KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justas Jokubauskas

Abstrakčios sintaksės medžių pertvarkymo algoritmų tyrimas

Magistro darbas

Darbo vadovas
prof. dr. L. Nemuraitė

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Justas Jokubauskas

Abstrakčios sintaksės medžių pertvarkymo algoritmų tyrimas

Magistro darbas

Recenzentas
doc. dr. V. Pilkauskas
2010-05-31

Vadovas
prof. dr. L. Nemuraitė
2010-05-31

Atliko
IFM-4/2 gr. stud.
Justas Jokubauskas
2010-05-31

Kaunas, 2010

Turinys

Investigation of Refactoring Algorithms for Abstract Syntax Tree

SUMMARY

Over the past years, agile development methodologies have attracted a lot of attention. Refactoring has become one of the most heavily used practices, especially in Extreme Programming. Therefore the need to have powerful refactoring tools has grown to such an extent, that there is a required feature for modern-day IDEs to have implemented refactoring tools.

The aim of this work is to build a refactoring library for the generic abstract syntax tree and test the algorithms for speed. Generic abstract syntax tree (GAST) is a tree structure that can store elements of several programming language.

Achieving this goal required doing certain tasks that you find in this document: analysis of technologies and existing software; several algorithms for refactoring; user need and specification of requirements; library model, expressed in UML diagrams; test plan and test procedure.

Experiments have shown that it is necessary to improve the refactoring algorithms. But the library and the AST structure is worth further development.

Keywords: refactoring, abstract syntax tree, generic abstract syntax tree, refactoring library.

1. ĮVADAS

Pastaraisiais metais populiarėja programų kūrimo metodikos, kurios leidžia kūrėjui sparčiau pateikti veikiančią sistemos versiją vartotojui, bei prisitaikyti prie kintančių reikalavimų. Vienas iš tokių yra XP, arba ekstremalus programavimas (angl. Extreme Programming), tai vienas iš lanksčiojo programavimo (angl. Agile) metodų. Kad būtų galima sparčiau rašyti programų kodą bei pagerinti kodo suprantamumą, kūrėjams būtini automatiniai kodo pertvarkymo įrankiai. Dėl šios priežasties kiekviena šiuolaikinė integruota programų kūrimo aplinka (angl. IDE - Integrated Development Environment) turi integruotus pertvarkymo įrankius.

Šiuo metu pasaulyje naudojama daug įvairių programavimo kalbų, kiekviena turi savo paskirtį ir jas naudojančius programuotojus. Kiekviena programavimo kalba savaip išskirtina, turi savo sintaksę, pritaikymo sritį ir t.t. Tačiau nemaža dalis kalbų turi panašių ar netgi vienodų struktūrų: ciklai, sąlyginiai sakiniai, funkcijos. Šie panašumai leidžia mums sukurti įrankius, galinčius apdoroti daugiau nei vieną programavimo kalbos kodą.

Kad nereiktų kiekvienai programavimo kalbai kurti atskiro programos kodo pertvarkymo įrankio, galima sukurti vieną įrankį numatytai programavimo kalbų aibei ir sutaupytą laiką skirti sudėtingesnių pertvarkymo algoritmų kūrimui. Šiame darbe sprendžiama **problema**, kaip sukurti tokį įrankį

Tyrimo sritis – abstrakčios sintaksės medžių pertvarkymo algoritmai.

Tyrimo objektas – abstrakčios sintaksės medžių pertvarkymo procesai skirtingose programavimo kalbose (C++ ir Java).

Darbe buvo analizuojama:

- Bendrinio AST struktūra [4, 10, 12].
- XML aprašomosios kalbos panaudojimas AST struktūrų perdavimui [5, 6, 8, 9, 13].
- Programavimo kalbų pertvarkymų metodika [1, 2, 3, 7, 11].

Šio **darbo tikslas** – ištirti, kaip efektyviai būtų galima vykdyti pertvarkymus bendriniam AST, tam sukuriant pertvarkymų biblioteką ir atliekant eksperimentus įvairių kalbų atvejais.

Šiam tikslui pasiekti reikia įvykdyti **uždavinius**:

1. Išanalizuoti bendrinio AST struktūrą ir programavimo kalbų pertvarkymų metodiką.
2. Sudaryti pertvarkymo algoritmus.
3. Realizuoti ir ištestuoti pertvarkymų biblioteką.
4. Atlikti eksperimentą ir įvertinti darbo rezultatus.

Tyrimo metodika. Tyrime buvo taikoma literatūros analizė, sprendimui konstruoti buvo taikomi ekstremalaus programavimo ir testais grindžiamo kūrimo principai, atliekamas eksperimentinis tyrimas.

Kokybės kriterijai: bendrinio AST pertvarkymų greitaveika, kuri šiame darbe yra svarbesnė už pertvarkymų tikslumą, nes norima nustatyti, ar verta šią biblioteką plėtoti toliau. Net jei pertvarkymai būtų visada tikslūs, bet vykdymo laikas būtų neproporcingai didelis, tai tokios bibliotekos naudojimas būtų nepraktiškas ir labai nepatogus programuotojui. Dėl šios priežasties pirma atliekama analizė, kaip greitai vykdomi pagrindiniai pertvarkymo algoritmai. Greitaveikai turi įtakos:

- Naudojamas algoritmas – jei dėl lėtos greitaveikos kaltas algoritmas, jį galima pakeisti.
- Bendrinės AST struktūros ir pertvarkymo bibliotekos realizacijos trūkumai – tuomet reikia atlikti pataisymus arba atsisakyti tolesnių plėtojimo darbų.

Darbo struktūra.

2 skyriuje pateikiama technologijų, panaudotų pertvarkymo bibliotekos kūrime, analizė. Taip pat pateikiamas panašias funkcijas atliekančių įrankių palyginimas.

3 skyriuje aprašomi realizuoti pertvarkymo algoritmai.

4 skyriuje pateikiamas sukurtos pertvarkymų bibliotekos projektas, kurį sudaro: bibliotekai keliami reikalavimai, bibliotekos architektūra, testavimo planas ir testavimo rezultatai.

5 skyriuje aprašomas atliktas tyrimas bei pateikiami įvykdyto eksperimento rezultatai ir išvados.

6 skyriuje pateikiamos atlikto darbo išvados.

Šiame darbe eksperimentui atlikti naudojama programinė įranga: bendrinio AST biblioteka ir jos pertvarkymų biblioteka yra sukurtos sistemos dalis, kurią sudaro: bendrinio AST biblioteka, bendrinio AST pertvarkymų biblioteka, AST transformavimo biblioteka bei pagalbiniai įrankiai. Bendrinio AST pertvarkymų biblioteka sukurta šio darbo autoriaus – Justo Jokubausko. AST transformavimo bibliotekos kūrėjas – kolega, Rytis Stankevičius. Bendrinio AST biblioteka sukurta abiejų kūrėjų kartu.

2. AST PERTVARKYMO TECHNOLOGIJŲ, STRUKTŪROS IR ALGORITMŲ ANALIZĖ

Šiame skyriuje aprašoma literatūros šaltinių analizė bei panašias funkcijas atliekančių įrankių tyrimas, pagrindžiamas atitinkamų technologijų naudojimas pertvarkymų bibliotekos kūrime.

2.1 Programų kodo pertvarkymas

Programos kodo pertvarkymas (angl. refactoring) – tai procesas, kurio metu programinės įrangos kodas keičiamas taip, kad nepasikeistų sistemos išorinė elgsena, o vidinė struktūra pagerėtų [1]. Programos kodo pertvarkymas yra procesas kuris blogą programos kodą perverčia į gerą kodą [7]. Pertvarkomas kodas nebūtinai turi būti blogas, pertvarkymas gali būti atliekamas programuotojo darbo palengvinimui.

Esminės kodo pertvarkymo savybės:

- Pertvarkymo įrankiai yra reikalingi ir svarbūs programavimui, bet jie taip pat sudėtingi, nes nelengva sutvarkyti chaosą.
- Pertvarkymas vykdomas pagal logikos taisykles. Išspręsti uždavinį paprastai galima bent keletu būdų, todėl skirtingi žmonės pasirenka skirtingus sprendimo variantus. Kai tai daro programuotojai, kodas gali tapti sunkiai skaitomas, todėl pravartu turėti įrankius galinčius suvienodinti kodą.
- Pertvarkymo algoritmai gali padaryti kodo struktūras lengviau suprantamas didesniam ratui žmonių.

Pertvarkymo mastai gali būti dideli arba maži. Idealiu atveju maži pertvarkymai naudojami taip dažnai, kad didelių net neprireikia [2]. Pertvarkymas nėra programos kodo perrašymas naujai, aiškiau ir geriau, nes tai per didelė rizika. Pertvarkymas tai esamo kodo perskirstymas taip, kad jo struktūra taptų geresnė. Vieno iš pertvarkymų pavyzdys:

Metodo pavadinimo keitimas [1].

Atliekami tokie žingsniai:

- Patikrinkite ar keičiamo metodo aprašas nėra naudojamas tėvo ar vaikų klasėse, jei tokių yra. Jei naudojamas, tai sekančius žingsnius reikia atlikti kiekvienai klasei atskirai.
- Sukurkite naują metodą su pageidaujamu pavadinimu ir įdėkite seno metodo turinį. Jei reikia, atlikite pakeitimus.
- Sukompiliuokite
- Pakeiskite seno metodo turinį taip, kad jis iškviestų naująjį metodą

- Sukompiliuokite ir išbandykite
- Suraskite visas nuorodas į seną metodą ir pakeiskite jas nuorodomis į naują metodą.
- Jeigu galima, ištrinkite seną metodą, kitu atveju pažymėkite, kad jis pasenęs.
- Sukompiliuokite ir išbandykite.

Atlikti visiems šiems žingsniams žmogui reikėtų sugaišti daug laiko ir pastangų. Jeigu šis pertvarkymo metodas yra automatizuotas ir atliekamas kompiuterio, tai atlikimo laikas žymiai sumažinamas.

2.2 XML ir JAVA technologijų taikymas pertvarkymo procese

XML (angl. Extensible Markup Language) – duomenų struktūrų ir jų turinio aprašomoji kalba [6]. Šią kalbą rekomenduoja W3C. Šios kalbos paskirtis užtikrinti nesudėtingą duomenų apsikeitimą tarp sistemų. Pagrindinis XML kalbos vienetas yra elementas. Kiekvienas elementas turi savo vardą ir gali turėti atributus, kitus elementus bei tekstą. Atributas turi vardą ir reikšmę. Elemento viduje esantys elementai dar vadinami dukteriniais elementais [13].

Pavyzdys:

```
<vartotojai>
  <vartotojas numeris="123456">
    <vardas>Jonas</vardas>
    <pavarde>Jonaitis</pavarde>
  </vartotojas >
  <vartotojas numeris="123457">
    <vardas>Petras</vardas>
    <pavarde>Petraitis</pavarde>
  </vartotojas>
  <papildoma_informacija node="node"/>
</vartotojai>
```

Elementai: vartotojai, vartotojas, vardas, pavarde, papildoma_informacija.

Elementas „vartotojas“ turi du dukterinius elementus: vardas ir pavarde. Elementas „vartotojas“ turi atributą pavadinimu „numeris“ su reikšme 123456.

Kadangi mūsų naudojami duomenys saugomi medžio struktūroje, XML dokumentai idealiai tinka jiems perduoti, todėl ir buvo pasirinkta būtent ši aprašomoji kalba.

Pasirinkus geriausiai tinkamą duomenų struktūros aprašomąją kalbą, telieka rasti geriausią būdą ją panaudoti JAVA programavimo kalba, nes ši kalba naudojama sistemos kūrimui.

Sukurti programą, kuri išvestų duomenis į XML dokumentą, yra nesudėtingas procesas, kurį aprašyti gali ir pradedantieji programuotojai [5]. Taip teigiama dėl to, kad duomenų išvedimas į

XML dokumentą toks pat, kaip standartinis tekstinis duomenų atvaizdavimas. Duomenų išvedimui naudojamas išvedimas srautais.

Duomenų nuskaitymas iš XML dokumento yra žymiai sudėtingesnis, nes reikia nuskaityti ir atrinkti:

- Elementus
- Dukterinius elementus
- Atributus
- Susijusį tekstą.

Nuskaitymo darbo palengvinimui yra sukurti XML analizatoriai (angl. parser) [8, 9]. Kad nereiktų kiekvienam programuotojui kurti nuosavo nuskaitymo įrankio, tam yra sukurti analizatoriai, kurie nuskaito dokumentą ir sudeda duomenis į elementus. Taip programuotojas sutaupo laiko ir gali būti užtikrintas dėl teisingo XML dokumentų nuskaitymo.

