4.1. Projektavimo aplinka

Įvertinus visus projektui iškeltus reikalavimus ir apribojimus nuspręsta projektą kurti Microsoft Windows operacinės sistemos aplinkoje naudojant Microsoft Visual Studio .NET projektavimo aplinką.

Microsoft Windows operacinė sistema buvo pasirinkta dėl didelės sukauptos patirties šioje srityje. Didelė patirtis ne vien tik palengvina projekto planavimo ir vykdymo darbus, bet tuo pačiu padeda sumažinti ir kuriamo projekto riziką bei apmokymo (darbuotojų ir būsimų vartotojų) kaštus. Didelę įtaką turėjo ir tai, kad ši operacinė sistema šiuo metu yra pati populiariausia rinkoje, kas savo ruožtu dar labiau palengvina sistemos projektuotojų darbą, nes tokioje aplinkoje yra didelis skaičius projektavimo įrankių, egzistuojančių modulių, problemų sprendimų ar net programinio kodo pavyzdžių.

Reikalavimai aparatūrinei įrangai pateikiami skyriuje „Numatoma eksploatacijos aplinka“. Ryšio pajungimo ir ryšio įrangos klausimais rūpinsis pats sistemos užsakovas.

Testavimui numatoma naudoti tikros kolonėlės simuliatorių. Tai įrenginys, kuris sukonstruotas iš pagrindinių tikros kolonėlės valdymo blokų. Šio simuliatoriaus dėka bus galima atlikti išsamesnius ir labiau atitinkančius tikrovę testavimo darbus.

### 4.4.2. Programavimo technologija

#### 4.4.2.1. Inžineriniai sprendimai

Projektą nuspręsta realizuoti objektiškai orientuotoje valdomoje (*managed*) programavimo kalboje C#. Šioje kalboje realizuotos programos dirba Microsoft .NET karkaso pagalba. Ryšio su kolonėlėmis ir kortelių skaitytuvais realizacijai pasirinkta C programavimo kalba. Skirtingas aplinkas pasirinkti buvo būtina, nes projektavimo metu .NET karkase vis dar nebuvo standartizuoto palaikymo nuoseklių jungčių (*serial port*) programavimui.

Numatoma išvengti didelių integravimo sunkumų, kurie galėtų iškilti programuojant skirtingomis programavimo kalbomis, nes visi programiniai moduliai bus kuriami, testuojami ir koreguojamo vienoje integruotoje projektavimo aplinkoje Visual Studio.

Testavimas bus vykdomas specializuotų testų pagalba. Šiam tikslui bus kuriami pagalbiniai programiniai įrankiai, kurių pagalba bus galima sudaryti norimus testus atskiriems moduliams arba galutinei programai.

Galutinėje programoje bus vykdomi trys darbiniai procesai (gijos). Du procesai reikalingi ryšiui su išoriniais įrenginiais palaikyti, o vienas (pagrindinis) procesas bus naudojamas veiksmams koordinuoti. Valdantis procesas bus paleidžiamas (žiūrėti statinį sistemos vaizdą) „Order processing“ klasės pagalba. Ryšį palaikantys procesai apklausia savo aptarnaujamus įrenginius ir gautus pranešimus saugo atmintyje, kol juos paima valdantis procesas. Šis procesas, priklausomai nuo sistemos būsenos, atlieka atitinkamus valdymo veiksmus.

#### 4.4.2.2. Duomenų bazė ir ataskaitos

Duomenų bazę nuspręsta saugoti Microsoft Access formate. Toks sprendimas priimtas dėl dviejų priežasčių. Pirmiausia, tai praktiškai visų projektavimo įrankių palaikomas universalus Microsoft programinės įrangos duomenų formatas, kurį lengva atsiradus poreikiams išplėsti iki normalios duomenų bazių valdymo sistemos. Naudojant šį duomenų formatą taip pat pavyksta išvengti specializuotos duomenų bazių valdymo sistemos diegimo bei palaikymo, kas ženkliai sumažina galutinės sistemos sudėtingumą ir kainą.

Pritaikius Microsoft Access duomenų bazę taip pat atsiranda galimybė panaudoti ir šios sistemos teikiamus duomenų analizės ir atvaizdavimo įrankius, kurie būna labai naudingi, kai sistemą reikia greitai ir nesudėtingai modifikuoti pagal kliento poreikius.

## *4.5. Numatoma eksploatacijos aplinka*

Planuojama programą diegti ir vartoti Windows tipo operacinėse sistemose su NT branduoliu (Windows 2000 ir naujesnėse sistemose). Aparatūrinė įranga turi būti suderinama su valdančiame kompiuteryje įdiegta operacine sistema, o tuo pačiu turi atitikti ir rekomenduojamus įrangos techninius parametrus.

### 4.5.1. Rekomendacijos kompiuterinei įrangai

Procesorius: Intel Pentium IV 2GHz (arba greitesnis);

Atmintis: 256 Mb (arba daugiau);

Kietas diskas: bent 10 Gb laisvos vietos;

COM portai: du laisvi portai arba daugiau (priklausomai nuo naudojamos įrangos);

### 4.5.2. Būtina programinė įranga

Windows 2000 SP4 (arba naujesnė operacinė sistema);
Microsoft Office 2000 (arba naujesnis);

### 4.5.3. Bendros rekomendacijos

Saugumo sumetimais rekomenduojama kompiuterį, kuriame bus naudojama kuro kolonėlių valdymo programa nejungti į bendrą kompiuterinį tinklą. Tuo pačiu rekomenduojama šio kompiuterio nejungti ir prie interneto. Siekiant užtikrinti duomenų saugumą taip pat reikia pasirūpinti fiziniu kompiuterio saugumu bei išjungti arba išimti visus įrenginius, kurių pagalba būtų galima daryti duomenų kopijas.
Jei sistemą izoliuoti nuo išorinių tinklų nėra praktiška, reikia pasirūpinti, kad būtų įdiegta visa reikalinga pagalbinė programinė įranga (pavyzdžiui, antivirusinė programa).
Rekomenduojama pastoviai parsiųsti ir įdiegti visus būtinus operacinės sistemos atnaujinimus.

## *4.6. Sprendimų įtaka sistemos architektūrai*

Sistemos architektūros vaizdas buvo sukurtas remiantis priimtais technologiniais bei projektavimo sprendimais. Ryšio palaikymo moduliai realizuoti C kalboje DLL bibliotekų pagalba. Jie dinamiškai prijungiami ir startuoja programos paleidimo metu. Visi kiti programos moduliai realizuoti objektiškai orientuotoje programavimo kalboje C#, kuri teikia puikias atminties ir duomenų bazių valdymo galimybes.
Sistemos projektavimo metu taip pat buvo nuspręsta atskirti sistemos valdymo modulį nuo duomenų atvaizdavimo modulio. Tai ne vien tik sistemą padaro lankstesne ir lengviau modifikuojama, bet tuo pačiu leidžia sumažinti visos galutinės sistemos sudėtingumą, klaidų skaičių ir išplėtimo galimybes. Duomenų atvaizdavimo ir ataskaitų generavimo modulis realizuotas Microsoft Access sistemoje pritaikius standartines priemones.

## *4.7. Projekto technologiniai apribojimai*

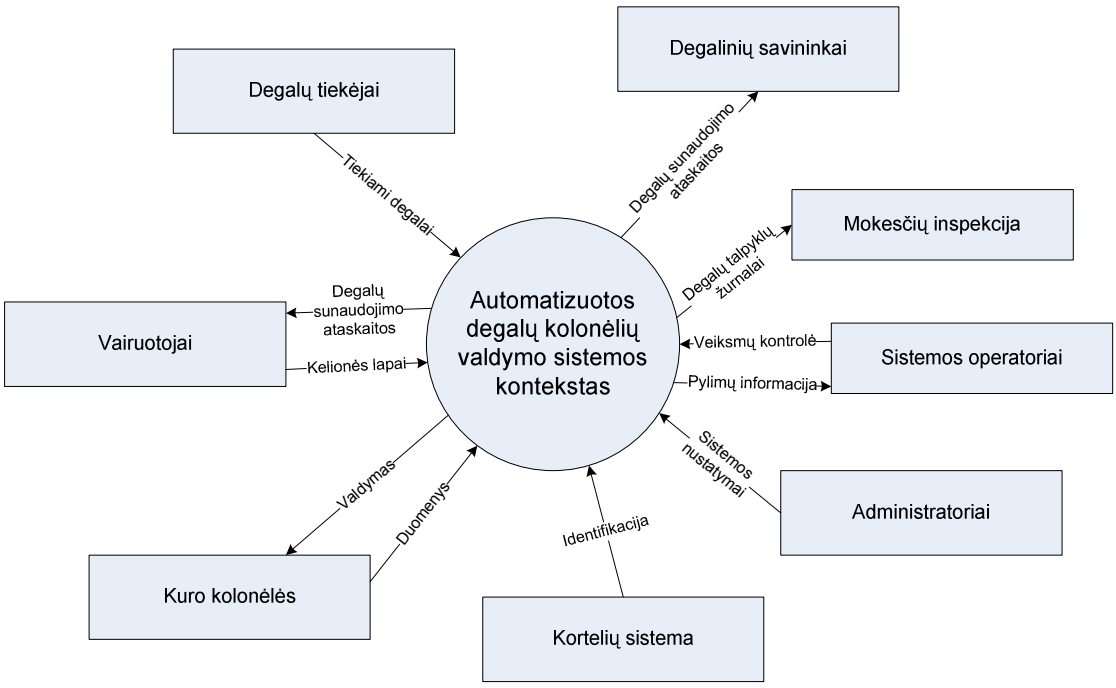
Pagrindiniai klausimai, kuriuos reikia išspręsti, tai degalų kolonėlių programinis valdymas ir šios programos modulių integravimas į vientisą sistemą.
Degalų kolonėlės dirba pagal standartinį DART protokolą. Bendravimas su jomis atliekamas per nuoseklų RS-232 prievadą. Visos kolonėlės dirba aptarnaujamame (*slave*) režime. Tai reiškia, kad atsakymai sugeneruojami tik tada, kai pateikiama teisinga užklausa. Taip pat reikia pažymėti ir tai, kad visos kolonėlės jungiamos (pagal protokolo specifikaciją) per vieną vytos poros laidą. Dėl šios priežasties galima vykdyti tik vienpusį bendravimą (*half-duplex*).

Su tokiais apribojimais reikia realizuoti programinį visų pajungtų kolonėlių valdymą. Didžiausia numatoma problema – laiko apribojimas. Komandos ir atsakymai turi būti siunčiami pagal protokolo aprašyme pateiktą tvarką, kas priverčia sistemoje taikyti apibrėžtus kodo vykdymo laiko apribojimus.

Antra sudėtinga problema, tai visų valdančių modulių integravimas į vientisą sistemą. Tai reiškia, kad visi sistemai pateikiami įėjimai ir išėjimai turi būti apdoroti teisingai ir laiku, o tuo pačiu priimant teisingus iš anksto numatytus valdymo sprendimus. Situaciją apsunkina dviejų realiu laiku dirbančių sistemų apjungimo klausimas (degalų kolonėlių valdymo modulis ir sprendimų priėmimo modulis). Reikės papildomą dėmesį skirti duomenų srautų ir veiksmų sinchronizacijai užtikrinti.

Microsoft Windows operacinės sistemos pasirinkimas taip pat įneša savotiškų apribojimų. Tokio tipo sistemose vienu metu vykdoma visa eilė tarpusavyje nesusijusių procesų, kurie gali neprognozuojamai pertraukti sistemos vykdymą, smarkiai apkrauti procesorių arba rezervuoti kitus kritiškai svarbius programos vykdymui resursus. Dėl šios priežasties patartina kuro kolonėlių valdymo sistemą diegti specialiai šiam darbui atlikti skirtame kompiuteryje. Ne visada bus galima vykdyti šią rekomendaciją, dėl to būtinai reikia į šią problemą atsižvelgti sistemos projektavimo metu.

## *4.8. Veiklos kontekstas*




**Pav. 1. Sistemos veiklos kontekstas**

Pagal šią schemą galime sudaryti bendrą supratimą apie funkcinius sistemos reikalavimus.

## 4.9. Veiklos padalinimas

Žvelgdami į sistemos funkcinius reikalavimus taip pat galime sistemos veiklas suskirstyti pagal apdorojamus informacijos srautus:

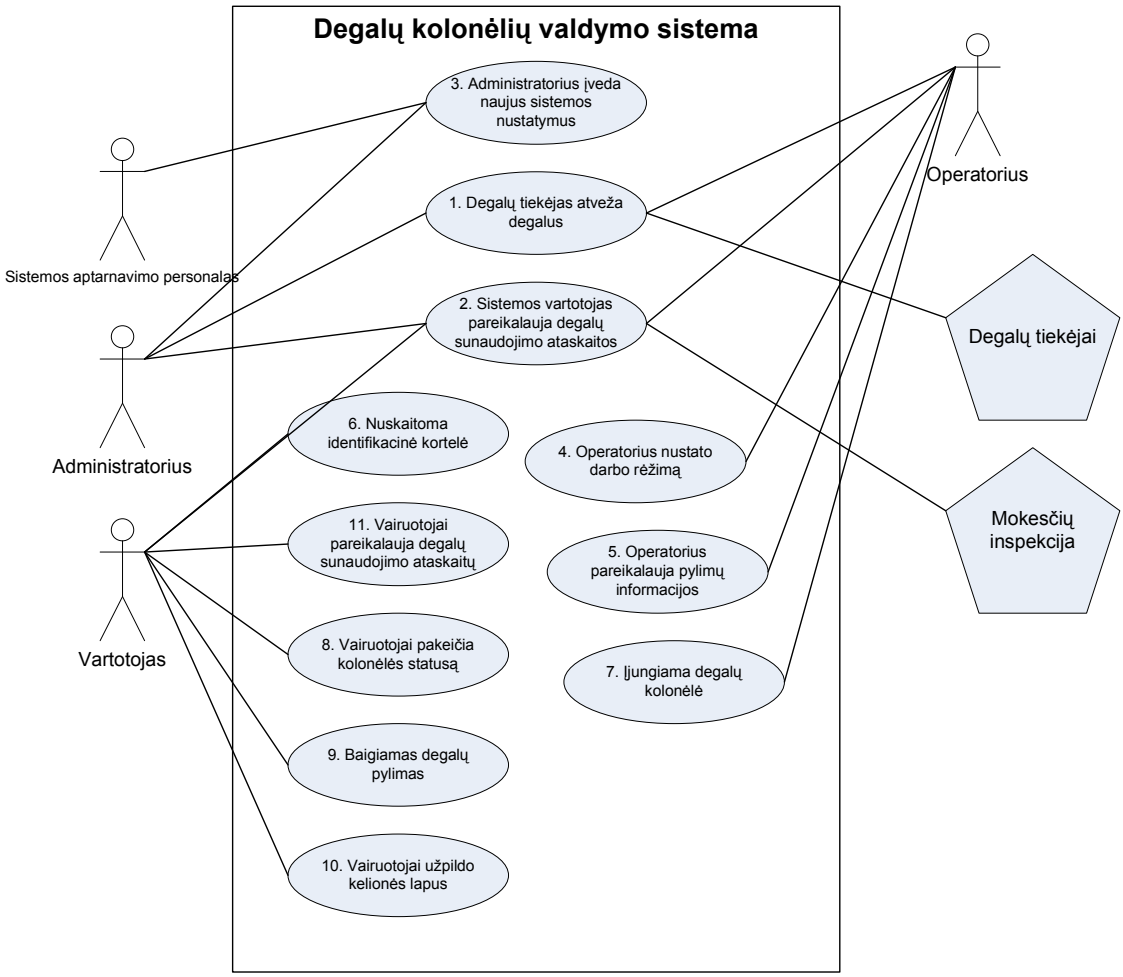
**Lentelė 3. Sistemos veiklos padalinimas**

| Eil. Nr. | Įvykio pavadinimas | Išeinantys/įeinantys informacijos srautai |
|---|---|---|
| 1 | Degalų tiekėjas atveža degalus | Degalų važtaraštis (*in*) |
| 2 | Sistemos vartotojas pareikalauja degalų sunaudojimo ataskaitos | Pylimų sąrašas (*out*) |
| 3 | Administratorius įveda naujus sistemos nustatymus | Sistemos nustatymai (*in*) |
| 4 | Operatorius nustato darbo režimą | Sistemos nustatymai (*in*) |
| 5 | Operatorius pareikalauja pylimų informacijos | Pylimų sąrašas (*out*) |
| 6 | Nuskaitoma identifikacinė kortelė | Kortelės numeris (*in*) |
| 7 | Įjungiama degalų kolonėlė | Valdymo programa (*out*) |
| 8 | Vairuotojai pakeičia kolonėlės statusą | Kolonėlės statusas (*in*) |
| 9 | Baigiamas degalų pylimas | Pylimo informacija (*in*) |
| 10 | Vairuotojai užpildo kelionės lapus | Kelionės lapai (*in*) |
| 11 | Vairuotojai pareikalauja degalų sunaudojimo ataskaitų | Pylimų sąrašas (*out*) |

Pasitelkdami informacija iš šios lentelės, galime susidaryti bendrą sistemos architektūros vaizdą. Matome atskirus modulius, bendrus duomenis ir ryšius tarp šių modulių ir jų duomenų.

