Зборник радова Факултета техничких наука, Нови Сад

УДК: 004.9
DOI: https://doi.org/10.24867/05BE15Djuric

ИМПЛЕМЕНТАЦИЈА И ВИЗУАЛИЗАЦИЈА ГРАФА ТОКА КОНТРОЛЕ НА ЈЕЗИКУ PHARO

IMPLEMENTATION AND VISUALIZATION OF THE CONTROL FLOW GRAPH USING THE PHARO PROGRAMMING LANGUAGE

Светлана Ђурић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *У раду је пројектована хијерархија класа за моделовање графа контрола тока. Граф се преузима из синтаксног стабла произвољног Pharo програма и визуализује коришћењем Mondrian радног оквира.*
Кључне речи: *Граф тока контроле, апстрактно синтаксно стабло, визуализација, Pharo*
Abstract – *The paper presents a class hierarchy for control flow graph modeling. Control flow graph is created based on abstract syntax tree of a Pharo program. It is visualized using Mondrian framework.*
Кључне речи: *control flow graph, abstract syntax tree, visualization, Pharo*

1. УВОД

Анализа кода може бити корисна приликом креирања и одржавања софтверских система. Анализом кода се могу изоловати графови контроле тока који представљају редослед у коме се извршавају инструкције програма. Графови контроле тока описују начине на које се контрола тока преноси кроз програм [15]. Графовима контроле тока би се могло побољшати отклањање грешака у коду. Визуализација онога што је имплементирано може знатно убрзати рад и може се корак по корак пратити шта се дешава. Графови контроле тока такође би се могли искористити и приликом утврђивања да ли ток једног програма изгледа као ток другог програма, што омогућава да се лакше препозна да ли је дошло до копирања кода.

Циљ овог рада је да се омогући издвајање и визуализациаја графа тока контроле из програмског кода коришћењем програмског језика *Pharo*.

2. ТЕОРИЈСКЕ ОСНОВЕ

У овој секцији је приказан *Pharo* језик као основа за имплементацију решења и описани су појмови као што је: статичка анализа кода, апстрактна синтаксна стабла, графови контроле тока и њихова намена. Описан је шаблон који је коришћен приликом имплеметнације овог решења, који се назива *Visitor*. Радни оквир који је послужио за визуализацију назива се *Mondrian* и појашњено је како се ради са њим.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је била проф. др Гордана Милосављевић.

2.1 О Pharo језику

Pharo је савремени објектно-оријентисани језик отвореног кода. Погодан је за визуализацију, моделовање података, умрежавање, као и остале намене због бројних објектно-оријентисаних библиотека и радних оквира. Синтакса је једноставна и читљива (стаје на једну разгледницу) а објектни модел је лако прилагодљив, што одговара и почетницима и искусним програмерима.

2.2 Статичка анализа програма

Да би софтвер био квалитетан, добрих перформанси и без грешака, потребно је доста труда и времена уложити у његово пројектовање и развој. Како би се тај поступак олакшао, користи се анализа софтвера како би се открили грешке и мане софтвера још у раним фазама. Постоје два начина за анализу софтвера: статички и динамички.

Статичка анализа софтвера се базира на анализи изворног кода. Посматрају се апстрактна и конкретна синтаксна стабла и графови контроле тока.

Динамичка анализа софтвера темељи се на подацима који се добијају у време извршавања софтвера као и интеракцијом са софтвером. Акценат овог рада је на статичкој анализи.

Програм се најбоље анализира посматрајући међукод, који је независан од коришћеног програмског језика. У овом раду биће дискутоване две форме међукода: апстрактно синтаксно стабло (AST) и граф тока контроле (ЦФГ). AST је повезано са изворним кодом, док је ЦФГ потпуно независан од језика, али се не може помоћу њега извести код. AST има богатији приказ у односу на ЦФГ.

Граф контроле тока се изводи из апстрактног синтаскног стабла и представља приказ изворног кода. Језички је независан, што значи да може да се изведе за било који језик. Граф контроле тока у себи садржи чворове који су повезани линијама које су усмерене од претходног ка следећем. Да би се граф тока контроле могао брже и интуитивније растумачити, добра је пракса да се изнад или испод сваког чвора уведе његово име.

2.3 Visitor шаблон

Шаблон *Visitor* служи да одвоји алгоритам од структуре података над којом се примењује као и да,

код сложених структура података, одреди место за имплементацију операције, која ће се извршити над елементима те структуре. Пример за такву сложену структуру може бити стабло са својим чворовима.

2.4 Mondrian

Mondrian представља језик специфичан за домен који служи за визуализацију разних врста података у облику графа. *Mondrian* омогућава интеракцију са подацима који треба да се визуализују.

