KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. Egidijus Kazanavičius

UNIVERSALUS PROGRAMUOJAMAS INTERNETINIŲ ROBOTŲ KŪRIMO ĮRANKIS

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. katedros lekt.
dr. J. Mikelionienė

Vadovas
dr. doc. Stasys Maciulevičius

Recenzentas
doc. Eimutis Karčiauskas

Atliko
IFM-9/1 gr. stud.
Marijus Paškevičius

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Marijus Paškevičius

UNIVERSALUS PROGRAMUOJAMAS INTERNETINIŲ ROBOTŲ KŪRIMO ĮRANKIS

Magistro darbas

Vadovas
doc. dr. S. Maciulevičius

KAUNAS, 2005

SUMMARY

Actually, a whole new class of web user is developing. These users are computer programs that have the ability to access the Web in much the same way as a human user with a browser does. There are many names for these kinds of programs, and these names reflect many of the specialized tasks assigned to them. Spiders, bots, and aggregators are all so-called intelligent agents, which execute tasks on the Web without the intervention of a human being. In this research we will examine the differences between them and study possibilities to create user friendly programming toll which would be able to generate such kind of programs.

TURINYS

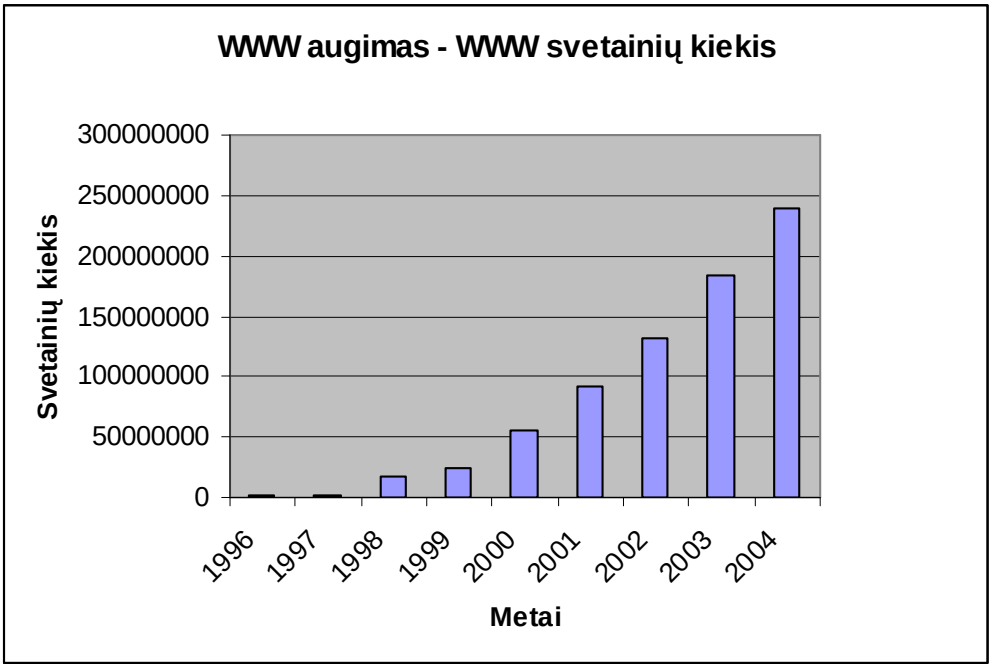
Paveikslėlių sąrašas

Lentelių sąrašas

1. BENDROJI DALIS

1.1. Internetiniai robotai

Vis daugiau laiko žmonės praleidžia ieškodami arba sekdami tam tikrą informaciją. Intensyviai plečiantis pasaulinio žiniatinklio apimtims, rankinis naršymas, norimai informacijai rasti, darosi nebeįmanomas. Tai sąlygoja vis didesnį naršymo automatizavimo poreikį, todėl kuriama bei tobulinama vis daugiau naujos žiniatinklio vartotojų klasės atstovų. Tai programiniai robotai gebantys naršyti po internetinius puslapius lygiai taip pat, kaip tai darytų bet kuris žmogus su savo internetine naršykle. Internetiniai robotai – programos standartinių žiniatinklio protokolų pagalba, naršančios ir nagrinėjančios informaciją, esančia žiniatinklyje. Per pastaruosius metus smarkiai auga internetinių robotų panaudojimas. Atsiranda naujų įrankių, siūlančių suprojektuoti savo agentus. Nemažai literatūros, aprašančios jų veikimo principus ir kūrimo technologijas. Tačiau vis dar trūksta universalių įrankių, suteikiančių galimybes vartotojui lengvai konstruoti neapibrėžto tipo robotus. Dabartinėje rinkoje esami įrankiai labai stipriai orientuoti į konkrečios problemos sprendimo metodika, jų siūlomi robotai siauros paskirties. Esant tokiai situacijai, asmuo, kiekvieną kart susidūręs su problema, turi daug laiko gaišti paieškai konkrečios paskirties roboto ar įrankio šiam robotui realizuoti.



1 pav. Pasaulinio žiniatinklio plėtimasis

1.2. Robotų panaudojimas.

1.2.1. Statistinė analizė

Pirmasis robotas buvo panaudotas žiniatinklio tarnybinių stočių radimui ir jų kiekio skaičiavimui. Statistinei analizei galima priskirti dokumentų vidurkį tenkantį tarnybinei stočiai, atitinkamų bylų tipų proporcijų skaičiavimą, nuorodų i kitas tarnybines stotis skaičiavimą.

1.2.2. Eksploatacija

Vienas pagrindinių sunkumų palaikant hipertekstinę struktūrą – tai, kad nuorodos į kitus šaltinius tampa „mirusiomis nuorodomis“, minėtajam šaltiniui persikėlus ar dingus. Šiomis dienomis nėra mechanizmo perspėjančio vartotojus, naudojančius nuorodas į kitus šaltinius, apie minimojo šaltinio pasikeitimus.

Robotai tikrinantys nuorodas, kaip *MOMspider*, gali padėti svetainės autoriui aptikti „mirusiąsias nuorodas“. Robotų pagalba galima ieškot stiliaus ar semantinių klaidų. Toks robotų panaudojimas gali ženkliai pagerinti esploatacijos kokybę ir sumažinti kaštus.

1.2.3. Atspindėjimas

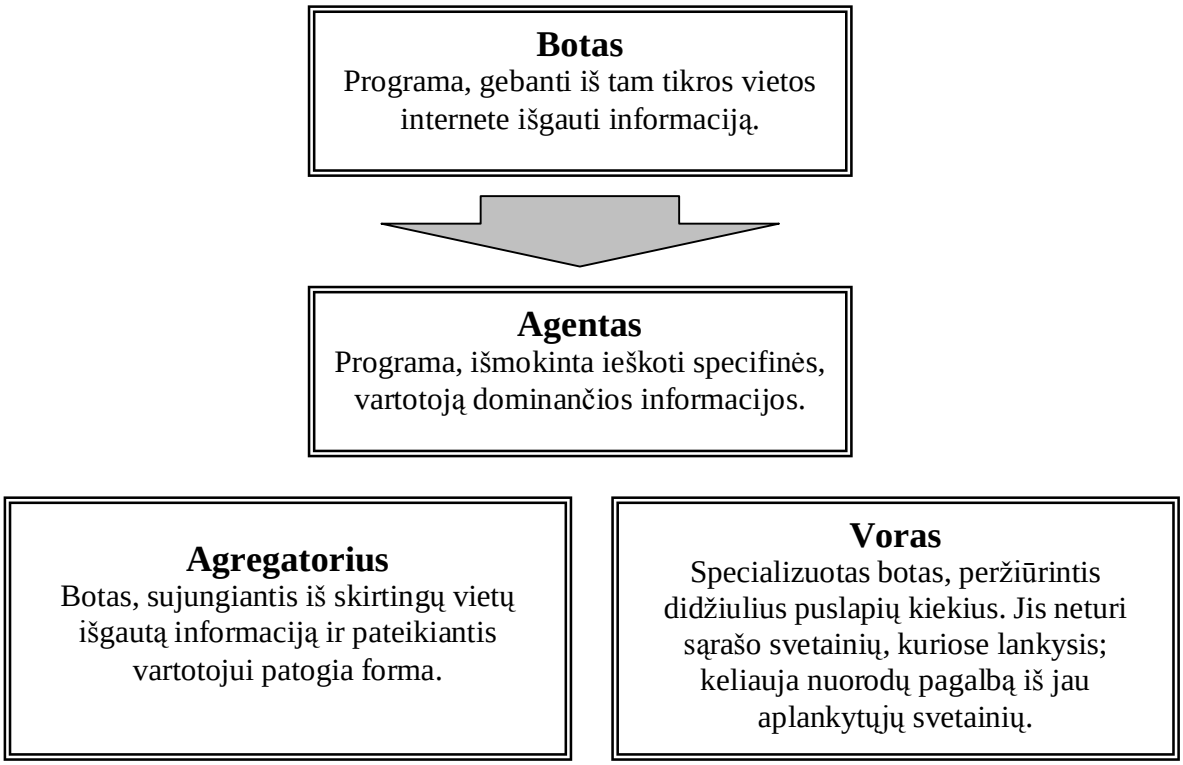
Atspindėjimas – populiari FTP archyvų palaikymo technologija. Veidrodis rekursiškai kopijuoja FTP katalogų medį ir nuolatos parsisiunčia dokumentus, kuriuose įvyko pokyčiai. Tai suteikia siuntimosi pasidalijimo galimybes, paspartina ir papigina vietinę prieigą. Žiniatinklio atspindėjimą galima realizuoti robotų pagalba.

1.2.4. Resurso radimas

Plačiausia robotų pritaikymo sritis – informacijos ieškojimas. Paieškos variklių kompanijos naudoja aibes robotų informacijai internete ieškoti ir rūšiuoti. Kuriamos milžiniškos duomenų bazės robotų apdorotai informacijai saugoti. Duomenų bazės reguliai peržiūrimos, informacija atnaujinama, šalinamos neveikiančios nuorodos. Be internetinių robotų, šių milžiniškų paieškos sistemų egzistavimas neįmanomas.

1.3. Internetinių robotų klasifikacija

Pagal savo specifiką ir atliekamą užduotį internetiniai robotai vadinami: vorais, botais, *agregatoriais*, agentais bei intelektualiaisiais agentais. Paveikslėlyje (žr. 2 pav.) pavaizduota šių programų hierarchija.



2 pav. Botai, vorai, agregatoriai ir agentai

1.3.1. Botas

Botais vadinamos paprasčiausios internetą pažįstančios programos. Jų pavadinimas kilęs iš termino robotas. Robotas – įrenginys, vykdantis jam palieptą dažnai pasikartojančią užduotį. Programiškasis robotas (botas) veikia tuo pačiu principu. Bet kurią programą, gebančią pasiekti informaciją internete ir ją „parnešti“, galima pavadinti botu: vorai, agentai, agregatoriai bei intelektualieji agentai – tai specializuotų, pagal veikimo principą bei atliekamą užduotį skirstomų, botų pavadinimai. Tam tikra prasme botai labai panašūs į kompiuterines

makrokomandas, leidžiančias vartotojui pakartoti tam tikrų komandų seką, tam tikrai užduočiai baigti. Taigi, iš esmės botas niekas kitas, kaip paprasčiausia makrokomanda, gebanti rasti vieną ar daugiau puslapių bei išgauti reikiamą informaciją iš jų.

Galima pateikti daug internete naudojamų botų pavydžių. Tarkim, paieškos varikliai Google ar Yahoo nuolat naudoja botus savo duomenų bazėse saugomų puslapių sąrašams peržiūrėti ar neatsirado kokių nors naujų nebeegzistuojančių nuorodų.

1.3.2. Voras

Voro (siejamo su vabzdžiu) pavadinimas šiai botų klasei prigijo dėl tam tikrų jam būdingų ypatybių: gebėjimo „suktis ratais“ ir aplankyti didelius painius puslapius, persikeliant iš vieno į kitą. Voras – tai specializuotas botas, kurio užduotis iš jau turimo svetainių sąrašo rasti nuorodas į kitas. Voras veikimą pradeda viename, kartais keliuose puslapiuose. Peržiūrėjęs ir sudaręs sąrašą visų nuorodų, jis keliauja į vieną iš jų ir vėl pildo sąrašą naujomis rastomis nuorodomis. Paprastai vorui dedami tam tikri apribojimai, kad paiešką vykdytų tik tam tikroje srityje, kitaip galimas variantas, jog voras savo paieškų taip ir nebaigs, kol neperžiūrės visų pasaulyje egzistuojančių internetinių svetainių puslapių. Patys pirmieji vorai pritaikyti internetiniuose paieškos varikliuose, gebančiuose pagal turimą žodžių rinkinį rasti reikiamas svetaines.

Vorų panaudojimas gali būti naudingas ne tik ieškant kitų svetainių. Jis taip pat sėkmingai gali būti panaudotas savo asmeninės svetainės struktūrai pavaizduoti, neegzistuojančių nuorodų ar net gramatinių klaidų paieškai.