## 4.10. Panaudojimo atvejai

Šioje diagramoje pavaizduotas ryšys tarp sistemos ir su ja susijusių elementų. Taip pat matome sistemos ribas, kokie objektai randasi už sistemos ribų ir kaip jie turės dirbti su sistemos teikiamomis funkcijomis:




**Pav. 2. Panaudos atvejų diagrama**

Pagal šią diagramą galime vartotojus suskirstyti pagal jiems reikalingas funkcijas. Šios schemos dėka taip pat galime apžvelgti ir bazinius sistemos saugumo reikalavimus (nes matome, prie kurių sistemos funkcijų vartotojai turi priėjimą, koks funkcionalumas jiems yra būtinai reikalingas).

## *4.11. Funkciniai reikalavimai*

Žemiau pateikiamas pilnas funkcinių sistemos reikalavimų sąrašas:

1) Kolonėlė degalus turi išduoti tik identifikuojamam objektui;
2) Sistemoje turi būti kaupiama detali informacija apie kiekvieną pylimą;
3) Kolonėlė turi dirbti tik tuo atveju, jei ją valdo kompiuteris;
4) Kolonėlė turi išduoti degalus nesustodama visą parą;
5) Sistemoje turi būti galimybė apriboti, kada galima naudotis sistemos paslaugomis (laiko apribojimai);
6) Sistemoje turi būti galimybė nurodyti maksimalų išduodamų degalų kiekį ir sumą;
7) Sistema turi unikaliai identifikuoti degalų vartotoją;

8) Sistemoje turi būti galimybė identifikuoti pylimą pagal dvigubą raktą (pvz., vairuotojas ir automobilis);

9) Turi būti galimybė apriboti vairuotojui leidžiamas užpilti transporto priemones;

10) Turi būti galimybė apriboti degalų išdavimą vartotojui per nurodytą laiko tarpą;

11) Sistemoje turi būti galimybė apriboti, kokias degalų rūšis gali naudoti vartotojas;

12) Sistema turi realizuoti DART protokolą;

13) Vartotojų identifikavimas turi būti vykdomas nenaudojant kompiuterio periferinius įrenginius (klaviatūra, ekranas, pelė);

14) Vartotojų identifikavimui turi būti naudojamos *proximity* tipo nuotolinio veikimo kortelės;

15) Sistema turi būti nesudėtingai pritaikoma darbui su bet kokia vartotojų identifikavimo sistema;

16) Vartotojas turi patvirtinti savo tapatybę įvesdamas PIN kodą ar kitokį slaptažodį;

17) Turi būti galimybė kolonėlę paleisti režime, kuris nereikalauja identifikacijos;

18) Turi būti galimybė pylimams priskirti vartotoją;

19) Sistema turi leisti vartotojus suskirstyti į grupes;

20) Priėjimas prie sistemos (darbinio kompiuterio) turi būti reguliuojamas pagal vartotojui priskirtas teises;

21) Turi būti galimybė kiekvienam pylimui priskirti vairuotojo kelionės lapo numerį;

22) Sistemos funkcionalumas turi būti prieinamas tik identifikuotiems vartotojams;

23) Sistemoje turi būti skaičiuojami teoriniai degalų sunaudojimo skaitikliai;

24) Turi būti įvesta rezervuarų informacija (tūris, numeris, kuro pavadinimas);

25) Turi būti sekami rezervuarų degalų likučiai;

26) Kolonėlės turi pilti degalus be papildomo operatoriaus įsikišimo;

27) Sistemoje turi būti kuro pajamavimų žurnalas;

28) Kolonėlė turi nepilti degalų, jei rezervuaruose nėra pakankamai likučių;

29) Degalų likučiai turi būti skaičiuojami pagal grupę;

30) Kortelėms turi būti įmanoma nustatyti galiojimo laiką;

31) Sistema turi dirbti balto sąrašo (white list) kortelių režime;

32) Turi būti galimybė uždrausti kortelės naudojimą;

33) Turi būti kolonėlės darbo režimas, kai operatorius nurodo kiek degalų ir kada vartotojas gali įsipilti (standartinės kasos režimas);
34) Sistemoje turi būti realizuota galimybė redaguoti standartinius sistemos nustatymus (pavyzdžiui, kortelių sąrašas, grupių sąrašas ir pan.);
35) Turi būti galimybė formuoti pylimų ataskaitas pagal pageidautinus parametrus (datą, laiką, vartotojus, grupes ir t.t.);
36) Reikia formuoti degalų pajamų-išlaidų ataskaitą;
37) Elektroninio žurnalo formavimas mokesčiams;
38) Galimybė duomenis eksportuoti į išorines sistemas.

## *4.12. Nefunkciniai reikalavimai*

Analitinėje darbo dalyje buvo bendrai užsiminta apie problemas, kurios kyla dėl sistemai keliamų nefunkcinių reikalavimų. Šiame skyriuje pateikiamas pilnas nefunkcinių reikalavimų sąrašas.

### 4.12.1. Reikalavimai sistemos išvaizdai

Ypatingų reikalavimų užsakovas vartotojo sąsajai nepateikė, todėl reikės vadovautis bendrais vartotojo sąsajų projektavimo principais.
Pagrindinis reikalavimas šiai sistemai – aiškus ir lengvai suprantamas pranešimas vartotojui apie sistemos darbinę būseną. Operatorius ar vartotojas turi, vien tik pažiūrėjęs į pagrindinį sistemos langą, suprasti ar sistema teisingai veikia. Jei sistema susidūrė su klaidomis, tai turi aiškiai būti pavaizduota ekrane. Tokie paprasti vartotojo sąsajos elementai ne vien tik padeda vartotojui lengviau suprasti ir išspręsti dažniausiai pasitaikančias sistemos problemas, bet tuo pačiu ir sumažina iki minimumo sistemos aptarnavimui (personalui) reikalingą laiką ir žmones.
Pagrindiniai vartotojo sąsajos projektavimo principai, kurių reikia laikytis šios sistemos projektavimo metu yra šie:

- lengvai skaitoma ir suprantama sąsaja;
- paprastas ir nesudėtingas sąsajos naudojimas;
- neįkyri sąsaja (kad vartotojas galėtų netrukdomai naudoti ir kitus programinės įrangos produktus);
- bendradarbiaujanti (save paaiškinanti) sąsaja.

Ši sistema kuriama tam, kad maksimaliai sumažinti reikalingus žmogaus darbo resursus. Tai reiškia, kad sistema didžiausią savo darbo laiko dirbs autonomiškai, be vartotojo įsikišimo.

Dėl šios priežasties vartotojo sąsajos projektavimui numatoma neskirti labai daug projektavimo resursų.

### 4.12.2. Reikalavimai panaudojamumui

Šioje srityje, panašiai kaip ir sistemos išvaizdai, daug resursų skirti nereikės (dėl aukščiau išvardintų priežasčių). Visi pagrindiniai sistemos pranešimai bus pateikiami nacionaline (lietuvių) kalba, todėl bet kuris darbuotojas galės atlikti bent minimalius sistemos diagnostikos ir palaikymo darbus.

Sistemos panaudojamumas bus projektuojamas atsižvelgiant į tokias prielaidas apie su sistema dirbantį žmogų:

- minimalūs darbo su kompiuteriu įgūdžiai (įjungti, išjungti, perkrauti kompiuterį, paleisti programas);
- minimalios failų sistemos žinios, failų kopijavimas, paieška;
- darbas su spausdintuvu;
- darbas Windows operacinėje sistemoje;
- kitos sistemos funkcijos vartotojui pateikiamos pagal pareigas/kvalifikaciją.

Siekiant sumažinti sistemos klaidų kainą sistemoje bus apribotas priėjimas prie kritinių sistemos funkcijų ir duomenų. Pagal priėjimo prie sistemos lygį galime išskirti tokias vartotojų grupes:

- sistemos aptarnavimo personalas – gali atlikti visas funkcijas;
- administratorius – gali keisti svarbius sistemos nustatymus, tačiau negali keisti tokių nustatymų, kurie sustabdytų sistemos darbą arba sąlygotų duomenų praradimą sistemoje;
- operatoriai – tai maža kvalifikaciją turintys vartotojai, kurie gali kviesti tik trumpalaikius pokyčius darančias funkcijas. Operatoriai, taip pat kaip ir administratoriai, gali atlikti visas duomenų (ataskaitų) formavimo operacijas.

### 4.12.3. Reikalavimai vykdymo charakteristikoms

Vykdymo charakteristikų parinkimą šioje sistemoje reikėjo derinti su reikalavimais sistemos saugumui. Operacijas sistema turi atlikti realiame laike, tačiau tai nesukelia jokių papildomų apribojimų sistemos vykdymo charakteristikoms, nes duomenų transakcijų skaičius sistemoje yra sąlyginai nedidelis.

Numatoma duomenų bazės eksploatavimo aplinka – apie 100 transakcijų per minutę. Prognozuojamas darbinis duomenų bazės dydis – nuo 10 iki 500 Mb. Tai sąlyginai nedidelės

apimties duomenų srautai, todėl papildomų resursų šiam projektavimo etapui skirti nenumatoma.
Pagrindinis reikalavimas sistemos vykdymui – sistemos nepertraukiamas veikimas. Dėl šios priežasties reikia ypatingą dėmesį atkreipti į sistemos patikimumą ir resursų panaudojimo efektyvumą.
Sistema gali valdyti nuo 1 iki 32 degalų kolonėlių.

### 4.12.4. Reikalavimai veikimo sąlygoms

Darbinį sistemos kompiuterį numatoma eksploatuoti normaliomis (ofiso) sąlygomis.
Pagrindinis reikalavimas sistemos veikimui – sistema neturi prarasti duomenų perkrovimo metu (pavyzdžiui, dingus elektrai). Po sėkmingo sistemos startavimo, visada reikia patikrinti ar nėra neužbaigtų transakcijų ir jei tokių yra – vykdyti atstatomuosius veiksmus.

### 4.12.5. Reikalavimai sistemos priežiūrai ir palaikymui

Sistemą numatoma kurti Microsoft Windows aplinkoje. Būtina užtikrinti sėkmingą ir nebrangų tolimesnį sistemos eksploatavimą šioje aplinkoje, dėl ko projektavimo metu reiktų naudoti naujausias šiuo metu prieinamas operacinės sistemos programavimo sąsajas, o tuo pačiu vengti pasenusių programavimo konstrukcijų.
Svarbu atsižvelgti į tai, kad ateityje gali tekti sistemą pritaikyti daugiakalbėje aplinkoje. Galimas reikalavimas – veikimas su keletu kalbų vienu metu.
Būtina įvertinti galimybę ateityje sistemą perkelti į kitas operacines sistemas.

### 4.12.6. Reikalavimai saugumui

Dirbant su degalų apskaitos programomis reikia labai rimtai atsižvelgti į saugumo klausimą. Sistemos saugumo reikalavimus galima apžvelgti trimis pagrindiniais aspektais:

- konfidencialumas – sistemoje esantys duomenys apsaugoti nuo neteisėtos prieigos. Šis bruožas šiek tiek mažiau svarbus, nei kiti du saugumo parametrai, tačiau vis vien labai svarbu duomenis saugoti tokiu būdu, kad nebūtų lengvai prieinami bet kokiems asmenims. Rekomenduojama kompiuterį nejungti prie vietinio tinklo arba bent jau atriboti nuo interneto.
- vientisumas – sistemos duomenys vienareikšmiai atitinka šaltinio perduotus. Šis reikalavimas labai svarbus sistemoje ne vien dėl to, kad pagal duomenis vykdoma materiali apskaita, bet ir dėl to, kad pagal sistemos duomenis reikia daryti ataskaitas mokesčių inspekcijai. Reikia numatyti atsarginių sistemos duomenų kopijų darymo galimybę. Pagal sistemos saugomus duomenis taip pat gali būti skaičiuojamos degalų

sąnaudos ir iš to atitinkamai premijos arba skolos. Šiam reikalavimui įvykdyti patartina pasinaudoti standartine komercine duomenų bazių valdymo sistema.

- pasiekiamumas – galimybė pasinaudoti duomenimis. Taip pat labai svarbus reikalavimas sistemai. Pasiekiami turi būti ne vien sistemos duomenys, bet ir kitos paslaugos. Neveikianti degalų išdavimo sistema gali sustabdyti visą organizacijos logistinį tinklą, todėl labai svarbu į sistemą įdiegti apsaugos funkcijas, kurios padėtų tokių nesklandumų išvengti.

### 4.12.7. Kultūriniai ir politiniai reikalavimai

Sistema turi dirbti pagal Lietuvoje priimtus ir įsigaliojusius įstatymus, kurie nurodo kaip įmonėse turi būti eksploatuojamos privačios degalų saugyklos. Dauguma šių niuansų liečia tik įmonę ir jos darbuotojus arba kelia tam tikrus reikalavimus degalinės techninei įrangai (kolonėlėms), tačiau kuriama sistema yra neatsiejama privačios degalų saugyklos eksploatacijos dalis, todėl būtina atsižvelgti į visus nurodymus.

Pagrindinis reikalavimas – visi sistemos duomenys sutaptų su popieriniais dokumentais. Šiam reikalavimui išpildyti sistemoje turės būti įdiegta dokumentų apskaitos sistema, o taip pat degalų kolonėlių skaitliukų apskaita.

Prireikus dirbti su kainomis arba savikainomis, sistemoje būtinai turi būti numatytas teisingas valiutos kodavimas.

Visos pagrindinės sistemos funkcijos vykdomos nacionaline (lietuvių) kalba.

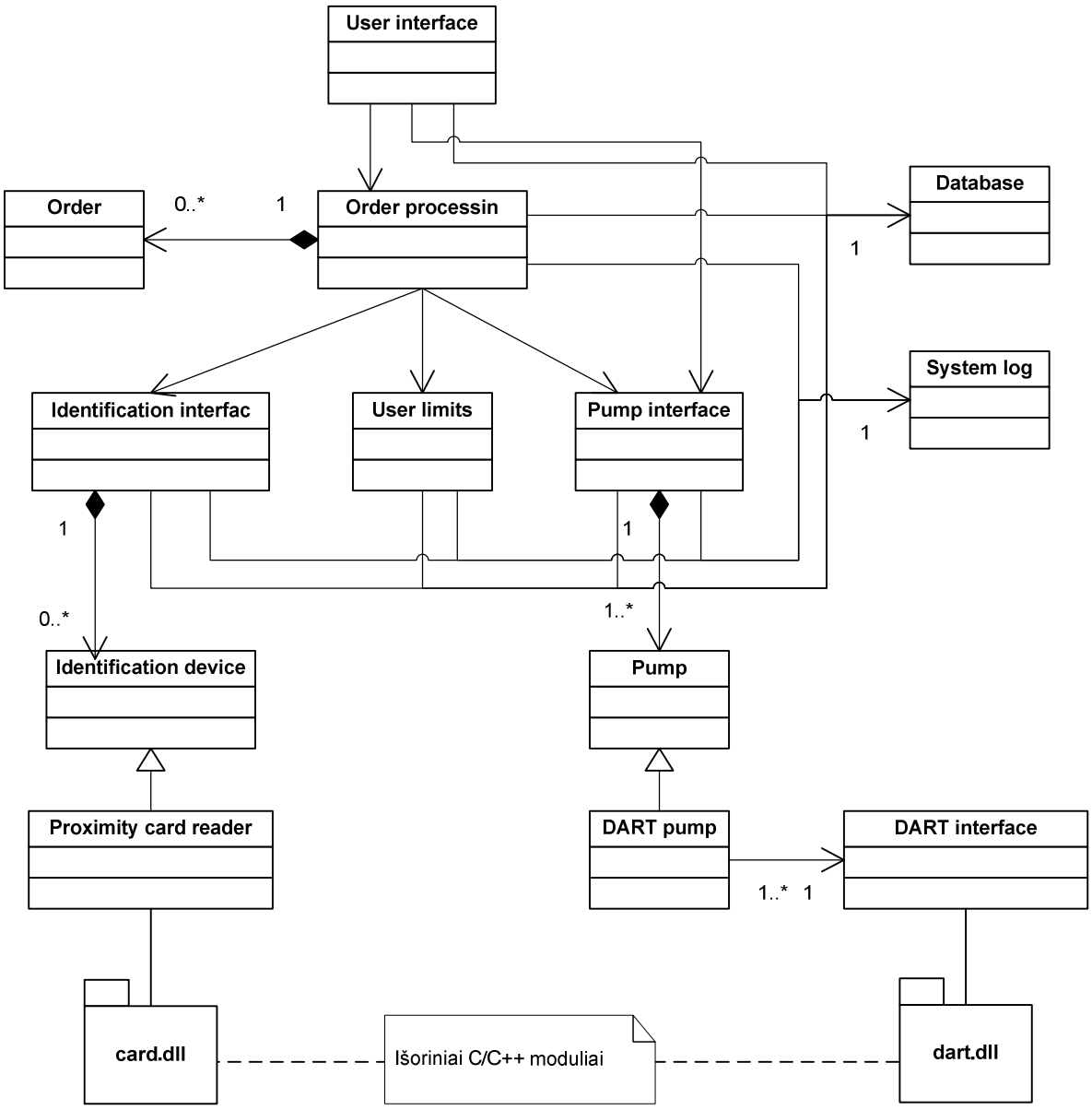
### 4.12.8. Teisiniai reikalavimai

Ši sistema – tai privačios degalinės kuro apskaitos programa. Nėra jokių įstatymų, kurie reglamentuotų tokios sistemos veiklą, tačiau yra įstatymai, kurie nurodo kaip turi dirbti tokio tipo degalinėse įdiegtos kuro talpyklos. Pagal užsakovo reikalavimus, sistemoje turės būti įstatymus atitinkanti degalų apskaitos ataskaita (degalų talpyklų žurnalas), kuris bus naudojamas vietoje popierinio žurnalo arba siekiant palengvinti popierinio žurnalo pildymą.