Įvertinus visus aspektus, XML aprašomoji kalba buvo pasirinkta AST duomenų perdavimui. Dokumentų nuskaitymo įrankis nuskaito dokumentus atitinkančius struktūrą pavaizduotą 1 paveiksle.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <GenericAST:Root>
3      <GenericAST:Project id="1">
4          <OutterScope>2</OutterScope>
5          <Files>
6              <Element>3</Element>
7          </Files>
8      </GenericAST:Project>
9      <GenericAST:GlobalScope id="2"/>
10     <GenericAST:CompilationUnit id="3">
11         <LocationInfo>4</LocationInfo>
12         <Language>Java</Language>
13     </GenericAST:CompilationUnit>
14     <GenericAST:SourceLocation id="4">
15         <StartLine>10</StartLine>
16         <StartColumn>30</StartColumn>
17         <EndLine>20</EndLine>
18         <EndColumn>40</EndColumn>
19     </GenericAST:SourceLocation>
20 </GenericAST:Root>
```

1 pav. Bendrinės AST struktūros vaizdavimas XML faile

Pagrindinis elementas visada turi būti <GenericAST:Root>.

AST struktūrą aprašantis elementas turi būti sudarytas tokia tvarka:

- Elemento pavadinimas sudaromas iš priešdėlio „GenericAST:“ ir klasės, kurioje tas elementas saugomas, vardo.
- Kiekvienas elementas turi turėti individualų numerį – „id“. Šakninio struktūros elemento „id“ turi būti lygus vienetui (1 pav. matome šakninį elementą Project). Visų likusių elementų „id“ gali būti bet koks sveikasis teigimas skaičius.
- Kiekvieno elemento viduje esantys elementų pavadinimai sudaromi pagal kintamojo, kuriam jis skirtas, pavadinimą. Pavadinimas į žymę rašomas didžiąja raide.
- Jei elemento reikšmė yra kitas elementas, tai vietoj reikiamo elemento įrašomas jo „id“.
- Kiekvienas sąrašo elementas turi būti pažymimas <Element> žyme.

2.3 Bendrinio AST metamodelis

AST (angl. Abstract Syntax Tree) – abstrakčios sintaksės medis [10, 12].

Daug programavimo kalbų sudaryta iš daug panašių elementų ir dalies savitų, unikalių specifinei kalbai. Atsižvelgiant į programavimo kalbą ir naudojamą kompiliatorių, sudaromas AST, kuris kintant programavimo kalbai ir kompiliatoriui taip pat kinta. Dėl šios priežasties mums reikia sukurti skirtingus įrankius kiekvienam skirtingam AST. Kadangi skirtingi įrankiai, priklausomai nuo naudojamų algoritmų, pertvarko kodą skirtingai, tai gali apsunkinti kodo skaitomumą. Norėdami atsikratyti šių nepageidaujamų savybių, mes norime sukurti vieną įrankį, kuris galėtų atlikti pertvarkymus su populiariausių programavimo kalbų kodu.

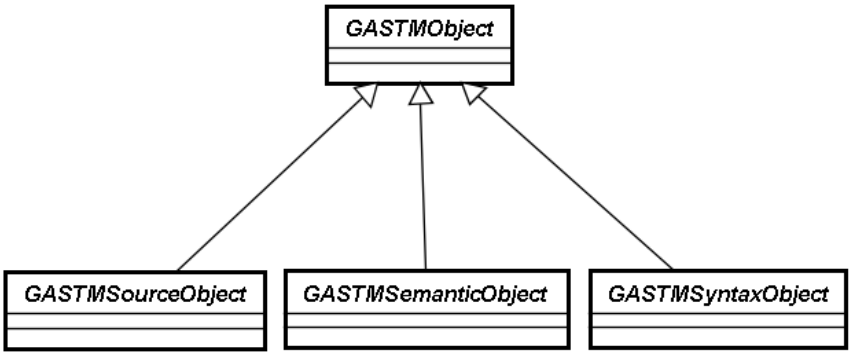
Kad galėtume pradėti įrankio projektavimą, mes privalome turėti AST struktūrą, kuri galėtų talpinti visų numatytų programavimo kalbų elementus [10].

Viena iš pagrindinių įgyvendinimo problemų – kaip sukurti tarpinę struktūrą, į kurią būtų galima pervesti standartinius AST, bei nepadaryti jos per daug didelės. Norint, kad minėta bazinė struktūra galėtų veikti, pakaktų sujungti daugumą AST į vieną didelį katalogą. Šiuo atveju struktūra taptų labai griozdiška ir tai atitiktų variantą, kai kiekvienam medžiui sukuriame po atskirą algoritmą. Todėl reikia atrinkti ir sujungti bendras struktūras visuose skirtinguose AST bei sugalvoti, kaip prijungti skirtingas dalis.

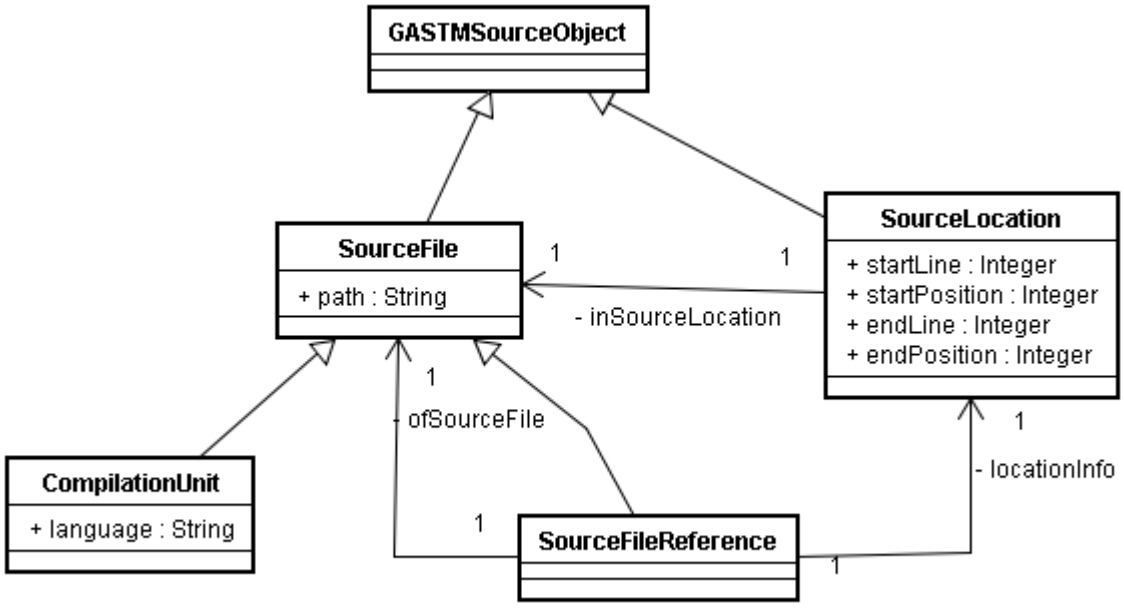
Antra problema – kaip padaryti bazinę struktūrą ne tik universalią, bet ir lengvai pritaikomą naujovėms, bei programavimo kalbų pokyčiams.

OMG sudaryta GASTM specifikacija [4] sukurta panašiu tikslu, todėl šiai užduočiai atlikti ji ir buvo pasirinkta. OMG sudarytoje ASTM specifikacijoje numatyta, kad GASTM, toliau vadinamas bendriniu AST, palaiko C, C++, Java, SQL, .NET ir kitas kalbas.

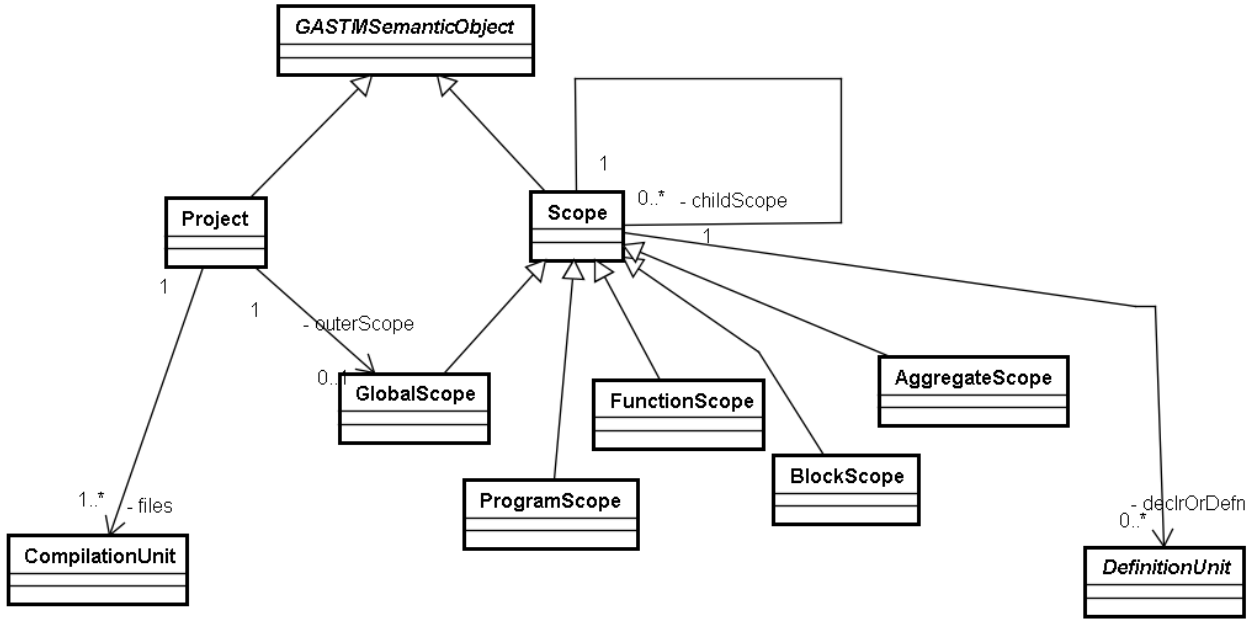
Bendrinė AST struktūra sudaryta iš beveik 200 elementų. Bendrinio AST pagrindas GASTMObject abstrakti klasė, kuri paveldima GASTMSourceObject, GASTMSemanticObject, GASTMSyntaxObject abstrakčių klasių. Visi kirti bendrinio AST elementai paveldi iš šių pagrindinių elementų, nusakančių objektų vietą, abstrakčią sintaksę ir pagrindinius semantinius ryšius. Pagrindinių elementų klasių diagramos pavaizduotos 1, 2, 3 ir 4 paveiksluose.



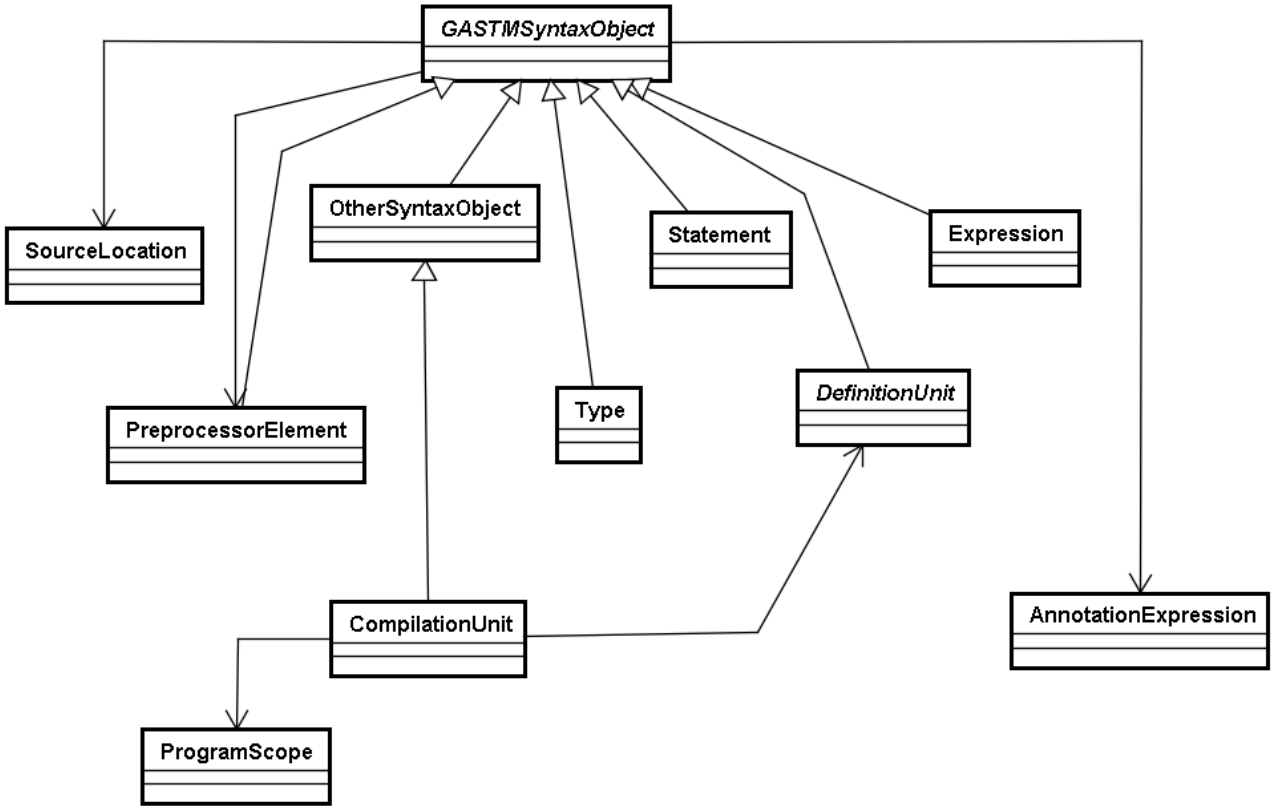
2 pav. Bendrinis duomenų modelis



3 pav. GASTMSourceObject objekto schema



4 pav. GASTMSemanticObject objekto schema



5 pav. GASTMSyntaxObject objekto schema

2.4 Pertvarkymo įrankių analizė

ReSharper – tai Microsoft Visusal Studio įskiepis, skirtas darbui su: C#, VB.NET, ASP.NET, XML, XAML[3]. Šis įrankis atlieka ar padeda atlikti daug įvairių operacijų, skirtų palengvinti programų kūrimo procesą. Įrankio suteikiamos priemonės:

- Kodo analizė. Analizuojamas rašomos programos kodas, nekompiliuojant kodo surandamos sintaksės bei kitos klaidos. Klaidos pažymimos raudona banguota linija, užvedus pelės žymeklį ant klaidingo kodo, parodomas klaidos aprašas. Rašant kodo eilutę, pasiūlo užbaigimo variantus. Automatiškai įterpia skliaustelius. Suranda nenaudojamus kodo fragmentus ir t.t.
- Navigacija ir paieška kode. Galima nesunkiai rasti reikiamas deklaracijas. Galima nesunkiai pereiti į paveldėjusias klases bei tėvo klasę. Galima atlikti paiešką kode, visuose projekto dokumentuose ir t.t.
- Kodo pertvarkymai. Leidžia atlikti numatytą rinkinį kodo pertvarkymo operacijų. Pertvarkymų vedlys gali būti iškviestas pažymėjus kodą ir pasirinkus ReSharper įrankio pertvarkymų meniu arba mygtukų kombinacija – Ctrl+Shift+R. Įrankis atlieka šias pertvarkymo operacijas:
  - perverčia abstrakčią klasę į sąsają(angl. Interface) ir atvirkščiai – pirmas atvejis pravartus, kai norima, kad viena klasė paveldėtų iš keleto kitų abstrakčių klasių.
  - iškelia metodą iš klasės,
  - pervadinti kintamuosius, metodus, klases ir t.t.
  - perkelia kintamuosius, metodus, klases ir t.t.
  - inkapsuliuoja(angl. encapsulate) kintamuosius -
- Kodo generavimas. Leidžia generuoti įvairias rašomo kodo dalis:
  - konstruktorius
  - kintamųjų priėjimo metodus
  - objektų palyginimo funkcijas
- Kodo valymas. Ši funkcija leidžia pašalinti perteklinį kodą, formatuoti kodą. Leidžia pašalinti nereikalingus tarpus, tuščias eilutes ir t.t.. Šios pertvarkymo operacijos leidžia vartotojui nesunkiai padaryti programos kodą lengviau skaitomą.

- Kodo šablonai. Ši funkcija leidžia vartotojui nesudėtingai įterpti į kodą dažnai naudojamus kodo fragmentus:

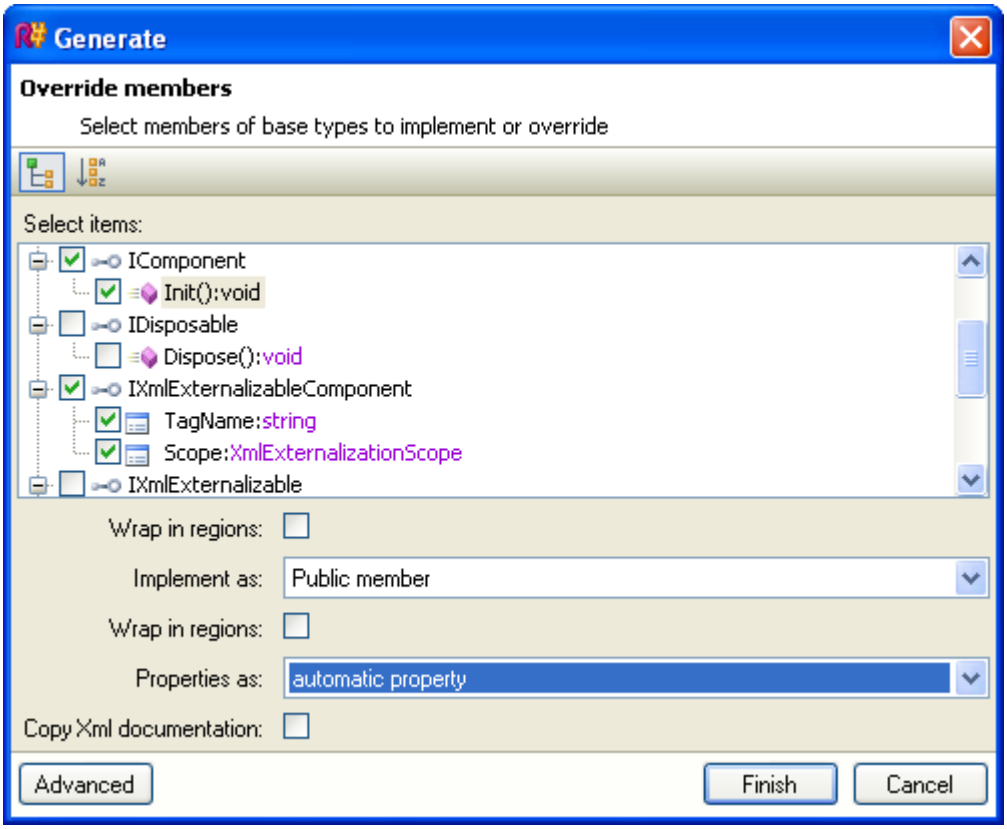
- Ciklai
- Aprašai
- Masyvai
- Ir kt.

Šios pertvarkymo operacijos padeda programuotojui paspartinti programos kodo rašymą.

- Vienetų testavimas. ReSharper automatiškai aptinka NUnit ir MSTest karkasų vienetų testus.

Tai leidžia MS Visual Studio aplinkoje leisti ir stebėti vienetų testavimo eigą.

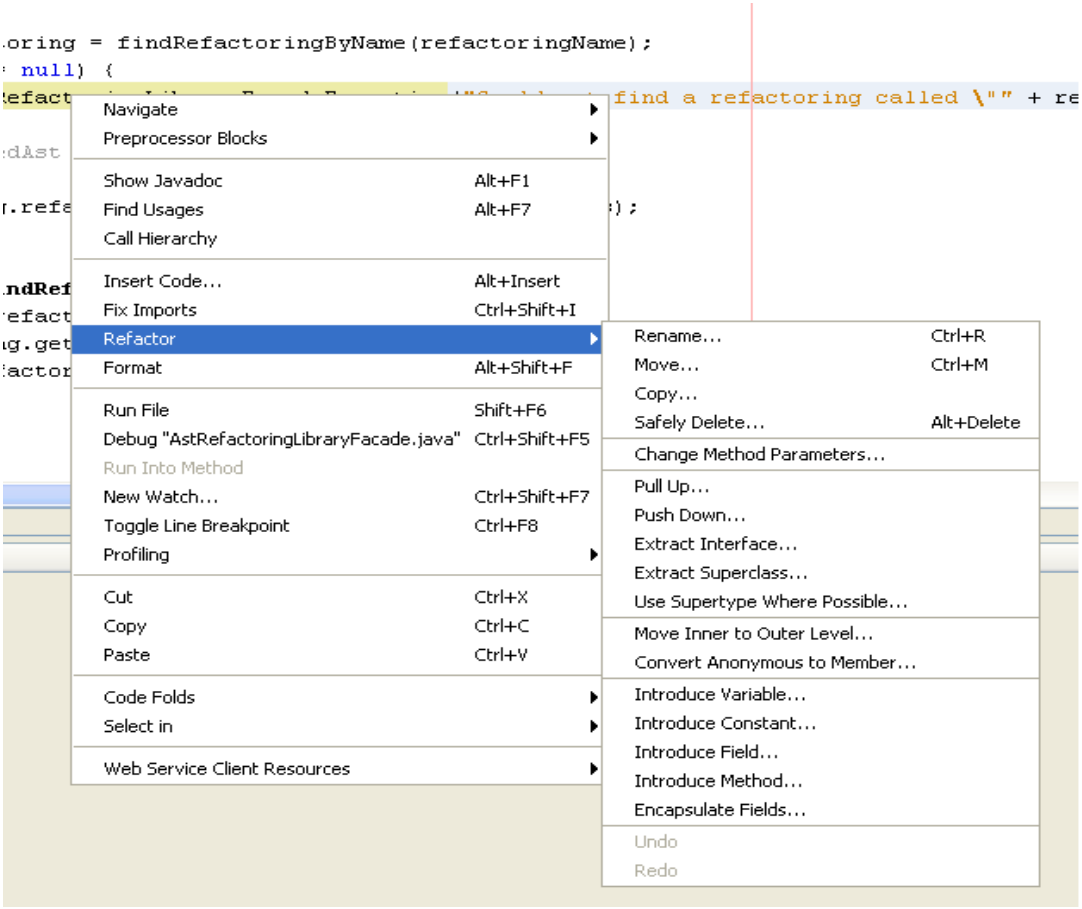
Visos šiuolaikinės programų kūrimo aplinkos turi integruotus kodo pertvarkymo įrankius, tačiau dažniausiai jie turi mažesnį pertvarkymo operacijų pasirinkimą. Sudėtingus pertvarkymus atliekantys įrankiai (įskiepiai) dažniausiai yra mokami.



6 pav. ReSharper programos langas, skirtas paveldėtų metodų realizacijos perrašymui

Netbeans programų kūrimo aplinkos pertvarkymo įrankiai [11] atlieka šiuos pertvarkymus (7 pav.):

- Pervadinimas. Netbeans kodo pertvarkymo įrankiu galima pervadinti: paketus, klases, metodus, kintamuosius nesukeliant klaidų programos kode. Pastaba: metodo pervadinimas negalimas, jei norimo keisti metodo aprašas yra naudojamas tėvo arba vaiko klasėse.
- Laukų inkapsuliavimas. Klasėje aprašytiems kintamiesiems sugeneruojami „set“ ir „get“ metodai.
- Kitos pertvarkymo operacijos: klasės perkėlimas į kitą paketą, saugus elementų trynimas, metodų parametrų keitimas.



7 pav. NetBeans galimų atlikti pertvarkymų sąrašas.

Programų kūrimo aplinkos **Eclipse** atliekamos pertvarkymo operacijos [14]:

- Vardų ir fizinio kodo išdėstymo keitimas – atliekami veiksmai: laukų, kintamųjų, klasių, sąsajų vardų keitimas bei saugūs paketų ir klasių perkėlimai.

- Loginio kodo išdėstymo keitimas klasės lygyje – atliekamos pertvarkymo operacijos: anoniminių klasių vertimas į lizdines klases, sąsajų kūrimas pagal konkrečią klasę, metodų perkėlimas į tėvo ar vaiko klasę.
- Kodo keitimas klasės ribose – atliekamos operacijos: lokalių kintamųjų vertimas į klasės laiką (angl. Class Field), nuskaitymo (angl. getter) ir priskyrimo (angl. setter) metodų klasės laukams generavimas.

Visi trys įrankiai turi skirtingą pertvarkymo operacijų kiekį. **ReSharper** yra geriausias iš trijų apžvelgtų įrankių, tačiau jis vienintelis yra mokamas. **NetBeans** ir **Eclipse** aplinkose veikiantys įrankiai turi panašų kiekį atliekamų pertvarkymo operacijų.

Visuose įrankiuose buvo bendrų pertvarkymo operacijų:

- Pervadinimo operacijos: klasių, metodų, kintamųjų, parametrų pervadinimai.
- Perkėlimo operacijos.: klasių, paketų saugūs perkėlimai.
- Nuskaitymo ir priskyrimo metodų klasės laukams generavimas.

Šios pertvarkymo operacijos taip pat yra dažniausiai naudojamos. Todėl buvo pasirinkta dalį šių operacijų realizuoti kuriamoje pertvarkymų bibliotekoje. Realizuotos šios pertvarkymų operacijos: klasių, klasių laukų, lokalių kintamųjų ir metodų parametrų pervadinimas bei nuskaitymo ir priskyrimo metodų klasės laukams generavimas.

2.5 Analizės išvados

Atlikus technologijų ir egzistuojančių pertvarkymo įrankių analizę buvo atlikti tokie pasirinkimai:

1. Bendrinių AST perdavimui buvo pasirinkta XML aprašomoji kalba, nes ji puikiai tinka tokiai užduočiai, bei egzistuoja įrankiai Java programavimo kalbai, leidžiantys palengvinti darbą su XML dokumentais.

2. Kaip bendrinio AST struktūra buvo pasirinktas OMG sudaryta bendrinis abstraktaus sintaksės medžio metamodelis (GASTM), nes jo kūrimo tikslai ir specifikacija sutapo su mūsų reikalavimais šiai struktūrai bei nebuvo rasta geresnių sprendimų.

3. Nuspręsta pradžioje ištirti paprastų pertvarkymo operacijų bendriniame AST greitaveiką, kad būtų galima daryti išvadas dėl tolesnių tyrimų perspektyvumo.

4. Išanalizuoti trys pertvarkymo įrankiai, vienas iš jų – ReSharper – galintis atlikti sudėtingus kodo pertvarkymus. NetBeans ir Eclipse aplinkose pateikiami įrankiai atlieka paprastesnius pertvarkymus. Remiantis šia analize, pasirinktas šiame darbe tiriamų pertvarkymų operacijų rinkinys. Pasirinkti dažniausiai naudojamos pertvarkymai.

3. SUDARYTI PERTVARKYMO ALGORITMAI

Šiame skyriuje aprašomi sukurti pertvarkymo algoritmai, kurių pagrindu bus kuriama pertvarkymų biblioteka. Šiame darbe buvo sukurti pagrindiniai pertvarkymo algoritmai, kadangi sudėtingesnių algoritmų kūrimas gali būti atliekamas tik ištyrus, ar verta juos kurti.

3.1 Metodo parametro pervadinimas

Šis algoritmas atlieka metodo parametro pervadinimą

Parametrai:

- *root – bendrinis AST..*
- *klase – klases vardas.*
- *metodas – metodo vardas.*
- *parametrai – metodo parametrai.*
- *parametras – keičiamo metodo parametro pavadinimas.*
- *naujas_parametras – naujas vardas keičiamam parametrui.*