1.3.3. Agentas bei intelektualusis agentas

Agentas – asmuo, vykdantis tam tikrą veiklą vietoj kito asmens. Panašiai ir kompiuterinis agentas vietoj vartotojo gali prieiti prie internetinio puslapio ir vykdyti tam tikrą veiklą, pvz., parduoti, pirkti. Kitas daugiau paplitęs agentų panaudojimo būdas – „kompiuterizuotas tyrinėjimo asistentas“. Toks agentas, susipažinęs su šeimininką dominančiomis sritimis gali pats aptikti ir pateikti tai, kas bent kiek susiję su šiom sritim. Nors ir turintys didelį potencialą, agentai dar nėra pasaulyje plačiai paplitę, dėl to, kad norint sukurti tikrai galingą, apibendrintą agentą reikalingi dirbtinio intelekto programavimo įgūdžiai.

Intelektualusis agentas – tai vartotojo suprojektuotas botas ir pagal tam tikrus kriterijus ieškantis informacijos. Intelektualusis agentas suradęs tam tikrą informaciją pasinaudodamas dirbtinio intelekto šablonų atpažinimo algoritmus gali atrinkti tai, kas jo vartotojui gali būti įdomu, o kas ne.

1.3.4. Agregatorius

Agregacija – procesas, sukuriantis sukomponuotą objektą iš kelių paprastesnių. Kompiuterizuota agregacija daro tą patį. Gali būt panaudojami prisijungti prie skirtingų vartotojo banko sąskaitų, prisijungimui panaudojant skirtingus ID bei slaptažodžius, gauti ir pateikti tam tikrą, apibendrintą informaciją. Esminis skirtumas nuo boto tas, jog informacija imama iš kelių skirtingų šaltinių.

1.4. Voro struktūra

Voras gali būt sukurtas dviem būdais: voras kaip rekursinė programa ir nerekursinis voras, saugantis puslapių sąrašą, kuriuose turės apsilankyti. Bandant pasirinkti vieną iš šių alternatyvų, reikia nepamiršti tai, jog voras turės tinkamai veikti ir su milžiniškais puslapių kiekiais.

1.4.1. Rekursinė programa

Rekursija – programavimo technika, kai metodas kreipiasi pats į save. Kai kuriuose projektuose kurti vorą naudojant rekursinį projektavimą būtų protingas, logiškai pagrįstas sprendimas. Tai naudinga praktiškai visur, kur paprasta užduotis kelis kartus pakartojama arba kai ateities informacija apie ateities užduotis gaunama ankstesnėm užduotims pasibaigus.

```
void RecursinisVoras(String url)
{
.... užkrauti URL ....
.... išnagrinėti URL ....
su kiekvienu rastu URL
   iškviesti RecursinisVoras(SuRastuURL)
.... ivykdyti ką tik užkrautą puslapį...
}
```

Šis metodas paleidžiamas nurodant vieną URL adresą. Išnagrinėjęs užkrautą puslapį bei radęs naujų nuorodų, metodas kreipiasi pats į save perduodamas naujai rastą URL adresą. Toks voro projektavimo būdas butų logiškas, tačiau jis visiškai netinka, jeigu norima vienu metu aplankyti kelias svetaines, nes kiekviena iteracija turi būt „padėta“ į dėklą ir laukti savo vykdymo laiko. Taip pat reikia nepamiršti ir to, jog rekursinei programai ilgai dirbant, dėklas gali stipriai išaugti bei suvartoti visą atmintį, tai sutrukdytų tolesniam programos veikimui.

1.4.2. Nerekursinė konstrukcija

Sekantis būdas suprojektuoti vorą yra nepanaudojant rekursinės programavimo technikos. Čia voras nenaudos metodų besikreipiančių patiems į save, tačiau vietoj to bus naudojamos eilės, kur visos rastos nuorodos bus talpinamos.

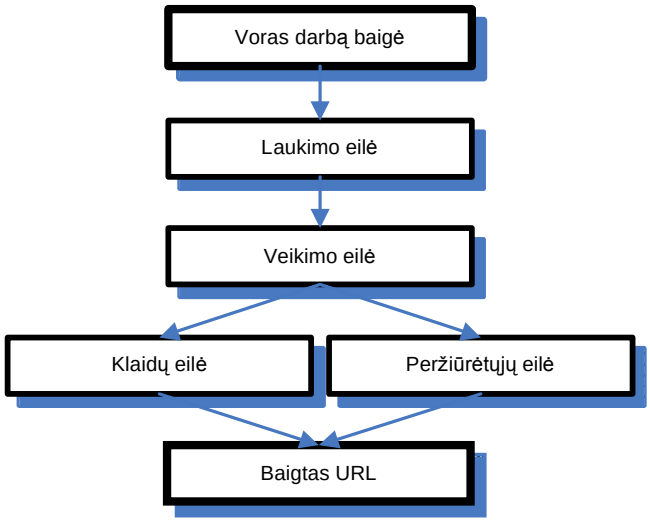
Voro eilės

Naudojant šį nerekursinį projektavimo metodą vorui bus pateikiamas sąrašas su pirmuoju puslapiu, nuo kurio jis pradės veikimą. Kiekviena nauja rasta nuoroda dedama į eilę. Baigęs paiešką esamame puslapy pereinama prie sekančios eilėje esančios nuorodos.

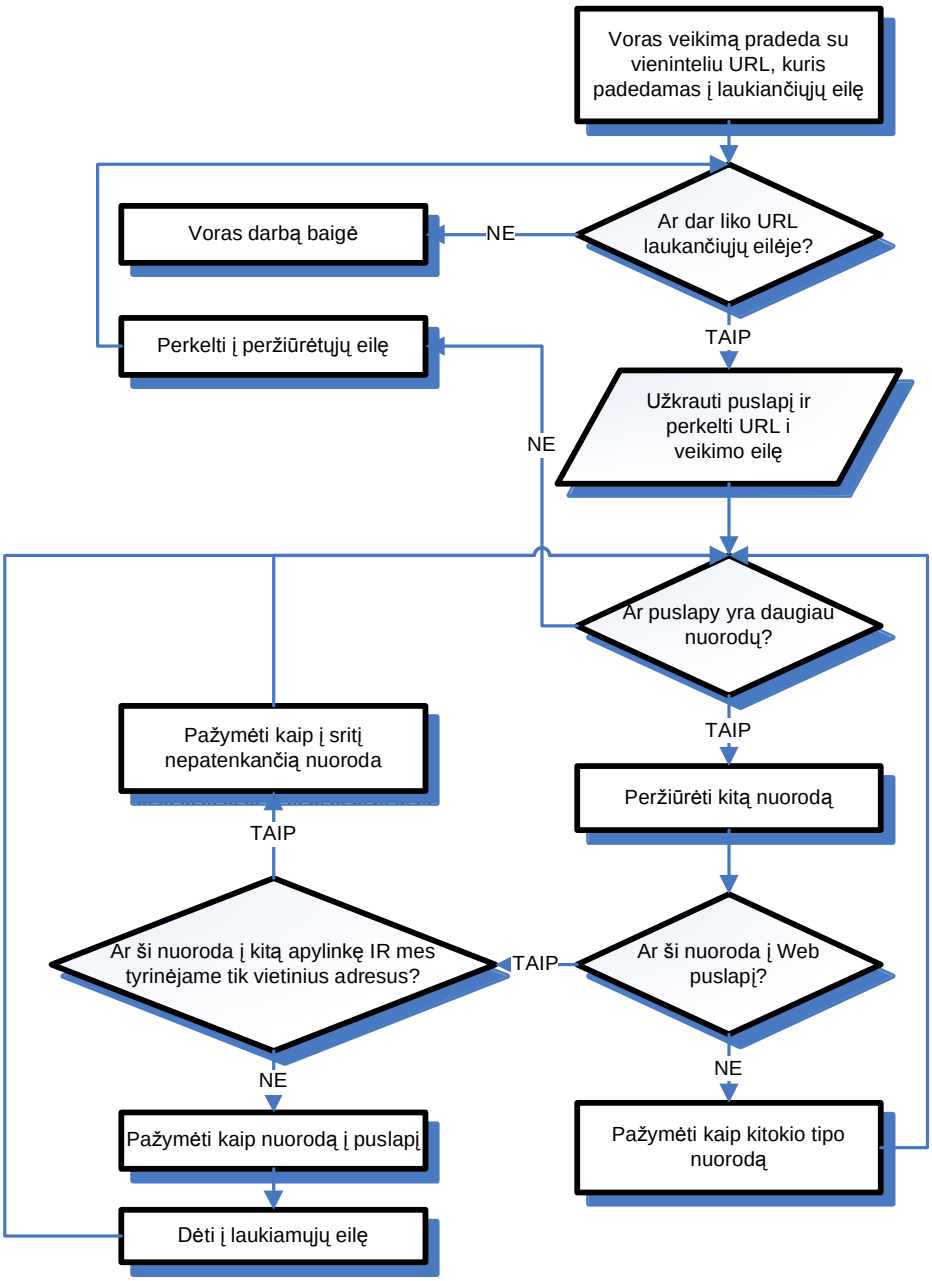
Laukimo eilė –	eilė su URL, kuriuos voras peržiūrės. Vorui suradus naują URL jie dedami į šią eilę.
Veikimo eilė –	čia patenka visi URL vos tik vorui pradėjus jį vykdyti. Naudojamas tam, kad voras vienu metu kelis kartus nenagrinėtų to paties puslapio. Baigus paiešką nuorodos dedamos arba į klaidų eilę arba į peržiūrėtųjų eilę.
Klaidų eilė –	įvykus klaidai bandant peržiūrėti puslapį jo URL bus dedamas čia.
Peržiūrėtųjų eilė –	sėkmingai baigus puslapio peržiūrą, jo URL patenka į peržiūrėtųjų eilę.

1 lentelė. Voro eilės

Adresui patekus į klaidų arba peržiūrėtųjų eilę, iš jos į kitas eiles jau nebepateks.



3 pav. URL būsenų kitimas



4 pav. Tipinio voro struktūrinė schema

1.4.3. Greitaeigis voras

Kadangi voro laukiančiųjų URL eilė didėja eksponentiniu greičių, yra labai svarbu sukonstruoti vorą kaip įmanoma efektyvesnį. Voro efektyvumą galima pakelti:

- Panaudojant lygiagrečiai veikiančius procesus.
- Panaudojant duomenų bazes URL eilėms kurti.

Kelių gijų panaudojimas

Multithreding'as – programos gebėjimas vienu metu vykdyti daugiau nei vieną užduotį. Ši galimybė turėtu ženkliai padidinti voro efektyvumą.

Vienas iš pagrindinių kelių optimizuojant programą – rasti „kamščius" apribojančius jos greitį. Kamščiai tai programos taškai smarkiai apribojantis jos veikimą. Nepriklausomai nuo to kaip gerai visa likusi programos dalis optimizuota, jei joje yra „kamščių", vis vien programa turės laukti savo pačios lėčiausios dalies.

Tarkim voras turi peržiūrėti 10 puslapių. Pradžioj jis voras pasiunčia serveriui užklausą, tada krauna šiuos puslapius iš serverio. Tuo metu kai voras laukia atsakymo į užklausą – susidaro taip vadinamas „kamštis". Šioje situacijoje kelių gijų panaudojimas gali smarkiai pagelbėti sukuriant 10 atskirų gijų, kurios vienu metu pasiųstų užklausas serveriui ir lauktų atsakymo.

Gijų sinchronizavimas

Projektuojant vorą nepamirštam ir to, jog gijos turi būt susinchronizuotos tarpusavyje. Daugelis gijų turi dirbti kartu dalindamiesi tuos pačius resursus, bei žinoti viena kitos veiksmus ir būsenas. Pirmas žingsnis kurį reikia atlikti – suskaldyti voro užduotis į daug mažyčių dalių, kurios bus lygiagrečiai vykdomos vienu metu atskirose gijose. Kad išvengtume veiksmų dubliavimosi šios gijos turės tarpusavy komunikuoti.

Duomenų bazių panaudojimas

Vorui dirbant su didelėm svetainėm, nepriekaištingam jo darbui būtinas efektyvus URL eilių valdymas. Valdyti šias eiles naudosim DBVS (duomenų bazių valdymo sistemas). Dėl savo populiarumo bei nesudėtingo valdymo puikiai tiktų SQL (*Structed Query Language*).

Tačiau būtina turėti omeny, jog DBVS panaudojimas atsiperka tiktai tuo atveju, jei voras turės aplankyti tikrai didelį puslapių kiekį (daugiau nei 10,000 psl.). Dirbant su mažesnėmis eilėmis, DBVS naudojimas ne tik nepasiteisina, bet ir ženkliai gali sumažinti jo veikimo greitį.

1.5. KatBotas

Daugelis standartinių botų yra projektuojami orientuojantis į tam tikrą internetinę svetainę . Taigi, pagrindinis jų trūkumas tame, jog pakeitus puslapio struktūrą, botas neras savo orientyrų ir neatliks jam paskirtos užduoties.

KatBot – botas gebantis išgauti informaciją iš visų tos pačios kategorijos puslapių. Priešingai nei paprasti standartiniai botai, Katbotai nėra orientuoti į tam tikrą specifinį puslapį, taigi jis be jokių sunkumu gali atlikti savo užduoti net ir pasikeitus puslapio struktūrai.