## 4.13. Architektūros specifikacija

### 4.13.1. Statinis sistemos vaizdas

Kiekviena svarbi sistemos dalis bus realizuota atskirame modulyje (šiuo atveju – klasėje):




**Pav. 3. Statinis sistemos vaizdas**

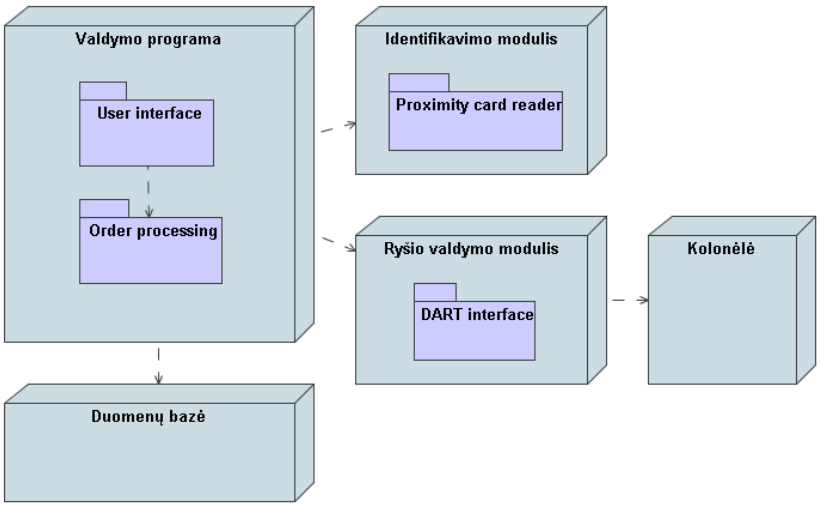
Toliau detalizuojant galima įvardinti visas klases ir jų programoje atliekamas funkcijas:

- User interface. Vartotojo sąsaja. Pateikia pranešimus vartotojui. Vizualiai parodo sistemos statusą. Sistemos būsenos informaciją surenka iš visų kitų susijusių klasių.
- Order. Užsakymo informaciją apimanti klasė.
- Order processing. Tai pagrindinis užsakymo valdymų sistemos procesas. Šis procesas savo vykdymo metu pagal tai, kokia sistemos būsena, aktyvuoja kitas darbines programos klases.

- Database. Duomenų bazės operacijas apimanti klasė. Realizuota „*singleton*“ projektavimo šablono principu. Šis funkcionalumas iškeltas į atskirą klasę, kad būtų galima palengvinti ir automatizuoti su duomenų baze susijusių veiksmų ir iškilsiančių problemų apdorojimą.
- Identification interface. Tai identifikavimo įrenginių valdymo klasė. Startavimo metu ši klasė sujungia visus sistemoje aprašytus vartotojų identifikavimo prietaisus.
- User limits. Kadangi užsakovams atiranda įvairių kintančių reikalavimų sistemai (kas susiję su sistemos vartotojais), nuspręsta vartotojų apribojimų ar kitokių veiksmų logiką perkelti į specializuotą klasę. Po vartotojo identifikacijos, surinkta informacija turi būti perduota šiai klasei, kuri pagal iš anksto suvestus nustatymus apskaičiuos kiek ir kaip vartotojui galima išduoti kuro.
- Pump interface. Kolonėlių valdymo klasė. Startavimo metu sujungia visas sistemoje aprašytas kolonėles.
- System log. Tai sistemos vykdomų veiksmų ir informacinių pranešimų saugojimo klasė. Realizuota „*singleton*“ projektavimo šablono principu. Šis funkcionalumas iškeltas į atskirą klasę, nes buvo planuota atlikti centralizuotą visų sistemos pranešimų apdorojimą ir saugojimą.
- Identification device ir Pump. Tai klasės – šablonai. Jų funkcionalumą realizuoja konkrečios klasės (Proximity card reader ir DART Pump). Teoriškai turi būti labai paprasta sistemoje įdiegti naują kuro kolonėlių ar identifikavimo įrenginio algoritmą, nes reikia tik realizuoti šių klasių sąsajas.
- DART interface. Tai „singleton“ projektavimo šablonu realizuota kolonėlių valdymo sąsaja. Ji skirta tik DART tipo protokolu dirbančių kolonėlių valdymui. Ši papildoma klasė sukurta, nes visos šio tipo kolonėlės jungiamos per vieną tarp procesų nedalijamą resursą – nuosekliąją sąsają.

Likę du moduliai – tai C programavimo kalba parašytos bibliotekos, kurios palaiko ryšį su aparatūrine įranga.

### 4.13.2. Paketų detalizavimas

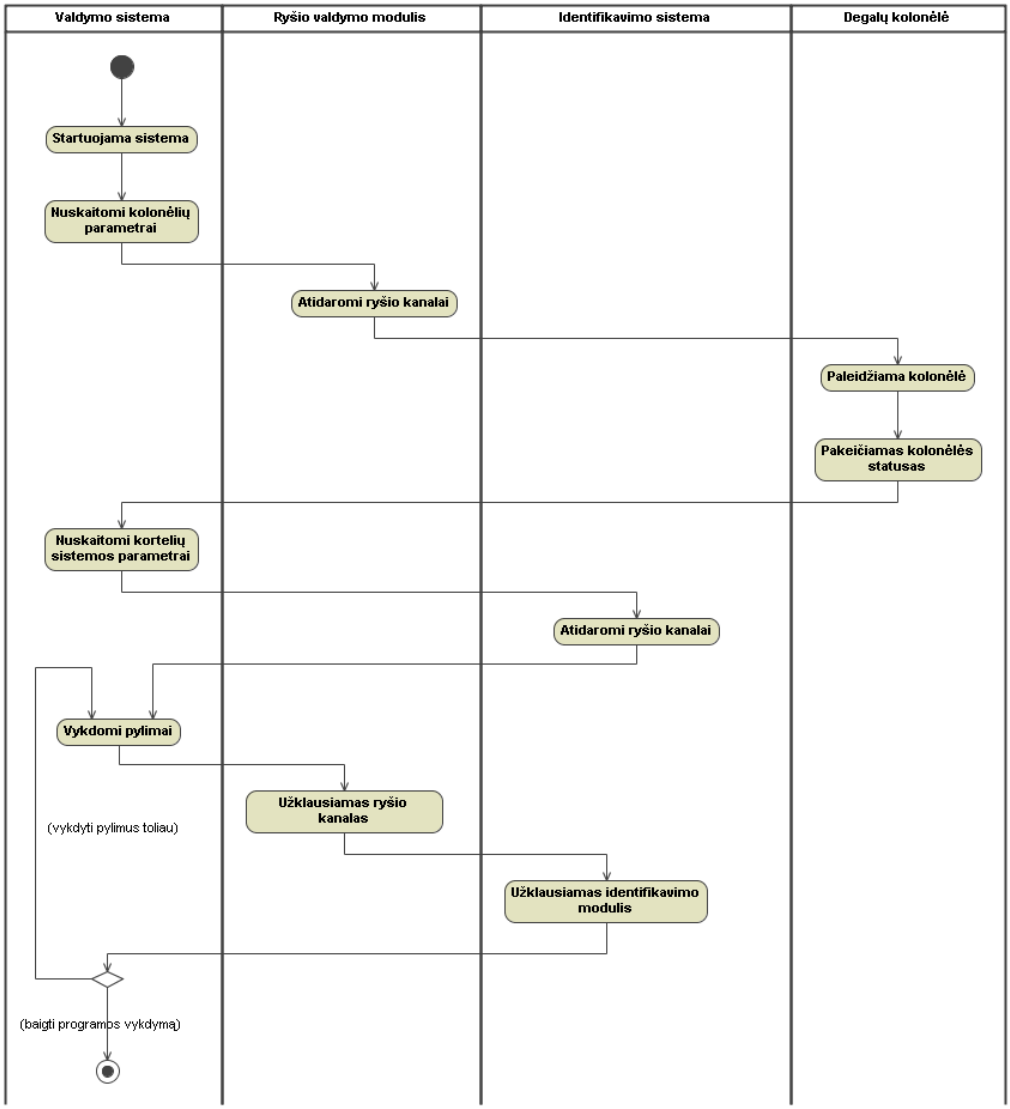



**Pav. 4. Paketų detalizavimo diagrama**

Šioje diagramoje pateikiamas bendras sistemos paketų vaizdas. Vėliau ryšiai tarp šių paketų bus detalizuojami kitų diagramų pagalba.

### 4.13.3. Veiklos diagramos

Šioje diagramoje pateikiamas standartinis programos veikimo scenarijus, nuo programos paleidimo iki jos uždarymo.

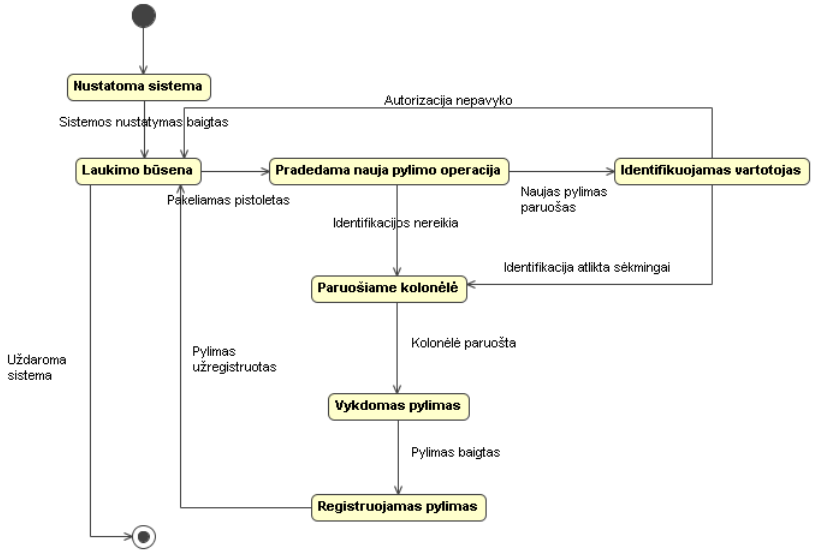



**Pav. 5. Pagrindinė programos veiklos diagrama**

Iš šios diagramos matome, kad pradėjus programos vykdymą, sukuriamas vienas pagrindinis ciklas, kuris ir vykdo pagrindinę programos operaciją (pila kurą). Prieš pradedant darbą taip pat reikia atlikti sudėtingą paleidimo procedūrą.

### 4.13.4. Būsenų diagramos

Visą savo darbo laiką programa praleidžia laukdama, kol vartotojas pareikalaus jos paslaugų. Šioje diagramoje parodoma kaip keisis pagrindinis programos darbo ciklas, kai paduodami reikiami įėjimo signalai:

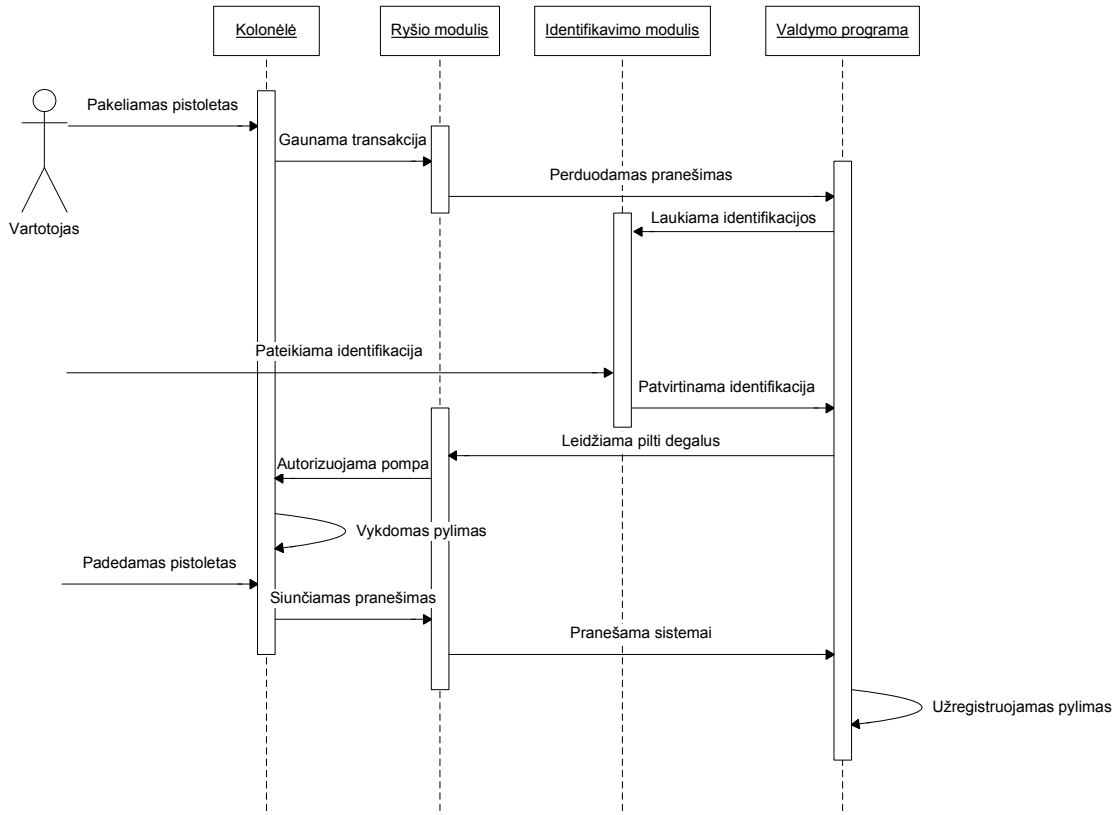



**Pav. 6. Sistemos būsenų diagrama**

Diagramoje pavaizduota valdančiojo ciklo būsenų diagrama. Pagal ją galime aptikti svarbiausius programos išsišakojimus ir būsenas. Matome galimų perėjimų iš vienos programos būsenos į kitą eiliškumą.