3. ИМПЛЕМЕНТАЦИЈА РЕШЕЊА

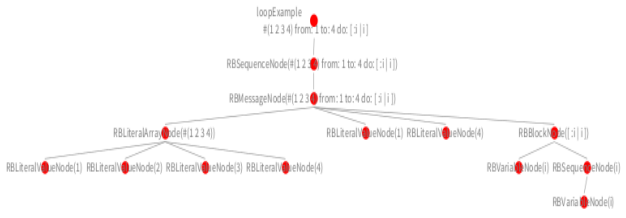
Овај рад је проширење пројекта под називом *OAK* [16]. Постојећи код за обраду графа тока контроле је проширен у оквиру овог рада и урађена је визуализација коришћењем програмског оквира *Mondrian*.

Основни принципи на којима се заснива имплементација решења јесу:

- обрађене су методе, блокови, петље, услови и једноставне структуре
- метода има своје тело
- тело може да има једну или више порука, а може да буде и празно
- блок може имати један или више аргумената; блок има своје тело; може се појавити било где.

3.1. Приказ апстрактног синтаксног стабла

Пре визуализације графа тока контроле одрађена је визуализација апстрактног синтаксног стабла. Подршка за апстрактно синтаксно стабло постоји у *Pharo* језику. Апстрактном синтаксном стаблу приступа се помоћу методе *getSource*. Прикупљају се чворови из апстрактног синтаксног стабла. Затим се апстрактном стаблу додељује посетилац. Креира се поглед и у њега се додају чворови и њихове везе. Чворови су деца из AST-а, а грана иде од родитеља ка деци, што је приказано на слици 1.



Слика 1 – Приказ AST за петљу

3.2. Опис имплементираног решења

У овој секцији је приказан дијаграм класа за моделовање графа контроле тока. Дат је опис класа.

3.2.1. Дијаграм класа за моделовање графа контроле тока

На слици 2 приказан је дијаграм класа којима се моделује граф контроле тока. Класе ће детаљније бити појашњене у наставку ове секције. Посматрајући аспстрактно синтаксно стабло, лакше је израдити граф контроле тока. Са слике 2 види се да постоје два типа чворова:

- *Childish* - сложени чворови
- *Non - Childish* - прости чворови.

Сложени чворови наслеђују класу *GrdChildishNode*. Предвиђено је да постоје две врсте веза:

- претходни - следећи
- родитељ - дете

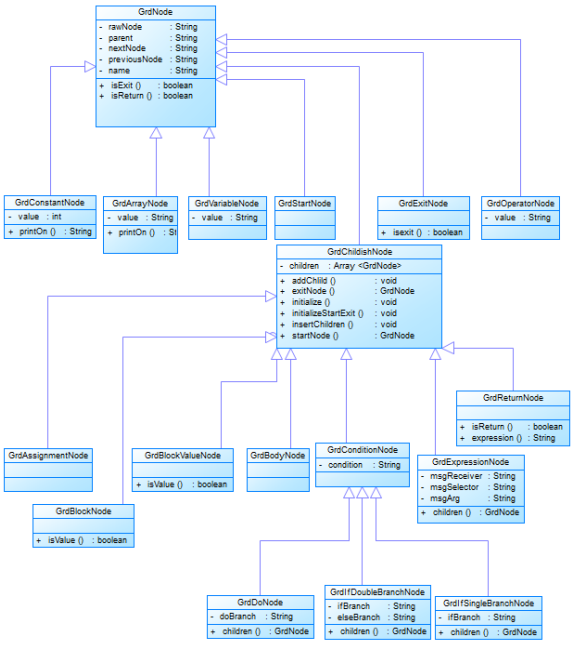
На слици 2 приказана је класа *GrdNode*, која представља чвор. Она служи за то да сваки чвор буде повезан са својим претходником и следбеником, ако их има.

Граф контроле тока почиње чворовима *GrdStartNode* и *GrdExitNode*. Они представљају улаз и излаз из графа. Ови чворови се такође користе и у оквиру сложених чворова како би било прегледније шта се дешава.

Класа *GrdChildishNode* представља сложени чвор, који у себи садржи и остале чворове. Она у себи има листу деце. У овој класи поред везе претходник - следбеник, јавља се веза родитељ - дете. Уколико је неки чвор сложен, унутар њега се приказују и његова деца и тако рекурзивно све док чвор који се посматра има деце у себи. У сложене чворове спадају чворови који представљају блокове, тело блокова, услове, чворове доделе, чворови који представљају петље, изразе и повратну вредност.

Класа *GrdBlockNode* представља блок. Класа која представља блок је наследница класе *GrdChildishNode*. Ако променљивама у блоку треба доделити вредност користи се порука *value:*. Тај случај је потребно посебно обрадити и за то служи чвор кога представља класа *GrdBlockValueNode*.

