KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Deividas Kreivys

Programų sistemų variantiškumo modelių, aprašytų požymių diagramomis, tyrimas

Magistro darbas

Vadovas
doc. dr. Robertas Damaševičius

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Deividas Kreivys

Programų sistemų variantiškumo modelių, aprašytų požymių diagramomis, tyrimas

Magistro darbas

Recenzentas
doc. dr. Giedrius Ziberkas

2010-05-26

Vadovas
doc. dr. Robertas Damaševičius

2010-05-26

Atliko
IFM 4/2 gr. stud. Deividas Kreivys

2010-05-26

Kaunas, 2010

SUMMARY

Feature Modeling is a domain modeling technique used in software product line development and generative software engineering that addresses the development of reusable software. A feature model defines common and variable elements of a family of software systems or products of a product line – the domain. It can be used to derive members of the system family built from a common set of reusable assets. The concept of product line, if applied systematically, allows for the dramatic increase of software design quality, productivity, provides a capability for mass customization and leads to the ‘industrial’ software design [26].

In this work, the author describes the way of product line variability specification using feature diagrams. The presented approach deals with specification of feature model elements, syntax validation, complexity evaluation and feature diagram configuration aspects. The developed software, described in this thesis, allows the user to specify features, design, validate, evaluate and document system product line variability models.

TURINYS

SUMMARY 3

TURINYS 4

Terminų ir santrumpų žodynas 8

Lentelių sąrašas 10

Paveikslėlių sąrašas 11

1. ĮVADAS 12

1.1. Dokumento paskirtis 13

1.2. Mokslinis naujumas. 13

1.3. Darbo tikslas. 13

1.4. Uždaviniai 13

2. LITERATUROS APŽVALGA IR TECHNOLOGINIŲ SPRENDIMŲ ANALIZĖ. 14

2.1. Požymių diagramų sintaksė 14

2.2. Požymių diagramų sintaksės variantų palyginimas 15

2.2.1. Požymių diagramos pagal Gomaa 15

2.2.2. Požymių diagramos pagal Riebisch 15

2.2.3. OFT FODA Požymių medžio kalba pagal Kang (1990) 16

2.2.4. OFD FORM Požymių diagramų kalba pagal Kang (1998) 16

2.2.5. RFD RSEB Požymių diagramų kalba pagal Griss (1998) 17

2.2.6. VBFD požymių diagramų kalba pagal van Gurp (2001) 18

2.2.7. GPFT Eisenecker ir Czarnecki notacija (2000) 18

2.2.8. EFD Požymių diagramų išplėtimas pagal Riebisch (2002) 19

2.3. Bendra požymių diagramų modeliavimo įrankių apžvalga 20

2.3.1. Įrankis Feature IDE 20

2.3.2. Įrankis XFeature 20

2.3.3. Įrankis Feature modeling plugin 20

2.3.4. Įrankis Ecore Feature modeling plugin 20

2.3.5. Įrankis RequiLine 21

2.3.6. Įrankis FAMA 21
2.3.7. KumbangTools 21
2.3.8. pure::variants 21
2.3.9. Požymių modeliavimo ir UML įrankiai 21
2.4. Požymių diagramų įrankių palyginimas 21
2.5. Įrankių pasirinkimo įvertinimas 23
3. PROŽYMIŲ DIAGRAMŲ MODELIAVIMO ĮRANKIO KŪRIMO PROJEKTINĖ DALIS 25
3.1. Sistemos paskirtis 25
3.2. Kuriamos sistemos tikslai 25
3.3. Funkciniai reikalavimai 25
3.4. Nefunkciniai reikalavimai 26
3.4.1. Reikalavimai sistemos išvaizdai 26
3.4.2. Reikalavimai panaudojamumui 27
3.4.3. Reikalavimai vykdymo charakteristikoms 27
3.4.4. Reikalavimai veikimo sąlygoms 28
3.4.5. Reikalavimai sistemos priežiūrai 28
3.4.6. Reikalavimai saugumui 28
3.4.7. Kultūriniai-politiniai reikalavimai 29
3.4.8. Teisiniai reikalavimai 29
3.5. Architektūros tikslai ir apribojimai 29
3.6. Apribojimai sprendimui 30
3.7. Diegimo aplinka 30
3.8. Sistemos statinis vaizdas 30
3.9. Paketas Eclipse sistema 30
3.10. Paketas FD komponentas 32
3.11. Duomenų vaizdas 33
3.12. Sistemos sudėtis 34
3.13. Panaudojimo atveju aprašymai 35