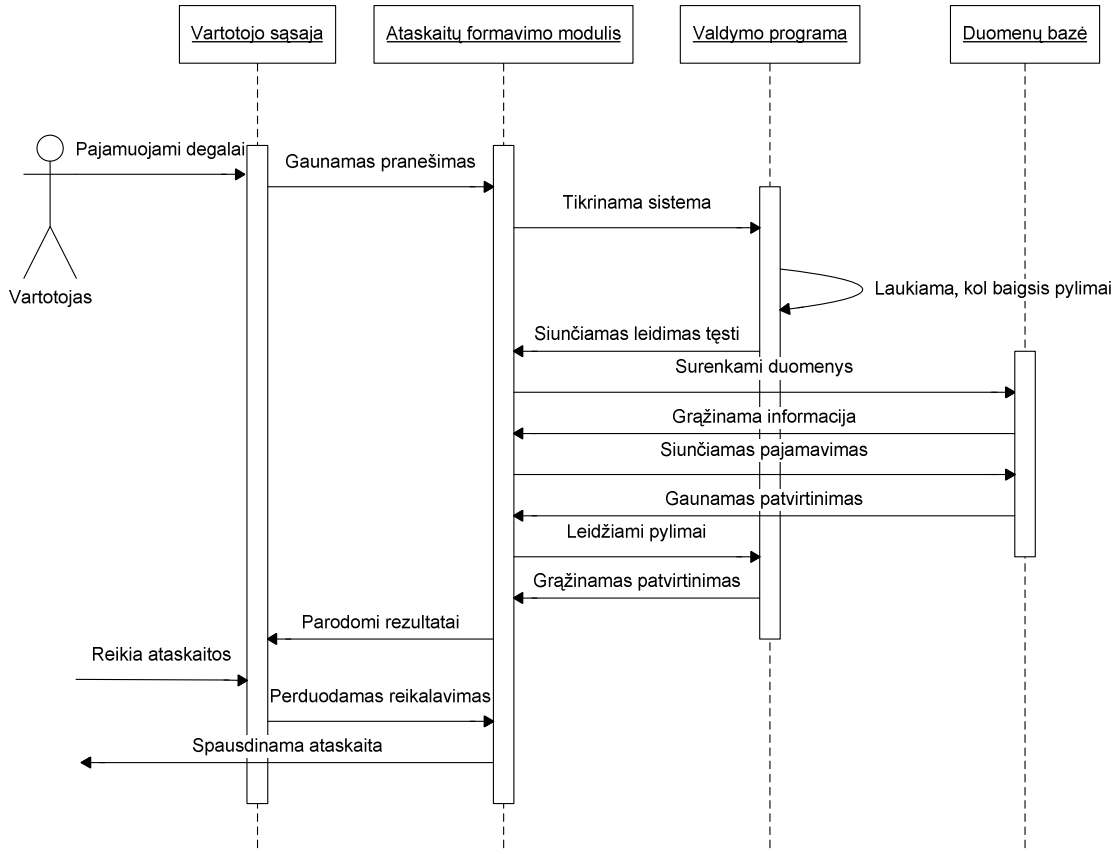
## 4.13.5. Sekų diagramos

Šiame skyriuje atvaizduojami svarbiausi sistemos atliekami veiksmai. Pirmiausia reikia aptarti pagrindinį programos atliekamą veiksmą – tai kuro pylimas.




**Pav. 7. Degalų pylimo proceso sekų diagrama**

Iš šios sekų diagramos matome visas su šiuo veiksmu susijusias klases ir jų bendravimo eiliškumą. Vartotojas tiesiogiai bendrauti su sistema negali. Vartotojas keičia tik kolonėlės būseną, kuri savo ruožtu šią informaciją perduoda per ryšio modulį į valdymo programą. Prieš pradedant pilti kurą, vartotojas privalo identifikuoti save sistemai. Po šio veiksmo pradedamas kuro pylimo procesas. Kuro pylimas baigiamas, kai kolonėlė pranešą, jog buvo padėtas kuro pistoletas.

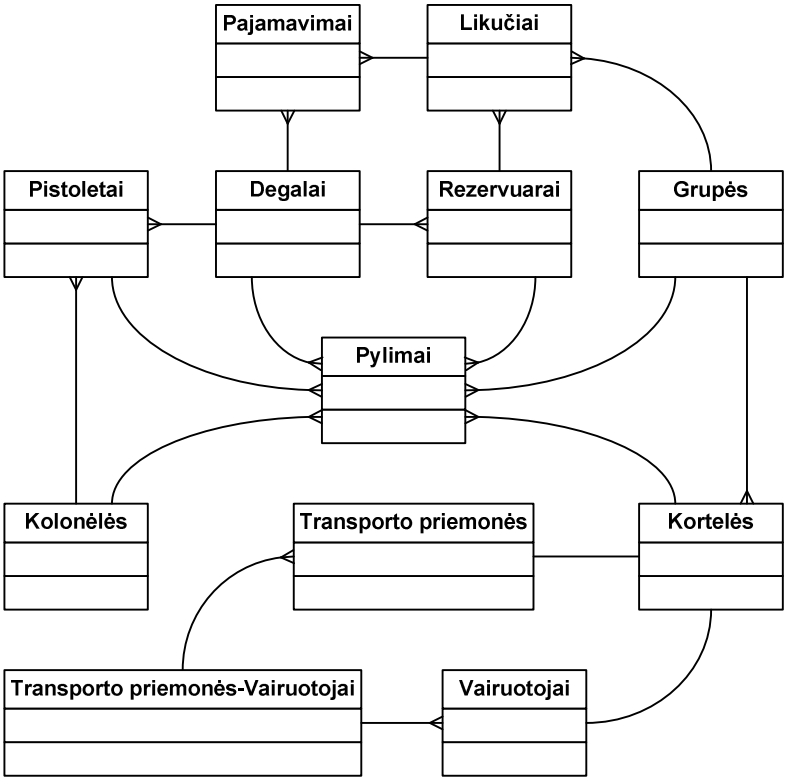



**Pav. 8. Degalų pajamavimo proceso sekų diagrama**

Sekų diagramoje matome kaip sistemoje pajamuojami degalai. Svarbiausia atkreipti dėmesį į tai, kad prieš atliekant bet kokį duomenų bazės koregavimą, pirmiausia reikia įsitikinti, kad sistemoje tuo metu nėra vykdomi pylimai. Toks reikalavimas atsiranda, nes pajamavimo operacijos rezultatai gali įtakoti valdymo programos priimamus sprendimus.

## 4.14. Duomenų modelis

Žemiau pateikiamas apibendrintas duomenų modelis:




**Pav. 9. Duomenų modelis**

Pagal šią schemą buvo realizuota projekto duomenų bazė. Kiekvienos lentelės laukai gali keistis priklausomai nuo užsakovo reikalavimų duomenims ir ataskaitoms, todėl iš anksto reikėjo numatyti lengvai išplečiamą ir modifikuojamą duomenų bazės modelį. Duomenų lentelės bus naudojamos šiais tikslais:

- Rezervuarai, Degalai, Grupės. Šiose lentelėse pateikiama atitinkamų objektų informaciją. Pavyzdžiui, rezervuarų lentelėje nurodomas rezervuaro numeris, saugomų degalų rūšis, minimalus bei maksimalus leistinas tūris ir kita informacija. Grupių lentelėje suvedamas sistemoje naudojamų įmonių sąrašas.
- Pajamavimai, Likučiai. Lentelės naudojamos degalų apskaitai atlikti.
- Kortelės, Vairuotojai, Transporto priemonės. Šiose lentelėse aprašomos vartotojų atpažinimo informacija.
- Kolonėlės, Pistoletai. Lentelėse saugoma kolonėlių nustatymų informacija.
- Pylimai. Tai pagrindinė sistemos duomenų lentelė, kurioje bus saugoma sistemos atliekamų pylimų informacija.

## *4.15. Projekto išeiga*

### 4.15.1. Atviri klausimai

Bandant pritaikyti sistemą specialiems klientų poreikiams ir tuo pačiu praplečiant užsakovo klientų ratą gali tekti atlikti sistemos funkcinės specifikacijos pakeitimus.

Tiksliai neaišku, kokią vartotojų identifikacijos sistemą reikės naudoti galutinėje sistemoje. Dėl to buvo priimtas sprendimas sukurti universalią programinę sąsają, prie kurios būtų nesunku prijungti papildomas identifikacines sistemas.

Vienas realiausių naujų sistemos reikalavimų – veikimas su keletu kalbų vienu metu. Į tai atsižvelgta sistemos kūrimo metu. Visa programinė įranga ir pati projektavimo aplinka dirba Unicode UTF-8 koduotės pagrindu.

Daugiau nenumatytų problemų šiuo metu nėra.

### 4.15.2. Naujos problemos

#### 4.15.2.1. Problemos diegimo aplinkai

Manoma, kad įdiegus naują sistemą pirmiausia pasikeis kuro apskaitos darbų vykdymas. Problemos gali atsirasti tada, jei reikėtų sistemą diegti kompiuteryje, kuriame jau vykdoma daugelis kritinio svarbumo programų, arba kuriuo nuolatos intensyviai naudojasi vartotojai (pavyzdžiui, failų tarnybinė stotis). Dėl šios priežasties ir ypač norint pasiekti aukštesnį saugumo lygį, patartina sistemą diegti tik šiai sistemai skirtame kompiuteryje.

#### 4.15.2.2. Įtaka jau instaliuotoms sistemoms

Sistemos diegimo metu neplanuojama atlikti didelius operacinės sistemos pakeitimus. Prieš diegiant degalinių valdymo programą, patartina operacinę sistemą atnaujinti (saugumo ir stabilumo sumetimais). Dėl šių atnaujinimų gali nustoti veikti kitos liktinės sistemos.

Papildomų problemų gali atsirasti specializuotos duomenų bazės diegimo metu. Jei sistemoje jau yra įdiegta duomenų bazių valdymo sistema, tai patartina programą pritaikyti darbui su įdiegta DBVS.

#### 4.15.2.3. Neigiamas vartotojų nusiteikimas

Tokio tipo sistemos dažniausia susilaukia neigiamo darbuotojų nusiteikimo ir teigiamo vadovų požiūrio. Reikia aptarti visų pagrindinių vartotojų grupių požiūrį į naują sistemą.

- Administratoriai – neigiamas požiūris, nes atsiranda papildomas darbas – naujos sistemos eksploatavimas. Reikia atlikti bazinius apmokymus, tačiau sistemos eksploatavimo eigoje administratorių požiūris pasidarys neutralus (rutina).

- Operatoriai – laukiamas didesnis pasipriešinimas, nei iš administratorių grupės, nes naujos sistemos diegimas, gali reikšti operatorių etatų mažinimą. Taip pat reikės atlikti minimalius apmokymus darbui su sistema įgūdžiams išvystyti. Ilgam laiko tarpui numatomas teigiamas nusiteikimas sistemos požiūrių, nes sistema leidžia išvengti rankinės apskaitos darbų.
- Vartotojai – neutralus požiūris, tačiau gali būti neigiama reakcija sistemos eksploatacijos pradžioje (neigiamas požiūris į pokyčius). Taip pat gali atsirasti neigiamas nusiteikimas, jei sena liktinė sistema leisdavo apeiti buhalterinę apskaitą ir piktnaudžiauti įmonės turtu.

#### 4.15.2.4. Kliudantys diegimo aplinkos apribojimai

Sistemai diegti reikės naujo personalinio kompiuterio, spausdintuvo ir darbo vietos. Taip pat reikės pirkti naują licenziją operacinei sistemai ir visiems susijusiems komponentams (pavyzdžiui, duomenų bazė arba Microsoft Office). Atsižvelgiant į santykinai mažą numatomą visos sistemos kainą, tai gali būti pakankamai dideliu sistemos diegimo apribojimu.

#### 4.15.2.5. Galimos naujos sistemos sukeltos problemos

Pradėjus diegti naują sistemą reikės apmokinti esamus arba naujus sistemų aptarnavimo specialistus. Reikės teikti aptarnavimą ir atlikti naujų vartotojų apmokymus. Smarkiai padidėjus paklausai reikės samdyti naujus žmones.

# 5. Tiriamoji dalis

## 5.1. Įvadas

Šioje dokumento dalyje pateikiama magistrinio darbo metu sukurtos programinės įrangos kokybės analizė. Pagrindiniai klausimai, kuriuos bus bandoma atsakyti šiame skyriuje, tai:

- Kaip galima vertinti sprendimo rezultato patikimumą?
- Kaip galima vertinti programų sistemos kūrimo rezultatus?
- Kokios programų sistemos kokybės tobulinimo galimybės?
- Koks rekomenduojamų pakeitimų pagrindimas?

Šiame skyriuje padarytų išvadų teisingumas bus tikrinamas eksperimentinėje darbo dalyje. Trumpai bus aptariamas ir pačios programinės įrangos kokybės valdymas. Kokybės valdymo procesas naudojamos programinės įrangos kokybei užtikrinti, o tuo pačiu ir kai kuriems programinės įrangos procesams apibrėžti bei kontroliuoti. Šios sistemos kūrimo metu buvo taikomi tik bendro pobūdžio kokybės kontrolės darbai, kurie bus išsamiau detalizuojami sekančiuose skyriuose. Konkrečių kokybės standartų reikalavimai programinei įrangai nebuvo taikomi.

## 5.2. Priimto sprendimo pagrindimas

Architektūros specifikacijoje pateikto sprendimo realizacija leido pasiekti visus sistemos kūrimo pradžioje užsibrėžtus tikslus. Projektavimo metu neiškilo kritinės problemos, kurios galėjo atsirasti dėl pasirinktų sprendimų. Šiuo metu užbaigta, ištestuota ir patikrinta sistema sėkmingai veikia užsakovo klientų privačiose degalinėse.

Moduliari sistemos architektūra leidžia lengvai atlikti mažas ir vidutines programos modulių modifikacijas, tačiau iškyla problemos, kai reikia atlikti modulių integravimo testavimą. Tai ir padėjo, ir tuo pačiu trukdė pasiekti norėtą kokybės lygį.

Siekiant užtikrinti kokybę buvo sudarytas testavimo įrankių rinkinys ir standartinis testavimo planas. Programinė įranga vartotojams buvo išleidžiama be žinomų defektų, tačiau sąlyginai mažas programinės įrangos testavimo finansavimas neleido atlikti išsamios programos kokybės analizės.

Pati programos architektūra galutinio produkto kokybę veikė palankiai, nes projektavimo darbai buvo vykdomi pagal nuoseklų, detalų ir suderintą programos architektūros planą.

Galutinis produktas veikia stabiliai, be didelių defektų. Problemos iškyla tik aparatūrinės įrangos defektų pasirodymo metu. Tokias situacijas labai sunku tinkamai ištestuoti normalių testų metu.

## *5.3. Kokybės analizės tikslai*

### 5.3.1. Aptikti klaidas funkcionavime, logikoje, realizacijoje

Kokybės analizės darbai atliekami pirmiausia tam, kad įsitikinti, jog programa veikia pagal pradinę programos specifikaciją. Reikia patikrinti, ar sistemos programuotojo sudarytas sistemos veikimo vaizdas sutampa su pradiniais užsakovo reikalavimais. Šiame etape tikrinamas konkretus sistemos funkcijų veikimas. Tikrinamos logikos klaidos ir atliekami bendro pobūdžio darbo su programa veiksmai. Aptiktos klaidos registruojamos, išanalizuojamos ir joms pateikiami pataisymai.

### 5.3.2. Patikrinti ar programų sistema atitinka reikalavimų specifikaciją

Patikrinus programos logiką ir realizaciją, reikia atlikti aukštesnio lygio sistemos kontrolės darbus. Sekančiame etape tikrinama ar programa atitinka reikalavimų specifikacijoje iškeltus reikalavimus. Tikrinama, ar realizuoti visi pradiniai panaudos atvejai. Svarbu kuo anksčiau projektavimo eigoje aptikti neatikimus su reikalavimų specifikacija, nes ankstyvoje stadijoje pokyčiams įgyvendinti reikia mažiausiai pastangų.

### 5.3.3. Įsitikinti, ar programų sistema sukurta pagal standartus

Kai pirmieji pagrindiniai kokybės kontrolės etapai jau atlikti belieka tik patikrinti ar programinė įranga atitinka įmonėje keliamų standartų reikalavimus. Šios kontrolės svarba gali keistis skirtingose įmonėse. Pavyzdžiui, banke programinei įrangai keliami reikalavimai žymiai griežtesni ir turi būtinai atitikti tam tikrus bankams keliamus standartus.

### 5.3.4. Programinės įrangos kokybės vertinimo kriterijai

Sistemos kokybė bus vertinama pagal tokius pagrindinius kokybės vertinimo kriterijus [9]:

- Funkcionalumas:
    - tinkamumas;
    - tikslumas;
    - bendradarbiavimas;
    - suderinamumas;
    - apsauga (saugumas);

- Patikimumas:
    - brandumas;
    - gedimų tolerancija;
    - atkuriamumas;
- Panaudojamumas:
    - suprantamumas;
    - išmokstamumas;
    - veikimas;
- Našumas:
    - laikinė elgsena;
    - elgsena resursų atžvilgiu;
- Palaikomumas:
    - analizuojamumas;
    - keičiamumas;
    - stabilumas;
    - testuojamumas;
- Portatyvumas:
    - prisitaikomumas;
    - įdiegiamumas;
    - atitikimas;
    - pakeičiamumas.

## *5.4. Kokybės vertinimo rezultatai*

### 5.4.1. Atitikimas reikalavimams

Atitikimas reikalavimams buvo atliekamas dviejų veiklų pagalba. Pirmiausia buvo atlikta formali techninė peržiūra, kurios metu nustatyta, kad sistemoje realizuoti visi reikalavimų specifikacijoje pateikti panaudojimo atvejai. Vėliau buvo atliekamas bendro pobūdžio sistemos auditas, kurio metu tikrinta, kaip sistemoje atsižvelgta į nefunkcinius reikalavimus. Nustatyta, kad sistemoje yra pateikta adekvati nefunkcinių reikalavimų realizacija arba pateiktos atitinkamos rekomendacijos, kurios leidžia įvykyti šiuose reikalavimuose keliamus tikslus.