1.5.1. KatBoto Identifikatoriai

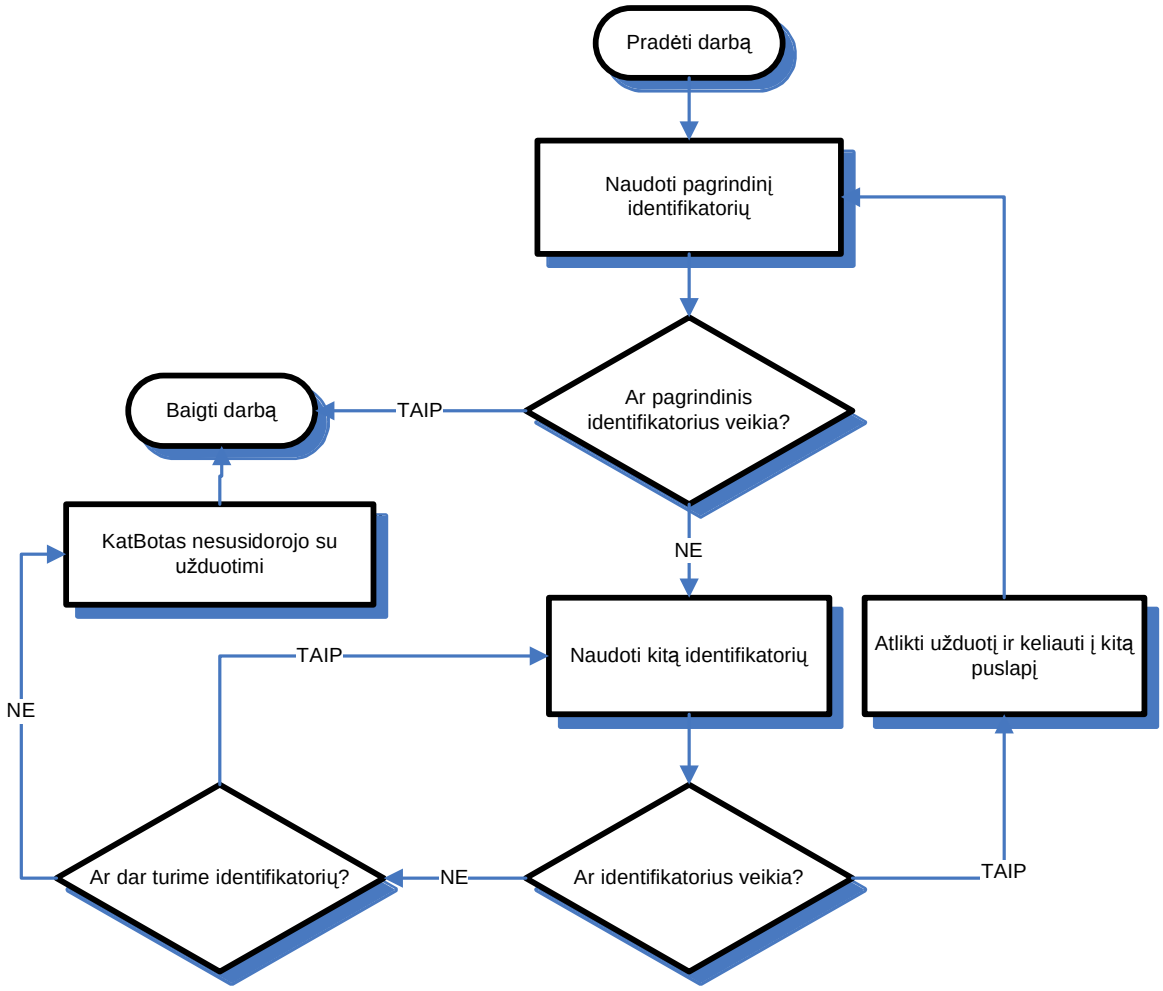
Katbotai naršo po svetaines naudodami tam tikrų klasių rinkinius vadinamus identifikatoriais. Identifikatorius – klasė suprojektuota atpažinti tam tikrus, specifinius puslapius. Katbotas veiks sėkmingai jei jis turi bent po viena gerai veikiantį identifikatorių kiekvienam puslapio tipui atpažinti.

Prieš paleidžiant Katbotą, jam nurodoma:

- URL, nuo kurio jis pradės savo darbą.
- Identifikatorių, su kuriais botas dirbs, rinkinys.
- Pagrindinis identifikatorius. Botas laiko užduotį sėkmingai atlikta, jei pagrindinis identifikatorius atpažįsta puslapį.

Katbotui pradėjus veikimą startiniame URL, bandomas pritaikyti pagrindinis identifikatorius. Jei reikiama informacija gaunama – darbas baigtas, priešingu atveju bandoma panaudot kitus identifikatorius. Jei nė vienas iš identifikatorių nesugebėjo išgauti duomenų, Katbotas darbą baigia.

Jei nors vienas iš identifikatorių puslapį sėkmingai atpažino, jam suteikiamas valdymas. Šis identifikatorius žino, kaip persikelti į kitus puslapius, kur vėl bus atliekama paieška. Katbotas darbą baigia tada, kai randa tai, ko nori, arba kai nė vienas iš identifikatorių nesuveikia.



5 pav. *Katboto* atpažinimo struktūrinė diagrama

1.6. Agregatorius

Galima rasti daug savo prigimtimi panašių puslapių, atliekančių panašią funkciją. Duomenis, paimtus iš šių puslapių, galima sujungti, palyginti. Vartotojas, besidomintis konkrečia informacija, greičiausiai turės kelis šaltinius, iš kurių surinkęs duomenis galėtų susidaryti bendrą vaizdą.

Agregatorius – tai botas ar botų rinkinys, renkantys informaciją iš kelių skirtingų šaltinių ir pateikiantis juos bendrame vaizde. Agregatorius gali būt projektuojamas kaip atskirų botų rinkinys arba kaip vienas KatBotas, išsiųstas į kelis puslapius. Surinkus visus duomenis valdymas perduodamas kitai programai, kuri šiuos sujungia į vieną bendrą vaizdą.

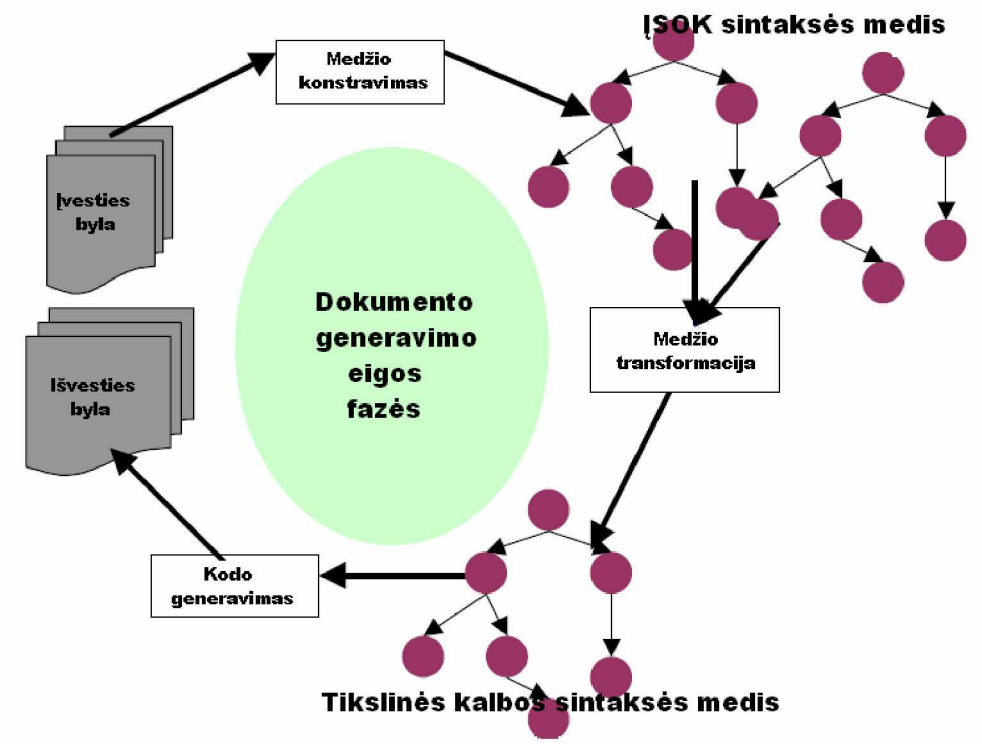
1.7. Generatoriai

Programavimo inžinerija anksčiau buvo labiau orientuota į originalų programinės įrangos kūrimą, bet dabar pripažinta, kad norint geriau, pigiau ir greičiau sukurti programinę įrangą, reikia pritaikyti projektavimo procesą pagrįstą sistemingu pakartotiniu panaudojimu. Viena iš pakartotinio panaudojimo paradigmų – programų generatorių naudojimas.
Patys paprasčiausi metodai naudojami programų generatoriuose kodui generuoti :

- Eilučių atstovavimas programos fragmentams**;**
- Duomenų tipų („abstrakčios sintaksės medžių“) atstovavimas programos fragmentams.

Deja, abu šie metodai turi savų trūkumų. Naudojant eilučių kodavimą, kodo fragmentą f(x, y) atvaizduojame simbolių eilute „f(x,y)“. Toks kodo fragmentų konstravimas ir apjungimas yra gana lengvas, glaustas ir praprastas. Tačiau nėra jokios garantijos, jog generuojamas programos kodas bus sintaksiškai teisingas.

Naudojant duomenų tipų formavimą mes esame tikri, jog sugeneruota programa yra sintaksiškai teisinga, tačiau šis metodas nesuteikia jokios garantijos, jog pati programa gerai parašyta. Pagrindinis trūkumas, jog programa koduojama labai sudėtingais duomenų tipais. Vienas seniausių metodų palengvinančių meta programavimą yra konkrečios sintaksės, į sritį orientuotų kalbų (ĮSOK) naudojimas. Į sritį orientuota kalba (angliškas terminas: domain specific laguage-DLS) yra programavimo kalba pritaikyta tam tikros konkrečios taikomosios srities uždaviniams, t.y. vaizdžiai, suprantamai ir efektyviai išreiškia taikomosios srities semantika.

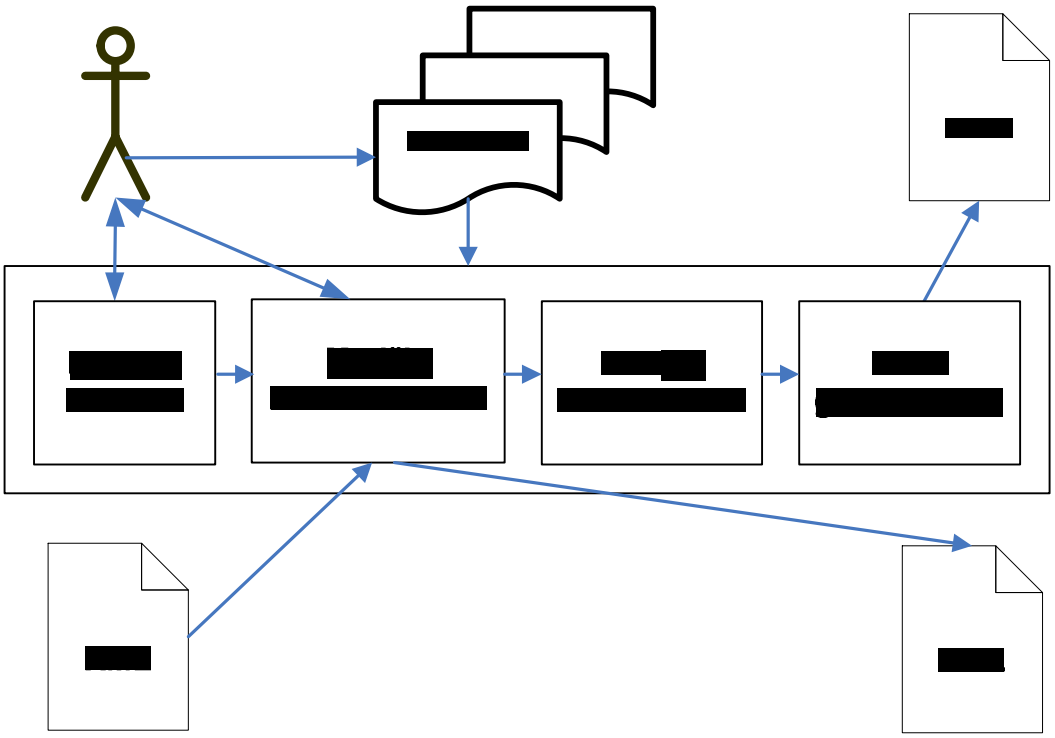


6 pav. Dokumento generavimo eigos fazės

Įvesties byloje saugoma į sritį orientuota programavimo kalba aprašyta sistemos specifikacija. Jei specifikacija aprašyta korektiškai, konstruojamas į sritį orientuotos kalbos sintaksės medis. Generatorius ĮSOK sintaksės medį transformuoja į tikslinės kalbos sintaksės medį. Jei medžio transformacija sėkminga, iš gauto medžio formuojama išvesties byla su sistemos specifikacija, aprašyta tiksline programavimo kalba.