4. POŽYMIŲ DIAGRAMŲ METAMODELIS IR JO ELEMENTAI 41

4.1. Siūlomas požymių diagramų metamodelis.....41

4.2. Realizuoto FD sintaksės varianto aprašymas 44

4.3. Siūloma produktų linijų projektavimo metodika naudojant požymių diagramas 46

4.4. Atskiros produktų linijų projektavimo problemos ir siūlomi jų sprendimo metodai 48

4.4.1. Požymių modelių specifikavimas 48

4.4.2. Požymių modelių validavimas 49

4.4.3. Požymių modelių sudėtingumo charakteristikų įvertinimas 51

4.4.4. Požymių modelių konfigūravimas 53

5. SUKURTO POŽYMIŲ MODELIAVIMO ĮRANKIO KOKYBĖS ĮVERTINIMAS 56

5.1. Įvadas.....56

5.2. Sistemos testavimas.....56

5.2.1. Testavimo tikslai ir objektai.....56

5.2.2. Testavimo apimtis ir tipai.....56

5.2.3. Reikalavimai testavimui.....56

5.2.4. Testavimo prioritetai 56

5.3. Projektavimo komandos narių peržiūrų aprašymas.....57

5.4. Interviu su užsakovu. Aptariami reikalingi patobulinimai. Sukuriamas pakeitimų sąrašas. 57

5.5. Formalios techninės peržiūros (ar korektiškai buvo sukurtas programų kodas) 57

5.6. Atlikto darbo kaina, darbo našumas 58

5.7. Reikalavimų išpildymas 58

5.8. Kokybės įvertinimas.....59

5.9. Išvados.....60

6. POŽYMIŲ DIAGRAMŲ MODELIO EKSPERIMENTINĖ DALIS 61

6.1. E-parduotuvė 61
6.1.1. Problema 61
6.1.2. Reikalavimai 61
6.1.3. Požymių sąrašas 61
6.1.4. E-parduotuvės požymių modelis (diagrama) 67
6.1.5. Požymių modelio charakteristikų tyrimas....68
6.1.6. Požymių modelio validavimo tyrimas 69
6.1.7. Požymių modelio konfigūravimo tyrimas....70
7. IŠVADOS 72
8. LITERATŪRA....73
9. PRIEDAS 77

Terminų ir santrumpų žodynas

1. FODA (angl. Feature-Orented Domain Analysis) – požymiais paremta srities analizė.
2. Eclipse (platforma) – programa, priedų platforma, kurią galima papildyti individualiais priedais
3. Eclipse priedas – į Eclipse programą integruotas įrankis, kuris atlieka tam tikrą funkciją.
4. Kodo generavimas – Programinio kodo sukūrimas, pagal pasirinktą klasių diagramą.
5. XML – (angl. eXtensible Markup Language) yra W3C kompanijos rekomenduojama bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.
6. XMI – (angl. XML Metadata Interchange) standartas metaduomenų keitimuisi tarp programų, panaudojant XML kalgą.
7. OWL – (angl. Web Ontology Language) – Internetinių duomenų failų aprašymo panaudojant ontologijas kalba.
8. EPL – (angl. The Eclipse Public License) – Eclipse programos kūrėjų sukurta viešam naudojimui skirta licenzija. Pagal šią licenziją kuriamos atviro kodo programos.
9. CPL – (angl. Common Public License) – IBM korporacijos sukurta vieša licencija leidžianti laisvai naudoti turinį. Gali būti vartojama atviro ir uždaro kodo programinei įrangai.
10. GMF – Grafinio modeliavimo įrankiai.
11. EMF – Modeliavimo įrankiai.
12. JAVA - objektiškai orientuota programavimo kalba, 1991 metais sukurta Džeimso Goslingo ir kitų Sun Microsystems inžinierių.
13. UML – (angl. Unified Modeling Language**)** – modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus.
14. C++ – bendros paskirties programavimo kalba, viena populiariausių programavimo kalbų.
15. FD – (angl. Feature Diagram) – požymių diagrama.
16. Dialektas – Tam tikras savitas požymių diagramų vaizdavimas.