Interviu su užsakovu ir sistemos pridavimo metu šie vertinimai pasitvirtino. Sistema buvo priduota sėkmingai.

### 5.4.2. Funkcionavimo, logikos ir realizacijos klaidos

Realizacijos klaidų buvo ieškoma testavimo etape. Testavimo darbams atlikti buvo sukurta keletas testavimo įrankių, kuriems vėliau buvo galima pagal poreikius kurti testinius atvejus. Visus sudarytus testus moduliai atliko sėkmingai. Galutinė programa užsakovui buvo pristatyta be jokių žinomų defektų.

Vėliau vyko programos sistemos diegimo darbai ir sistema pradėjo eksploatuoti praktikoje. Didelių programos defektų neaptikta.

Tikrinant programos realizaciją buvo aptikta keletas architektūros trūkumų, kuriuos nuspręsta taisyti ir analizuoti magistrinio darbo vykdymo metu.

### 5.4.3. Kokybės vertinimo rezultatai

Žemiau pateiktoje kokybės įvertinimo lentelėje kriterijai vertinami penkiabalėje sistemoje. 5 – reiškia jokių priekaištų nėra. 1 – visiškai netinkama.

**Lentelė 4. Kokybės vertinimo rezultatai**

| Kokybės kriterijus | Įvertinimas | Komentaras |
|---|---|---|
| **Funkcionalumas** | | |
| tinkamumas | 5 | Trūkumų nėra. |
| tikslumas | 5 | Trūkumų nėra. |
| bendradarbiavimas | 4 | Kai kurios galimybės yra, tačiau reikalauja nemažai projektavimo išlaidų. |
| suderinamumas | 4 | Suderinama su kitomis sistemomis, tačiau ribotai. |
| apsauga (saugumas) | 4 | Nėra specifinių programinių apsaugų. |
| **Patikimumas** | | |
| brandumas | 4 | Pirmoji sistemos versija. |
| gedimų tolerancija | 4 | Daugumą gedimų įveikia sėkmingai, tačiau yra išimčių. |
| atkuriamumas | 5 | Trūkumų nėra. |
| **Panaudojamumas** | | |
| suprantamumas | 5 | Trūkumų nėra. |
| išmokstamumas | 5 | Trūkumų nėra. |
| veikimas | 4 | Pasitaiko neprognozuojamų trukdžių. |
| **Našumas** | | |
| laikinė elgsena | 4 | Kartais sistemos darbas neleistinai sulėtėja. |

| Kokybės kriterijus | Įvertinimas | Komentaras |
|---|---|---|
| elgsena resursų atžvilgiu | 5 | Trūkumų nėra. |
| **Palaikomumas** | | |
| analizuojamumas | 5 | Trūkumų nėra. |
| keičiamumas | 5 | Trūkumų nėra. |
| stabilumas | 4 | Kartais pasitaiko stabilumo problemų. |
| testuojamumas | 4 | Sistemą testuoti problematiška, nes programa dirba su išorine aparatūrine įranga. |
| **Portatyvumas** | | |
| prisitaikomumas | 3 | Sistemą galima pritaikyti darbui tik Windows aplinkoje. |
| įdiegiamumas | 5 | Trūkumų nėra. |
| atitikimas | 5 | Trūkumų nėra. |
| pakeičiamumas | 5 | Trūkumų nėra. |

Galime daryti išvadą, kad galutinės sistemos kokybė atitinka visus iškeltus reikalavimus. Sistemos vertinimo metu nebuvo aptikta nė viena rimta problema. Dėl šios priežasties toliau magistriniame darbe bus kalbama ne apie sistemos funkcionalumo tobulinimą, bet apie sistemos architektūros tobulinimą apskritai.

## *5.5. Projektavimo problemos*

Nors ir pačio darbo metu sukurta sistema atitinka visas pradines specifikacijas, galime rasti keletą probleminių sričių, kurias reikėtų patobulinti ir ištirti eksperimentinėje darbo dalyje.

Vienas iš problematiškų projektavimo sprendimų, kuris buvo priimtas pačioje projekto pradžioje, tai ryšio modulių realizacija kitoje programavimo kalboje, nei pagrindinė programa. Nors vertinant priimtus sprendimus buvo padaryta išvada, kad dėl tokio sprendimo didelių sunkumų nebus, galutiniame projekte šie moduliai liko kaip atskira tinkamai neprižiūrėta sistemos dalis.

Pradinis sprendimas buvo priimtas iš dalies dėl techninių .NET programavimo platformos apribojimų. Praėjus dviems metams, buvo išleista nauja .NET versija, kuri leistų įveikti anksčiau buvusius techninius apribojimus.

Reikia C kalboje parašytas DLL bibliotekas perrašyti valdomoje .NET aplinkoje. Planuojama panaudoti didžiąją dalį egzistuojančio programinio kodo. Perdarius ryšio valdymą visi sistemos moduliai bus integruoti į vieningą, nepriklausančią nuo išorinių modulių sistemą.

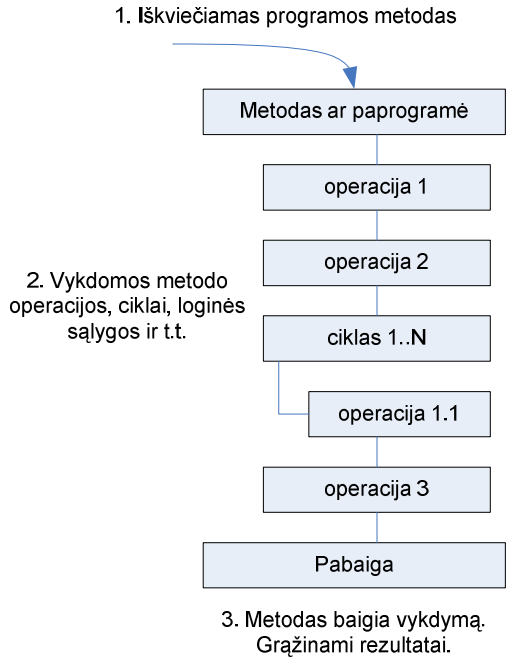
Pagrindinis pranašumas, tai vieninga kodo bazė, palengvėjęs testavimas ir derinimas. Galutinėje sistemos versijoje norint atidaryti ryšį su išorinėmis DLL bibliotekomis reikia kviesti nepatikimą (*unmanaged*) programos kodą, kas iškart padaro neigiamą įtaką sistemos stabilumui ir patikimumui, bei prieštarauja pagrindinei .NET programavimo koncepcijai.

Naudojant išorines bibliotekas taip pat buvo susidurta ir su priklausomybe nuo kitų Microsoft Windows sisteminių bibliotekų, kurios skirtingose operacinės sistemos versijose būna ne vienodų versijų. Atsikračius šio priklausomybių sąrašo pagerės bendras sistemos architektūros vaizdas, o tuo pačiu ir statinė klasių diagrama.

## 5.6. Realizacijos ir architektūros trūkumai

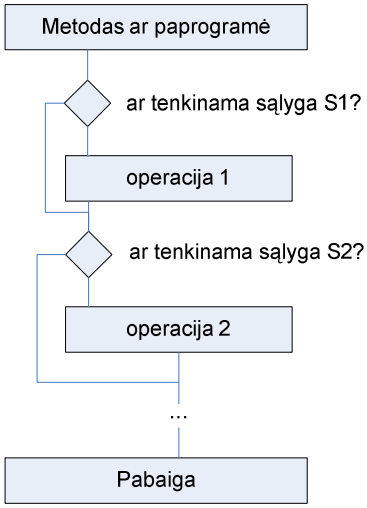
Pagrindinis sistemos architektūros trūkumas slypi pasirinktoje projektavimo metodikoje ir iš dalies atsirado dėl poreikio palaikyti stabilų ir patikimą ryšį su kuro kolonėlėmis. Netgi pritaikius visus analitinėje dalyje išvardintus metodus programos kokybei užtikrinti, galutinis programos realizavimas kai kuriose srityse buvo be reikalo sunkus ir problematiškas.

Sistemoje metodai buvo realizuoti standartinio programavimo principais, kuriuos galima iliustruoti tokia schema:




**Pav. 10. Įprastos programos vykdymo principas**

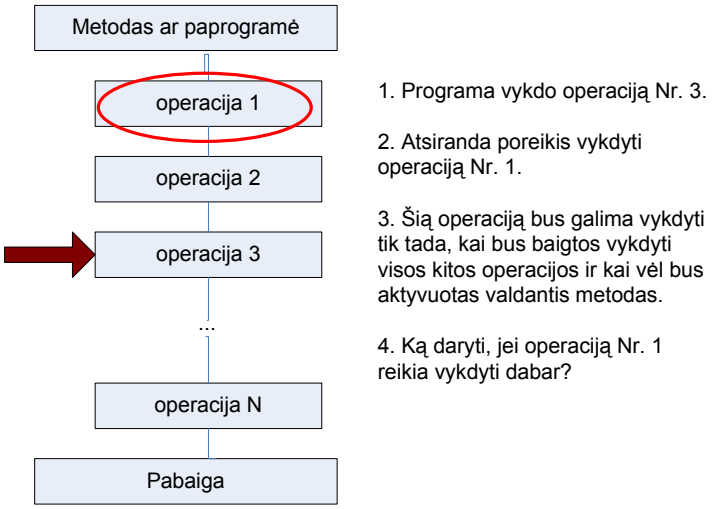
Tokiu būdu programuojama ir vykdoma didžioji dalis programinės įrangos, todėl ir šio projekto metu nuspręsta taikyti būtent šią metodologiją. Bendraujant su išoriniais įrenginiais, kurių veikimas priklauso nuo laiko komponentės šitokia realizacija greitai tampa neadekvati. Viršuje pateikiamas supaprastinta metodo vykdymo schema. Praktikoje, prieš vykdant su išoriniu įrenginiu susijusias operacijas dažniausia reikės iš pradžių patikrinti ar įrenginys pasiruošęs, ar įrenginys tinkamoje būsenoje ir pan. Taigi metodo vykdymo algoritmas transformuojamas į tokį:




**Pav. 11. Tikroviška metodo vykdymo schema**

Iš schemos galime pastebėti, kad kodas iškart pasidaro sudėtingesnis, sunkiau suprantamas ir testuojamas. Kiekviena loginė sąlyga ar programos išsišakojimas didina galutinės sistemos sudėtingumą ir apsunkina būsimus programinės įrangos palaikymo darbus. Paprastumo dėlei toliau einančiose schemose loginės sąlygos nebus vaizduojamos, tačiau reikia prisiminti, kad norint įvykdyti bet kokį veiksmą, prieš tai turi būti patikrintos visos pradinės sąlygos.

Toliau bandysime aptarti esmines problemines situacijas, kurios atsiranda taikant tokį programavimo stilių. Pirmiausia pažvelkime šią schemą:

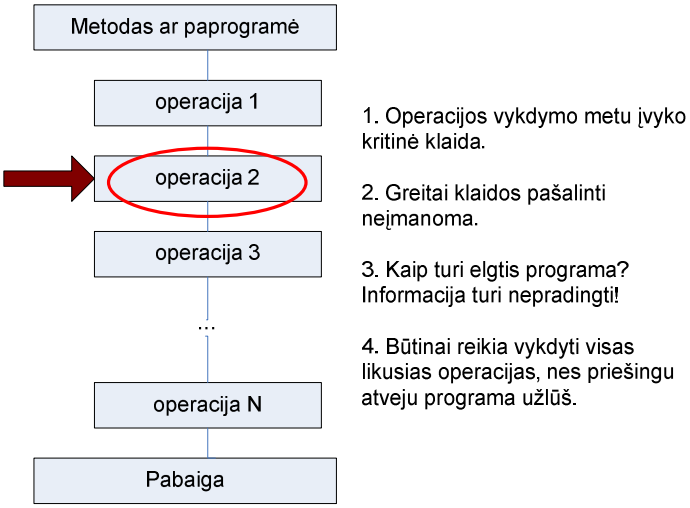



**Pav. 12. Veiksmų eiliškumo problema**

Čia pavaizduota veiksmų eiliškumo problema. Normalioje programavimo kalboje visi veiksmai vykdomi iš eilės ir šokinėjimas nuo vienos operacijos prie kitos yra negalimas. Tačiau čia yra sistema atliekanti kolonėlės valdymo darbą, dėl ko veiksmų negalima užvėlinti neapibrėžtą laiko tarpą (pavyzdžiui, vairuotojas nelauks minutę laiko, kol sistema jam paleis kurą). Reikia rasti optimalų šios problemos sprendimą. Dabartinėje sistemoje ši problema sprendžiama tik apribojant visų operacijų vykdymo laiką ir uždraudžiant vykdymo blokavimą.

Tačiau atsiranda tokių situacijų, kai būtų pageidautina blokuoti programos vykdymą, kol, pavyzdžiui, nebus išspręsta tam tikra būtina sąlyga. Tokio tipo problemas galima spręsti tik pritaikius asinchroninius programavimo metodus.

Sekanti problema susijusi su klaidų valdymu, tačiau gali būti aptinkama ir kitose programos dalyse:




**Pav. 13. Kitos programos vykdymo problemos**

Optimalaus šios situacijos sprendimo, pritaikant dabartinę projektavimo technologiją padaryti neįmanoma. Esančioje programoje tokio tipo problemą šiuo metu bandoma apdoroti tokia veiksmų seka:

- bandoma kartoti operaciją;
- visa kritinė informacija išsaugoma į tekstinį failą, išvedamas pranešimas apie klaidą ir programa vykdoma toliau.

Reikia ieškoti geresnio asinchroninio programavimo sprendimo, kuris leistų nestabdant programos vykdymo eigos atlikti visus reikiamus veiksmus. Šis sprendimas detaliau aprašomas sekančiame skyriuje.

## *5.7. Įvykiais valdomos sistemos*

### 5.7.1. Įvadas

Praktikoje yra daug programavimo pavyzdžių (paradigmų), kurie savo esme yra visiškai priešingi, tačiau padeda pasiekti tą patį bendrą tikslą. Vienas iš pavyzdžių būtų objektinis programavimas lyginant su procedūriniu programavimu [10]. Šiame skyriuje bus bandoma apžvelgti kaip įvykiais valdomas projektavimas skiriasi nuo tradicinio procedūrinio programavimo metodų ir kodėl pritaikius šį metodą šioje sistemoje galima sukurti paprastesnį,

lengviau suprantamą ir palaikomą programinį produktą, kuris tuo pačiu įgauna ir geresnes vykdymo charakteristikas.

### 5.7.2. Bendra informacija

Pirmiausia reikia suprasti pagrindines šių dviejų programavimo metodų savybes [11]:

- procedūrinis programavimas:
    - kodo tėkmė valdoma iš anksto numatytomis sąlygomis;
    - kodas vykdomas nuosekliai, tik kartais išsišakodamas loginių sąlygų ar ciklų pagalba;
    - programos atliekamų veiksmų seką nusako kodo išdėstymas;
    - visi veiksmai yra numatyti iš anksto;
- įvykiais valdomas programavimas:
    - programos vykdymas priklauso nuo išorinių įvykių (pavyzdžiui, išorinių įrenginių veiklos, ar vartotojo darbo);
    - programos kodas sistemoje vykdomas asinchroniškai, t. y. nepriklausomai nuo kitų kodo modulių.

Moderniose programavimo kalbose įvykiai turi tokias tris pagrindines savybes:

- šaltinis – nurodo, kas iškvietė įvykį;
- tipas – nusako įvykio pobūdį;
- tikslas – objektas, kuriam skirtas įvykis.

Pačių įvykių realizacija gali būti programuojama skirtingai, priklausomai nuo sistemos poreikių ir egzistuojančių projektavimo aplinkos įrankių. Įvykiai gali būti realizuojami kaip:

- Periodinė apklausa. Tikrinama sistemos informacija ir iškviečiamos atitinkamos sistemos funkcijos.
- Asinchroniniai metodai. Įvykiai formuojami standartinių Unix ar Windows sistemų asinchroninio programavimo bibliotekų pagalba.
- Žinučių perdavimas. Sudaromas žinučių perdavimo ir žinučių gavėjų mechanizmas. Įvykis formuojamas kaip pranešimas.
- Aparatūriniai pertraukimai. Metodai registruojami kaip aparatūrinės įrangos pertraukimo apdorojimo mechanizmai.
- Kiti būdai. Pavyzdžiui, transakcijų apdorojimo sistemos ar stebėtojo projektavimo šablonas.

Apibendrinant galime sakyti, kad įvykiais valdomas programavimas – tai tiesiog patogesnis ir priimtinesnis būdas bendradarbiaujančių sistemų informacijos srautams valdyti.