Тело блока (*GrdBodyNode*), се може налазити у оквиру блока, а може и посебно, ван других чворова. Цео граф контроле тока представља чвор који је типа тело (*GrdBodyNode*). Класа која га представља је наследница класе *GrdChildishNode*.



Слика 2 - Приказ дијаграма класа за ЦФГ

Што се тиче услова, направљена је класа која представља апстракцију, *GrdConditionNode*. Она у себи садржи услов. Њене наследнице су конкретне класе за гранање. Прва је *GrdIfSingleBranchNode*, која у себи садржи грану која представља *if* грану.

Уколико се ради о чвору који има више грана, *if* и *else*, он се представља класом *GrdIfDoubleBranchNode*. Слично као код горе поменуте класе, наслеђује *GrdConditionNode*.

Петља је представљена помоћу класе *GrdDoNode*. Класа која представља обраду низова је *GrdArrayNode*. За доделу се користи класа *GrdAssignmentNode*. Чвор који је повратна вредност неке методе је представљен помоћу *GrdOperatorNode* класе, константе помоћу *GrdConstantNode*, а променљиве помоћу *GrdVariableNode*

Уколико се ради о неком изразу, за то служи класа *GrdExpressionNode*. Класа *GrdExpressionNode* служи за обраду израза који у себи садрже бинарне операторе (*GrdOperatorNode*).

3.2.2. Посета чворовима

У овом решењу коришћен је шаблон *Visitor* који је објашњен у секцији број 2.3. У иницијализацији класе *GrdVisitor* прави се нови објекат *GrdBodyNode*, у коме ће бити смештени сви остали чворови. За сваки тип чвора направљена је посебна метода, која служи за посету истог.

У овом решењу је граф контроле тока направљен само на нивоу једног метода, где се поруке гледају само као искази. Проблем који је остао нерешен јесте повезивање позива метода, али то спада у проблеме динамичког везивања. Тај проблем би се могао решити на статичком нивоу помоћу техника као што је *type inference*. Ова техника анализира типове променљивих или израза. То значи да компајлер сам одређује тип израза, функција или вредности.

3.2.3. Визуализација графа контроле тока

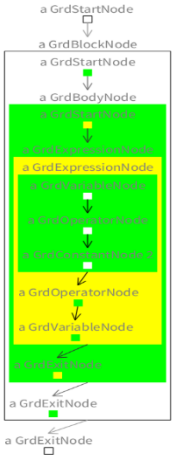
За визуализацију коришћена је верзија *Pharo*-а *Moose Suite 6.0* јер се у њој налази све потребно за рад са *Roassal*-ом и *Mondrian*-ом. Алгоритам који се користи за обилазак графа јесте „први у дубину“. То значи да се креће од корена и да се иде дуж сваке гране докле год је то могуће.

4. ПРИКАЗ ИМПЛЕМЕНТИРАНОГ РЕШЕЊА

У овој секцији биће приказане слике екрана на којима се виде визуализовани графови тока контроле за примере са блоковима и условима са двоструким гранама. Слика екрана за пример блока : *[:a :b | a = 2 + b]* дата је на слици 3.

На слици 3 се види граф контроле тока који се састоји од почетног чвора, чвора за блок и крајњег чвора. Чвор који представља блок има чворове приказане на слици 3. Деца од сваког чвора су увучена у односу на њега.

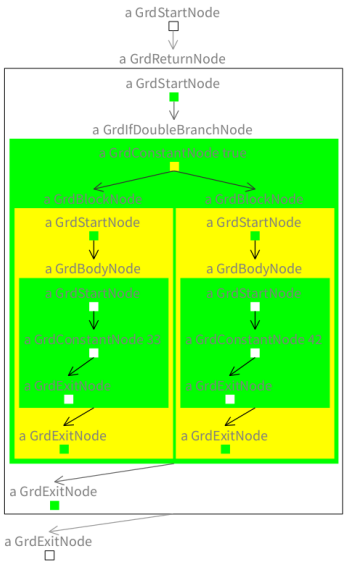
1) почетни чвор
2) тело
    a. почетни чвор
    b. први израз
        - другi израз
            - променљива a,
            - оператор =,
            - константа 2
        - оператор +
        - променљива b
    c. крајњи чвор
3) крајњи чвор



Слика 3 – Приказ графа контроле тока за блок

На слици 4 приказана је визуализација примера:

^ *true ifTrue: [33] ifFalse: [42]* .