```
algoritmas(root, klase, metodas, parametrai, parametras,
naujas_parametras)
{
     klase = rastiKlase(root, klase);
     elementas = rastiMetoda(klase, metodas, parametrai);
     parametras = rastiParametra(elementas, parametras);
     KeistiPavadinima(parametras, parametras, naujas_parametras);
     KeistiLaukusSrityje(pradinsVardas, naujasVardas);
     If(jei reikia)
          keistiMetodoDeklaracija(klase, metodas,
          parametrai,parametras, naujas_parametras);
}
```

Surandamas ieškomos klases elementas. Pateikiamas bendrinis AST arba jo dalis(root), bei klasės vardas(klase).

```
rastiKlase(root, klase)
{
     fragments = gauti_elementu_kieki(root);
     for(int i = 0; i < fragments.kiekis; i++)
     {
```

```
            if(fragment[i] yra klasės aprašymo elementas)
                  if(fragments[i] yra reikiamas elementas)
                        return fragments[i];
      }
      return null;
}
```

Surandamas ieškomas metodo elementas. Pateikiamas klasės srities elementas(root), metodo vardas(metodas) ir jo parametrai.

```
rastiMetoda(root, metodas, parametrai)
{
      klases_nariai = gautiKlasesNarius(klase);
      for(int i = 0; i < klases_nariai.kiekis; i++)
      {
            if(klases_nariai[i] yra metodo aprašymo elementas)
                  if(klases_nariai[i] yra reikiamas elementas)
                        return klases_nariai[i];
      }
      return null;
}
```

Surandamas ieškomas metodo parametro elementas. Pateikiamas metodo parametrų srities elementas(root) ir jo parametro vardas..

```
rastiParametra(root, parametras)
{
      parametrai = gautiParametrus(elementas);
      for(int i = 0; i < parametrai.kiekis; i++)
      {
            if(parametrai[i] yra reikiamas elementas)
                  return parametrai[i];
      }

}
```

3.2 Klasės kintamųjų pervadinimas

Klasės kintamasis – tai kintamasis aprašytas klasėje.

Jei kintamasis privatus, tai keitimas atliekamas tik toje klasėje. Jei viešas – visame bendriniame AST.

```
algoritmas(root, klase, kintamasis, naujas_kintamasis)
{
     klase = rastiKlase(root, klase);
     kint = rastiKintamaji(klase, kintamasis);
     if(kintamasis.tipas yra privatus)
          keistiLaukusSrityje(klase, kint, naujas_kintamasis);
     else
          keistiLaukusSrityje(root, kint, naujas_kintamasis);
}
```

3.3 Lokalaus kintamojo pervadinimas

Lokalus kintamasis – tai kintamasis aprašytas metode.

```
algoritmas(root, klase, metodas, parametrai, kintamasis,
naujas_kintamasis)
{
     klase = rastiKlase(root, klase);
     elementas = rastiMetoda(klase, metodas, parametrai);
     kintamasis = rastiKintamaji(elementas, kintamasis);
     keistiLaukusSrityje(elementas, kintamasis, naujas_kintamasis);

}
```

3.4 Klasės pavadinimo keitimas

Pakeičiamas norimos klasės pavadinimas.

```
algoritmas(root, klase, naujas_klase)
{
     klase = rastiKlase(root, klase);
     keistiLaukusSrityje(root, klase, naujas_klase);

}
```

3.5 Klasės kintamojo inkapsuliacija

Klasės kintamasis – tai kintamasis aprašytas klasėje.

Sugeneruojami „set“ ir „get“ metodai pagal nurodytą klasės kintamąjį.
Parametrai set ir get yra boolean tipo.

```
algoritmas(root, kintamasis, klase, set, get)
{
     elementas = rastiKlase(root, klase);
     kint = rastiKintamaji(elementas, kintamasis);
     if(set){

          if(set metodas neegzistoja){
               elementas.add(generuoti_set_metoda(kint));
               padaryti_privatu(kint);
               If(jei reikia)
                    elementas.add(generuoti_set_metodo_dekl(kint));
          }
     }