### 5.7.3. Kodėl nenaudoti gijų arba procesų?

Analizuojant pateiktą informaciją galima prieiti išvados, kad viską, ką galima padaryti pritaikius įvykiais valdomą programavimą, galima padaryti ir panaudojus standartinį gijų ar procesų programavimą. Šiame skyriuje bus bandoma paaiškinti, kodėl gijų programavimas ne visada yra geriausias sprendimas.

Pirmas sunkumas, su kuriuo susiduriame programų su daug vykdomų gijų realizavimo metu, tai dideli programos kodo kokybei keliami reikalavimai [12]. Reikia itin atsargiai dirbti su bendrais duomenimis. Būtina užtikrinti, kad neatsiras blokuojanti (*deadlock*) vykdymo situacija. Yra daug uždavinių, kuriuose įvykiais valdomos ir gijų pagrindu realizuotos programos pasiekia tokius pat gerus rezultatus, bet gijų metodo programiniam kodui keliami žymiai didesni reikalavimai.

Galime rasti įvykiais valdomą programavimą propaguojančių autorių, kurie teigia, kad „gijos veda link nepatikimos programinės įrangos“ [13, 1 p.]. Šiame straipsnyje aptarinėjami pagrindiniai gijų programavimo metodų pranašumai ir trūkumai:

- leidžia lygiagrečiai vykdyti duomenų ir ryšio palaikymo operacijas tam, kad sudaryti nuoseklaus ir sklandaus programos vykdymo iliuziją;
- tačiau įneša bereikalingas sinchronizacijos problemas ten, kur jos tikrai nėra būtinos;
- praktikoje programos realizuotos gijų pagrindu visada turi dalintis duomenis, atlikti sinchronizaciją ir lenktyniauti tarpusavyje, kas apsunkina gerų programų kūrimą.

Praktikoje yra pademonstruota, kad didelio apkrovimo metu, kai naudojamas daugiau nei vienas procesorius, įvykiais valdomos programos kai kuriais atvejais ženkliai lenkia tradicines sistemas, kuriuose programos realizuotos gijų pagrindu.

### 5.7.4. Įvykiais valdomų programų pranašumai

Atsisakius gijų ir perėjus prie įvykiais valdomų programų galima išvengti daugumos problemų, susijusių su duomenų pasidalijimu. Taip pat, smarkiai apkrautose sistemose galima sutaupyti nemažai atminties resursų ir procesoriaus darbo laiko, nes blokuojami įvykiai gali būti tiesiog dedami į eilę tol, kol juos bus galima vykdyti vietoj to, kad jiems būtų pastoviai skiriamas procesoriaus darbo laikas (kaip gijoms).

Įvykiai leidžia išvengti blokuojančių programų vykdymo situacijų, o tuo pačiu ir lengviau leidžia planuoti procesoriaus resursus. Pritaikius tokio tipo metodiką taip pat galima lengviau atlikti tikru laiku vykdomų operacijų duomenų priėmimą ir analizę.

Didžiausias įvykiais valdomų sistemų minusas, kurį dažnai cituoja autoriai, tai šio tipo sistemų programavimo sudėtingumas. Įvykiams realizuoti naudojama daug dinaminės atminties valdymo operacijų, kas gali įnešti nepageidaujamų klaidų. Tačiau tokio tipo problemos sutinkamos tik žemo lygio kalbose (pavyzdžiui, C ar C++).
Šioje sistemoje įvykiais valdomas programavimas leis išspręsti problemas, kurių nepavyko pašalinti pritaikius įprastines projektavimo priemones.

## 5.8. Naujų technologijų pasirodymas rinkoje

„Projektavimo problemos“ skyriuje aptartas užduotis galima išspręsti pritaikius naujai rinkoje pasirodžiusias projektavimo technologijas. Nuo tada, kai sistemos specifikavimas ir kūrimas buvo baigtas, rinkoje pasirodė nauja Microsoft .NET programavimo karkaso versija. Daugelis šios atnaujintos versijos funkcijų neturi didelės įtakos sistemos tobulinimui, tačiau yra keletas vertų dėmesio savybių [14]:

- .NET Framework 2.0:
    - 64-bitų platformų palaikymas. Tai galimybė sistemą pritaikyti šiuo metu naujausiose galingiausiose tarnybinėse stotyse;
    - Papildomos priemonės programinės sistemos saugumui užtikrinti;
    - Duomenų apsaugos programavimo sąsaja;
    - Naujos derintuvo galimybės;
    - Tinklo nustatymų diagnostika;
    - Paskirstyto programavimo įrankiai;
    - Naujos log failų generavimo priemonės ir daug kitų;
- C# 2.0:
    - Apibendrinti tipai (šablonų kūrimas);
    - Nauji iteracijos įrankiai;
    - Statinės klasės;
    - Fiksuoto dydžio struktūros ir kt.

Svarbiausias naujas patobulinimas, tai *SerialPort* klasė, kuri įgalina atlikti bendravimą per nuosekliąją sąsają nesinaudojant nesaugiu (*unmanaged*) programos kodu.

## 5.9. Rekomendacijos

Atnaujintoje sistemos versijoje rekomenduojama atlikti tokius pakeitimus:

- perrašyti C kalba parašytus valdymo modulius valdomoje C# kalboje (tuo pačiu bus sumažintas ir sistemos priklausomybių skaičių);
- esminiuose programos valdymo cikluose panaudoti įvykiais valdomą programavimą;
- sistemoje panaudoti naują .NET 2.0 programavimo karkasą.

Visa tai turi būti padaryta nekeičiant sistemos funkcionalumo ir nebloginant vykdymo charakteristikų. Šių pakeitimų analizė ir įvertinimas pateikiama eksperimentinėje darbo dalyje.

# 6. Eksperimentinė dalis

## *6.1. Pakeitimų apžvalga*

Atnaujintoje sistemos versijoje reikės atlikti tokius pakeitimus:

1) perrašyti ryšio modulius panaudojant .NET 2.0 karkasą;

2) perrašyti valdymo ciklus panaudojant įvykius (*events*).

### 6.1.1. Ryšio modulių perrašymas

Šių modulių perrašymas pagerins sistemos architektūrą keletu aspektu.

Visų pirma, bus panaikintas išorinių, sistemai reikalingų modulių sąrašas. Pradinė sistemos versija ryšiui su išoriniais įrenginiais palaikyti naudoja dvi dinamiškai prijungiamas bibliotekas, kurios parašytos C programavimo kalboje. Šios bibliotekos savo ruožtu naudoja *MFC* programavimo bibliotekas, kurių naujausios versijos nebūna laisvai prieinamos visose Windows aplinkose.

Kitas svarbus aspektas, tai sistemos palaikymo darbų palengvinimas. Skirtingose programavimo aplinkose kurtas programas, kurios turi dirbti kartu, palaikyti ir testuoti gana problematiška. Ypač pasunkėja darbas, kai reikia atrasti galimas problemas galutiniame produkte (kai nėra priėjimo prie išeities tekstų). Apjungus valdymo modulius su pagrindine sistema ir perkėlus juos į tą pačią projektavimo aplinką turėtų pastebimai palengvėti sistemos palaikymo darbai. Po pakeitimų bus panaikintas ir visas su išorinėmis operacinės sistemos funkcijomis bendraujantis programinis kodas, kuris .NET karkase yra laikomas nepatikimu (*unsafe*).

Naujo ryšio modulio realizacija neturėtų būti labai sudėtinga, nes šiam darbui atlikti bus galima pakartotinai panaudoti visą egzistuojantį programinį C kodą. Naujas ryšio modulis turės būti realizuotas C# programavimo kalboje, kuri yra tiesioginis žemesnio lygio kalbų (C/C++) palikuonis. Keistis turės tik kai kurios funkcijos ir darbas su tiesiogine atmintimi, kuriam metodų valdomoje projektavimo aplinkoje nėra. C# aplinkoje taip pat galima panaudoti visas pagalbines .NET karkaso funkcijas, kurių dėka galima pastebimai supaprastinti kai kurias kodo dalis.

### 6.1.2. Valdymo algoritmų perrašymas

Šios modifikacijos metu turi būti pakeisti valdantieji sistemos algoritmai. Jų realizacijai turi būti pritaikytos įvykiais valdomo programavimo priemonės.

Pačioje projektavimo aplinkoje yra puikios priemonės šiam tikslui realizuoti. C# programavimo kalboje kiekviena klasė gali turėti atributus, metodus, savybes (*properties*) ir taip pat įvykius (*events*). Įvykis kalboje aprašomas kaip specializuotas metodas delegatas. Tai reiškia, kad prireikus atlikti tam tikrą funkciją, šis veiksmas deleguojamas išoriniam metodui, kuris buvo įtrauktas į galinčių apdoroti tokio tipo įvykį sąrašą. Tokiu būdu tą patį įvykį apdoroti gali keletas išorinių funkcijų, kas moduliui leidžia labai lanksčiai dirbti su kitomis sistemos dalimis.

Sudėtingesnė pakeitimo dalis bus pačios sistemos perrašymas. Pagrinde reikės perrašyti modulius, kurie turi bendrauti su išoriniais įrenginiais, nes jie ir yra visų sistemos įvykių generatoriai (pavyzdžiui, įvykis gali būti generuojamas, kai pakeliamas kolonėlės pistoletas, nuskaitoma identifikacinė kortelė ar baigiamas pilti kuras). Šis darbas gali suderintas su pirmąja pakeitimų užduotimi, nes ryšio modulius vis tiek numatoma perrašyti. Neskaitant šių modulių, papildomai reikės perrašyti ir patį centrinį programos vykdymo ciklą. Šis modulis bus perrašomas ne tam, kad galėtų įvykius generuoti (panašiai kaip ryšio moduliai), bet tam, kad galėtų teisingai juos apdoroti.

Šiuos darbus reikės atlikti atsižvelgiant į vieną labai svarbų reikalavimą – neturi pasikeisti programos veikimas (funkcionalumas).

## 6.2. Galimos problemos

Didžiausia darbo dalis bus ne pačių pakeitimų realizavimas, bet pakeistos sistemos testavimas ir jos kokybės užtikrinimas. Reikės iš naujo atlikti pilną sistemos testavimą, nes pasikeis beveik visi valdantieji sistemos moduliai.

Projektavimo problemų planuojama išvengti, nes kaip jau aptarta skyriuje „Pakeitimų apžvalga“, pačioje kūrimo aplinkoje yra visos pakeitimams programuoti reikiamos priemonės.

Gali iškilti problemos C kodo transformavimo į C# programavimo kalbą metu, nes skiriasi šiuose aplinkose egzistuojančių funkcijų specifika. C kalba yra žemo lygio kalba, kurioje galima tiesiogiai dirbti su kompiuterio atmintimi, įmantriai manipuliuoti kintamuosius, masyvus, nesudėtingai kurti dinamines struktūras ir atlikti kitus veiksmus, kurie yra neprieinami valdomoje programavimo aplinkoje.

Ryšio modulių perrašymo metu gali iškilti papildomų problemų, nes juose realizuotą kodą reikės modifikuoti pagal abu pasiūlytus programos pakeitimų reikalavimus.

## 6.3. Pakeitimų realizacijos planas

Apibendrinant galima sudaryti tokį preliminarų programos pakeitimų planą:

- Ryšio modulių pakeitimas:

1) aptikti visas svarbiausias ryšio valdymo funkcijas;
2) įvertinti jų realizacijos transformavimo į kitą programavimo kalbą pasekmes;
3) perrašyti ryšio modulius C# programavimo kalboje (nauji moduliai užims senų ryšio palaikymui naudojamų klasių vietą);
4) sukurti įvykių delegatų sąsajas, per kuriuos bus kviečiamos deleguotos funkcijos;
5) realizuoti darbą su metodais įvykiais;
6) atlikti modulių testavimą;

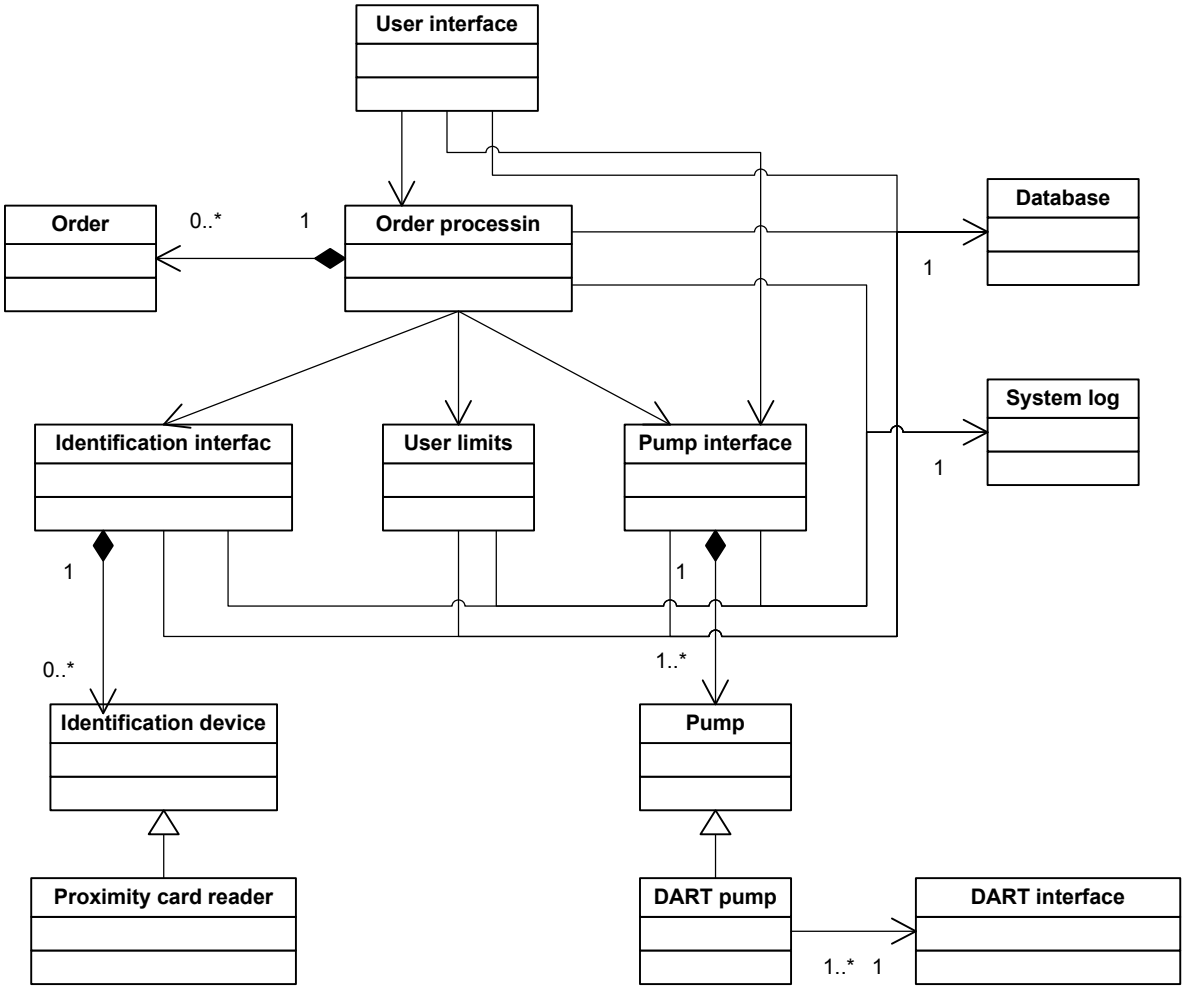
- Valdymo algoritmo perrašymas:
  1) aptikti pagrindines valdymo operacijas;
  2) šias operacijas realizuoti kaip deleguotus metodus, kurie turi būti registruojami ryšio palaikymo klasėje;
  3) sukurti įvykių delegatų sąsajas, per kurias bus kviečiamos deleguotos funkcijos;
  4) atlikti valdymo modulių testavimą;
- Pilnas sistemos testavimas.

Labiausiai neapibrėžti šiame plane yra testavimo darbai, kurių sudėtingumą ir trukmę iš anksto labai sunku prognozuoti.

## *6.4. Pakeitimų įtaka sistemos architektūrai*

### 6.4.1. Atnaujintas sistemos statinis vaizdas

Vienas iš pakeitimų reikalavimų buvo išlaikyti tokį patį sistemos funkcionalumą, nepriklausomai nuo to, kokie pakeitimai bus atliekami sistemos viduje. Tai atsispindi ir atnaujintoje sistemos klasių diagramoje:




**Pav. 14. Atnaujinta klasių diagrama**

Matome, kad iš esmės sistemos architektūros vaizdas nepasikeitė, tačiau buvo pašalinti moduliai, kurie palaiko ryšį su išoriniais įrenginiais. Šių modulių funkcionalumas perkeltas į atitinkamas „*Proximity card reader*“ ir „*DART interface*“ klases. Ryšiai tarp klasių ir paveldimumas liko nepakitęs, nes nepakito ir sistemos duomenų srautai. Atnaujinimo metu pasikeitė tik vidiniai klasių metodai.