Слика 4 – Приказ графа са двоструком граном

Граф са слике 4 састоји се од почетног чвора, чвора за повратну вредност и крајњег чвора. Сва три чвора су означена белом бојом. Чвор који представља повратну вредност је сложен, и он се састоји од почетног, чвора са двоструком граном и крајњег чвора. Сва деца су му зелене боје. Са слике 4 види се да је чвор са двоструком граном сложен чвор и да се у њему налазе:

1) константа која представља услов (true)
2) блок који се извршава ако је услов тачан
    a. почетни чвор
    b. тело
        - почетни чвор
        - константа 33

- крајњи чвор

c. крајњи чвор

3) блок који се извршава ако је услов нетачан

a. почетни чвор

b. тело

- почетни чвор
- константа 42
- крајњи чвор

c. крајњи чвор

5. ЗАКЉУЧАК

У оквиру пројекта који је послужио као основа за овај рад одрађена је имплементација и визуализација графа контроле тока на програмском језику Pharo. Граф контроле тока је имплементиран и визуализован за апликације које су написане на програмском језику Pharo. Направљене су класе које представљају чворове који ће се приказивати у графу контроле тока. Написане су методе које служе за посету одговарајућим чворовима.

За просте чворове су направљене класе које описују низове, константе, променљиве и операторе који се користе у аритметичким операцијама. За сложене чворове су направљене класе које означавају доделу вредности, блокове, тело блокова, условне чворове, петље и чворове који представљају изразе. За обраду блокова је направљена додатна класа која служи за блокове са поруком *value:*. Поруком *value*: се додељује вредност променљивама које се користе у блоковима. За чворове који означавају услове су направљене класе за обраду чворова са једноструком и двоструком граном. Поруке које су обрађене су једноставне поруке и каскадне поруке.

Даљи развој подразумева рад на петљама попут: *to:do*, *to:do:by*, *timesRepeat:*. Још један вид проширења би била обрада порука попут: *ifNone:*, *ifAbsent:*, *detectIfNone:*, *at:ifAbsent:*.

Код метода се може одрадити апстракција. То значи да се направи посетилац за методу генерално и да се та апстракција користи за сваки пример. Тако би се избегла појединачна обрада за конректне поруке. Што се тиче визуализације, било би добро увести различите геометријске симболе и доделити им значење, тако да би корисник на први поглед одмах могао да види о чему се ради, без читања назива чвора.

ЛИТЕРАТУРА

[1] Alexandre Bergel, Damien Cassou, Stephane Ducasse, Jannik Laval, *Deep into Pharo*, Square Bracket Associates, Switzerland, 2013

[2] Stéphane Ducasse, Dimitris Chloupis, Nicolai Hess, and Dmitri Zagidulin, *Pharo By Example 5*, September 29, 2018

[3] Janusz Laski, William Stanley, *Software Verification and Analysis*, Springer, 2009

[4] Наташа Сукур, *Репрезентација тока извршавања програма дијаграмом стања, независна од улазног језика*, Мастер рад, Природно - математички факултет, Универзитет у Новом Саду, 2016.

[5] Visualizing Polymetric Graphs using Mondrian, http://agilevisualization.com/AgileVisualization/Mondrian/0202-Mondrian.html

[6] Еволуција софтвера, http://rti.etf.bg.ac.rs/rti/ms1es/Predavanja/ProcenaTroskova.ppt

[7] Оптимизација, http://www.acs.uns.ac.rs/sites/default/files/PP9-optimizacija.pdf

[8] Обрасци понашања, http://www.igordejanovic.net/courses/sok/obrasci-ponasanja.html#53

[9] Identifying messages , https://ci.inria.fr/pharo-contribution/job/UpdatedPharoByExample/lastSuccessfulBuild/artifact/book-result/UnderstandingMessage/UnderstandingMessage.html

[10] The Pharo MOOC , https://mooc.pharo.org/

[11] Smalltalk, https://sr.wikipedia.org/sr-ec/Smalltalk

[12] Abstract Syntax Tree, https://vinaytech.wordpress.com/2008/10/04/abstract-syntax-tree/

[13] All Pharo syntax on a postcard , https://pharoweekly.wordpress.com/2018/06/02/all-pharo-syntax-on-a-postcard/

[14] Стабло парсирања, https://sh.wikipedia.org/wiki/Stablo_parsiranja

[15] Тестирање софтвера , http://etf.beastweb.org/index.php/site/download/P5_BelaKutija.pdf

[16] OAK, http://www.smalltalkhub.com/#!/~GordanaRakic/Oak

Кратка биографија:

Светлана Ђурић рођена је 15.10.1995 у Зрењанину. Основне студије из области Рачунарство и аутоматика – Примењене рачунарске науке и информатика је завршила 2017. године када је уписала и мастер студије из исте области.