     if(get)
     {
          if(get metodas neegzistoja)
          {
               padaryti_privatu(kint);
               elementas.add(generuoti_get_metoda(kint));
               If(jei reikia)
                    elementas.add(generuoti_set_metodo_dekl(kint));
          }
     }
}
```

Sugeneruojamos „set“ ir „get“ metodų struktūros, pridedami tik tipai ir kintamieji kur jų reikia. Deklaracijos sugeneruojamos tik toms kalboms kurioms jų reikia.

Pagal šį algoritmą, generavimo ir metodų įterpimo laikas visada bus vienodas. Laikas keisis, priklausomai nuo metodų kiekio klasėje, tik atliekant kintamojo ir metodų paiešką.

4. PERTVARKYMO BIBLIOTEKOS PROJEKTAS

4.1 Reikalavimų specifikacija

4.1.1 Bibliotekos vartotojai:

Paprastas vartotojas:

Sistemos vartotojas: Programuotojas

- Vartotojo kategorija: inžinierius.
- Vartotojo sprendžiami uždaviniai: Iš AST struktūrų transformavimo komponento paduodama bendroji AST struktūra. Vartotojas nustato, kokius pakeitimus atlikti šiai struktūrai bei atlieka pakeitimus. AST struktūra perduodama atgal į transformavimo komponentą.
- Patirtis dalykinėje srityje: gali būti tiek naujokas, tiek specialistas.
- Patirtis informacinėse technologijose: gali būti tiek patyręs vartotojas, tiek informatikas.
- Papildomos vartotojo charakteristikos – nėra.
- Vartotojų prioritetai: įprasti vartotojai.

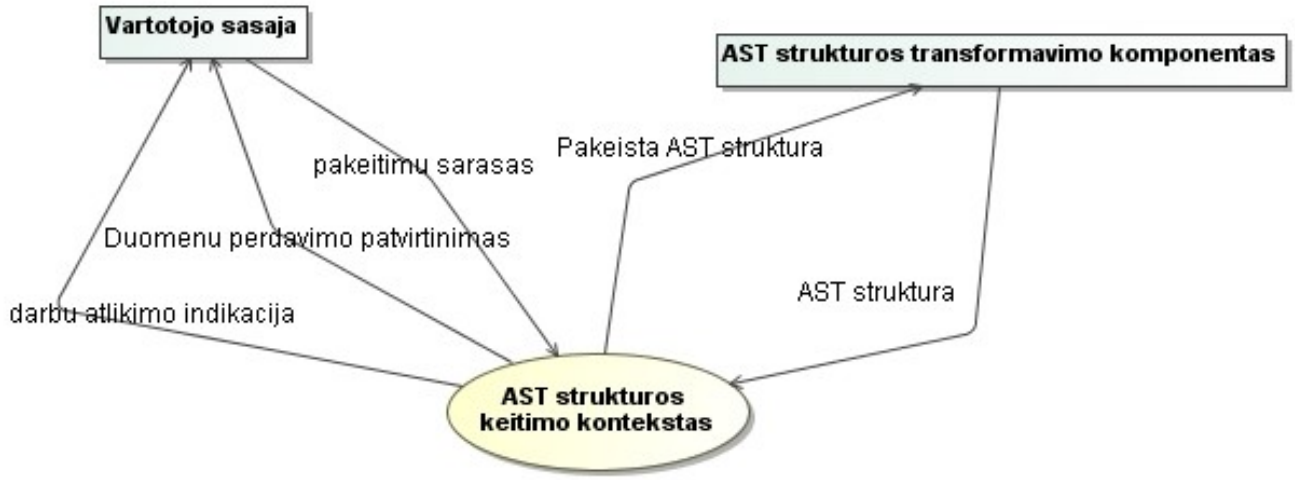
Prižiūrėtojas, tobulintojas:

Sistemos vartotojas: Programuotojas

- Vartotojo kategorija: inžinierius.
- Vartotojo sprendžiami uždaviniai: Kurti pertvarkymo algoritmus ir įterpti juos į sistemos konstrukciją.
- Patirtis dalykinėje srityje: srities specialistas.
- Patirtis informacinėse technologijose: informatikas.
- Papildomos vartotojo charakteristikos – nėra.
- Vartotojų prioritetai: svarbūs vartotojai.

4.1.2 Veiklos sfera

4.1.2.1 Veiklos kontekstas



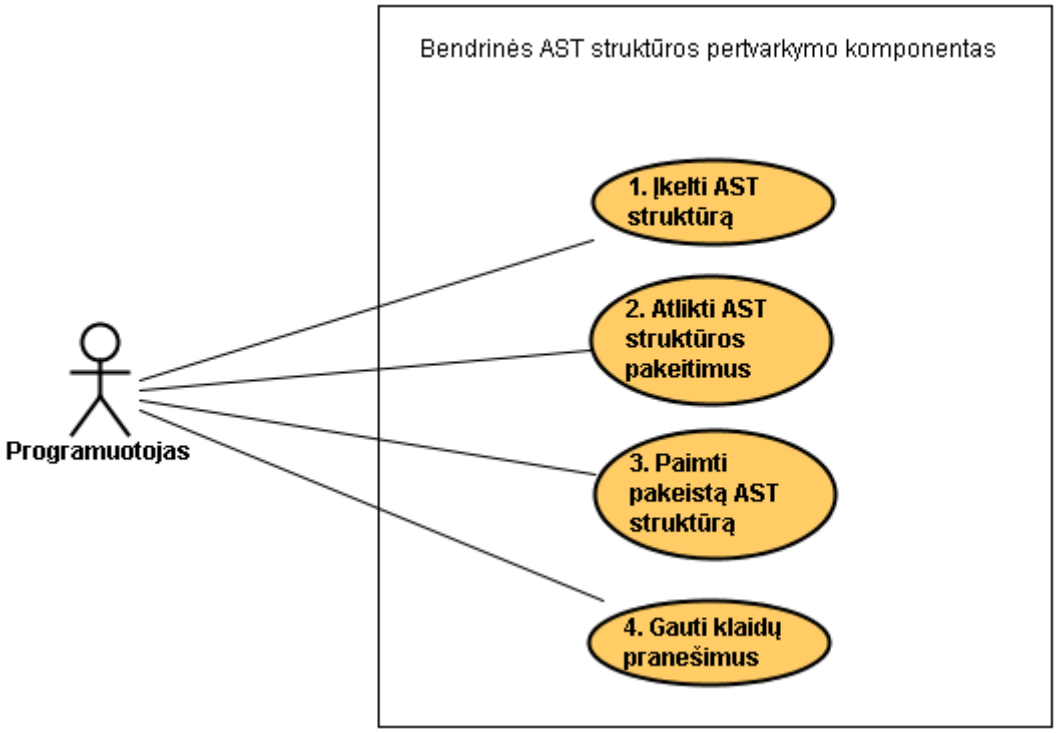
8 pav. Veiklos konteksto diagrama.

4.1.2.2 Veiklos įvykiai ir duomenų srautai

1. *Lentelė. Veiklos įvykių sąrašas*

Eil. Nr.	Įvykio pavadinimas	Įeinantys/Išeinantys informacijos srautai
1	Vartotojas įkelia AST struktūrą	AST struktūra(in)
2	Vartotojas atlieka pakeitimus AST struktūrai	Pakeitimu sąrašas(in) Darbų atlikimo indikacija(out)
3	Vartotojas pasiima AST struktūrą	Pakeista AST struktūra(out)
4	Vartotojas stebi duomenų perdavimą	AST struktūra(in) Duomenų perdavimo patvirtinimas(out)

4.1.3 Panaudojimo atvejų diagrama



9 pav. Panaudos atvejų diagrama

4.1.4 Panaudojimo atvejų sąrašas

1. PANAUDOJIMO ATVEJIS: Įkelti AST struktūrą **Vartotojas/Aktorius:** Programuotojas **Aprašas:** Apima procesą, kurio metu vartotojas įkelia savo kodo versiją AST struktūros pavidalu. **Prieš sąlyga:** Komponento atmintinė laisva. **Sužadinimo sąlyga:** Vartotojas nori įkelti AST struktūrą. **Po-sąlyga:** AST struktūra įkelta į komponento atmintį.

2. PANAUDOJIMO ATVEJIS: Atlikti AST struktūros pakeitimus **Vartotojas/Aktorius:** Programuotojas

Aprašas: Apima procesą, kurio metu vartotojas atlieka pakeitimus AST struktūroje pagal pasirinktus kriterijus.
Prieš sąlyga: AST struktūra išsaugota komponento atmintinėje.
Sužadinimo sąlyga: Vartotojas pasirenka AST struktūros pakeitimus.
Po-sąlyga: AST struktūra pakeista.

3. PANAUDOJIMO ATVEJIS: Paimti pakeistą AST struktūrą
Vartotojas/Aktorius: Programuotojas
Aprašas: Apima procesą, kurio metu paima AST struktūrą iš komponento.
Prieš sąlyga: AST struktūra yra komponento atmintinėje.
Sužadinimo sąlyga: Vartotojas nori pamatyti AST keitimo rezultatus.
Po-sąlyga: AST struktūra iškelta iš komponento ir jo atmintinė laisva.

4. PANAUDOJIMO ATVEJIS: Gauti klaidų pranešimus
Vartotojas/Aktorius: Programuotojas
Aprašas: Apima procesą, kurio metu komponentas praneša apie AST struktūros perdavimo klaidą.
Prieš sąlyga: AST paruošta perdavimui.
Sužadinimo sąlyga: AST struktūra perduodama į komponentą.
Po-sąlyga: Perduotas pranešimas vartotojui ir komponento atmintis laisva.

4.1.1 Funkciniai reikalavimai ir reikalavimai duomenims

4.1.1.1 Funkciniai reikalavimai

Reikalavimas#: 1 Reikalavimo tipas: 9 Įvykis/panaudojimo atvejis#: 1

Aprašymas: Komponentas turi įsiminti jam paduodamą AST struktūrą.

Pagrindimas: Po transformacijos AST struktūra turi būti išsaugoma komponente, kad būtų galima atlikti pakeitimus paeiliui.

Šaltinis: Užsakovas
Tikimo kriterijus: Visa AST struktūra išsaugoma komponente ir apie tai informuojamas vartotojas.

Užsakovo tenkinimas: 5 Užsakovo netenkinimas: 1
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 2 Reikalavimo tipas: 9 Įvykis/panaudojimo atvejis#: 2

Aprašymas: Komponentas leidžia atlikti iš anksto numatytus pertvarkymus AST struktūroje.

Pagrindimas: AST struktūrą, perteikiančia programos kodą, galima papildyti numatytomis kodo eilutėmis arba atlikti kodo optimizavimą.

Šaltinis: Užsakovas
Tikimo kriterijus: Komponentas atlieka užduotus pakeitimus AST struktūroje ir apie tai informuoja vartotoją.

Užsakovo tenkinimas: 5 Užsakovo netenkinimas: 4
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 3 Reikalavimo tipas: 9 Įvykis/panaudojimo atvejis#: 3

Aprašymas: Komponentas leidžia iš atminties perduoti AST struktūrą į transformacijos komponentą

Pagrindimas: Pabaigus visus AST struktūros pertvarkymus, turi būti galimybė išsiimti pakeistą struktūrą, kad būtų galima naudotis rezultatu.

Šaltinis: Užsakovas
Tikimo kriterijus: Komponentas perduoda visą išsaugotą informaciją apie AST struktūrą laikomą atmintyje.

Užsakovo tenkinimas: 5 Užsakovo netenkinimas: 1
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 4 Reikalavimo tipas: 9 Įvykis/panaudojimo atvejis#: 4

Aprašymas: Komponentas įspėja, jei AST struktūros perdavimo metu įvyko klaida

Pagrindimas: Dėl sistemos sutrikimų ar išorinių trikdžių gali persiduoti ne visą struktūrą.

Šaltinis: Užsakovas
Tikimo kriterijus: Komponentas perduoda pranešimą vartotojui, jeigu AST struktūroje pastebėti netikslumai.

Užsakovo tenkinimas: 3 Užsakovo netenkinimas: 3
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

4.1.1.2 Reikalavimai duomenims

AST struktūros modelio schema GASTMObject yra modelio pagrindas, jis susideda iš trijų abstrakčių klasių, kurios skirtos programos kodo abstrakčios sintaksės, pagrindinių semantikos ryšių ir objektų vietos saugojimui.

GASTMSourceObject skirtas objektų vietai, GASTMSemanticObject – pagrindiniams semantikos ryšiams, bei GASTMSyntaxObject – abstrakčiai sintaksei.

Plačiau apie duomenų struktūrą galima paskaityti skyriuje 2.3 Bendrinio AST metamodelis. Ten taip pat pateiktos pagrindinių struktūrą sudarančių klasių diagramos.

4.1.2 Nefunkciniai reikalavimai

4.1.2.1 Reikalavimai sistemos išvaizdai

Vartotojo sąsaja privalo būti kuo paprastesnė ir suteikianti visas numatytas valdymo galimybes vartotojui. Sąsaja privalo būti intuityvi ir lengvai skaitoma, nes sistema skirta tiek patyrusiam paprastam vartotojui, tiek informatikos specialistui.

4.1.2.2 Reikalavimai panaudojamumui

Komponentas apdoros AST struktūras, tai yra programos kodo išraiškas. AST struktūras sudarantys duomenys naudoja anglų kalbos raidyną. Indikaciniai pranešimai vartotojo sąsajai bus lietuvių kalba. Kalbą bus galima pakeisti.

4.1.2.3 Reikalavimai vykdymo charakteristikoms

Duomenų apdorojimo greitis priklausys nuo paduotos AST struktūros dydžio ir nuo struktūros pakeitimo darbų sudėtingumo. Kadangi AST struktūrai atlikti pakeitimą galima keletu būdų, todėl nesvarbu kuriuo būdu bus realizuotas pakeitimas, svarbu, kad galutinis rezultatas turėtų norimą reikšmę. Sudėtingi skaičiavimai nėra atliekami, todėl sistemai nereikia skirti daug resursų kuriamam komponentui.

4.1.2.4 Reikalavimai veikimo sąlygoms

Komponentas turi atlikti skaičiavimus sistemai visada, išskyrus tuomet, kai sutrinka kompiuterio, kuriame yra komponentas, veikimas arba dėl išorinių ar vidinių priežasčių komponentas negali susisiekti su kitais sistemos komponentais.

4.1.2.5 Reikalavimui sistemos priežiūrai

Kuriamas produktas galės veikti tiek Windows, tiek Linux sistemos, jei jose bus atitinkama programinė įranga. Jeigu keisis reikalavimai produktui po to, kai jis bus sukurtas, tai pakeitimo sunkumas priklausys nuo to, ką norima keisti. Pakeisti vartotojo sąsają nebus sudėtinga. Kadangi produktas dirba su iš anksto numatyta AST struktūra, tai pasikeitimai struktūros bazinei formai nenumatomi. Komponentas turi būti sukurtas taip, kad prireikus atlikti pakeitimus AST struktūros bazinės formos apraše, nereikėtų perrašyti viso komponento. Norint pridėti daugiau AST struktūros