2. TIRIAMOJI DALIS

Mano tikslas – sukurti lengvai suprantamą, paprastą ir patogų internetinių robotų kūrimo įrankį. Įrankis turi būt kiek įmanoma universalesnis, suteikiantis galimybę konstruoti visų tipų internetinius robotus. Realizacijai siūlau vieną iš pakartotinio naudojimo paradigmų – programų generavimą. Siūlomas generatoriaus modelis (žr. Pav. 7).



7 pav. Generatoriaus modelis

Generatoriaus funkcionavimas prasideda dialogu. Vartotojui siūloma pasirinkti: pradėt roboto konstravimą nuo pradžios, tęsti roboto konstravimą pasirinkus šabloną, tęsti anksčiau išsaugoto roboto konstravimą. Medžio konstruktorius atvaizduoja roboto struktūrą medžio pavidalu, leidžia vartotojui grafinės sąsajos pagalba lengvai kurti/redaguoti roboto elgsenos scenarijų. Vartotojas konstruodamas medį iš bibliotekos renkasi klases/metodus tinkamus roboto elgsenai aprašyti. Biblioteką vartotojas pagal poreikį turi galimybę papildyti. Jei medis sukonstruotas korektiškai, kodo generatorius generuoja robotą, aprašytą Java programavimo kalba. Medžio konstrukcija saugoma XML tipo byloje.

2.1. Medis

Kadangi įrankis labiau orientuotas į vartotojus, mokančius algoritmuot, įrankyje suteikiama galimybė pačiam reguliuoti roboto atliekamų komandų sekas. Vartotojas, naudodamas grafinę sąsają, iš pateiktojo sąrašo pasirenka norimas komandas, palaipsniui formuodamas programos medį. Medis puikiai tinka roboto elgsenos scenarijui atvaizduoti, jis lengvai suvokiamas bei redaguojamas.

Įrankyje naudojami kelių rūšių medžiai:

- Programos medžiai – atskiriems programos fragmentas atvaizduoti,
  - § Pagrindinis „*main()*“ medis – vykdomajam „*main()*“ metodui atvaizduoti,
  - § Pagalbinių metodų medžiai – naudojamiems metodams atvaizduoti,
  - § Gijų medžiai – proceso vykdomoms gijoms atvaizduoti,
- Matematinės išraiškos medžiai – matematinėms išraiškoms atvaizduoti,
- Loginės išraiškos medžiai – loginėms išraiškoms atvaizduoti.

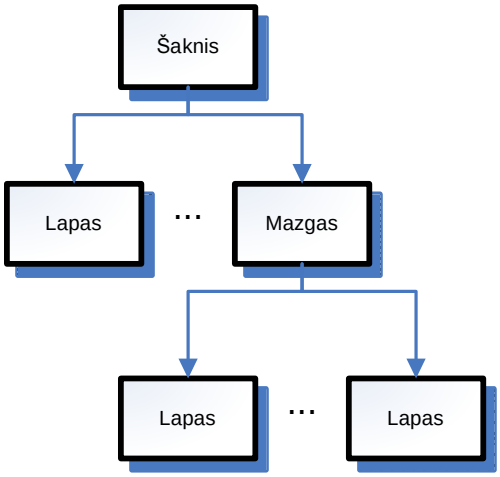
2.1.1. Programos medžiai

Programos medžiai skirti kuriamo roboto atskiriems fragmentams vaizduoti. Roboto konstravimas pradedamas pagrindiniame medyje „main()“, kuris yra privalomas ir gali būti tik vienas. Medis „main()“ – atitikmuo pagrindiniam programos metodui Java kalboje. Pagalbinių metodų ir gijų medžių kiekis įrankyje neribojamas.

Visų programos medžių konstravimo principas identiškas. Tačiau, priešingai nei medis „main()“, kitų medžių pavadinimus vartotojas parenka pats. Pagalbinių metodų medžiai gali turėti grįžtamąją reikšmę**.**

Programos medžio konstravimas

Medis susideda iš šaknies, mazgų ir lapų. Medžio šaknyje saugoma informacija, apibūdinanti medžio tipą, visi veiksmai realizuojami mazguose bei lapuose. Medžio lapuose saugomos komandos, kurių vykdymo seką aprašome mazgų pagalba.



8 pav. Programos medis

IF	**TRUE**	Vykdoma programos atšaka jei sąlygos sakinys teisingas.
	FALSE	Vykdoma programos atšaka jei sąlygos sakinys nėra teisingas.
Loop	**while**	Vykdomas ciklas „wile“, kol sąlygos sakinys teisingas (sąlyga tikrinama ciklo pradžioje).
	do	Vykdomas ciklas „do“, kol sąlygos sakinys teisingas (sąlyga tikrinama ciklo pabaigoje).
	for	Vykdomas ciklas „for“, kol sąlygos sakinys teisingas (sąlyga tikrinama ciklo pradžioje).

2 lentelė. Programos medžio mazgai

Declare	**var**	**boolean**	Paskelbiamas naujas boolean tipo kintamasis.
		char	Paskelbiamas naujas char tipo kintamasis.
		string	Paskelbiamas naujas string tipo kintamasis.
		byte	Paskelbiamas naujas byte tipo kintamasis.
		int	Paskelbiamas naujas int tipo kintamasis.
		long	Paskelbiamas naujas long tipo kintamasis.
		foal	Paskelbiamas naujas float tipo kintamasis.
		double	Paskelbiamas naujas double tipo kintamasis.
	array	Paskelbiamas ir konstruojamas masyvas.	
Set	**var**	Kintamajam priskiriama konkreti arba kito kintamojo reikšmė.	
	var-math	Kintamajam priskiriama matematinė išraiška (formuojama medžiu).	
	var-logic	Kintamajam priskiriama loginė išraiška (formuojama medžiu).	
	array	Masyvo elementui priskiriama konkreti arba kito kintamojo reikšmė.	
	array-math	Masyvo elementui priskiriama matematinė išraiška (formuojama medžiu).	
	array-logic	Masyvo elementui priskiriama loginė išraiška (formuojama medžiu).	
Import	Naujų klasių ar metodų įterpimas.		
New	Kuriamas naujas nurodytos klasės objektas.		
Execute	Pasirinktojo metodo paleidimas (metodas pasirenkamas iš sąrašo).		
Start	**class**	Paleidžiama pasirinktos klasės gija.	
	tree	Paleidžiama medžio „tree-Thread()“ gija.	

3 lentelė. Programos medžio lapai

Programos medžio konstravimo taisyklės

- Medis pradedamas konstruoti nuo šaknies („main()“ – jei tai yra pagrindinis programos medis), iš kairės į dešinę nuosekliai pridedant naujus mazgus ar lapus.
- Sąlygos sakinys „IF(condition)“ konstruoja sąlygos mazgą ir dvi vaikines atšakas „True“, „False“. Prie mazgo „True“ nuosekliai lipdomi kiti mazgai ar lapai, kurie bus vykdomi jei sąlyga „condition“ tenkinama. Prie mazgo „False“ lipdomi mazgai ar lapai, kurių vykdymas pageidaujamas esant netenkinamai sąlygai „condition“.
- Ciklas „Loop(condition)“ formuoja sąlygos mazgą, prie kurio nuosekliai jungiami kiti mazgai ar lapai. Visi mazgai bei lapai išeinantys iš „loop“ mazgo bus nuosekliai kartojami, kol sąlyga „condition“ bus tenkinama. Sąlygos tikrinimo vieta priklauso nuo ciklo tipo.

Programos kodo generavimas

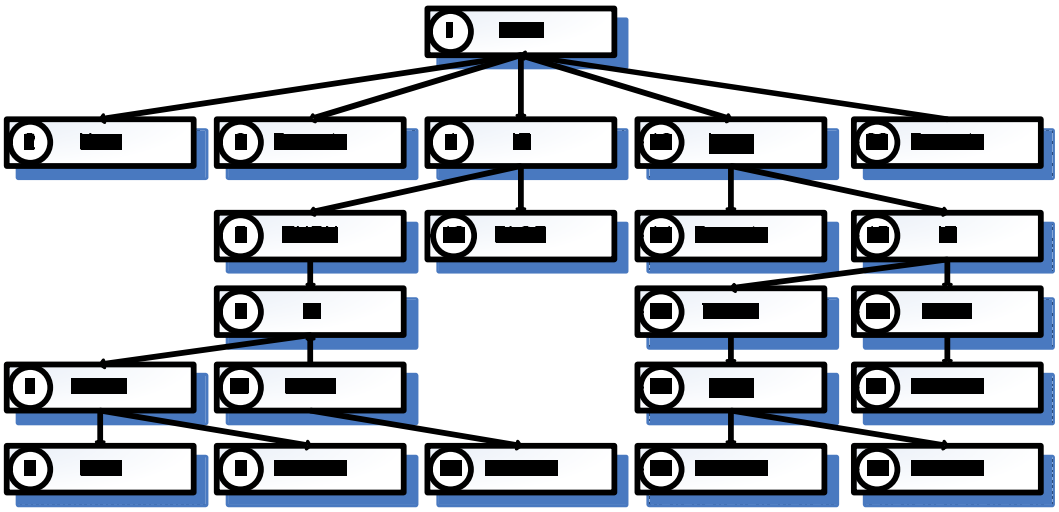
Įrankio pagalba sukonstruoti medžiai naudojami java kodui generuoti. Visų programos medžių peržiūros ir kodo konstravimo principai vienodi, skiriasi tik antraštės aprašas.

Programos kodo generavimas peržiūrint medį

- Medžio peržiūra pradedama nuo pagrindinės šaknies („main()“ – jei tai yra pagrindinis programos medis). Generuojama metodo antraštė ir atidaromi skliaustai.
- Nuosekliai peržvelgiamos visos šakos išeinančios iš šaknies ir generuojamos jas atitinkančios komandos.
- Sutikus sąlygos mazgą „IF“ generuojama sąlyga, peržvelgiamos abi vaikinės atšakos „True“ ir „False“.
  - Mazge „True“ atidaromi skliaustai, nuosekliai generuojamos visos komandos išeinančios iš šio mago. Peržiūrėjus visas komandas ir grįžus atgal į ši mazgą skliaustai uždaromi.
  - Mazge „False“ tikrinama ar mazgas turi vaikinių šakų, jei taip, atidaromi alternatyvaus pasirinkimo skliaustai („else {“) ir nuosekliai generuojamos visos komandos išeinančios iš šio mazgo. Sugeneravus visas komandas ir grįžus atgal į šį mazgą skliaustai uždaromi.
- Sutikus ciklo mazgą „Loop“ tikrinama ar mazgas turi vaikinių šaknų, jei taip, atidaromi skliaustai ir nuosekliai generuojamos visos komandos, atitinkančios

šakas, išeinančias iš šio mazgo. Sugeneravus visas komandas ir grįžus atgal į šį mazgą, skliaustai uždaromi. Priklausomai nuo ciklo tipo („While“ ar „Until“) sąlygos tikrinimo kodas generuojamas prieš atidarant skliaustus arba skliaustus uždarius.

- Baigus generuoti visas komandas išeinančias iš pagrindinės šaknies uždaromi metodo skliaustai.



9 pav. Pagrindinio metodo medis

Paveikslėlyje (žr. 9 pav.) pateiktas pagrindinio metodo „main()“ medis. Medžio peržiūros seka, generuojamas programos kodas, bei metodai, iškviečiami kodui generuoti, pateikiami lentelėje (žr. 3 lentelė.).

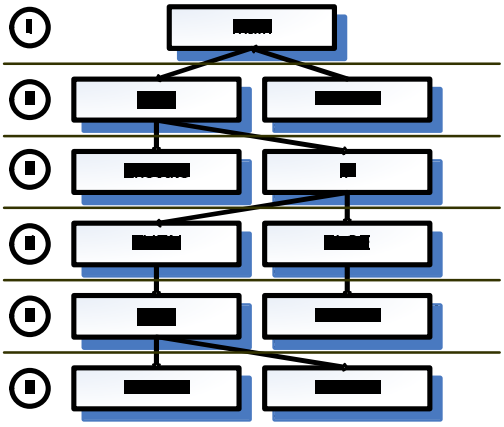
Objekto Nr.	Generuojamas kodas	Kviečiami metodai
1.	`main()`	`generateBeginString()`
1.	`{`	
2.	`New();`	`generateString();`
3.	`Execute();`	`generateString();`
4.	`if ( … )`	`generateIfBegintString();`
5.	`{`	`generateThenBeginString();`
6.	`if ( … )`	`generateIfBegintString();`

7.	`{`	`generateThenBeginString();`
8.	`New;`	`generateString();`
9.	`Execute();`	`generateString();`
11.	`}`	`generateElseBeginString();`
	`else`	
	`{`	
12.	`Execute();`	`generateString();`
6.	`}`	`generateIfEndString();`
4.	`}`	`generateIfEndString();`
13.	`while ( … )`	`generateLoopBegintString();`
	`{`	
14.	`Execute();`	`generateString();`
15.	`if ( … )`	`generateIfBegintString();`
16.	`{`	`generateThenBeginString();`
17.	`while ( … )`	`generateLoopBegintString();`
	`{`	
18.	`Execute();`	`generateString();`
19.	`Execute();`	`generateString();`
17.	`}`	`generateLoopEndString()`
20.	`}`	`generateElseBeginString();`
	`else`	
	`{`	
21.	`Execute();`	`generateString();`
15.	`}`	`generateIfEndString();`
13.	`}`	`generateLoopEndString();`
22.	`Execute();`	`generateString();`
1.	`}`	`generateEndString();`

4 lentelė. Kodo generavimas

Kultūringas kodo generavimas

Sugeneruotas programos kodas bus lengviau skaitomas ir suprantamas, jei laikysimės tam tikrų programavimo kultūros taisyklų. Vienas pagrindinių metodų programos skaitomumui pagerinti yra teisingas ir tvarkingas komandų lygiavimas, atitraukimas nuo kairiojo krašto. Darbe nagrinėjama programos medžio architektūra leidžia lengvai organizuoti tvarkingą kodo generavimą. Norint rasti atitraukimo nuo kairiojo krašto dydį, generuojant kodą pakanka žinoti, kokiame gylyje medžio lapas/mazgas yra.