### 6.4.2. Ryšio modulių realizacija

Ryšio moduliui realizuoti programinis kodas buvo pakartotinai panaudotas iš jau egzistuojančių C kalba parašytų bibliotekų. Šioje bibliotekoje naudojamos funkcijos buvo tiesiogiai transformuojamos į „*DART Interface*“ klasės metodus. Senas šios klasės kodas, kuris buvo kurtas tam, kad palaikytų ryšį su išorine C biblioteka, buvo pašalintas.

Galutinio modulio veikimas liko nepakitęs, tačiau vietoj dviejų programavimo kalbų ir metodologijų, visas programinis kodas buvo perrašytas naujoje vieningoje projektavimo aplinkoje.

Perdaryti programos kodo fragmentų pavyzdžiai pateikiami pirmame priede.

### 6.4.3. Įvykiais valdomų modulių realizacija

Įvykiais valdomos sistemos perdarymas susideda iš šių pagrindinių etapų:

- aptikti funkcijas, kurios atlieka periodinį bendravimą su kitais sistemos moduliais;
- šias funkcijas perdaryti į deleguojamus metodus;
- metodus užregistruoti į generuojamų įvykių eilę;
- aptikti funkcijas, kurios prieina prie sistemos statuso informacijos;
- šias funkcijas modifikuoti taip, kad pasikeitus sistemos statusui jos automatiškai kviečia visus deleguotus metodus.

Šios modifikacijos buvo atliktos išorinius įrenginius valdančiose moduliuose, o taip pat ir pačiame valdančiame cikle, kuris turėjo būti pritaikytas darbui su atnaujintais išorinių įrenginių valdymo moduliais.
Perdaryti programos kodo fragmentų pavyzdžiai pateikiami antrame priede.

## *6.5. Pakeitimų eksperimentinis tyrimas*

### 6.5.1. Tyrimo metodai

Naują sistemos variantą tirti paprastais būdais sunku, nes pakeitimų atlikimo metu sistema turėjo išlaikyti tokį patį funkcionalumą. Todėl tyrimo metu galime apžvelgti tik nefunkcines sistemos savybes, tokias kaip: programinio kodo kokybė, programos varianto kokybė ir priklausomybių tyrimas. Taip pat galime įvertinti naujo sistemos varianto naudojamų sistemos resursų pokytį.

### 6.5.2. Metrikos

Pagal pasirinktus tyrimo metodus galime sudaryti tokį tiriamų metrikų sąrašą:

- kodo pakeitimų analizė, kodo sudėtingumo pasikeitimas;
- patikimumas, brandumas, atkuriamumas ir stabilumas;
- gedimų tolerancija;
- portatyvumas;
- palaikomumas;
- efektyvumas, naudojamų resursų tyrimas;
- pritaikomumas;
- sistemos priklausomybių sąrašas.

## 6.6. Tyrimo rezultatai

### 6.6.1. Apibendrintas rezultatas

Reikia paminėti vieną svarbų sistemos modifikavimo rezultatą, kurio neina paprastai ištirti egzistuojančiais programų įvertinimo metodais. Tai įvykiais orientuotos sistemos architektūros realizacija. Kadangi vienas iš pagrindinių sistemos pakeitimo reikalavimų buvo modifikavimo metu išsaugoti tokį patį sistemos funkcionalumą, galiausiai gavome atnaujintą sistemos variantą, kuris išoriškai veikia lygiai taip pat kaip pradinis sistemos variantas.

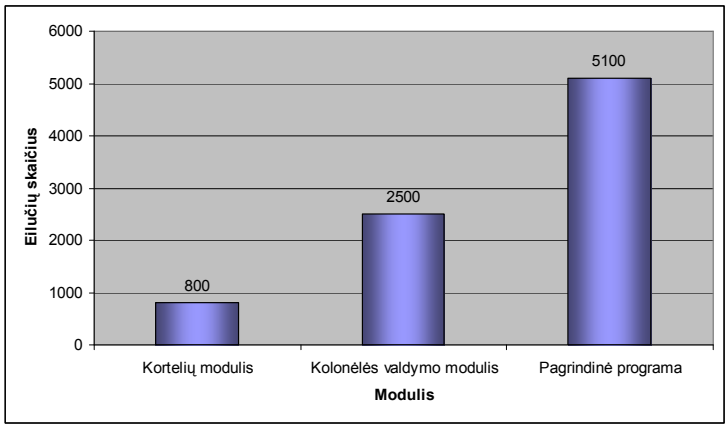
Svarbu suprasti, kad naujos sistemos architektūros pagalba pavyko išspręsti visas skyriuje „Realizacijos ir architektūros trūkumai“ paminėtas projektavimo problemas. Šių pakeitimų įtaką dalinai galime pastebėti ir kituose tyrimo rezultatuose, kurie bus analizuojami sekančiuose skyriuose.

### 6.6.2. Kodo analizė

Programinis kodas bus analizuojamas pagal tokius kriterijus:

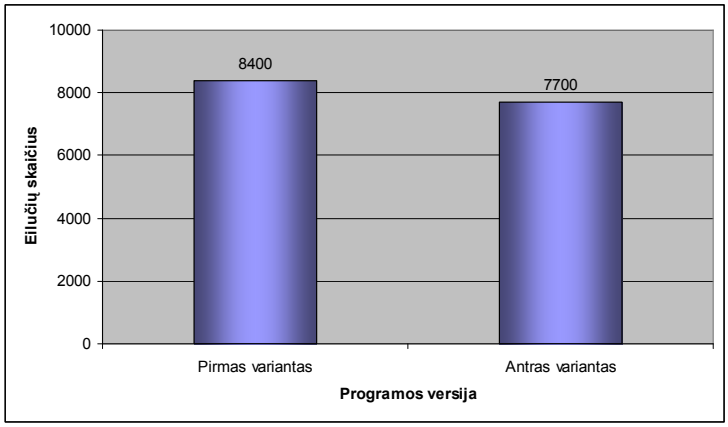
- kodo ilgis;
- kviečiamų ir kviečiančiųjų funkcijų skaičius;
- ciklomatinis sudėtingumas;
- sąlygos tikrinimo gylis;
- metodų ilgis ir skaičius;

Kodo eilučių skaičių pradinėje programoje galima pavaizduoti tokiame grafike:




**Pav. 15. Programinio kodo eilučių skaičius pradiniame programos variante**

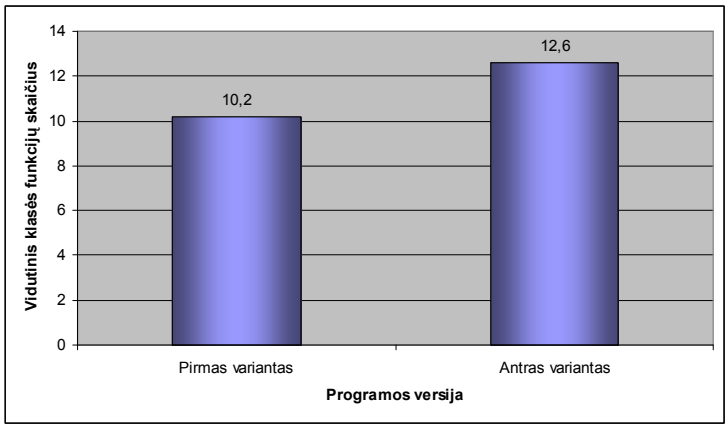
Iš grafiko matome, kad didžiąją kodo dalį sudaro pagrindinė programa. Antras pagal sudėtingumą yra kolonėlių valdymo modulis. Pakeistame programos variante visi trys moduliai bus sujungti į vieną programą. Galime palyginti pirmo ir antro programų variantų programinio kodo eilučių skaičių:




**Pav. 16. Pirmojo ir antrojo programų variantų palyginimas**

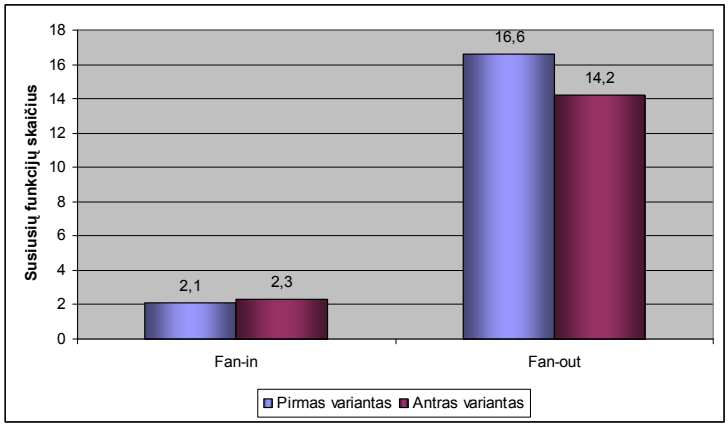
Matome, kad po programos modifikavimo kodo eilučių skaičius sumažėjo. Tai galime paaiškinti tuo, kad C# programavimo kalba, kurios pagalba buvo kuriamas antrasis programos variantas yra aukštesnio lygio ir teikia platesnį integruotų pagalbinių funkcijų sąrašą. Įtakos galėjo turėti it pačios sistemos architektūros keitimas, ką bus galima pamatyti ir kituose tyrimų rezultatuose.

Sekančiame grafike pavaizduotas vidutinis vienos programos klasės funkcijų skaičius:




**Pav. 17. Vidutinis vienos klasės funkcijų skaičius**

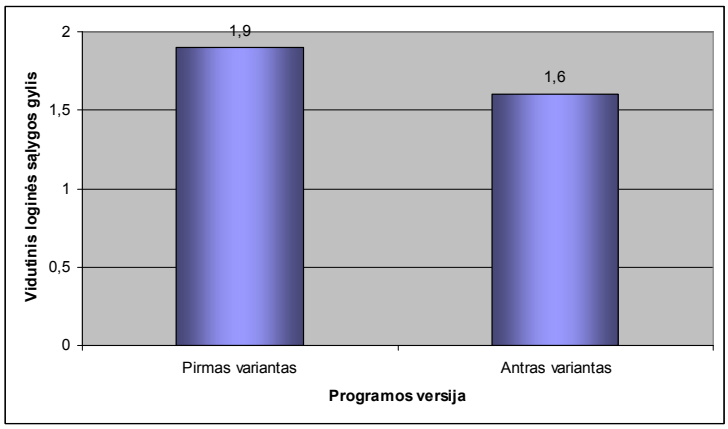
Iš grafiko matome, kad antroje programos versijoje padidėjo funkcijų skaičius. Tačiau atsižvelgiant į tai, kad bendras programos eilučių skaičius sumažėjo, galime daryti išvadą, kad ir kiekvienos funkcijos kodo eilučių skaičius antrame programos variante yra mažesnis. Galime įvertinti ir programų kviečiamų bei kviečiančiųjų funkcijų (*fan-in, fan-out*) skaičių:




**Pav. 18. Kviečiančių ir kviečiamųjų funkcijų palyginimas**

Pagal grafiko rezultatus matome, kad funkcijos sistemoje mažai įtakoja kitas funkcijas, bet yra stipriai priklausomos nuo kitų funkcijų (šiuo atvejų tai dauguma sisteminių funkcijų). Antrame programos variante šiek tiek padidėja funkcijų įtaka kitoms funkcijoms ir sumažėja jų priklausomybė nuo kitų funkcijų. Tai galime paaiškinti tuo, kad įvykiais valdomoje sistemoje metodai dažniausia nesaugo jokios vidinės būsenos ir būna iškviečiami su visais reikalingais parametrais, dėl ko šie metodai savo darbo metu gali mažiau kreiptis į išorines funkcijas.

Sekančiame grafike pavaizduotas vidutinis sąlygos tikrinimo gylis:




**Pav. 19. Vidutinis sąlygos tikrinimo gylis**

Iš grafiko matome, kad atnaujintoje programos versijoje vidutinis sąlygos tikrinimo gylis sumažėjo. Tai galime paaiškinti tuo, kad antroje sistemos versijoje sudėtingesni metodai buvo išskaidyti į paprastesnius, kas leido sumažinti galutinį programos sudėtingumą.

### 6.6.3. Patikimumas ir stabilumas

Iš programos kodo analizės matome, kad vidutinis sistemos sudėtingumas šiek tiek sumažėjo. Tai yra vienas iš sistemos kokybę ir patikimumą įtakojančių faktorių. Kitas faktorius, kuris teigiamai paveikė sistemos stabilumą, tai valdymo modulių perkėlimas į vientisą

projektavimo aplinką. Tai ne tik palengvina šių modulių kūrimo, testavimo bei derinimo darbus, bet tuo pačiu užtikrina ir lengvesnius palaikymo darbus vėlesniuose sistemos gyvavimo ciklo etapuose.
Galime sakyti, kad priimtų techninių sprendimų pagalba sistemos patikimumas ir stabilumas padidėjo.

### 6.6.4. Gedimų tolerancija

Valdymo modulius įjungus į pagrindinės sistemos kodo medį atsirado geresnės galimybės gedimams nustatyti, įveikti ir išvengti. Pirmoje programos versijoje šios galimybės buvo žymiai labiau ribotos, nes visas ryšys su išorine aparatūrine įranga buvo palaikomas per išorinius ryšio modulius. Jų sutrikimo atveju, pagrindinė programa praktiškai neturėjo jokių realių reagavimo galimybių išskyrus pakartotinį sistemos užkrovimą. Dėl šių pakeitimų naujoje sistemoje galima lengviau aptikti, išanalizuoti ir išvengti galimus sistemos gedimus.

### 6.6.5. Portatyvumas

Tai vienas iš programos kokybės reikalavimų, kuris pasidaro labai svarbus, kai sistema pradedama taikyti kompiuteriuose, kurių aparatūrinė įranga nėra lengvai tarpusavyje suderinama. Portatyvus kodas kai kuriose sistemose laikomas būtinu sistemos reikalavimu. Šios sistemos atveju, pritaikius atnaujintą .NET programavimo karkasą atsirado galimybė kurti gimtąsias 64 bitų programas. Ateityje (kai kuriose srityse jau ir šiuo metu) tai gali būti vienas iš perspektyviausių rinkos segmentų.

### 6.6.6. Palaikomumas

Programinės įrangos palaikomumas tiesiogiai siejasi su programinio kodo kokybe ir pačios programinės sistemos sudėtingumu. Vienas iš pradinių šio darbo tikslų buvo sukurti lengvai palaikomą ir išplečiamą sistemą. Realizavus visus rekomenduotus sistemos pakeitimus, palaikymo darbai turėtų pastebimai palengvėti. Vietoj trijų skirtingų programos modulių ir dviejų programavimo kalbų, atnaujintoje sistemoje lieka tik viena pagrindinė valdymo programa.

### 6.6.7. Naudojami resursai

**Lentelė 5. Sistemos naudojami resursai**

| Resursas | Programos versijos | |
|---|---|---|
| | Pirmoji programos versija | Antroji programos versija |
| Gijų skaičius | 11 | 7 |
| Fizinės atminties panaudojimas | 9.616 KB | 8.340 KB |
| Viso darbinės atminties | 16.616 KB | 10.644 KB |
| Procesoriaus resursų panaudojimas ($10^{-2}$) | 0,234% | 0,187% |

Kaip ir buvo tikėtasi, patobulinta sistemos versija naudoja mažiau sistemos resursų. Tai galima paaiškinti tuo, kad buvo atsisakyta skirtingose programavimo aplinkose parašytų modulių. Antroje programos versijoje vykdoma viena gija mažiau, nei pradinėje versijoje (dar trys gijos buvo naudojamos išorinėms bibliotekoms prijungti). Dėl tos pačios priežasties sumažėjo ir sistemos naudojamos atminties kiekis. Galime daryti išvadą, kad procesoriaus naudojamų resursų skaičius, kaip ir buvo planuota, buvo sumažintas dėl pasirinktos įvykiais valdomos programavimo metodikos.

### 6.6.8. Sistemos priklausomybės

Šiame skyriuje bus aptariamas abiejų programos versijų išorinių priklausomybių sąrašas. Geriausia šią informaciją lyginti lentelėje:

**Lentelė 6. Programos versijų priklausomybės**

| Savybė | Programos versijos | |
|---|---|---|
| | Pirmoji programos versija | Antroji programos versija |
| Programavimo kalbos | C, C++, C# | C# |
| Naudojamos programavimo bibliotekos | C, C++, MFC ir .NET | .NET |
| Būtinos programinės bibliotekos | MFC71.dll, MSVCR71.dll, mscoree.dll ir ktios Windows bibliotekos | mscoree.dll |
| Išorinės bibliotekos | dart.dll, card.dll | nėra |
| .NET karkaso versija | 1.0 | 2.0 |
| Kodo tipas | Maišytas (C ir nevaldomas C#) | C# |

Iš lentelės matome, kad atnaujintoje programos versijoje atsisakoma didelio skaičiaus iki tol naudotų programavimo priemonių. Kaip buvo aptarta ir ankstesniuose skyriuose, tai ne vien tik leidžia sukurti kokybiškesnę ir patikimesnę sistemą, bet taip pat leidžia ir sumažinti programinės įrangos palaikymo darbų išlaidas.