keitimo funkcijų, pakaks tik papildyti komponentą, nekeičiant anksčiau sukurto kodo, išskyrus sąsajos aprašą.

Reikalavimas#: 5 Įvykis/panaudojimo atvejis#: 1 – 4

Aprašymas: Vartotojo sąsaja turi būti paprasta ir intuityvi

Pagrindimas: Neturėtų iškilti poreikis baigti specialius kursus prieš naudojantis sistema.

Šaltinis: Justas Jokubauskas – projekto kūrėjas.
Tiekimo kriterijus: Informatikas gali įsisavinti sąsajos valdymo ypatumus per 1 dieną nuo darbo pradžios.

Užsakovo tenkinimas: 4 Užsakovo netenkinimas: 2
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 6 Įvykis/panaudojimo atvejis#: 1 – 4

Aprašymas: Aiški dokumentacija

Pagrindimas: Vartotojas turėtų sugebėti susirasti reikiamas funkcijas naudodamasis dokumentacija.

Šaltinis: Užsakovas.
Tiekimo kriterijus: Vartotojas gali susirasti reikiamas funkcijas ir išmokti jomis naudotis be pagalbos.

Užsakovo tenkinimas: 3 Užsakovo netenkinimas: 2
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 7 Įvykis/panaudojimo atvejis#: 1 – 4

Aprašymas: Išlaikomi gero komponento principai

Pagrindimas: Prireikus keisti programos kodą turi būti lengva suprasti kodo sandarą.

Šaltinis: Užsakovas.
Tikimo kriterijus: Programuotojas gali nesunkiai įsisavinti ir keisti programos kodą.

Užsakovo tenkinimas: 3 Užsakovo netenkinimas: 3
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

Reikalavimas#: 7 vykis/panaudojimo atvejis#: 1 – 4

Aprašymas: Realizuota JAVA kalba.

Pagrindimas: Sistemos prižiūrėtojai yra JAVA kalbos specialistai, todėl jiems turi būti paprasta keisti kodą.

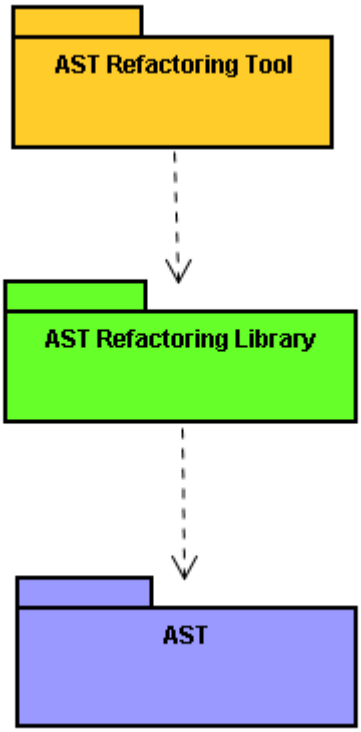
Šaltinis: Užsakovas.
Tikimo kriterijus: Kodas turi buti parašytas JAVA kalba.

Užsakovo tenkinimas: 5 Užsakovo netenkinimas: 0
Priklausomybės: Nėra Konfliktai: Nėra
Papildoma medžiaga: Nėra
Istorija: Užregistruotas 2009 – 03 – 04

4.2 Pertvarkymų bibliotekos architektūra

4.2.1 Sistemos statinis modelis

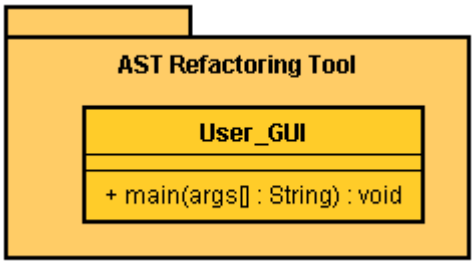
4.2.1.1 Paketų diagrama



10 pav. Bendra paketų diagrama

4.2.1.2 Detalizuotas paketų aprašas

4.2.1.2.1 AST Refactoring Tool

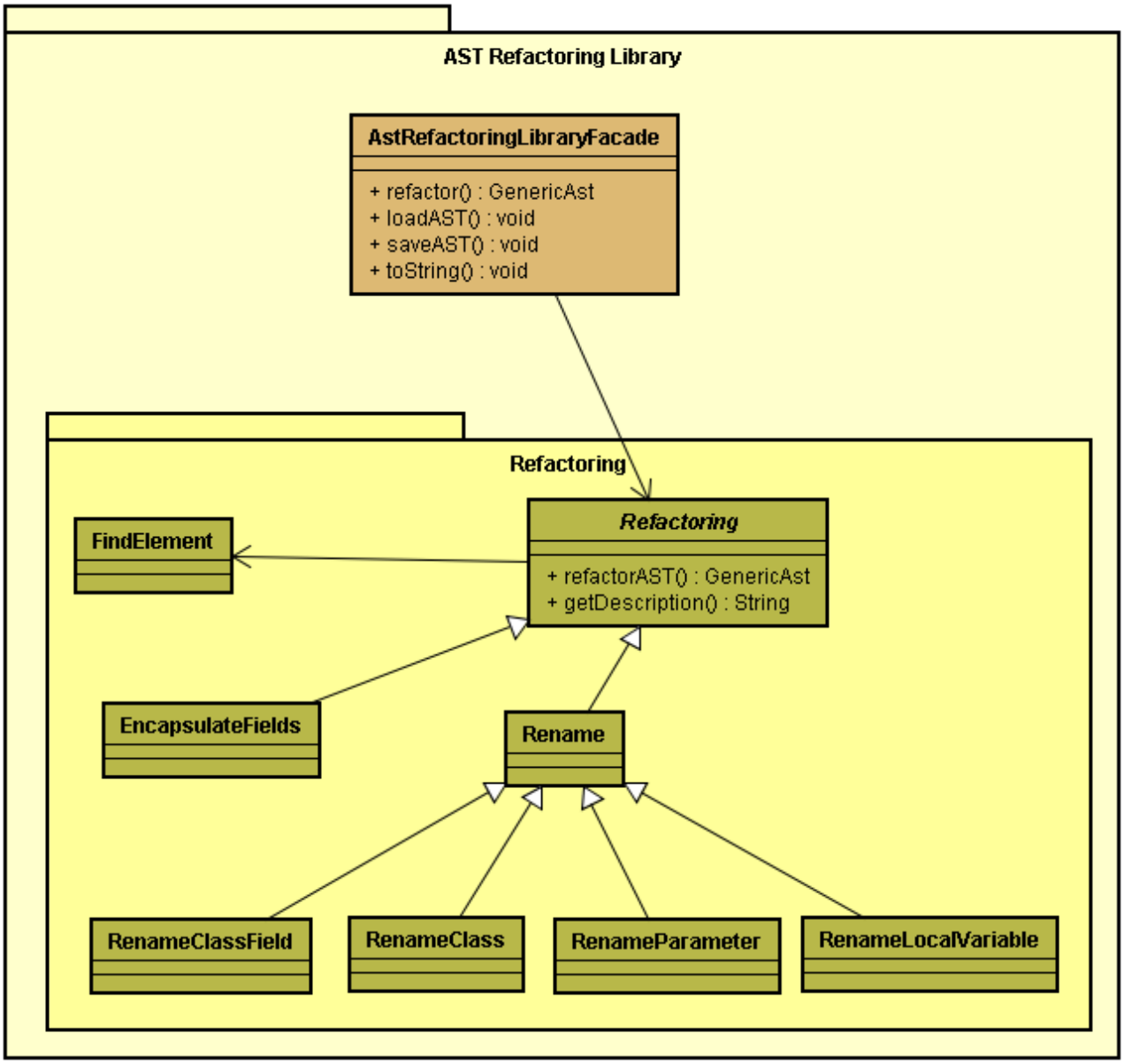


11 pav. AST refactoring tool paketo diagrama

Šį paketą sudaro klasės realizuojančios bendrinio AST pertvarkymo įrankį. Šis įrankis yra vartotojo sąsaja valdanti pertvarkymų biblioteką. Pagrindinė klasė pakete – User_GUI, per ja paleidžiama programa.

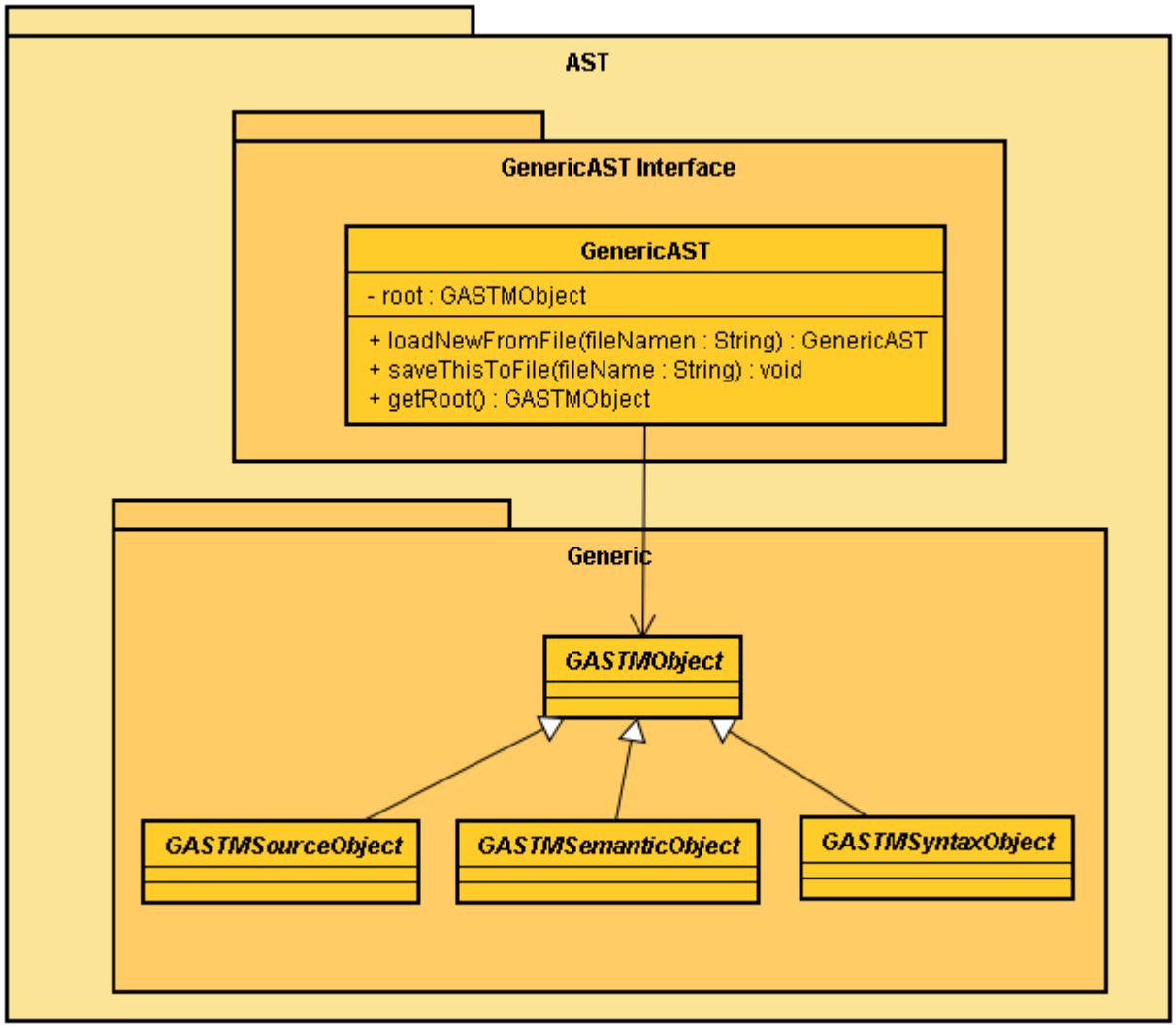
4.2.1.2.2 AST Refactoring Library

AstRefactoringLibraryFacade – klasė, per kurią pasiekiamos visos bibliotekos funkcijos.
Refactoring paketas – šiame pakete laikomos klasės atliekančios bendrinio AST pertvarkymus.



12 pav. AST Refactoring Library paketo diagrama

4.2.1.2.3 AST



13 pav. Paketo AST diagrama

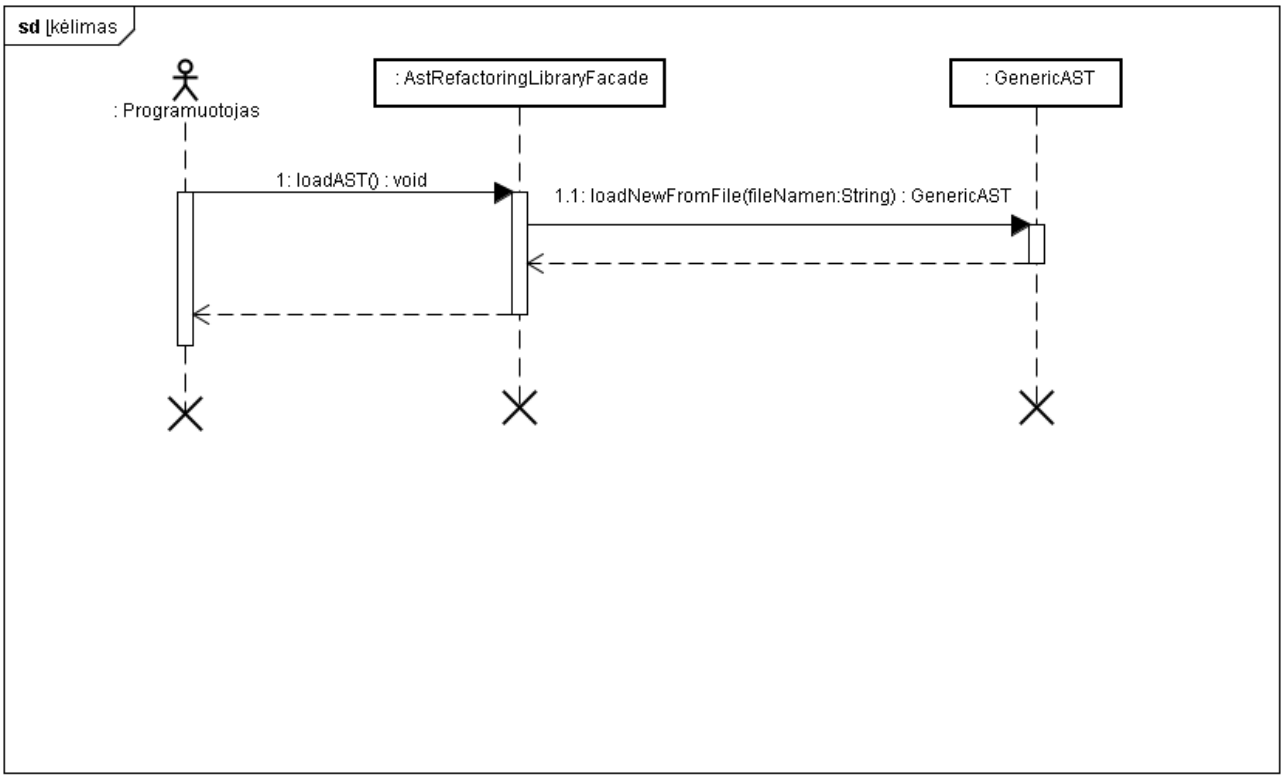
GenericAST Interface paketas sudarytas iš klasių pagalbinių klasių reikalingų darbui su bendrine AST struktūra: nuskaitymas, spausdinimas, struktūros perdavimas ir t.t.

Generic pakete sudėtos klasės reikalingos bendrinio AST saugojimui.

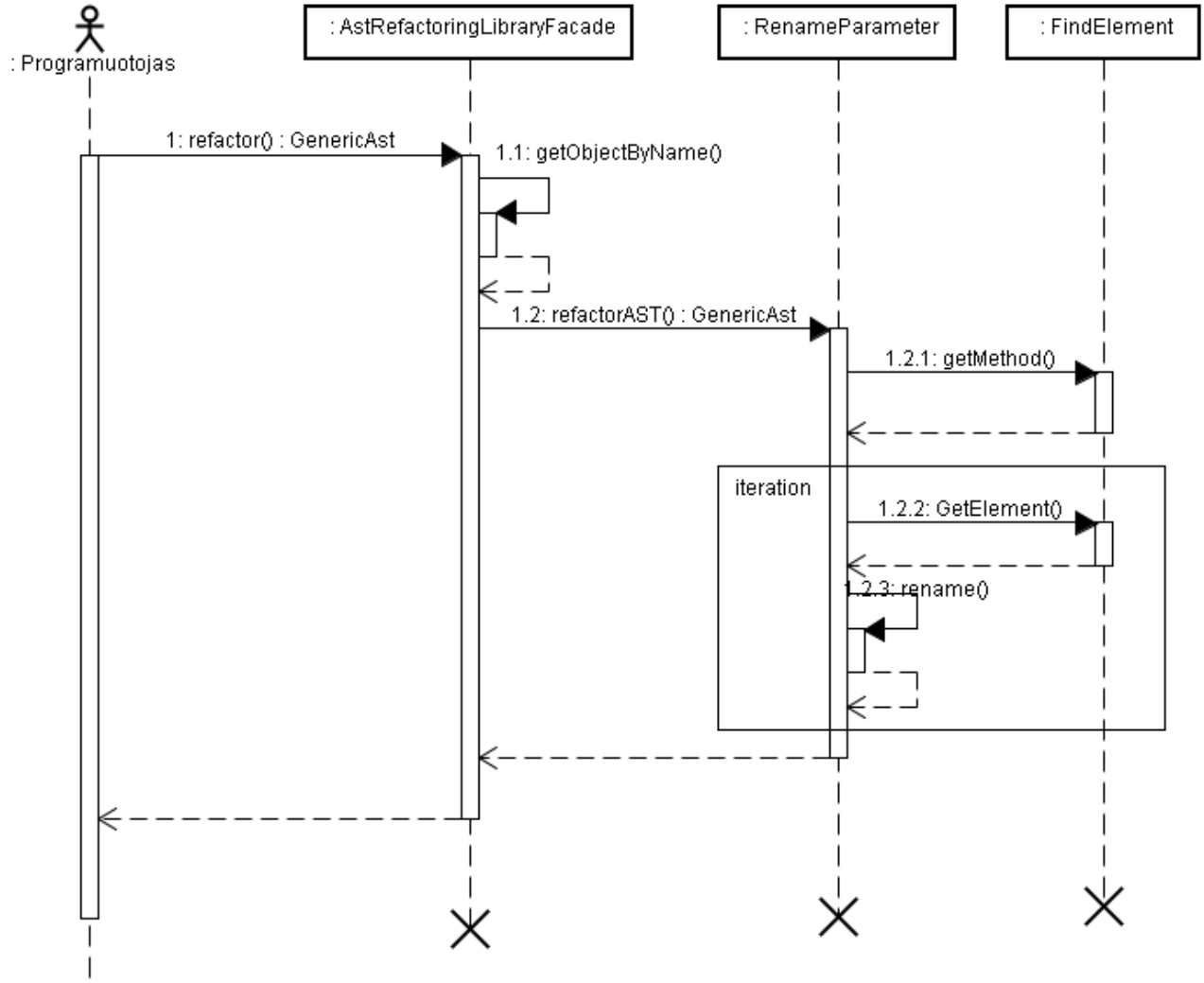
Generic pakete talpinamos visos klasės reikalingos bendrinio AST elementų saugojimui. Plačiau apie jas 2.3 Bendrinio AST metamodelis ir [4] šaltinyje, AST metamodelio specifikacijoje.

4.2.2 Sistemos elgsenos modelis

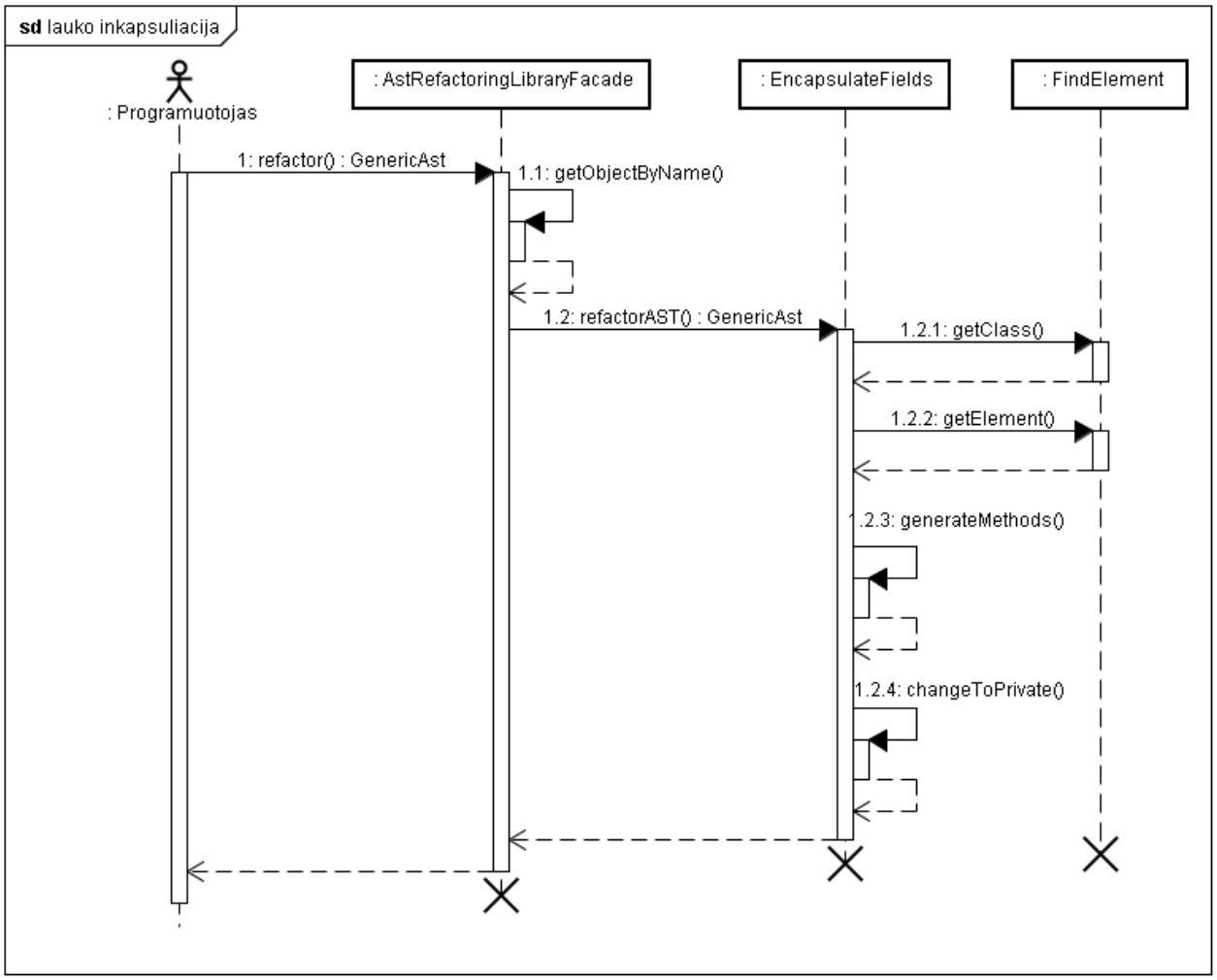
4.2.2.1 Sekų diagramos



14 pav. Sekų diagrama, duomenų įkėlimas

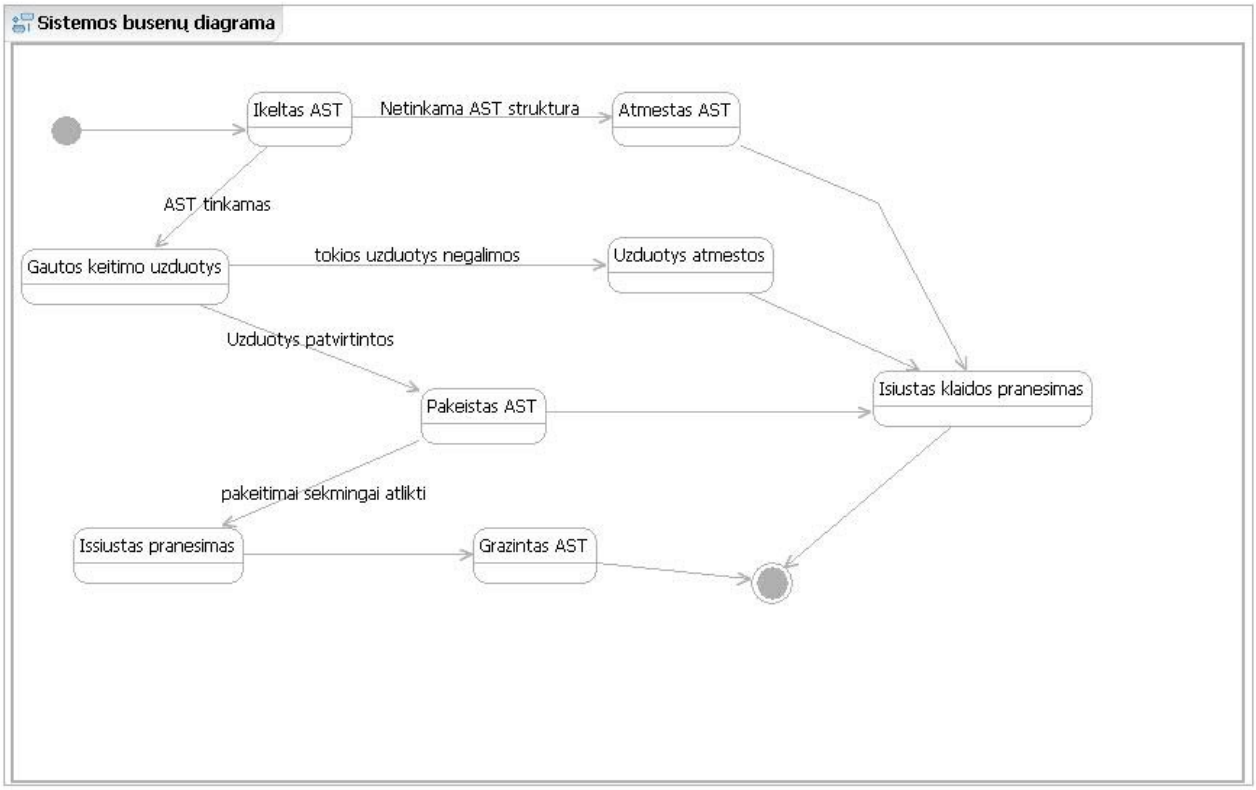


15 pav. Sekų diagrama veiksmui – pervadinti parametrą.



16 pav. Sekų diagrama veiksmui – inkapsuliuoti klasės kintamąjį.

4.2.2.2 Būsenų diagrama



17 pav. Sistemos būsenų diagrama.

4.3 Testavimo planas

4.3.1 Testavimo tikslai ir objektai

Kuriamo produkto testavimo tikslas yra pašalinti visus programinės įrangos defektus bei kitokius trūkumus. Tačiau visų klaidų suradimas gali būti tik teorinis, nes reali naudojama programinė įranga visuomet turės bent nedidelį kiekį klaidų. Produkto testavimo galimybes riboja įvairūs veiksniai: ribotas laikas, biudžetas ir pan. Todėl realų gyvenimą atitinkantis testavimo tikslas - sukurti kiek įmanoma geriau ištestuotą programinę įrangą.

Pagrindinis užsibrėžtas siekis - sukurti stabiliai veikiantį programinės įrangos produktą, turintį kiek įmanoma mažesnį kiekį loginių klaidų. Taip pat svarbu užtikrinti tinkamumą bei patogumą naudotis sukurta įranga.

4.3.2 Testavimo apimtis

AST medžių pertvarkymo programinė įranga bus testuojama naudojantis šiomis testavimo metodikomis:

- Vienetų testavimas – šiame etape bus atliekami vienetų testai. Testavimas vykdomas baltosios dėžės principu.
- Integracinis testavimas – atliekami integraciniai testai. Testavimas vykdomas juodos dėžes principu.
- Priėmimo testavimas – užsakovui pristatomi testavimo rezultatai.
- Aukšto lygio testavimas – bus testuojamos sukurtos bibliotekos. Atliekami apkrovos ir greitaveikos testai.

Testuojama programinė įranga susideda iš dviejų komponentų: bendrinio AST bibliotekos ir bendrinio AST pertvarkymų bibliotekos bei pagalbinių įrankių bibliotekų valdymui.

Vienetų testavimas. Šiame etape visos programinės įrangos dalys testuojamos atskirai. Apjungiant visas programinės įrangos dalis į vieną produktą bus vykdomi integraciniai testai. Kai bus baigti integraciniai testai ir visos dalys sujungtos į vieną sistemą, tuomet bus atliekamas aukšto lygio testavimas

Projekto dokumentacija bus testuojama patikros būdu. Bus tikrinama dokumentų struktūros teisingumas, aprašymų aiškumas, teisingumas, išsamumas ir t.t. Bibliotekų kūrimo aplinkybės yra unikalios tuo, kad dokumentacija bus tikrinama nepriklausomos šalies – Kauno technologijos universiteto dėstytojų.

Bibliotekos kuriamos naudojantis JAVA technologijomis, taigi, bibliotekos veiks visose operacinėse sistemose palaikančiose JAVA technologijas. Dėl šios priežasties testavimas bus atliekamas Windows ir Linux operacinėse sistemose.

4.3.3 Pagrindiniai apribojimai

Kuriamos programinės įrangos testavimą ribojantys veiksmai:

1) Laiko apribojimas – testavimui skiriamas ribotas laiko tarpas. Kadangi programinė įrangos pristatymo data jau numatyta, testavimo veikla negali būti vėluoti, todėl atliekami tik svarbiausi testai.

2) Riboti ištekliai – Testavimo vykdymui prieinamos tik Windows ir Linux operacinės sistemos. Testavimo veiklai lėšos neskiriamos, todėl galimų panaudoti įrankių aibė yra ribota.

4.3.4 Testavimo strategija

4.3.4.1 Vienetų testavimas

Vienetų testavimas vykdomas programinės įrangos kūrėjų. Vienetų testavimas atliekamas remiantis baltos dėžės principu. Kadangi dalis darbo kuriama testais grįsto kūrimo principu, todėl vienetų testų kūrimas ir vykdymas yra natūralus kūrimo procesas. Ypatingai didelis dėmesys buvo skiriamas bendrinės AST struktūros kūrimui, nes ši struktūra yra darbo pagrindas, klaidos joje atsilieptų visoms kitoms programinės įrangos dalims. Testais grįsto kūrimo principai nebus griežtai taikomi visos programinės įrangos kūrimui, nes mums to neleidžia griežti laiko apribojimai.

Programinės įrangos kūrimas vykdomas tokiu principu:

- Parašomas testas kuriamai funkcijai. Parašytas testas turi rodyti klaidą, nes funkcija dar nerealizuota.
- Realizuojam reikalinga funkcija. Paleidžiame testą, kad įsitikintume sukurtos funkcijos teisingumu.
- Vėl rašomas arba papildomas egzistuojantis testas naujoms funkcijoms, realizuojamos funkcijos ir t.t..

Toks kūrimo principas naudojamas tik didelės svarbos komponentas kurti. Mažesnės svarbos komponentų kūrimui testai iš anksto nerašomi. Dėl laiko stokos testai realizuojami jau po funkcijų sukūrimo.

4.3.4.2 Integravimo testavimas

Šis testavimas taip pat vykdomas programinės įrangos kūrėjų. Testavimas vykdomas naudojant juodos dėžės testavimo strategiją. Kaip ir vienetų testavimo metu, didesnis dėmesys skiriamas bendrinio AST bibliotekai. Šio testavimo metu tikimasi atskleisti vienetų testais neaptinkamas klaidas. Po to, kai bus ištestuotas bendrinis AST, bus pradedamos testuoti pertvarkymų bibliotekos sudedamosios dalys. Jei testavimas bus sėkmingas, tai sujungus visus komponentus turi būti įmanoma atlikti numatytus skaičiavimus. Jei abi bibliotekos veiks, bus prijungiami ir ištestuojami bibliotekų valdymo įrankiai. Jeigu viskas bus atlikta teisingai, tai vartotojas jau galės naudotis bibliotekų teikiamomis funkcijomis.