17. OCL – (angl. Object Constraint Language) – tai konstruojamoji skirta aprašinėti taisykles naudojamas UML modelyje.
18. CASE – (angl. Computer-Aided Software Engineering) – programinės įrangos kūrimo procesas panaudojant automatizavimo įrankius.

Lentelių sąrašas

1 lentelė. Pagrindiniai Požymių diagramos sintaksės elementai. [29].....14

2 lentelė. Požymių diagramos pagal Riebich.....15

3 lentelė. OFT FODA Požymių medžio kalba pagal Kang (1990).....16

4 lentelė. OFD FORM Požymių diagramų kalba pagal Kang (1998) 17

5 lentelė. RFD RSEB Požymių diagramų kalba pagal Griss (1998) 17

6 lentelė. VBFD požymių diagramų kalba pagal van Gurp (2001).....18

7 lentelė. GPFT Eisenecker ir Czarnecki notacija (2000) 19

8 lentelė. EFD Požymių diagramų išplėtimas pagal Riebisch (2002) 19

9 lentelė. Požymių diagramų braižymo įrankiai 22

10 lentelė. Aktoriai 35

11 lentelė. Panaudos atvejai.....35

12 lentelė. Realizuoto FD sintaksės pirmo varianto aprašymas 44

13 lentelė. Realizuoto FD sintaksės antro varianto aprašymas.....46

14 lentelė. Požymių diagramos elementų pažinimo svoriai. 52

15 lentelė. Programos kokybės vertinimo kriterijai.....59

16 lentelė. Elementų kiekis požymių modelyje (pagal 21 pav).....68

Paveikslėlių sąrašas

1 pav. Eclipse – priedų platforma [28]. 29
2 pav. Statinis sistemos vaizdas. 30
3 pav. Eclipse darbalaukio klasių diagrama [8]. 31
4 pav. Klasių diagrama FD komponentas. 32
5 pav. Duomenų diagrama. 33
6 pav. Sistemos panaudojimo atvejai 34
7 pav. Požymių diagramos metamodelis....42
8 pav. Požymis 42
9 pav. Būtinasis ryšys....43
10 pav. Nebūtinas ryšys 43
11 pav. Pasirenkamas (*case* - pasirinkimas)....43
12 pav. Pasirenkamas (*OR* - pasirinkimas) 44
13 pav. Ryšys *XOR*....44
14 pav. Reikalavimas (*requires*) 44
15 pav. Rezervacijų sistemos UML klasių diagrama....47
16 pav. Rezervacijų sistemos Požymių diagrama 47
17 pav. Privalomų ryšiu šaka 54
18 pav. Šaka su vienu neprivalomu ryšiu....54
19 pav. XOR ryšio problemos pavyzdys....54
20 pav. CASE pasirinkimo problemos pavyzdys....55
21 pav. E-parduotuvės požymių modelis 67
22 pav. E-parduotuvės požymių modelio charakteristikos 69
23 pav. E-parduotuvės konfigūravimo pavyzdys....70

1. ĮVADAS

Programų inžinerijoje, nuo 1968 metų programų pakartotinis panaudojimas tapo aktualus programų kūrimo procesuose [23]. Šiomis dienomis programų produktų linijos inžinerija [6] [26] skatina sisteminį pakartotinį panaudojimą visame programos kūrimo procese. Produktų linijos tai rinkinys programinės įrangos produktų, kurie dalinasi panašiomis savybėmis, tenkina specifinius reikalavimus ir naudoja tam tikras pagrindinės sistemos dalis.