10 pav. Medžio gylis

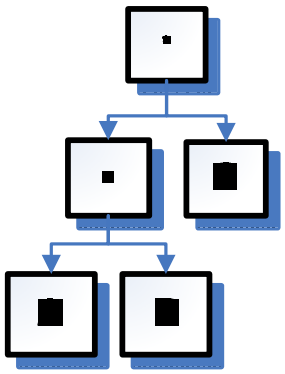
Pagal paveikslėlyje (žr. 10 pav.) pateiktą medį sugeneruotas programos fragmentas ir atitraukimų skaičius, kuris apskaičiuojamas tikrinant lapo/mazgo padėtį medyje, pateikiamas lentelėje (žr. 5 lentelė)**.**

Lapo/mazgo padėtis	Generuojamas programos kodas
1.	`public static void main() {`
2.	`    while (condition) {`
3.	`        Execute();`
3.	`        if (condition)`
4.	`            {`
5.	`                while (condition)`
5.	`                {`
6.	`                    Execute();`
6.	`                    Execute();`
5.	`                }`
4.	`            }`
	`            else`
	`            {`
5.	`                Execute();`
4.	`            }`
2.	`    }`
2.	`    Execute();`
1.	`}`

5 lentelė. Lygiuotas kodo generavimas

2.1.2. Matematinės išraiškos medžiai

Norint sumažinti galimų klaidų tikimybę, matematinėms išraiškoms įrankyje formuojami medžiai. Medžio lapuose saugomi duomenys, o mazguose bei šaknyje – elementariosios matematinės operacijos. Duomenimis gali būt laikomos konkrečios reikšmės, kintamieji arba metodai gražinantys rezultatą. Elementarios matematinės išraiškos (A + B) * C medis pavaizduotas (žr. 11 pav.) paveikslėlyje.



11 pav. Matematinės išraiškos medis

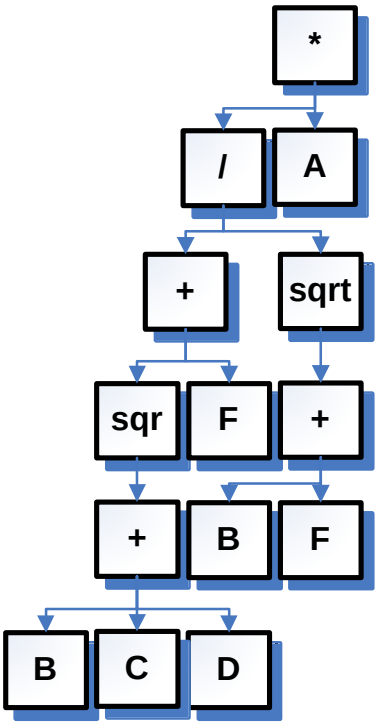
Kai kurios operacijos gali turėti daugiau nei vieną ar dvi atšakas, tai reiškia, jog ta pati operacija bus taikoma visoms iš jos išeinančioms atšakoms. Lentelėje (žr. 6 lentelė) pavaizduotos kelios galimos matematinės operacijos ir galimų atšakų kiekis.

Operacija	Operacijos pavadinimas	Atšakų kiekis
+	suma	neribotas
-	skirtumas	neribotas
*	sandauga	neribotas
/	dalyba	2
div	dalmuo	2
mod	liekana	2
abs	modulis	1
sqrt	kvadratinė šaknis	1
sqr	kėlimas kvadratu	1

6 lentelė. Elementariosios matematinės operacijos

Matematinės išraiškos medžiu galima generuoti ir dideles bei sudėtingas matematines išraiškas. Pateikiama matematinės formulės pavyzdį ir jos suformuotą medį (žr. 12 pav.).

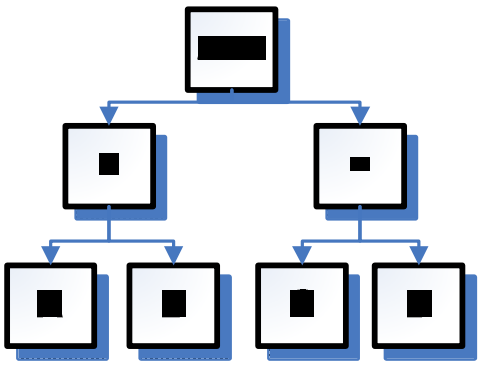
$$A \cdot \frac{(B+C+D)^2+F}{\sqrt{B+F}}$$



12 pav. Matematinės išraiškos medis

2.1.3. Loginės išraiškos medžiai

Kaip ir matematinėms išraiškoms, loginės išraiškos įrankyje formuojamos medžiu. Medžio lapuose saugomi duomenys, o mazguose bei šaknyje – elementariosios loginės operacijos. Duomenimis gali būt laikomos konkrečios reikšmės, kintamieji arba metodai, gražinantys rezultatą. Loginės išraiškos (A > B) && (C == D) medis pavaizduotas (žr. 13 pav.) paveikslėlyje.



13 pav. Loginės išraiškos medis

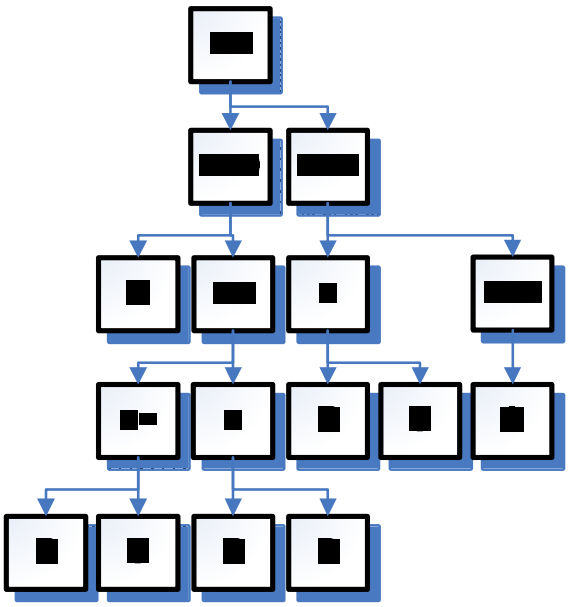
Kai kurios operacijos gali turėti daugiau nei vieną ar dvi atšakas, tai reiškia, jog ta pati loginė operacija bus taikoma visoms iš jos išeinančioms atšakoms. Lentelėje (žr. 7 lentelė) pavaizduotos kelios galimos loginės operacijos, bei galimų atšakų kiekis.

Operacija	**Operacijos atitikmuo Java kalboje**	**Atšakų kiekis**
>	>	2
<	<	2
>=	>=	2
<=	<=	2
=	= =	2
<>	!=	2
AND	&&	neribotas
OR	| |	neribotas
NOT	!	1

7 lentelė. Elementariosios loginės operacijos

Loginės išraiškos medžiu galima generuoti ir dideles bei sudėtingas logines išraiškas. Pateikiamas loginės išraiškos pavyzdys ir jos suformuotas medis (žr. 14 pav.).

(A && ((B >= C) | | (B < D))) | | ((B < C) && (!(A))



14 pav. Loginės išraiškos medis

2.2. Primityvūs duomenų tipai.

Primityviais (ne objektiniais) duomenų tipais vadiname duomenis, atstovaujančius vieną kintamąjį. Primityviems duomenų tipams priskiriam:

- boolen
- char
- byte
- chort
- int
- long
- float
- double

Įrankis suteikia galimybę visais šiais duomenų tipais operuoti. Tačiau kaip ir kiekvienoje programavimo kalboje egzistuoja tam tikros taisyklės. Prieš pradedant naudoti kurį nors kintamąjį, jis turi būt paskelbtas. Kintamąjį paskelbus, galimos operacijos: priskirti kintamajam konkrečią ar kito kintamojo reikšmę; priskirti kintamajam matematines išraiškas; naudoti kintamąjį kito kintamojo radimui; naudoti kintamąjį metodo iškvietimui.

Kintamojo deklaravimas

Kintamieji deklaruojami programos įrankio funkcija „Declare(var)“. Menių punktų, nurodžius, jog norima paskelbti kintamąjį, ir pateiktoje savybių formoje pasirinkus kintamojo pavadinimą, tipą, bei reikšmę (pasirinktinai), sukuriamas naujas medžio lapas „Declare(var)“.

Medžio lapą „Declare(var)“ atspindintis objektas turi tris parametrus:

- DataType – nusakantį kintamojo tipą,
- DataName – nusakantį kintamojo pavadinimą,
- DataValue – nusakantį kintamojo reikšmę.

Sukurtojo objekto metodas GenerateString() generuoja „string“ tipo eilutę. Šioje eilutėje laikomas sugeneruotas Java kodo fragmentas. Eilutės realizavimas atrodo taip:

```
str.add(DataType + “ ” + DataName);
if (DataValue != null) { str.append(“ ” + “=” + DataValue + “;”}
else {str.append(“;”)};
```

Reikšmės priskyrimas kintamajam

Jei kintamojo sukūrimo metu jam reikšmė nebuvo priskirta, tai galima padaryti komandos „Set(var)“ pagalba, leidžiančia elementui priskirti konkrečią reikšmę arba kintamąjį. Kuriamas naujas programos medžio lapas. Pasirinktinai kintamajam priskiriame arba konkrečią reikšmę arba kito kintamojo reikšmę.

Medžio lapą „Set(var)“ atspindintis objektas turi tris parametrus:

- DataName – nusakantį kintamojo pavadinimą,
- DataValue – nusakantį kintamojo reikšmę,
- SourceName – nusakantį kintamojo pavadinimą iš kurio bus paimta reikšmė.

Šiuo atveju *GenerateString()* metodo realizavimas atrodytu taip:

```
str.add(DataName);
if (SourceName != null) { str.append(“ ” + “=” + SourceName + “;”}
else {“ ” + “=” + DataValue + “;”}};
```

Matematinės išraiškos priskyrimas kintamajam

Matematinių operacijų seka kintamajam priskiriama komanda „Set(var-Math)“. Įrankio pagalba kuriamas matematinės išraiškos medis, kurio mazguose bei lapuose gali būt saugomi kintamieji, masyvo elementai, vartotojo susikurtos funkcijos ir visos elementariosios matematinės operacijos („+“, „-“, „*“, „/“, „sqrt()“ ir kt.).

Medžio lapą „Set(var-Math)“ atspindintis objektas turi du parametrus:

- DataName – nusakantį kintamojo pavadinimą,
- MathTreeName – nusakantį medžio pavadinimą.

GenerateString() metodo realizavimas atrodytu taip:

str.add(DataName + “ ” + “=” + getMathFormulaString(MathTreeName) + “;”);

Funkcija *getMathFormulaString(MathTree)* gražina *String* tipo eilutę, kurioje saugoma medžio pagalba sugeneruota funkcija.

2.3. Masyvai

Norint sukurti ir naudoti masyvą turime atlikti tris žingsnius:

- Deklaruoti masyvą,
- Konstruoti masyvą,
- Inicijuoti masyvą.

Vėliau, kaip ir su primityvių duomenų tipų kintamaisiais, su masyvo elementais galimos operacijos: priskirti masyvo elementui konkrečią ar kito kintamojo reikšmę; priskirti masyvo elementui matematinę išraišką; naudoti masyvo elementą kito kintamojo radimui; naudoti masyvo elementą metodo iškvietimui.

Masyvo deklaravimas ir konstravimas

Įrankyje paprastumo sumetimais masyvo deklaravimas ir konstravimas atliekamas vienu žingsniu, kurio metu kuriamas naujas medžio lapas „Declare(array)“. Įrankyje naudojami dviejų tipų masyvai: masyvas-eilutė (mas(n)); masyvas-matrica (mas(nxm)).

Medžio lapą „Declare(array)“ atitinkantis objektas turi parametrus:

- DataType – nusakantį masyvo tipą,
- DataName – nusakantį masyvo pavadinimą,
- Range1, Range2 – nusakančius masyvo gylį (jei masyvas vienmatis, Range2 = 0),
- Mas1[] – nusakančius vienmačio masyvo reikšmes,
- Mas2[][] – nusakančius masyvo reikšmes (dvimačio masyvo atveju).