4.3.4.3 Priėmimo testavimas

Testavimo rezultatai pristatomi projekto vadovui ir užsakovui. Reikalui esant testavimas atliekamas vietoje, bei parodomas sukurtos programinės įrangos funkcionalumas. Funkcionalumo pademonstravimui reikalingi iš anksto sugeneruoti duomenų failai, jie paruošiami iš anksto.

Kadangi bibliotekos pateikiami rezultatai yra sunkiai suprantami žmogui, todėl bus naudojami mažos apimties duomenų failai.

4.3.4.4 Aukšto lygio testavimas

Šioje dalyje bus atliekami šie testai: apkrovos, greitaveikos ir patogumo naudotis.

Apkrovos testavimas. Sukurtiems įrankiams pateikiami labai didelės apimties duomenų failai ir stebimas kompiuterio resursų sunaudojimas bei skaičiavimų galimumą su itin dideliu duomenų kieku. Rezultatų teisingumas nebus tikrinamas, testas nepavyks tik tais atvejais, kai dėl atsiradusių sutrikimų bus nenatūraliai nutraukiamas skaičiavimas. Dalis duomenų bus specialiai sugadinti, tai atliekama, kad patikrintume ar teisingai veikia nuskaitomų duomenų patikra.

Greitaveikos teistai. Bus tikrinama kaip duomenų kiekio didėjimas veikia pertvarkymų bibliotekos greitaveiką. Tam bus naudojami įvairių dydžių duomenų failai.

Patogumo naudotis. Projekto vadovui ir užsakovui bus pateikiama vartotojo dokumentacija. Vėliau pateikiama programinė įranga ir įvertinamas sąsajų patogumas. Patogumas naudotis yra subjektyvus dydis, todėl šiuo klausimu užsakovo nuomonė yra svariausia.

4.3.5 Testavimo resursai

4.3.5.1 Techninė įranga

Pagrindinis testavimo įrenginys – kūrėjo asmeninis kompiuteris. Jeigu to reikia, testavimui gali būti panaudoti universiteto kompiuteriai.

4.3.5.2 Testavimo įrankiai ir aplinka

Vienetų ir integravimo testavimui bus naudojamas JUnit testavimo karkasas. JUnit suteikia galimybę kurti testus, juos leisti ir stebėti jų rezultatus. JUnit yra atviro kodo projektas, jis tinkamas vienetų ir integracinių testų kūrimui.

JUnit karkasas naudojamas NetBeans aplinkoje. NetBeans turi patogią testų stebėjimo langą, kur testai išdėstomi hierarchiškai, pavykę ir nepavykę testai vaizduojami skirtingomis spalvomis (18 pav.).

4.3.6 Testavimo procedūra

4.3.6.1 Testuojama programinė įranga

Testuojama sistema – AST pertvarkymo programinė įranga skirta atlikti bendrinio AST pertvarkymus. Kuriama sistema išskirtina tuo, kad gali apdoroti įvairių kalbų kodą patalpintą į bendrinį AST. Sistema susideda iš pertvarkymų bibliotekos, bendrinio AST bibliotekos ir vartotojo sąsajos, kuri yra tik pagalbinė priemonė naudojama pademonstruoti bibliotekų veikimą.

4.3.6.2 Testavimo procedūros

4.3.6.2.1 Vienetų testavimas

Bus atliekami tokie pagrindiniai vienetų testavimo etapai:

1) **Bendrinio AST elementų testavimas** – testuojami visi medžio elementai, naudojamas vienetų testavimas.
2) **Bendrinio AST elementų nuskaitymo ir saugojimo testavimas** – kiekvienas medžio elementas nuskaitomas ir įrašomas į failą. Norima įsitikinti, kad elementų duomenų pasiekiamumo metodai („set“ ir „get“ metodai) veikia korektiškai;
3) **Bendrinio AST pertvarkymo bibliotekos fasado testavimas –** bus tikrinama, ar iškvietus atitinkamus fasado metodus bus kreipiamasi į teisingus kitų klasių metodus su teisingais parametrais;
4) **Bendrinio AST elementų šalinimo pertvarkymų testavimas –** bus tikrinama ar teisingai atliekamas elementų šalinimas iš bendrinės AST struktūros (ar teisingai sujungiami nutraukti ryšiai ir pan.);
5) **Bendrinio AST elementų įterpimo pertvarkymų testavimas –** bus tikrinama ar teisingai įterpiami nauji elementai į bendrinę AST struktūrą;
6) **Bendrinio AST elementų perkėlimo pertvarkymų testavimas –** bus tikrinama ar teisingai pakeičiama AST elemento vieta;

Visi vienetų testai, kurie turės atskleisti kelių objektų tarpusavio bendravimą, naudos tariamas klasių realizacijas (angl. „mock objects“).

4.3.6.2.2 AST pertvarkymų korektiškumo testas

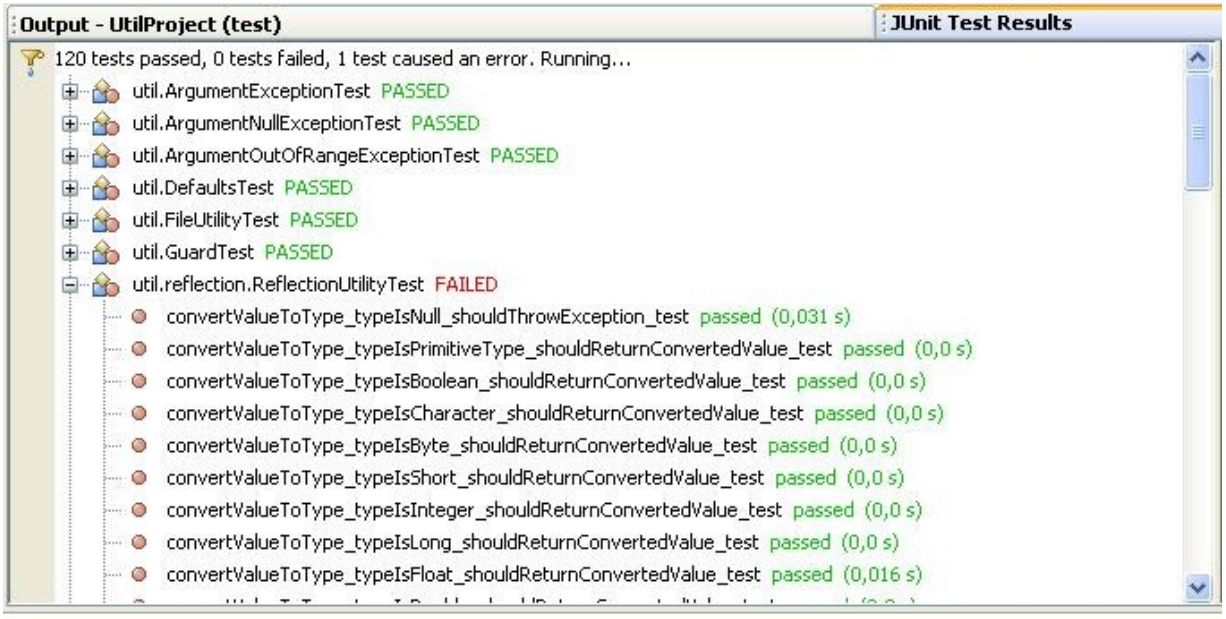
Priėmimo testavimas bus atliekamas pagal tokius punktus:

1) Atidaromas AST pertvarkymų įrankis;
2) Įdedamas iš anksto paruoštas 100-300 elementų bendrinis AST medis;
3) Atliekamas pagal panaudos atvejus sudarytas pertvarkymo operacijų rinkinys;
4) Gautas medis išsaugomas į pasirinktą failą;
5) Pradinis ir pertvarkytas medžiai lyginami.

4.3.6.3 Testavimo rezultatai ir išvados

4.3.6.3.1 Vienetų testavimo rezultatai

Vienetų testavimas atliekamas naudojant JUnit 4.5 testavimo karkasą. Pradžioje realizuojant projektą visoms kuriamoms klasėms ir jų metodams iš anksto realizuojami vienetų testai. Realizacijos eigoje buvo nuspręsta testus rašyti iš anksto tik sudėtingiems metodams, dėl laiko trūkumo. Vienetų testų rezultatai buvo patogiai pateikiami JUnit karkaso, NetBeans kūrimo aplinkoje(18 pav.)



18 pav. Testavimo rezultatų pavyzdys

4.3.6.3.2 Integraciniai testai

Integraciniai testai taip pat buvo atliekami naudojant JUnit 4.5 testavimo karkasą. Bet kartu su 7.2.1 skyriuje aptartu rezultatų kaupimu, buvo naudojama paprasta lentelė – iškilusių klaidų aprašymui.

2. lentelė. Testavimo rezultatų kaupimo Exel lentelėje pavyzdys.

Numeris	Testo Aprašymas	Testo rezultatas	Pastabos	Būsena
1	AST struktūros sudarymas saugykloje Nr.4	nesėkmingas	klaida susidaro DerivesFrom klasėje. (GenericAST)	Pataisyta
2	AST struktūros įkėlimas/saugojimas Nr.12	nesėkmingas	klaida įvyksta išsaugant AST struktūrą. Išsaugomi nereikalingi, tušti elementai. Reikia pataisyti elementu išsaugojimo metodą	Pataisyta
3	AST struktūros sudarymas saugykloje Nr6	nesėkmingas	klaida susidaro klasei ClassType sąveikaujant su kitomis klasėmis. Pati klasė yra teisinga. (GenericAST)	Pataisyta
4	AST struktūros įkėlimas/saugojimas Nr.13	nesėkmingas	klaida įvyksta nuskaitant sąrašų elementus(GenericAST)	Pataisyta
5	AST struktūros įkėlimas/saugojimas Nr.14	nesėkmingas	klaida įvyksta išsaugant AST struktūrą į failą.	Pataisyta
6	AST struktūros įkėlimas/saugojimas Nr.15	nesėkmingas	klaida įvyksta nuskaitant AST struktūrą. Klaida serializavimo metoduose, neteisingai interpretuojami klasių kintamųjų tipai	Pataisyta
7	AST pertvarkymas/elemento pervadinimas	nesėkmingas	buvo pervadinti ne visi elementai	Pataisyta
8	AST pertvarkymas/inkapsuliacija	nesėkmingas	set/get metodai buvo sukurti neegzistuojančiam elementui. Įdėti tikrinimą	Pataisyta

Papildomai informacija buvo kaupiama testavimo rezultatus įrašant į atskirus failus, kad būtų galima nesunkiai dalintis tarp abiejų kūrėjų. Taip buvo galima greičiau nustatyto klaidos vietą ir ištaisyti klaidas.

4.4 Testavimo veiklos vertinimas

1. Pasirinkta kūrimo ir testavimo strategija davė neblogus rezultatus projekto pradžioje ir jam įpusėjus. Iš anksto realizuoti vienetų testai neleisdavo įvelti kritinių klaidų į kuriamus komponentus. Integraciniai testai leido aptikti užsilikusias klaidas tiek AST struktūroje, tiek ją naudojančiuose įrankiuose.
2. Testavimą taip pat palengvino pasirinktas projekto valdymo tipas, kas leido kūrėjams peržiūrėti vienas kito kodą, padėti taisyti ir aptikti klaidas.
3. Projektui įpusėjus, kai testų kodo eilučių skaičius keletą kartų viršijo programos kodą, bei testų kūrimas užtruko žymiai ilgiau nei testuojamų funkcijų, testavimo apimtys buvo apribotos. Tam didelės įtakos turėjo tai, kai projekto pabaigimas negalėjo būti atidėtas. Buvo nuspręsta ženkliai sumažinti testų kiekį paprastoms bei nelabai sudėtingoms funkcijoms. Tai galėjo įtakoti didesnį užsilikusių klaidų kiekį kuriamuose įrankiuose.

PERTVARKYMO BIBLIOTEKOS EKSPERIMENTINIS TYRIMAS

4.5 Eksperimento tikslas

Tikslas – išmatuoti algoritmų greitaveiką sukurtoje pertvarkymų bibliotekoje.

4.6 Eksperimento apibrėžimas

Nepriklausomi kintamieji šiame eksperimente yra:

- 2 programavimo kalbos (JAVA ir C++)
- pertvarkymo algoritmai
- AST elementų skaičius ir klasės metodų skaičius.

Kadangi šie kintamieji šiek tiek skirtingi skirtingų pertvarkymo algoritmų atveju, detalus kiekvieno eksperimento aprašymas pateikiamas toliau.

Visi algoritmai buvo tiriami naudojant dviejų programavimo kalbų vienodą reikšmę turintį kodą bendrinėje AST struktūroje.

Dalis eksperimentų buvo atliekami didinant elementų skaičių ir matuojant skaičiavimo laiką.

Dalis eksperimentų buvo atliekami didinant klasių metodų skaičių. Jie buvo taikomi pertvarkymo algoritmams kuriems svarbi tiriamo metodo vieta klasėje.