## 6.7. Eksperimentinio tyrimo išvados

Pats svarbiausias rezultatas, kuris buvo pasiektas eksperimentinio tyrimo metu – tai įvykiais orientuotos sistemos architektūros sukūrimas, kuris leido išspręsti visas tiriamojoje dalyje atskleistas sistemos problemas.

Eksperimentinio tyrimo metu buvo ištirta rekomenduotų sistemos pakeitimų realizacija. Iš rezultatų matome, kad siūlyti pakeitimai leido sistemą patobulinti keletu aspektu. Visų pirma, sukurta nauja vieninga programinio kodo bazė. Tai leido užtikrinti geresnį sistemos patikimumą bei stabilumą. Tuo pačiu buvo pagerintos ir sistemos vykdymo bei resursų naudojimo charakteristikos. Rezultate gavome atsparesnę ir efektyvesnę sistemą, kuri reikalauja mažiau palaikymo darbų išlaidų.

# 7. Išvados

Analitinėje darbo dalyje buvo sudarytas bendras kuro kolonėlių valdymo sistemos vaizdas, aptartos galimos projektavimo problemos ir jų sprendimo būdai. Sistemos architektūra plačiau detalizuota ir apibrėžta projektinėje dalyje, kur pateiktas funkcinių ir nefunkcinių reikalavimų sąrašas bei svarbiausios sistemos UML diagramos.

Tiriamoje darbo dalyje atliktas galutinės sistemos tyrimas, kur buvo vertinama sistemos kokybė ir realizuotos architektūros pranašumai ir trūkumai. Sudarytas rekomenduojamų pataisymų sąrašas, kurių pagrindu ir buvo atliekamas eksperimentinis sistemos tyrimas. Šio tyrimo metu buvo ištirta pakeisto sistemos varianto realizacija. Iš rezultatų matome, kad rekomenduoti pakeitimai leido sistemą patobulinti keletu svarbių aspektų. Pirmiausia buvo sukurta nauja vieninga programinio kodo bazė. Tai leido užtikrinti geresnį sistemos patikimumą bei stabilumą. Taip pat buvo pagerintos ir sistemos vykdymo bei resursų naudojimo charakteristikos. Gavome atsparesnę gedimams ir tuo pačiu efektyvesnę sistemą, kuri reikalauja mažiau palaikymo išlaidų. Svarbus rezultatas, kuris buvo pasiektas eksperimentinio tyrimo metu – tai įvykiais orientuotos sistemos architektūros sukūrimas, kuris leido išspręsti visas tiriamojoje dalyje atskleistas sistemos architektūros problemas.

Apibendrinant galima daryti išvadą, kad tiriamoje dalyje siūlyti pakeitimai pasiteisino praktikoje. Tai patvirtina eksperimentinio tyrimo rezultatai.

# 8. Literatūra


[1] *SADOSKI, D. Client/Server Software Architectures. An Overview. 1997 m. rugpjūtis, žiūrėta 2006-04-15.*

Prieiga per internetą: <http://www.sei.cmu.edu/str/descriptions/clientserver_body.html>.

[2] *GERARD, J.; RAJEEV, J. Reliable Software Systems Design: Defect Prevention, Detection, and Containment. Jet Propulsion Laboratory, California Institute of Technology. Žiūrėta 2006-05-01.*

Prieiga per internetą: <http://vstte.ethz.ch/Files/holzmann-joshi.pdf>.

[3] *HOWARD, M. The 19 Deadly Sins of Software Security. Michael Howard's Web Log, žiūrėta 2006-04-20.*

Prieiga per internetą: <http://blogs.msdn.com/michael_howard/archive/2005/07/11/437875.aspx>.

[4] *ŠTUIKYS, V. Programų priežiūra [skaidrės]. Programų keitimas [skaidrės]. Žiūrėta 2006-04-20.*

[5] *MONTLICK, T. What is Object-Oriented Software? Software Design Consultants, LLC. 1999 m., žiūrėta 2006-05-05.*

Prieiga per internetą: <http://www.softwaredesign.com/objects.html>.

[6] *C. MYERS, N. Memory Management in C++. 1993 m., žiūrėta 2006-04-10.*

Prieiga per internetą: <http://www.cantrip.org/wave12.html>.

[7] *Real-time computing. Wikipedia, žiūrėta 2006-05-02.*

Prieiga per internetą: <http://en.wikipedia.org/wiki/Real-time_computing>.

[8] *KATZ, D. Error codes or Exceptions? Why is Reliable Software so Hard? 2006 m. balandis., žiūrėta 2006-05-27.*

Prieiga per internetą: <http://damienkatz.net/2006/04/error_code_vs_e.html>.

[9] *MOTIEJŪNAS, K. Kokybė [skaidrės]. Žiūrėta 2006-05-22.*

[10] *Programming paradigm. Wikipedia, žiūrėta 2006-05-03.*

Prieiga per internetą: <http://en.wikipedia.org/wiki/Programming_paradigm>.

[11] *RAMAMURPHY, B. Event Driven Systems and Modelling [skaidrės]. Žiūrėta: 2006-04-20.*

[12] *STOLPMAN, G. The Enqueue user's guide. 2000 m.*

[13] *DABEK, F.; ZELDOVICH, N., KAASHOEK F., MORRIS, R. Event-driven Programming for Robust Software. MIT laboratory for Computer Science.*

[14] *What‘s new in the .NET Framework Version 2.0. .NET Framework Developer‘s Guide, Microsoft, žiūrėta 2006-05-12.*

Prieiga per internetą: <http://msdn2.microsoft.com/en-US/library/t357fb32.aspx>.

[15] *FREEMAN, A.; JONES, A. Programming .NET Security. O‘Reilly 2003.*

[16] *WHEELER, D. Secure Programming For Linux and Unix. Žiūrėta 2005-10-15.*

Prieiga per internetą: <http://www.tldp.org/HOWTO/Secure-Programs-HOWTO/index.html>.

[17] *STENEK, J. Introduction to RS 422 & RS 485. Žiūrėta 2004-12-01.*

Prieiga per internetą: <http://www.hw.cz/english/docs/rs485/rs485.html>.

# 9. Terminų ir santrumpų žodynas

| Terminas | Paaiškinimas |
|---|---|
| **Administratorius** | Žmogus arba žmonių grupė, kurie atsakingi už sistemos nustatymų keitimą. Administratoriai gali keisti beveik visus sistemos nustatymus, tame tarpe ir kas gali pilti degalus, kaip turi veikti sistemoje naudojamos kortelės ir t.t. |
| **DLL** | Dynamically linked library. Dinamiškai prijungiama biblioteka. |
| **DART protokolas** | Dresser Wayne tipo kolonėlių valdymo protokolas. |
| **DBVS** | Duomenų bazių valdymo sistema. |
| **Degalų kolonėlė** | Įrenginys, kuris atlieka valdomos degalų pompos vaidmenį. |
| **Degalų tiekėjas** | Žmogus arba įmonė, kuri tiekia degalus degalinei. |
| **GC** | *Garbage collection*, atminties šiukšlių surinkimas. Terminas vartojamas sistemose, kuriuose įdiegti atminties valdymo metodai. |
| **LAN** | *Local area network. V*ietinis kompiuterių tinklas. |
| **Metrikos** | Specifiniai indikatoriai, pagal kuriuos galima matuoti pasiekiamų rezultatų progresą. |
| **MFC** | *Microsoft foundation classes*. C++ programavimo kalbai skirtas Microsoft klasių rinkinys. |
| **Memory leak** | Tai terminas, kuris apibūdina vieną iš sunkiausiai išsprendžiamų programavimo klaidų. Atmintis priskiriama programiniam moduliui, kuris vėliau neatlaisvinęs priskirtos atminties baigia savo vykdymą. |
| **Managed environment** | Programavimo aplinka, kurioje griežtai kontroliuojamos programos ir atminties valdymo operacijos. Menkiausias nusižengimas programoje iškart stabdomas apie tai atitinkamai pranešant sistemos vartotojui. |
| **Proximity card** | Plastmasinė (kreditinės kortelės dydžio) kortelė, kurią nuskaityti galima jos fiziškai nepalietus (t.y. magnetinių bangų pagalba). |
| **Pompa, dispenseris, kuro kolonėlė** | Įrenginys, kuris išduoda (pila) kurą vartotojui. |
| **POS** | Point- of- sale, vieta, kurioje vykdomi pardavimai (kalbant apie techninę ar programinę įrangą). |
| **RTS, RLS** | *Real time system.* Tikralaikiu dirbanti sistema. |
| **OS** | Operacinė sistema. Personalinio kompiuterio valdymo programinė įranga. |
| **ODBC** | *Open database connectivity*. Atvira duomenų bazių ryšio sąsaja. |
| **Operatorius** | Žmogus, atsakingas už teisingą sistemos darbą. Turi numatyti, kada reikia stabdyti/leisti sistemą, pajamuoti kurą, prižiūrėti degalų talpyklas ir kolonėles, nustatyti degalų kolonėlių kuro išdavimo normas ir t.t. |

| Terminas | Paaiškinimas |
|---|---|
| **PK** | Personalinis kompiuteris. |
| **Proximity card** | Nuotolinio veikimo kortelė, iš kurios per atstumą galima nuskaityti unikalų numerį. |
| **RS-232** | Sąsaja tarp bendraujančių sistemų, kuri duomenų perdavimui naudoja nuoseklius dvejetainius duomenų srautus. |
| **RS-485** | Tai panašus į RS-232 protokolas, kuris leidžia pritaikius kabelį iš vienos vytos poros laidų, sujungti iki 32 mazgų maksimaliu atstumu iki 1200 metrų. |
| **Ryšio linija** | Ryšio terpė, per kurią gali būti perduota informaciją tarp dviejų ar daugiau mazgų. |
| **Simuliatorius** | Degalų kolonėlių kontekste, simuliatorius yra toks įrenginys, kuris leidžia pritaikius minimalią techninės įrangos komplektaciją, imituoti tam tikro įrenginio darbo charakteristikas. Dažniausia naudojamas degalų kolonėlės simuliatorius. |
| **Valdantis kompiuteris** | Kompiuteris, kuris bus ryšio linijos pagalba sujungiamas su degalų kolonėlėmis. Šiame kompiuteryje turi būti įdiegta valdanti programinė įranga. |
| **Vartotojas** | Žmogus (ar transporto priemonė), kuri dirba su degalų kolonėle ir degalų kolonėlių valdymo programa tam, kad atliktų degalų pylimo veiksmą. |
| **UML** | *Unified Modeling Language*. Suvienyta modeliavimo kalba. |
| **UTP** | *Unshielded twisted pair* (neekranuota vyta pora, kalbant apie kabelius). |
| **Unicode** | 16 bitų raidžių kodavimo sistema, leidžianti kompiuteryje atvaizduoti beveik visus pasaulyje naudojamus rašmenis. |
| **UTF-8** | Vienas iš Unicode kodavimo variantų. Jis leidžia vienoje programoje naudoti pačių įvairiausių pasaulio kalbų rašmenis, netgi vienoje eilutėje. |

# 10. Priedai

## *10.1. Ryšio modulio kodo fragmentai*

Pradinis ryšio su kolonėlėmis palaikymo modulio kodas:

```
// dart.def
LIBRARY      "dart"
EXPORTS
    ; Explicit exports can go here
      ; Prijungimo/atjungimo funkcijos
      PumpConnect                       @1
      PumpDisconnect                    @2

      ; Pagrindines pompos komandos
      CommandToPump                     @3
      AllowedNozzleNumbers              @4
      PresetVolume                      @5
      PresetAmount                      @6
      PriceUpdate                       @7
{...}

// pump.h
{...}
// Prijungimo/atjungimo funkcijos
DLLEXPORT BOOL PumpConnect(BYTE PumpNumber, BYTE PortNumber, BYTE PumpAddress);
DLLEXPORT BOOL PumpDisconnect(BYTE PumpNumber);

// Pompos komandos
DLLEXPORT BOOL CommandToPump(BYTE PumpNumber, BYTE Command);
DLLEXPORT BOOL AllowedNozzleNumbers(BYTE PumpNumber, BYTE nozzles[], BYTE length);
DLLEXPORT BOOL PresetVolume(BYTE PumpNumber, LONG volume);
DLLEXPORT BOOL PresetAmount(BYTE PumpNumber, LONG amount);
DLLEXPORT BOOL PriceUpdate(BYTE PumpNumber, LONG prices[], BYTE length);
{...}

// dart.cpp
#include <stdio.h>
#include <stdlib.h>
#include "stdafx.h"
#include "dart.h"
#include "pump.h"

// Control characters
#define POLL            0x20
#define DATA            0x30
#define ACK             0xC0
#define NAK             0x50
#define EOT             0x70
#define ACKPOLL         0xE0

// Buffer sizes (maximum dart message size is 256 bytes)
#define DART INPUT BUFFER SIZE                 256
#define DART OUTPUT BUFFER SIZE                256

// Functions
DWORD WINAPI      MainCycle(LPVOID pParam);
HANDLE            OpenPort(BYTE PortNumber);
WORD              CalculateCRC16(BYTE *buffer, DWORD buffer size);

BOOL DartConnectPump(BYTE PumpAddress, BYTE PortNumber)
{
      AFX_MANAGE_STATE(AfxGetStaticModuleState( ));
      PumpInfo *pi;
      POSITION pos;

      if (!ports[PortNumber])
      {
            ports[PortNumber] = OpenPort(PortNumber);
            if (ports[PortNumber] == INVALID HANDLE VALUE)
                  return FALSE;
      }
      {...}
}
```

```
{...}
```

Sukompiliavę šį kodą gauname kolonėlių ryšio valdomo biblioteką „dart.dll“.

Transformuotas programos kodas tiesiog tampa C# klase:

```
using System;
using Unitechna;

namespace Unitechna.Unimachine
{
      public delegate void FillingInformationDelegate(double volume, double amount);
      public delegate PumpStatus PumpStatusDelegate();
      {...}

      public class DARTInterface
      {
            public DARTInterface()
            {
                  {...}
            }
            private bool PumpConnect(byte PumpNumber, byte PortNumber, byte PumpAddress)
            {
                  {...}
            }
            private bool PumpDisconnect(byte PumpNumber)
            {
                  {...}
            }

            public event PumpStatusDelegate OnPumpStatus();
            public event FillingInformationDelegate OnFillingInformation();

            private void MainCycle()
            {
                  // Nuskaitoma informacija is COM porto
                  {...}
                  if (OnPumpStatus != null)
                        OnPumpStatus();
                  {...}
            }
      }
}
```

## *10.2. Valdymo modulio kodo fragmentai*

Valdymo modulio realizacija atlikta panašiai, kaip ir ryšio modulio. Ciklai transformuoti į metodus, kurie pastatomi į ryšio modulių įvykių eilę. Pačiame valdymo modulyje taip pat realizuota keletas įvykių, kurie gali perduoti reikalingą informaciją apie sistemos statusą kitoms sistemos dalims.

Pradinio kodo fragmentai:

```
class OrderProcessing
{
      {...}

      public void ProcessOrders()
      {
            PumpStatus status;
            bool NozzleStatus = false;
            bool OldStatus = false;
            int retry;
            int i;

            while (!MainThreadEvent.WaitOne(100, false))
            {
                  for (i = 0; i < PumpCount; i++)
                  {
                        // Surenkami pradiniai duomenys
                        status = pump[i].GetStatus();

                        // reset ready pump
```

```
                        {...}

                        // pylimo pradejimas
                        {...}

                        // pylimo uzbaigimas
                        {...}
                  }
            }
      }

      {...}
}
```

Atnaujinto kodo fragmentai:

```
class OrderProcessing
{
      public delegate void StartTransactionDelegate();
      public delegate void EndTransactionDelegate();

      public void OrderProcessing(Pumps[] pumps)
      {
            {...}
            for (i = 0; i < pumps.Count(); i++)
            {
                  // uzregistruojami event handleriai
                  pump[i].OnStatus += ResetPump;
                  pump[i].OnFillingInformation += FinishTransaction;
                  {...}
            }
            {...}
      }

      public void ResetPump()
      {
            {...}
      }

      public void StartTransaction()
      {
            {...}
      }

      public void FinishTransaction()
      {
            {...}
      }

      public event StartTransactionDelegate OnStartFilling();
      public event EndTransactionDelegate OnFinishFilling();

      {...}
}
```

Naujame sistemos variante sistemos statusas valdymo modulį pasiekia tik tada, kai jis iš tiesų pasikeičia. Pats valdymo modulis taip pat atveria lankstesnę sąsają savo būsenai sekti.