Sistemingai taikant produktų linijos idėją, galima žymiai pagerinti projektuojamos sistemos kokybę, produktyvumą, suteikti galimybę dideliems pritaikymams pagal vartotojo reikalavimus ir šis metodas veda prie pramoninio programų kūrimo [20]. Pagrindas produktų linijų įgyvendinimo tai, kad jos naudojamos srities analizėje ir srities modeliavimo metoduose. Produktų linijų modeliavimui geriausia tinka naudoti požymių diagramas, jos apibrėžia srities turinį požymių forma ir leidžia modeliuoti sritį identifikuojant panašumus ir struktūros variantiškumą, funkcionalumą ir kitas charakteristikas (vadinamas požymiais) ir jų ryšius.

Požymis – tai savitas, charakteringas sistemos atributas. FODA (angl. Feature Orented Domain Analysis) [17] požymius apibūdina kaip žinomas, savitas bei vartotojui matomas sistemos charakteristikas, tuo tarpu funkcijos, objektai ir aspektai yra naudojami apibūdinti vidines sistemos detales. Požymių modeliavimas susitelkia ties labai matomų išorinių produkto charakteristikų apibūdinimu, kalbant apie produkto bendrumą bei variantiškumą [13], o ne apie detalų sistemos apibūdinimą [30].

Požymių modeliavimo rezultatas yra požymių diagramos. Tai yra grafinė kalba naudojama atvaizduoti bei modeliuoti sistemos arba komponento variantiškumus aukštesniame abstrakcijos lygyje, dažniausiai pradiniuose projektavimo lygiuose, tokiuose kaip reikalavimų specifikavime kuriant programinę įrangą [37]. Požymiai yra apibūdinami ir klasifikuojami galimybių specifikacijose, srities inžinerijoje, sistemų kūrimo metodikose bei operacinėse aplinkose. Galimybės yra vartotojui matomos charakteristikos kurios gali būti apibūdinamos kaip skirtingos paslaugos, operacijos ir nefunkcinės charakteristikos. Srities inžinerija apima srities žinių pakartotinio panaudojimo (atkartojimo) procesus. Kūrimo metodai yra bendros funkcijos arba metodai kurie naudojamos kurti paslaugas, operacijas ar srities funkcijas. Operacinės aplinkos apibūdina kurioje aplinkoje programinė įranga yra naudojama.

Požymių diagramų modeliavimas yra labai svarbus pakartotinio panaudojimo procesuose. Pakartotinis panaudojimas yra sistemų kūrimas naudojant jau turimus programų komponentus (modulius, kodo fragmentus) [3]. Programų produktų linijos pakartotinis

panaudojimas stipriai priklauso nuo požymių variantiškumo [16]. Modeliuojant sistemą aukštame abstrakcijos lygmenyje (požymių lygmenyje) galima lengvai apibrėžti kurios dalys iš jos gali būti pašalintos arba į ją įtrauktos, to pasekoje atliekant pakartotinį panaudojimą jau yra aiškus sistemos variantiškumas ir galima paprastai išgauti naują produkto konfigūraciją.

1.1. Dokumento paskirtis

Šiame darbe atliekamas programų sistemų variantiškumo modelių aprašytų požymių diagramomis tyrimas specifikavimo, sintaksės validavimo, sudėtingumo įvertinimo ir konfigūravimo aspektais. Darbe aprašomas autoriaus (bendraautorius: P. Žaliaduonis) sukurtas požymių modeliavimo įrankis leidžia vartotojui specifikuoti, modeliuoti, validuoti, įvertinti ir dokumentuoti programų sistemos produktų linijos požymių variantiškumo modelius.