```
public class Test {
        String void tiriamasMetodas(int n)
        {
                String a = “Helo World”;
                return a;
        }
        public void fillMetodas1()
        {
                String a = “Helo World”;
        }
        public void fillMetodas2()
        {
                String a = “Helo World”;
        }
        .
```

```
        .
        .
        public void fillMetodasn()
        {
                String a = “Helo World”;
        }
}
```

Atliekant eksperimentą, tiriamojo metodo vieta klasėje buvo keičiama taip: pirmas arba paskutinis. Taip atliekant eksperimentus bus galima nustatyti minimalų ir maksimalų vykdymo laiką. Eksperimentai buvo atlikti naudojantis sukurta pertvarkymo biblioteka.

4.7 Eksperimentų atlikimo aplinka

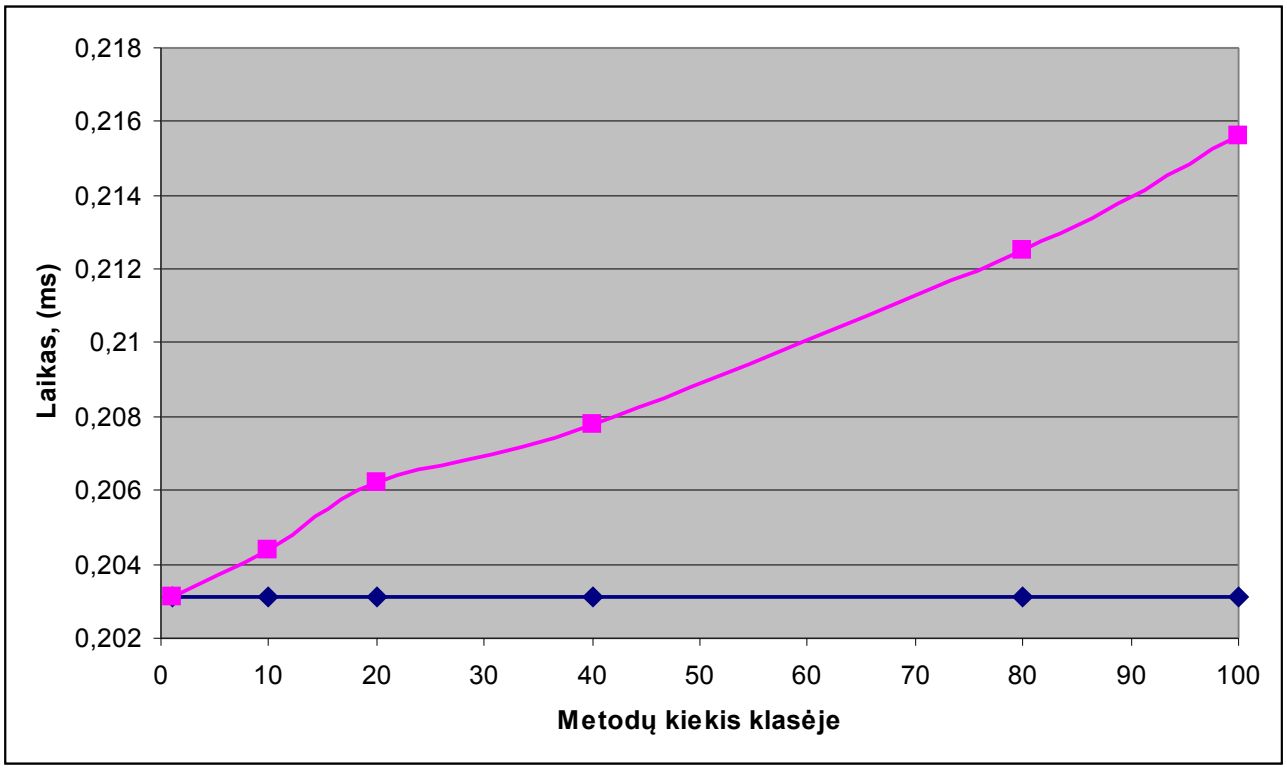
Eksperimentai buvo atliekami naudojantis asmeniniu kompiuteriu, juos atliko Justas Jokubauskas. Reikšmingi parametrai:

- Procesorius – 2GH Athlon.
- Darbinė atmintis – 1 GB.
- Naudojama NetBeans IDE 6.5.1 programų kūrimo aplinka.
- Windows XP operacinė aplinka.

4.8 Pertvarkymo algoritmų tyrimas

4.8.1 Metodo parametro pervadinimas

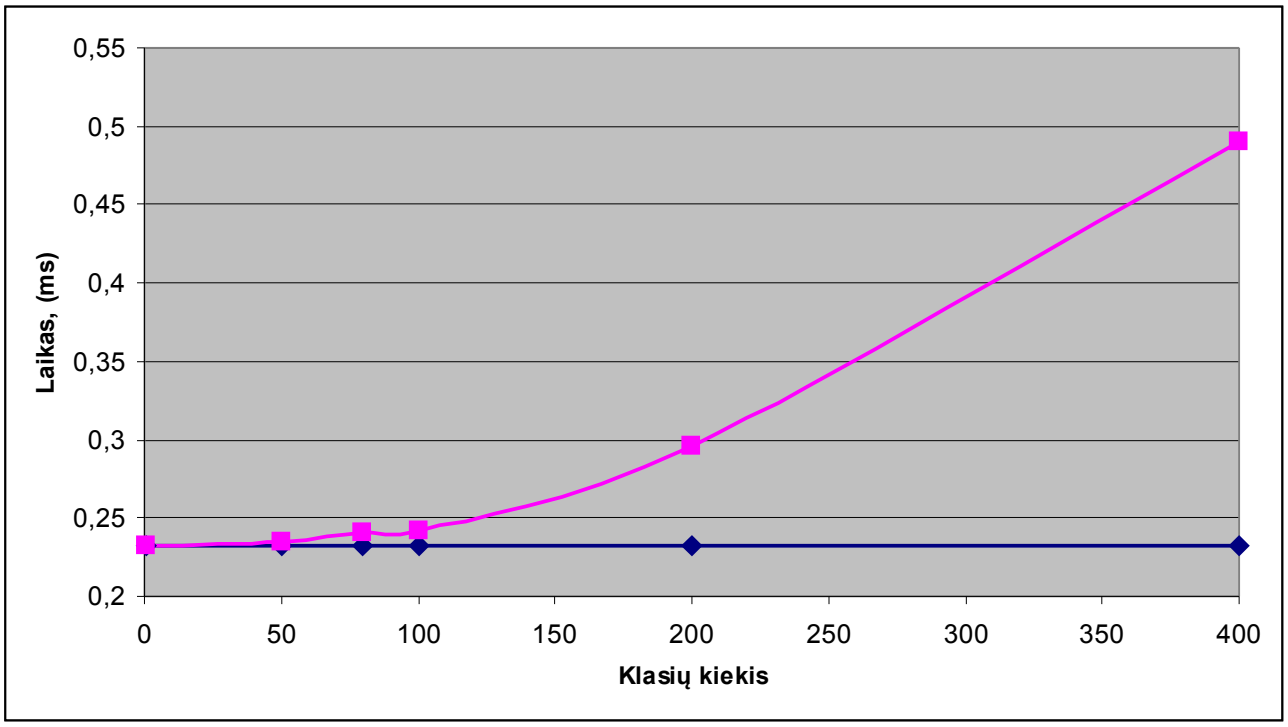
Kadangi šiam algoritmui nebūtina perrinkti visų elementų, todėl jo minimalus ir maksimalus vykdymo laikas priklauso nuo metodo, kurio parametro vardas keičiamas, buvimo vietos klasėje. Jei metodas yra pradžioje, tai vykdymo laikas trumpesnis. Atitinkamai, jei metodas yra klasės pabaigoje, tai vykdymo laikas bus didesnis. Taip yra dėl to, kad metodo suradimui taip pat reikia laiko. Viršutinė linija žymi maksimalų metodo vykdymo laika prie duotų parametrų, o apatinė – minimalų (19 pav.)



19 pav. Algoritmo – metodo parametro pervadinimas vykdymo laiko grafikas, pagal metodų kiekį klasėje(JAVA)

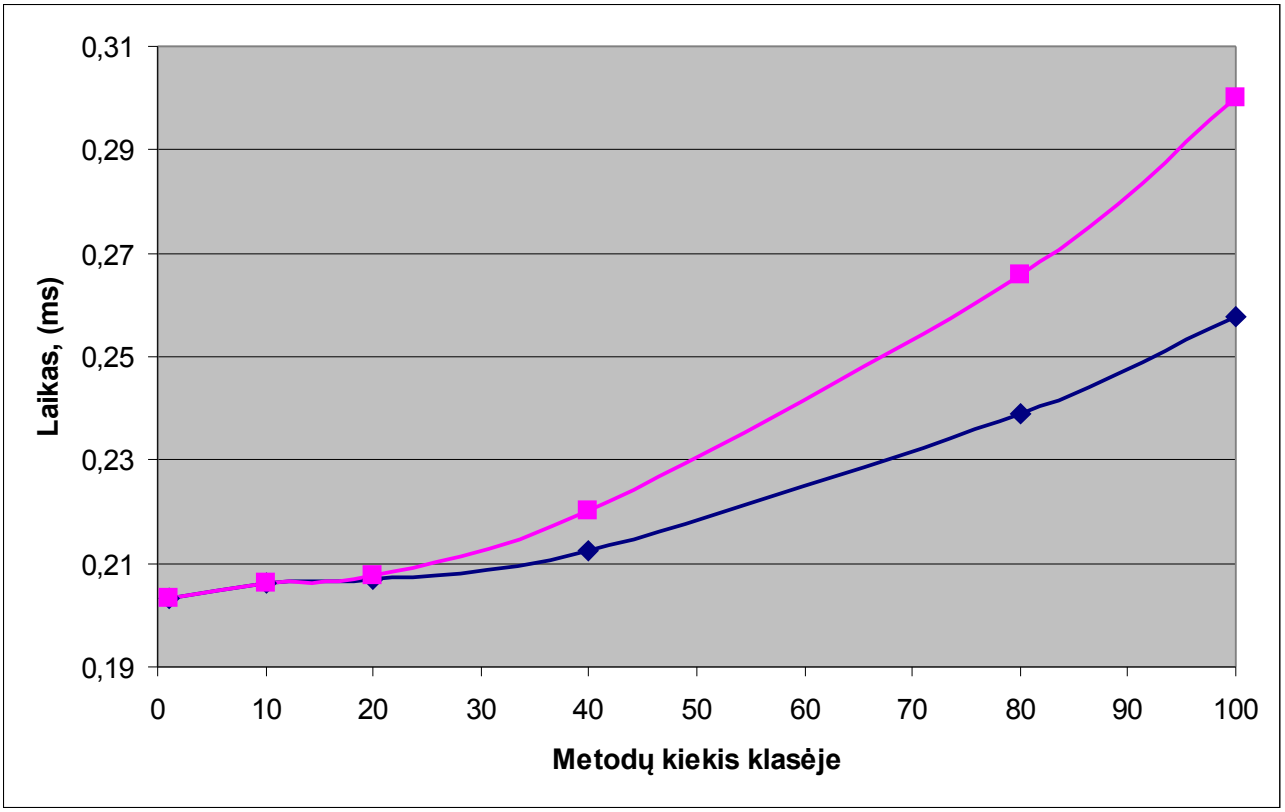
Buvo atlikti bandymai su 1-100 metodų klasėje. Realus algoritmo vykdomo laikas, bet kokioje situacijoje, būtų plote tarp dviejų grafiko laiko linijų. Pastaba: matavimai buvo atlikti su metodais, kurie nėra dideli. Jeigu bus keičiamas parametras metodo, kuriame yra daug kodo eilučių, tai vykdymo laikas gali būti didesnis už pateiktą maksimalų laiką.

Sekantis atvejis – metodų kiekis klasėje fiksuotas (50), o klasių kiekis didinamas (20 pav.). Žemesnė linija vaizduoja atvejus, kai klasė, kurioje yra pertvarkomas metodas, visada yra sąrašo viršuje, o viršutinė linija – kai toji klasė pasiekiama paskutinė.



20. pav. algoritmo – metodo parametro pervadinimas vykdymo laiko grafikas, pagal klasių kiekį(JAVA ir C++)

21 paveiksle pateiktas to paties algoritmo vykdymas su C++ kodu bendriniame AST. C++ turi metodo deklaracijas. Dėl šios priežasties, maksimalus vykdymo laikas yra didesnis nei Java kalbos AST. Grafike pateiktos 2 laiko linijos. Žemesnė linija gauta, kai metodo deklaracija yra klasės elementų viršuje, o apibrėžimas po deklaracijų. Viršutinė linija vaizduoja situaciją, kai tiriamojo metodo deklaracija yra deklaracijų pabaigoje, apibrėžimas – gale (tai reiškia – deklaracija įdėta paskutinė iš deklaracijų ir apibrėžimas įdėtas paskutinis iš apibrėžimų). Tikėtina, kad realioje situacijoje vykdymo laikas bus plote apribotam šių dviejų linijų.



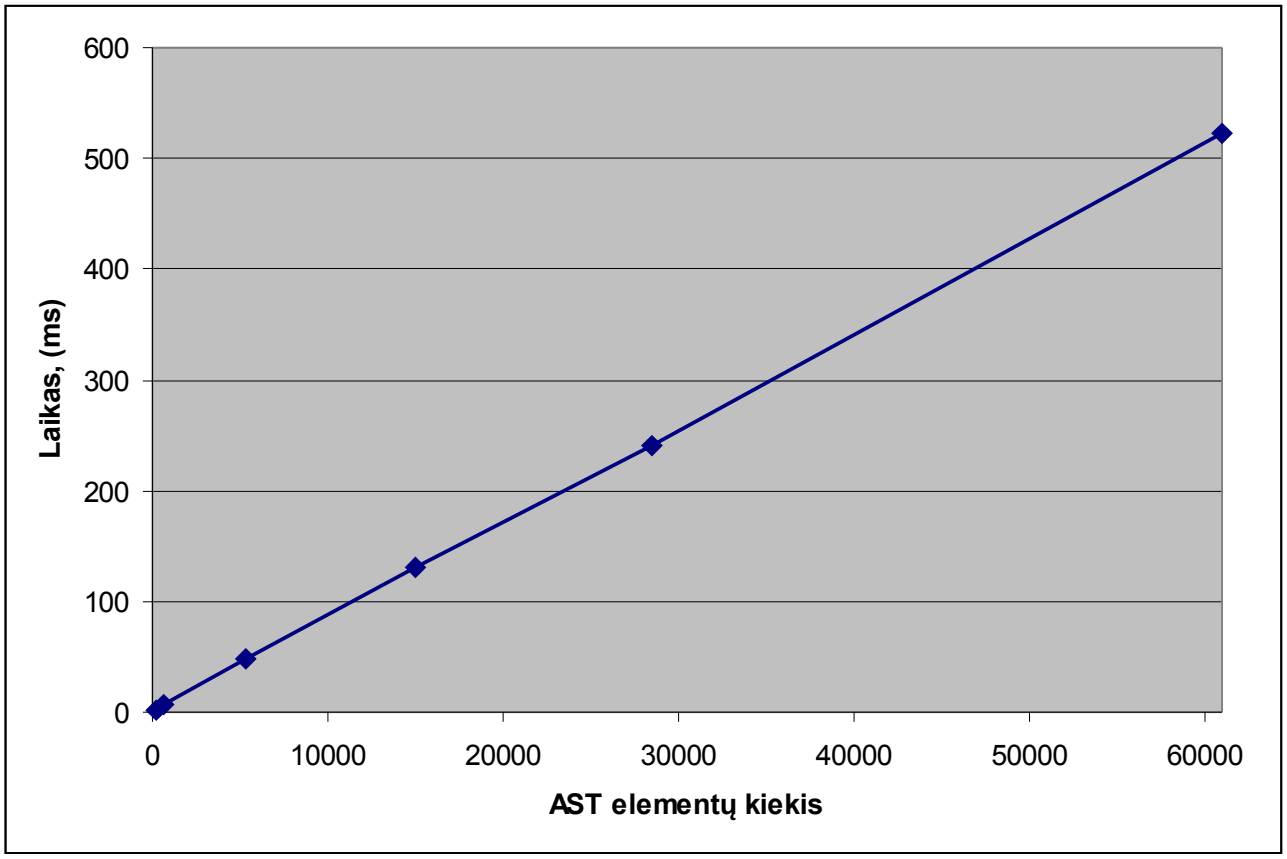
21 pav. Algoritmo – Metodo parametro pervadinimas vykdymo laiko grafikas, pagal metodų kiekį klasėje(C++).

4.8.2 Klasės kintamojo pervadinimas

Pagal grafika, algoritmo sudėtingumas yra O(n) eilės. Vykdymo laikas galėjo sulėti dėl to , kad kompiuteryje nėra laikoma tradicinė medžio struktūra – duomenys išdėstyti įvairiose klasėse. Todėl, kiekviename medžio elemente, norint sužinoti kiek ir kokius vaikų elementus turi esamas elementas, reikia surasti visus „get“ metodus elemento klasėje. Toliau vykdyti rastus metodus ir nagrinėti gautus rezultatus. Galimos reikšmės:

- Gaunamas vienas elementas, primityvaus tipo – tai lapo elementas, vaikų nebeturi.
- Gaunamas vienas elementas, sudėtingo tipo – tai šaka, reikia ieškoti vaikų.
- Gaunamas sąrašas, primityvaus tipo elementai – visi elementai yra lapai.
- Gaunamas sąrašas, elementai sudėtingo tipo – visi sąrašo elementai turi savo vaikų, juos reikia nagrinėti.

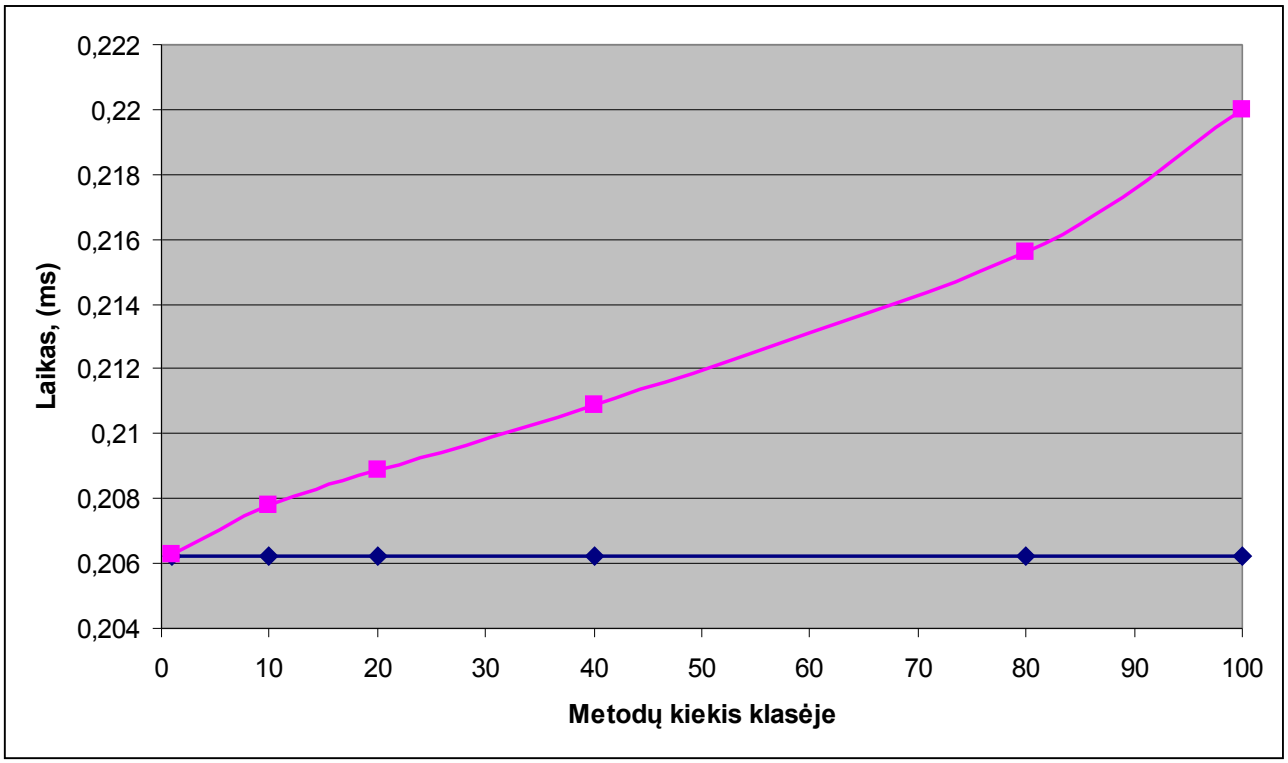
Dalis laiko sugaištama ėjimui per medžio elementus, todėl algoritmas su šia struktūrą dirba lėčiau. Kadangi peržiūrimi visi elementai, tai skirtumo tarp Java ir C++ vykdymo nėra.



22 pav. Algoritmo – klasės kintamojo keitimas, vykdymo laiko grafikas, pagal metodų elementų kiekį bendriniame medyje(JAVA ir C++).

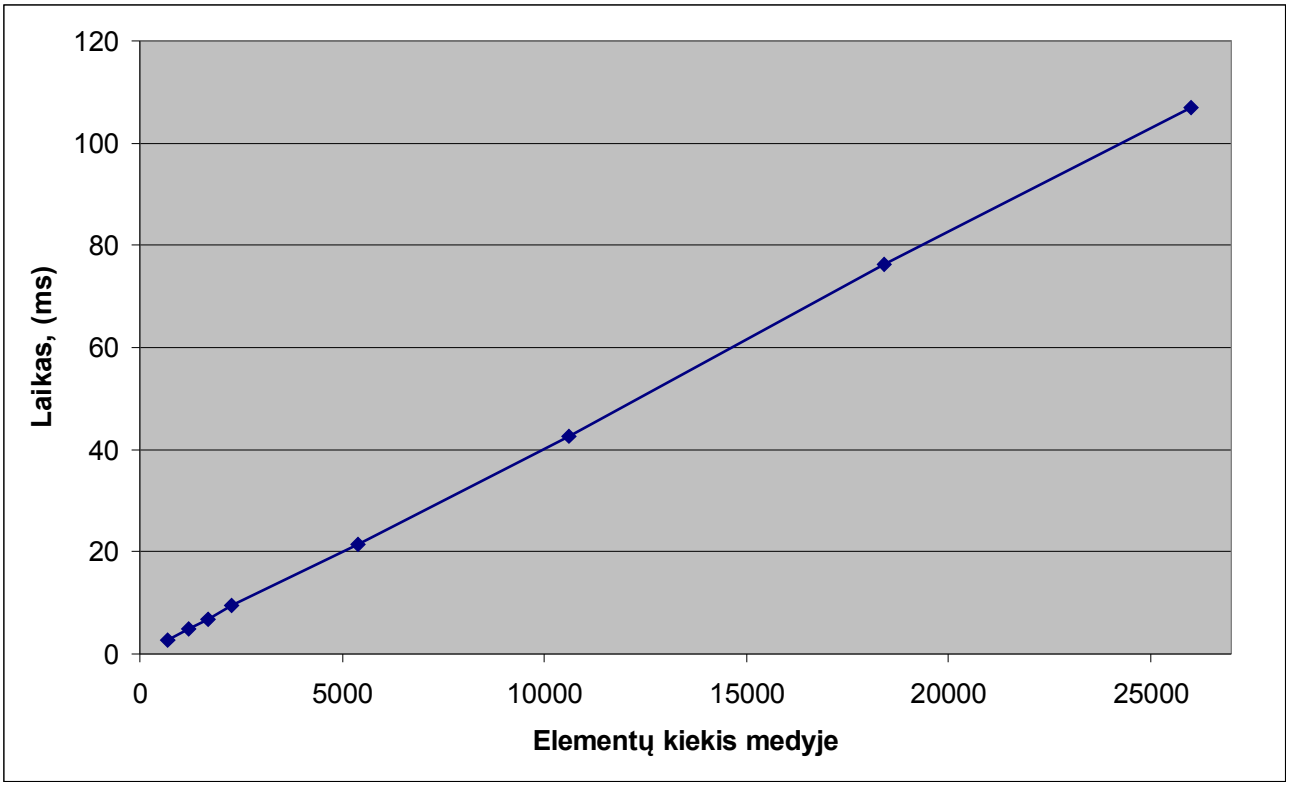
4.8.3 Lokalaus kintamojo pervadinimas

Algoritmas vykdomas 1000 kartų. Kaip matome, vėl yra dvi laiko linijos: minimali, kuri nekinta didinant metodų kiekį ir maksimali, kuri didėja priklausomai nuo metodų kiekio nagrinėjamoje klasėje. Metodų kiekis ir elementų kiekis bendriniame AST nėra lygūs, norint nustatyti ar metodas nėra ieškomasis, reikia patikrinti daugiau nei vieną elementą. Viršutinė laiko linija vaizduoja situaciją, kai tiriamasis metodas yra klasės gale. Apatinė laiko linija – kai tiriamasis metodas klasės viršuje, pirmas eilėje. Maksimalus laikas gali dar padidėti, priklausomai nuo metodo dydžio, kuriame keičiamas kintamojo vardas. Veiksmai metodo viduje vykdomi O(n) sudėtingumo algoritmu.