Vienmačio masyvo atveju metodas GenerateString() generuoja eilutę:

str.add(DataType + “ [] ” + DataName + “ = new” + DataType + “[” + Range + “];”)

Reikšmių priskyrimas masyvo elementams

Pradines reikšmes masyvui priskirti galima tiktai po jo paskelbimo. Masyvo elementams reikšmes priskirti įmanoma trimis būdais:

- inicijuojant,
- priskiriant konkrečias reikšmes arba kintamuosius,
- priskiriant matematines išraiškas.

Masyvo inicijavimas atliekamas masyvo paskelbimo metu. Tam realizuoti naudojamas ciklas.

```
str.add(DataType + “ [] ” + DataName + “ = new” + DataType + “ ={” );
for (int i = 0; i < Range -1; i ++){ str.append(Mas1[i] + “, ”) };
str.append(Data[Range-1] + “};”);
```

Dviejų ciklų pagalba analogiškai atliekamas kodo generavimas ir dvimačio masyvo atveju.

```
str.add(DataType + " [] " + DataName + " = new" + DataType + " ={{" );
for (int i = 0; i < Range1 -1; i ++){
  for (int u = 0; u < Range2 -1; u ++){
    str.append(Mas2[i][u] + ", ") };
  str.append(Mas2[Range1-1][u] + "},");
str.append(Mas2[Range1-1][Range2-1] + "}};");
```

Jei masyvo sukūrimo metu, jam reikšmės nebuvo priskirtos, tai galima padaryti komandos „Set(array)“ pagalba, leidžiančia kiekvienam elementui atskirai priskirti konstantas arba kintamojo reikšmę.

Medžio lapą „Set(array)“ atspindintis objektas turi du parametrus:

- DataName – nusakantį masyvo pavadinimą,
- DataIndex – nusakantį masyvo elementą,
- DataValue – nusakantį masyvo elemento reikšmę,
- SourceName – nusakantį kintamojo pavadinimą iš kurio bus paimta reikšmė.

Sukurtojo objekto metodas GenerateString() generuoja „string“ tipo eilutę. Šioje eilutėje laikomas sugeneruotas Java kodo fragmentas. Eilutės realizavimas atrodo taip:

```
str.add(DataName +"[" + DataIndex + "]");
if (SourceName != null) { str.append(" " + "=" + SourceName + ";"}
else {" " + "=" + DataValue + ";"}};
```

Matematinių operacijų sekos priskyrimas masyvo elementui

Matematinių operacijų seka masyvo elementui, kaip ir paprastajam kintamajam, priskiriama komanda „Set(array-Math)“. Įrankio pagalba kuriamas matematinės formulės medis, kurio mazguose bei lapuose gali būt saugomi kintamieji, masyvo elementai, vartotojo susikurtos funkcijos ir visos elementariosios matematinės operacijos.

Medžio lapą „Set(array-Math)“ atspindintis objektas turi tris parametrus:

- DataName – nusakantį masyvo pavadinimą,
- DataIndex – nusakantį masyvo elementą,
- MathTreeName – nusakantį medžio pavadinimą.

GenerateString() metodo realizavimas atrodytu taip:

str.add(DataName + “[” + DataIndex + “] =” +
getMathFormulaString(MathTreeName) + “;”);

Funkcija *getMathFormulaString(MathTree)* gražina *String* tipo eilutę, kurioje saugoma medžio pagalba sugeneruota matematinė išraiška.

2.4. Klasės ir metodai

Šiame įrankyje yra įdiegta nemaža klasių bei metodų aibė, padėsianti vartotojui kurti internetinį robotą. Norimi metodai iškviečiami komandomis „Execute(command)“ – jei metodas neturi jokios grįžtančiosios reikšmės, arba „Return(command)“ – jei metodas-funkcija gražina reikšmę. Komanda „Execute(command)“ sukuriamas naujas medžio lapas.

Medžio lapą „Execute(command)“ atspindintis objektas turi parametrus:

- CommandName – nusakantį metodo pavadinimą,
- DataArray – nusakantį metodui paduodamų kintamųjų sąrašą,

GenerateString() metodo realizavimas atrodytu taip:

str.add(CommandName +“(”);
if (DataArray != null) { str.append(DataArray + “);”}
else {“);”}};

Komanda „Return(command)“ programos medyje naujų lapų ar mazgų nekuria. Ji naudojama matematinių išraiškų medžiams konstruoti.

Naujų klasių bei metodų diegimas

Įrankis visas klases bei metodus įdiegia automatiškai. Tačiau vartotojui suteikiama galimybė esamų klasių, metodų sąrašą papildyti naujomis. Tereikia patalpinti „*.*class*“ laikmeną į „*lib*“ katalogą ir įrankio pagalba jas įdiegti naudojamą programą.

Komanda „Import()“ – naudojama naujoms, vartotojo pateiktoms klasėms, metodams įdiegti.

Metodo konstravimas medžiu

Jei programos medis „main()“ konstravimo metu „užauga“ ganėtinai didelis ir sudėtingas, tikslinga tam tikriems programos fragmentams kurti atskirus medžius. Įrankis teigia galimybę kurti netik pagrindinį medį „main()“ pagrindiniam metodui aprašyti, bet ir kitus medžius, programos pagalbiniams metodams aprašyti. Pagalbiniai medžiai konstruojami tokiu pat principu kaip ir „main()“ medis, jam galioja tos pačios taisyklės. Skirtumas, jog pagalbinio medžio pavadinimą vartotojas pats gali pasirinkti.

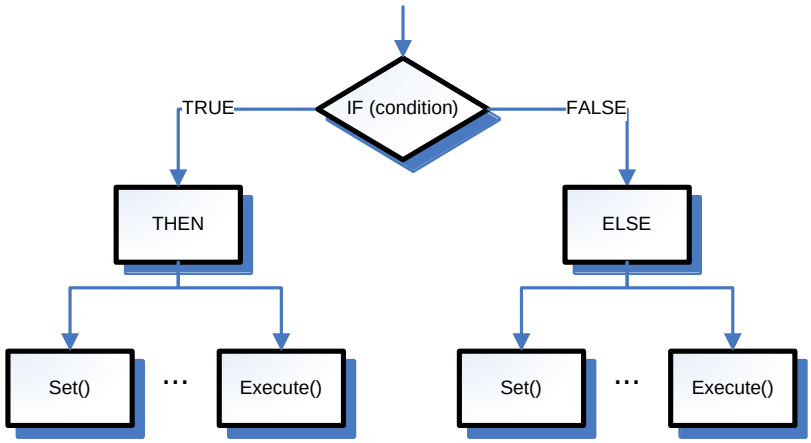
Sukonstravus pagalbinį medį, kaip ir bet kuris metodas, jis gali būt iškviestas naudojimui bet kurioje medžio „main()“ ar kito pagalbinio medžio vietoje. Jei pagalbinis medis turi grįžtamąją reikšmę, jis gali būt panaudotas kaip funkcija kintamųjų reikšmėms skaičiuoti arba matematiniam medžiui konstruoti.

Gijų vartojimas

Lygiagretiems procesams realizuoti galima naudoti gijas. Kuriamas naujas medis „tree-Thread()“, kuriame surašomos visos „run()“ metode atliekamos komandos. Medis konstruojamas tokiu pat principu kaip ir „main()“ medis. Pagal sukonstruotą medį įrankis generuos naują klasę. Vartotojas pagrindiniame ar pagalbiniuose medžiuose sukūręs šios klasės objektą, gali paleisti giją komandos „start()“ pagalba.

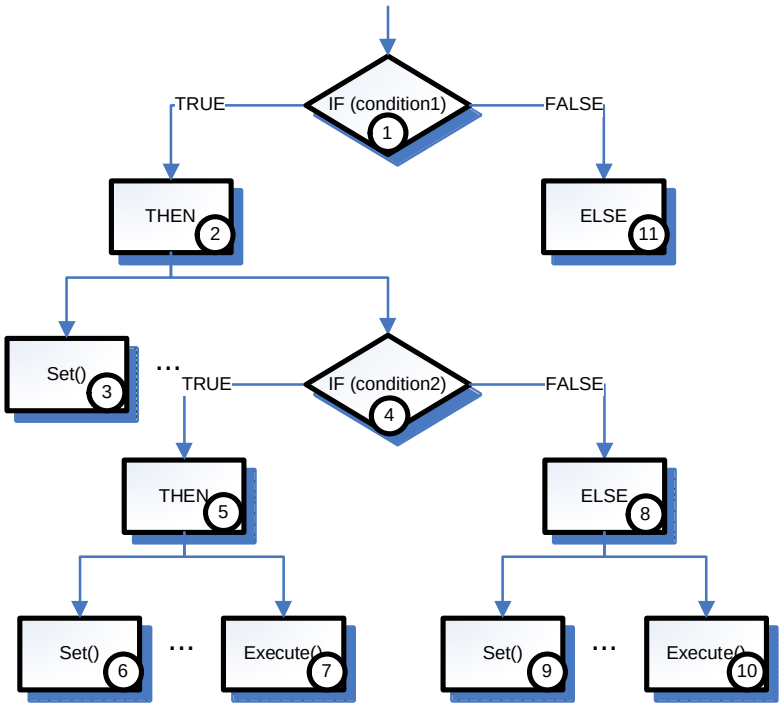
2.5. Sąlygos operatoriai

Įrankyje numatyta sąlygos operatorių panaudojimas. Sąlyga užduodama „IF()“ komanda. Programos medyje formuojamas naujas mazgas „IF(condition)“ ir du iš jo išeinantys mazgai „THEN“, „ELSE“. Sąlyga „condition“ konstruojama sąlygos medžiu.



15 pav. Sąlygos operatorius

Norimos vykdyti komandos esant teisingai sąlygai, nuosekliai išvesdinamos iš mazgo „THEN“. Jei pageidaujamas komandų vykdymas esant klaidingai sąlygai, komandos nuosekliai kuriamos po sąlygos „ELSE“ mazgu. Mazgas „ELSE“ pasirinktinai gali ir neturėti komandų.



16 pav. Medžio apėjimas

Lentelėje pateikiamas paveikslėlyje (žr. 16 pav.) pavaizduoto pomedžio sugeneruotas kodas, objektų numeriai ir metodai, kuriuos objektas iškviečia programos kodui generuoti.

Objekto Nr.	**Generuojamas kodas**	**Kviečiami metodai**
1.	*if (contidion1)*	generateIfBegintString(contition1);
2.	*{*	generateThenBeginString();
3.	*Set();*	generateString();
	...	...
4.	*if (condition2)*	generateIfBegintString(contition2);
5.	*{*	generateThenBeginString();
6.	*Set();*	generateString();
	...	...
7.	*Execute();*	generateString();
8.	*}*	generateElseBeginString();
	else	
	{	

9.	*Set();*	generateString();
	...	...
10.	*Execute();*	generateString();
4.	*}*	generateIfEndString();
1.	*}*	generateIfEndString();

8 lentelė. Kodo generavimas

Mazgą „IF“ atitinkantis objektas turi turi dvi klases.

- generateIfBeginString() – generuoja sąlygą,
- generateIfEndStrinf() – uždaro skliaustelius.

Mazgą „THEN“ atitinkantis objektas turi vieną klase:

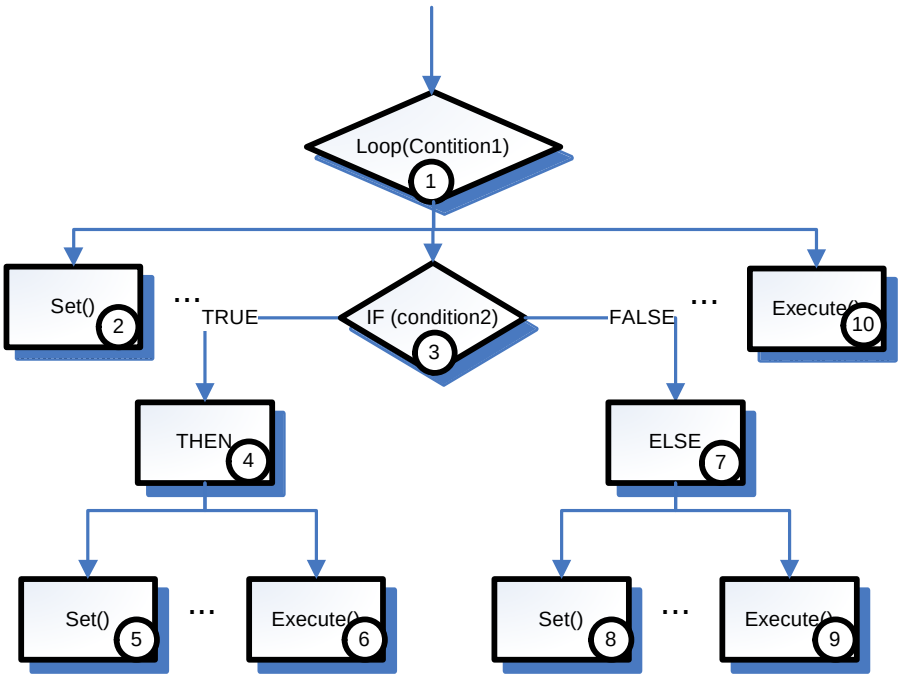
- generateThenBeginString() – atidaro „THEN“ mazgo skliaustelius,

Mazgą „ELSE“ atitinkantis objektas turi turi vieną klase:

- generateThenBeginString() – atidaro „ELSE“ mazgo skliaustelius,

2.6. Ciklai

Kaip ir natūralioje Java programavimo kalboje, šiame įrankyje realizuojama trijų tipų ciklų konstrukcijos: while(), do ir for(). Kol yra tenkinama tam tikra sąlyga, kiekvienas iš jų atkartoja pasirinktą komandą ar jų rinkinį. Ciklas sukuriamas komanda „Loop()“, kurios pagalba medyje sukuriamas naujas mazgas. Norimas atkartoti programos fragmentas išvesdinamas iš sukurtojo „Loop()“ mazgo.