23 pav. Algoritmo – lokalaus kintamojo pervadinimas, vykdymo laiko grafikas, pagal metodų kiekį klasėje (JAVA ir C++).

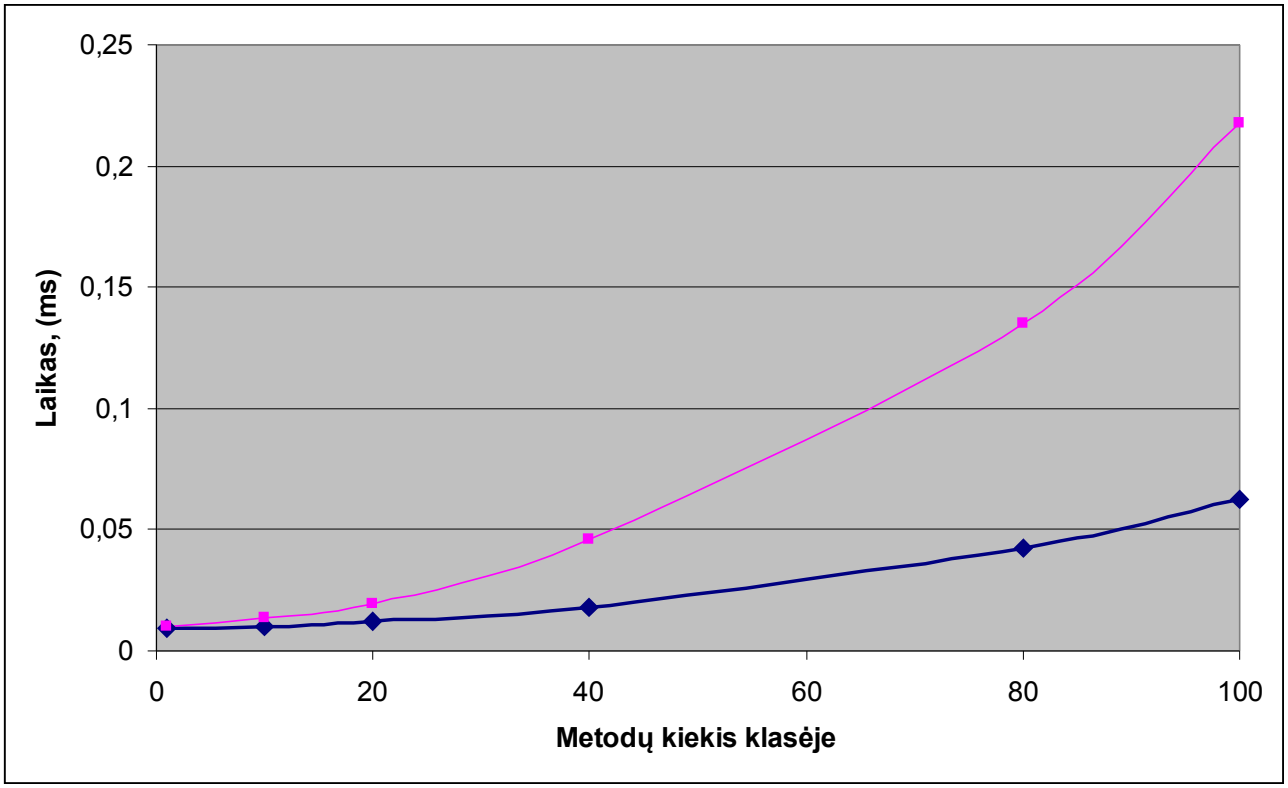
4.8.4 Klasės pavadinimo keitimas



24 pav. Algoritmo – klasės pavadinimo keitimas, vykdymo laiko grafikas, pagal metodų elementų kiekį bendriniame medyje.

Iš grafiko matyti, kad algoritmo sudėtingumas yra O(n) eilės. Šiai pertvarkymo operacijai reikia patikrinti visus medžio elementus, nes pakeitus klasės vardą, reikia pakeisti įrašus apie tą klasę visose kitose klasėse.

4.8.5 Klasės kintamojo inkapsuliacija



25 pav. Algoritmo – klasės kintamojo inkapsuliacija, vykdymo laiko grafikas, pagal metodų kiekį klasėje(JAVA, C++).

Kaip matome yra dvi laiko linijos: minimali, kuri vaizduoja bendrinio AST, sudaryto iš JAVA kodo, pertvarkymą ir maksimali – bendrinio AST, sudaryto iš C++ kodo, pertvarkymas. Skirtumas susidaro, nes C++ atveju reikia patikrinti metodų deklaracijas.

4.9 Eksperimento rezultatų įvertinimas

Iš gautų rezultatų galima spręsti, kad sudėtingesnių algoritmų sudėtingumas neturėtų tapti eksponentiniu. Taip pat įsitikinome, kad sukurtos bendrinės AST struktūros panaudojimas visiškai nedaro arba nedaro žymių pokyčių algoritmų vykdymo laikui. Atlikę eksperimentus buvo nustatyta, kad algoritmų greitaveika pertvarkymo bibliotekoje mus tenkina tik dalinai. Iš eksperimento rezultatų galima daryti išvadą, kad algoritmų pritaikymas C++ kalbai yra blogesnis nei JAVA kalbai, todėl algoritmus būtina tobulinti.

5. IŠVADOS

1. Atlikta analizė parodė, kad bendrinio AST struktūrą tikslinga aprašyti pagal OMG GASTM specifikaciją, o kuriant pertvarkymų biblioteką duomenų struktūras tikslinga perduoti XML dokumentais.

2. Siekiant nustatyti, ar tikslinga kurti sudėtingesnius pertvarkymų algoritmus, buvo nuspręsta sukurti pagrindinių pertvarkymų algoritmų rinkinį ir jį realizuoti pertvarkymo bibliotekoje.

3. Sukurtos pertvarkymų bibliotekos testavimas parodė, kad biblioteka sukurta be žymių klaidų.

4. Eksperimentinio tyrimo metu nustatyta, kad:

4.2. Algoritmų greitaveika blogesnė su C++ kalba.

4.3. Norint algoritmus naudoti C++ programavimo kalbai, reikia juos optimizuoti, kad greitaveikos skirtumas nebūtų toks didelis, lyginant su JAVA kalba.

5. Eksperimentas leidžia daryti prielaidą, kad sukurta pertvarkymų biblioteka ir bendrinio AST struktūra yra perspektyvios ir būtų galima jas plėtoti toliau.

6. LITERATŪROS ŠALTINIAI

[1] Fowler, Martin; et al. REFACTORING IMPROVING THE DESING OF EXISTING CODE, 2004., ISBN-0-201-48567-2.

[2] Wake, W. C. REFACTORING WORKBOOK, 2003., ISBN 0-32-110929-5.

[3] JetBrains, ReSharper [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://www.jetbrains.com/resharper/index.html>.

[4] OMG, Abstract Syntax Tree Metamodel Specification, žiūrėta 2010-04-29, Prieiga per internetą: <http://www.omg.org/spec/ASTM/1.0/Beta1/>.

[5] Harold, E. R. Processing XML with JAVA, 2002., ISBN 0-201-77186-1.

[6] W3C, Extensible Markup Language (XML) [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://www.w3.org/XML/>.

[7] Pissanetzky, S. Refactoring with Releations, 2006. ISBN 0-9762775-4-9.

[8] McLaughlin, B., Java & XML, 2001. Antras leidimas. ISBN 0-596-00197-5.

[9] Loukides, M.;McLaughlin, B. Java and XML, 2000. ISBN**,** 0-596-00016-2.

[10] OMG, Abstract Syntax Tree Metamodel Standard ASTM Tutorial 1,0 [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://www.omg.org/news/meetings/workshops/ADM_2005_Proceedings_FINAL/T-3_Newcomb.pdf>.

[11] NetBeans, Refactoring, [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://wiki.netbeans.org/Refactoring>.

[12] Eclipce, Abstract Syntax Tree [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html>.

[13] Wiki, Wikipedia -laisvoji enciklopedija [interaktyvus], [žiūrėta 2010 05 10]. Prieiga per internetą: <http://lt.wikipedia.org/wiki/XML>.

[14] David, G. Refactoring for Everyone How and why to use Eclipse's automated refactoring features [interaktyvus], [žiūrėta 2010 05 20]. Prieiga per internetą: <http://www.ibm.com/developerworks/library/os-ecref/>.

7. TERMINŲ IR SANTRUMPŲ ŽODYNAS

1) **AST (Abstract syntax tree)** – Tai medžio tipo duomenų struktūra, vaizduojanti abstrakčią (supaprastintą) sintaksinę išeities teksto, parašyto tam tikra programavimo kalba, struktūrą.
2) **Bendrinis AST (Generic Abstract Syntax Tree, GAST)** – AST medis, kuris išreiškia nuo konkrečios programavimo kalbos nepriklausančią duomenų struktūrą.
3) **GASTM (Generic Abstract Syntax Tree Metamodel)** – Bendrinio abstraktus sintaksės medžio metamodelis.
4) **Refactoring –** Kodo pertvarkymas. Tai toks programinės įrangos pakeitimo procesas, kuris nepakeičia išorinio programos kodo elgesio, tačiau pagerina jo vidinę struktūrą.
5) **Parser** – sintaksės analizatorius.