17 pav. Ciklas

Lentelėje pateikiamas paveikslyje (žr. 17 pav.) pavaizduoto pomedžio sugeneruotas kodas, objektų numeriai ir metodai, kuriuos objektas iškviečia programos kodui generuoti.

Objekto Nr.	**Generuojamas kodas**	**Kviečiami metodai**
1.	*while (condition1)*	generateWhileBegintString(contition1);
	{	
2.	*Set();*	generateString();
	...	...
3.	*if (condition2)*	generateIfBegintString(contition2);
4.	*{*	generateThenBeginString();
5.	*Set();*	generateString();
	...	...
6.	*Execute();*	generateString();
7.	*}*	generateElseBeginString();
	else	
	{	

8.	*Set();*	generateString();
	...	...
9.	*Execute();*	generateString();
3.	*}*	generateIfEndString();
10.	*Execute();*	generateString();
1.	*}*	generateWhileEndString();

9 lentelė. Kodo generavimas

Mazgą „Loop(while)“ atitinkantis objektas turi dvi klases:

- generateWhileBeginString() – generuoja sąlygą, atidaro skliaustelius,
- generateWhileEndStrinf() – uždaro skliaustelius.

Ciklą realizuojantis mazgas „Loop(do)“ konstruojamas analogiškai, skiriasi tik sąlygos tikrinimo vieta. Sąlyga „condition“ konstruojama loginės išraiškos medžiu.

Ciklą realizuojantis mazgas „Loop(for)“ nuo ankstesniųjų ciklo tipų skiriasi tuo, jog ciklo viduje kuriamas naujas kintamasis. Be to vartotojui čia nereikia kurti sąlygos medžio, pakanka tik nurodyti apatinį bei viršutinį rėžius, žingsnio dydį ir kintamojo pavadinimą.

GenerateForBeginString() metodo realizavimas atrodytu taip:

str.add(“for (int ” + LoopVar + “ = ” + LoopLowValue + “; ” + LoopVar);
str.append(“ < ” LoopHiValue + “; ” + LoopVar + “+=” + LoopStep + “);”);

3. PROGRAMINĖS ĮRANGOS REALIZACIJA

3.1. Funkcionalumo reikalavimai

Produktas turi atlikti šias funkcijas:

- užkrauti medžius iš failo XML formatu
- išsaugoti medžius į failą XML formatu
- redaguoti medį:
    - sukurti/ištrinti medžio mazgą/lapą
    - redaguoti mazgo/lapo parametrus
    - atstatyti atliktus veiksmus (undo/redo)
- tikrinti medžio mazgų/lapų sintaksės klaidas
- įterpti naujas klases/metodus
- generuoti medį atitinkantį programos kodą
- pagalbos sistema

Ypatingas dėmesys turi būti kreipiamas į naudojimosi programa patogumą bei paprastumą.

3.2. Reikalavimai vartotojo sąsajai

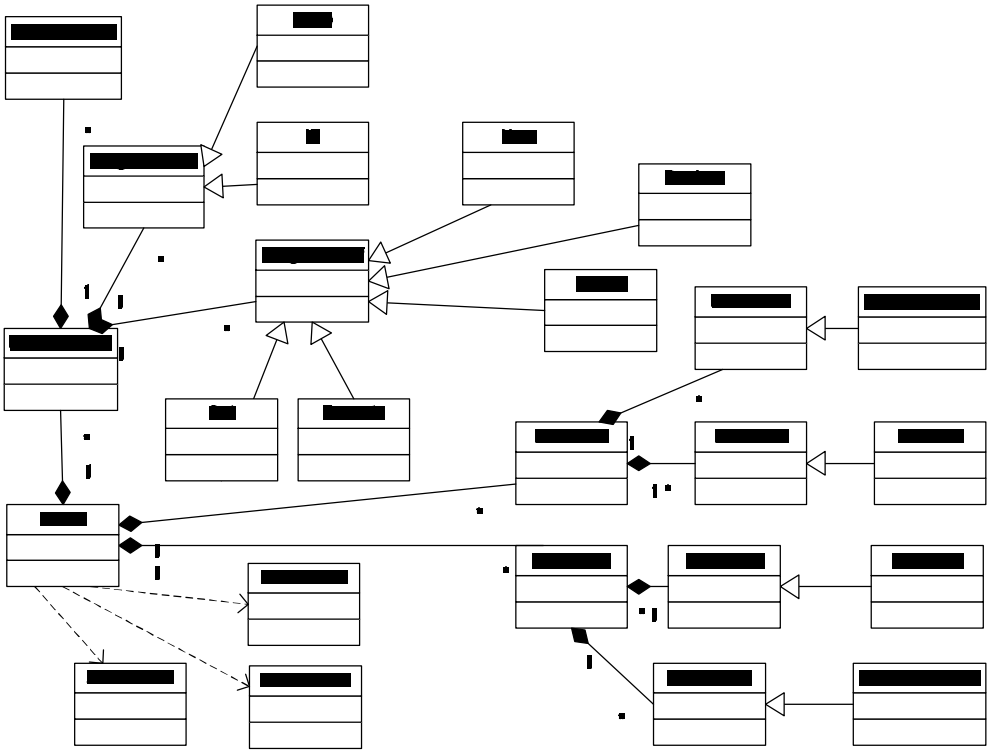
Vartotojo sąsaja turi būti grafinė, patogi, paprasta, lengvai suprantama taikomosios srities žinovui. Sąsaja privalo turėti veiksmų atstatymo galimybę (*undo/redo*). Spalvos turi būti ramios, netrukdančios darbui. Ryškios spalvos naudojamos tik išskirtiniais atvejais ar klaidų pažymėjimui. Redagavimas turi būti kuo patogesnis, panašus į kaladėlių dėliojimą. Grafiniai elementai turi automatiškai reaguoti į šalia esančių elementų išsidėstymo pasikeitimus ir atitinkamai pakeisti savo grafinę išvaizdą.

3.3. Projektavimo technologijos pasirinkimas

Internetinių robotų realizacijai pasirinkta Java programavimo kalba. Toks sprendimas padarytas dėl šių priežasčių:

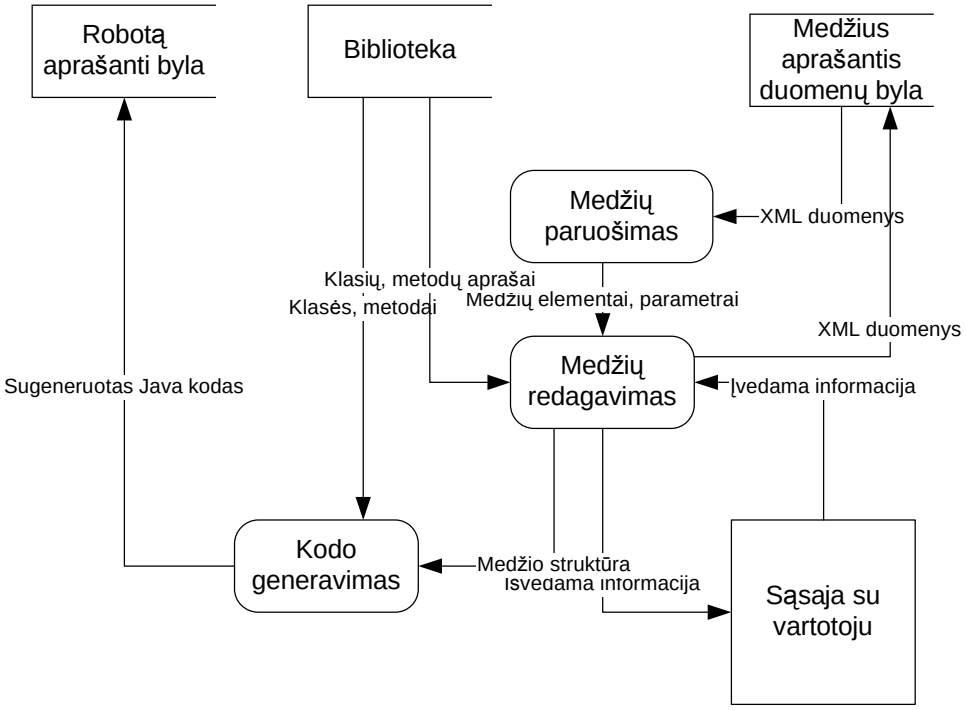
- tai labai lanksti kalba
- tai pilnai objektinė kalba
- kompiliatorius yra nemokamas
- sukompiliuotas projektas nepriklauso nuo platformos
- turi vienas geriausių priemonių vartotojo aplinkai kurti
- yra palaikoma daugelio CASE projektavimo įrankių
- kodas lengvai suprantamas ir modifikuojamas

3.4. Klasių diagrama



18 pav. Medžių redaktoriaus klasių diagrama

3.5. Duomenų srautų diagrama



19 pav. Programos duomenų srautų diagrama

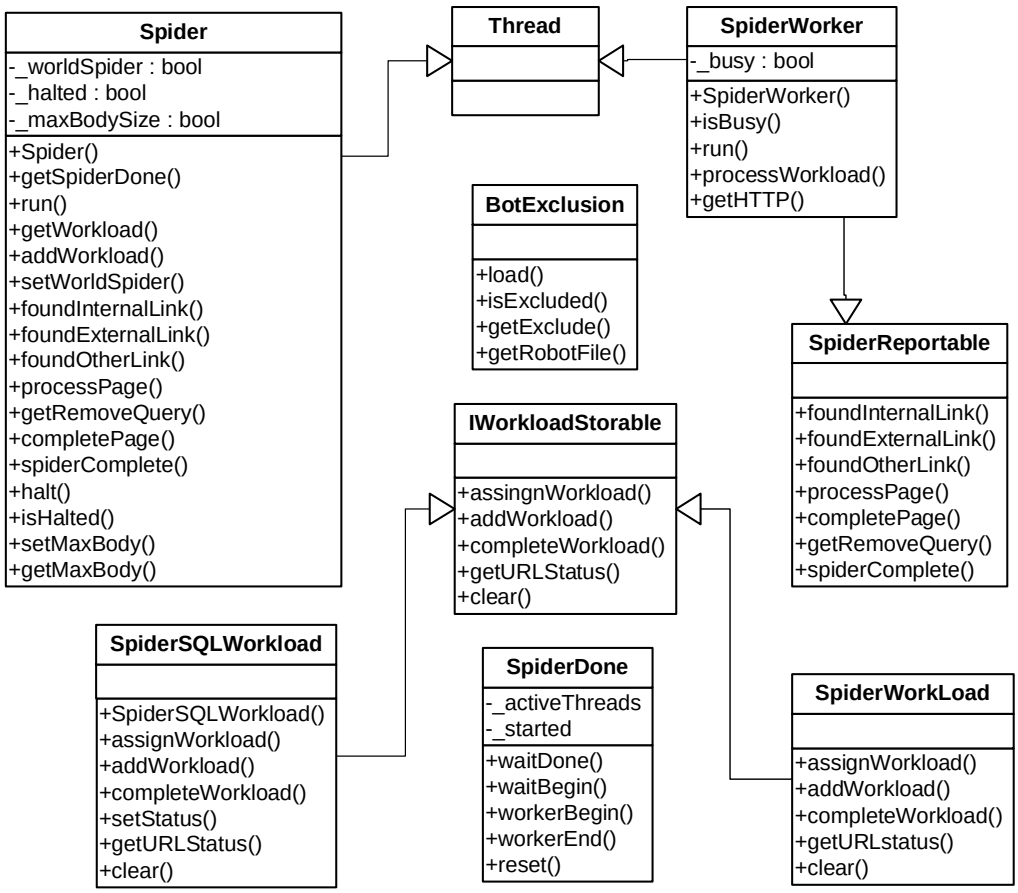
4. IŠVADOS

1. Intensyviai plečiantis pasaulinio žiniatinklio apimtims, didėja naršymo automatizavimo poreikis. Internete daugėja programinių robotų pavyzdžių. Nemažai literatūros, aprašančios jų veikimo principus ir kūrimo technologijas. Tačiau vis dar trūksta universalių įrankių, suteikiančių galimybes vartotojui lengvai konstruoti neapibrėžto tipo internetinius robotus.
2. Sukurtas programuojamas įrankis internetiniams robotams kurti. Įrankis nėra orientuotas į konkrečios problemos sprendimo metodika, siūlomas platus robotų pasirinkimo spektras.
3. Įrankyje programos fragmentai atvaizduojami medžiais. Grafine sąsaja medžiai yra lengvai konstruojami bei redaguojami. Vartotojui nebūtina turėti gerų programavimo įgūdžių, pakanka būt susipažinusiam su algoritmavimo bei medžio konstravimo technologijom.
4. Pasirinkta projektavimo architektūra leidžia lengvai išplėst generatoriaus galimybes. Poreikiui esant, įrankio vartojama biblioteka gali būti papildyta, praplėstas robotų šablonų pasirinkimas.
5. Pasirinkta Java programavimo kalba puikiai tinka internetinių robotų realizavimui. Generuojamas robotas yra nepriklausomas nuo operacinės sistemos, veikia visose platformose palaikančiose JAVA™ programinę įrangą.

1 PRIEDAS. Biblioteka

Voro klasės

Voro klasės skirtos robotui, naršančiam nuo vieno puslapio i kitą, sukurti.



20 pav. Voro klasės

BotExclusion klasė

Ši klasė naudojama perskaityti ir išnagrinėti internetinės svetainės robot.txt bylą. Robotas naudojantis šią klasę paklūsta viešai elgsenų bylai robot.txt, kurioje svetainės administratoriaus nurodyti, leidžiami nagrinėjimui katalogai ir kt.

Spider klasė
Pagrindinė klasė valdanti robotą, paleidžianti SpiderWorkerk klasę.

SpiderDone klasė
Ši klasė naudojama nurodyti robotui, kada baigti darbą.

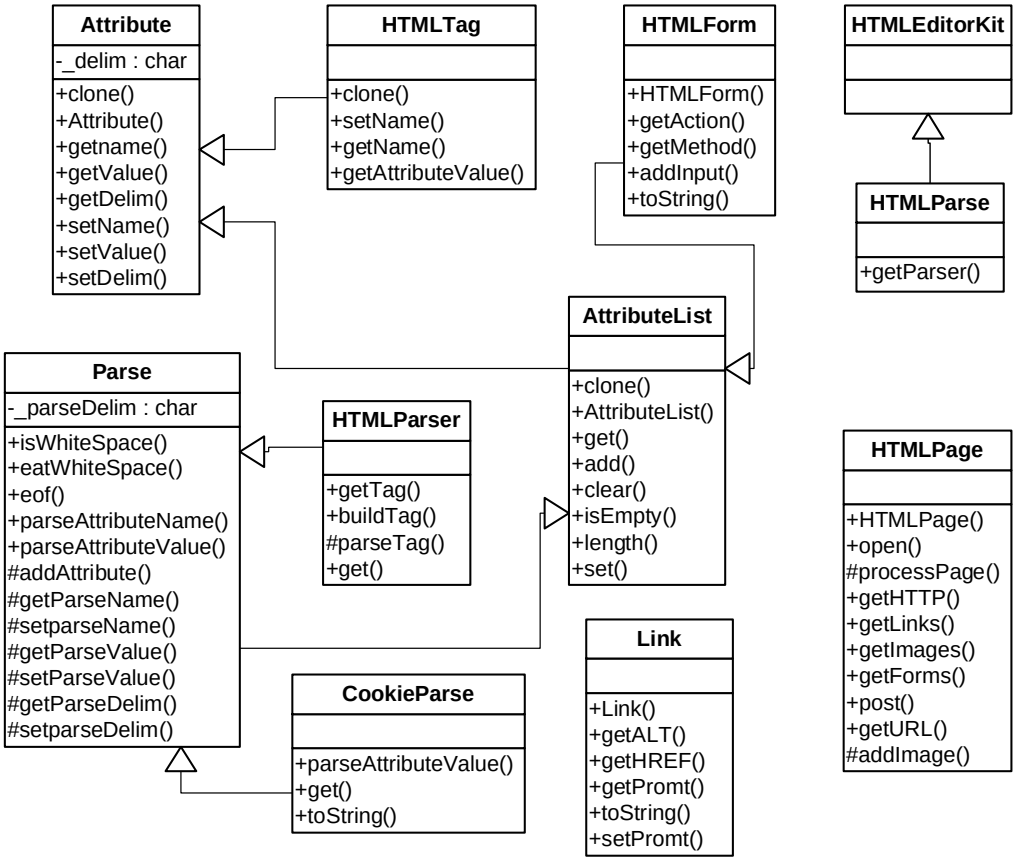
SpiderInternalWorkload klasė
Klasė skirta kurti bei saugoti nuorodų eiles atmintyje. Jeigu nėra nurodytas joks kitas metodas eilėms saugoti, naudojamas šis metodas.

SpiderSQLWorkload klasė
Klasė skirta kurti bei saugoti nuorodų eiles duomenų bazėje.

SpiderWorker klasė
Klasė koordinuojanti voro naršymą puslapiais eiga. Veikia kaip atskira Spider klasės sukurta gija.

Tyrinėjimo klasės

Šios klasės naudojamos nagrinėti įvairius duomenis. Skirstomi į klases skirtas nagrinėti „*sausainius*“ ir HTML.



21 pav. Tyrinėjimo klasės

Attribute klasė
Ši klasė saugo sąrašą įvardintų kintamųjų.

AttributeList klasė
Skirta saugoti „Attribute“ klasių sąrašui.

HTMLForm klasė

Ši klasė skirta atsakymui iš HTML formos generuoti bei perduoti ją žiniatinklio tarnybinei stočiai.

HTMLPage klasė
Klasė naudojama nagrinėti HTML puslapį bei įterpti jį išnagrinėtųjų formų sąrašą.

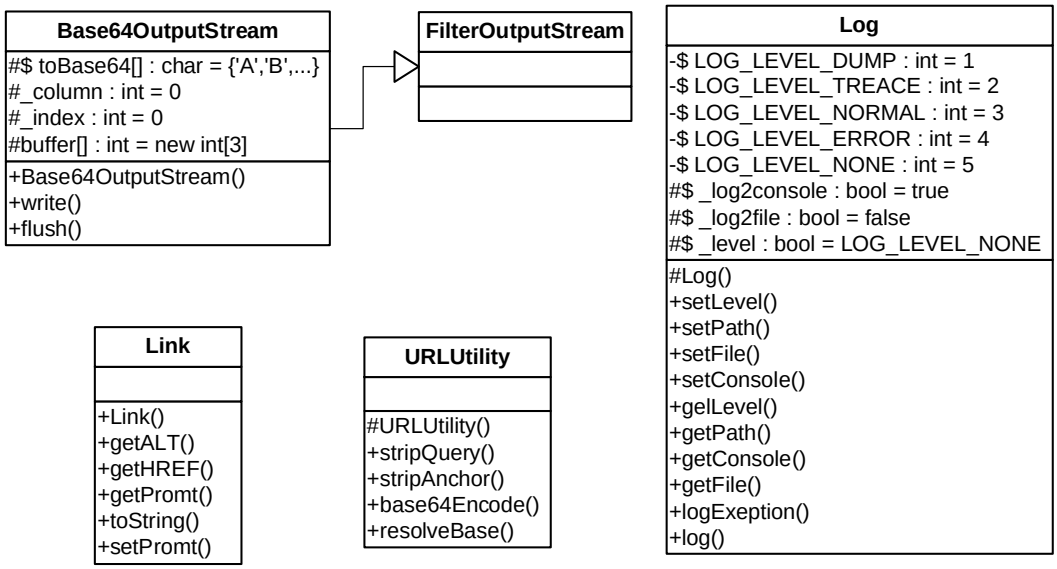
HTMLParser klasė
Ši klasė skirta nagrinėti HTML puslapį. Ši paprastutė klasė jokių duomenų savyje nesaugo, naudojama kaip pagalbininkė kitoms HTML naršymo metodams.

HTMLTag klasė
Klasė naudojama saugoti HTML žymes, įskaitant pavadinimus bei savybes.

Parse klasė
Naudojama žemo lygio tekstui nagrinėti, visų kitų nagrinėjimo klasių pagrindas.

Pagalbinės klasės

Tai klasės, kurias pakankamai sunku priskirti kuriai nors kategorijai. Robotas naudoja šias klases kaip pagalbininkes kitoms klasėms.



22 pav. Pagalbinės klasės

Base64OutputStream klasė
Šis filtras naudojamas 64-bitų kodavimui. Klasė leidžia *string* tipo eilute atvaizduoti ASCII simboliais, taipgi naudojama HTTP autorizacijai suteikti.

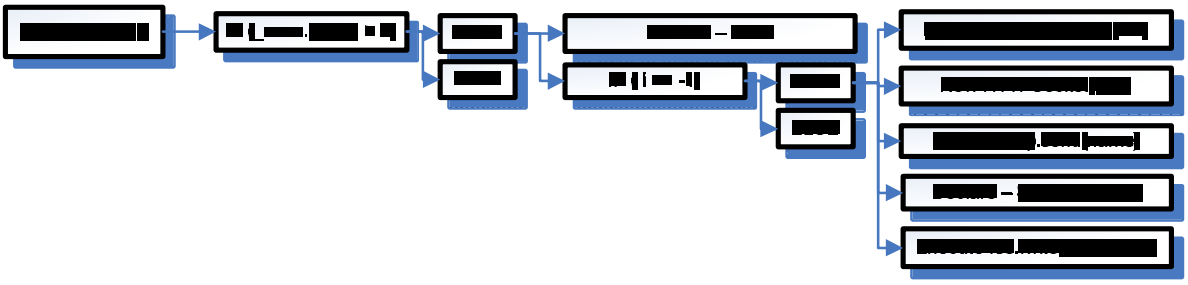
Link klasė
Šioje klasėje laikomos HTML nuorodos. Laiko tikrąsias URL nuorodas ir ALT žymes.

Log klasė
Klasė skirta jungimosi informacijai išvesti.

2 PRIEDAS. Robotas „GetImage“

Pateikiame elementaraus roboto „GetImage“ pavyzdį, kuris pagal duotą URL surenka iš puslapio paveikslėlius ir išsaugo juos kietame diske.

23 pav. Roboto „GetImage“ pagrindinis main() medis



24 pav. Roboto "GetImage" pagalbinis "processImage" medis

Sugeneruotas kodas Java kalboje :

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.io.*;

import bot.*;


public class GetImage extends javax.swing.JFrame
{

	public GetImage()
```

```
        {
                ...
        }

        javax.swing.JTextField _url = new javax.swing.JTextField();
        javax.swing.JTextField _save = new javax.swing.JTextField();
        javax.swing.JList _log = new javax.swing.JList();


protected void processImage(String name)
        {
                if ( _save.getText().length()>0 )
                {
                        int i = name.lastIndexOf('/');
                        if ( i!=-1 )
                        {
                                FileOutputStream fso
                                        = new FileOutputStream(
                                        new File(_save.getText(),name.substring(i)) );
                                HTTPSocket http = new HTTPSocket();
                                http.send(name,null);
                                fso.write( http.getBodyBytes() );
                                fso.close();
                        }
                }
        }


static public void main(String args[])
  {
      HTTPSocket http = new HTTPSocket();
      HTMLPage page = new HTMLPage(http);
      page.open(_url.getText(),null);

      Vector vec = page.getImages();
      if ( vec.size()>0 ) {
        String array[] = new String[vec.size()];
        vec.copyInto(array);
        _log.setListData( array );
        for ( int i=0;i < vec.size();i++ )
          processImage((String)vec.elementAt(i));
      }
  }
}
```

Santrumpų žodynas

DB – Duomenų bazė.
DBVS – Duomenų bazių valdymo sistema.
HTML – (Hyper Text Markup Language) hipertekstinė kalba.
SQL – (Structurer Query Language) struktūrinė užklausų kalba.
XML – (Extended Markup Language) išplėsta žymėjimų kalba.
URL – (Universal Resource Locators) universalios resursų žymės.
WWW – (Worl Wide Web) pasaulinis žiniatinklis.

LITERATŪRA

[1] Cheong, F. Internet Agents: Spiders, Wanderers, Brokers, and Bots. Indianapolis: New Riders, 1996. 337 p. ISBN 0-7356-1588-8.

[2] Heaton, J. Programming Spider, Bots, and Aggregators in Java. Marina Village Parkway Alameda: Sybex, 2002. 512p. ISBN 0-7645-7286-5.

[3] Herrington, J. Code Generation in Action. Indianapolis: Manning Publications, 2003. 372p. ISBN 0-7385-6384-8.

[4] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the World-Wide Web. Englewood Cliffs: Yourdon Press, 1997. 315p. ISBN 0-321-25727-8.

[5] Mohammadian, M. Intelligent Agents for Data Mining and Information Retrieval. Idea Group Publishing, 2004. 512p. ISBN 0-354-25641-5

[6] Alexandrescu, A.. Modern C++ Design: Generic Programming and Design Patterns Applied. Indianapolis: Addison-Wesley, 2001. 456p. ISBN 0-336-26584-3.

[7] Bravenboer M. Meta Programming with Concrete Object Syntax. Marina Village Parkway Alameda, CA, Sybex, 1996.

[8] The Java Tutorial. [žiūrėta 2005-05-01]. Prieiga per Internetą: <http://java.sun.com/docs/books/tutorial/>