1.2. Mokslinis naujumas.

1. Pasiūlytas ir aprašytas požymių diagramos metamodelis.
2. Išnagrinėta požymių diagramų sintaksės validavimo problema, pasiūlytos taisyklės, kurių reikia laikytis brėžiant taisyklingą požymių diagramą.
3. Atliktas požymių diagramų sudėtingumo įvertinimo tyrimas.
4. Ištirta požymių modelių konfigūravimo problema, pasiūlyti požymių konfigūravimo problemų sprendimo būdai.

1.3. Darbo tikslas.

Ištirti programų sistemų variantiškumo modelių, aprašytų požymių diagramomis (toliau – požymių modelių), specifikavimo, sintaksės validavimo, sudėtingumo įvertinimo ir konfigūravimo problemas.

1.4. Uždaviniai.

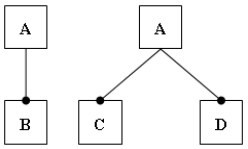
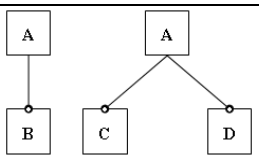
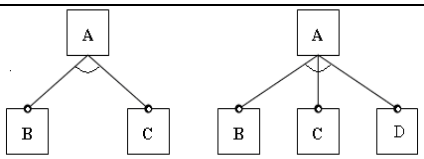
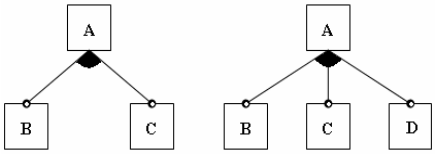
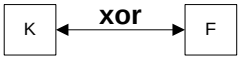
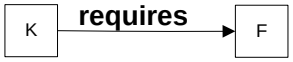
1. Aprašyti realizuoto požymių diagramų sintaksės varianto metamodelį.
2. Ištirti požymių diagramų specifikavimo problemas kuriant požymių modeliavimo įrankį.
3. Ištirti požymių modelių sintaksės patikrinimo (validavimo) problemas.
4. Atlikti požymių modelių sudėtingumo įvertinimo tyrimą.
5. Ištirti požymių modelių konfigūravimo problemas.
6. Pateikti, aprašyti ir ištirti tiriamos sistemos variantiškumo modelį.

2. LITERATUROS APŽVALGA IR TECHNOLOGINIŲ SPRENDIMŲ ANALIZĖ

2.1. Požymių diagramų sintaksė

Požymių diagrama yra medžio tipo arba tiesinis, aciklinis grafas, kurį sudaro rinkinys viršūnių, tiesinių briaunų ir briaunų kombinacijų. Šakninis elementas apibūdina aukščiausią požymio lygį (pvz. sistema, sritis). Vidutinės viršūnės apibūdina sudėtinius požymius, paskutinės – atominius požymius, kurie nėra skaidomi į mažesnius požymius duotoje srityje. Briaunos naudojamos palaipsniui suskaidyti sudėtinius požymius į labiau detalizuotus požymius. Grafo briaunos nurodo ryšius arba priklausomybes tarp požymių. [37]

1 lentelė. Pagrindiniai Požymių diagramos sintaksės elementai. [29]

Elementas	**Apibrėžimas**	**Grafinis pavyzdys**
Būtinas	Požymis B (C,D) yra įtrauktas, jeigu jo tėvas A yra įtrauktas **Jei** A **tuomet** B; **Jei** A **tuomet** B ir C	
Nebūtinas	Požymis B(C,D) gali būti įtrauktas jeigu jo tėvas A yra įtrauktas **Jei** A **tuomet** B arba niekas; **Jei** A **tuomet** C arba D arba niekas	
Pasirenkamas (*case* - pasirinkimas)	Tik vienas požymis B arba C arba D turi būti pasirinktas jeigu jo tėvas A yra parinktas **Jeigu** A **tuomet** *case iš*(B,C,D)	
Pasirenkamas (*OR* - pasirinkimas)	Mažiausiai vienas požymis (B,C arba D arba B ir C; arba B ir D; arba C ir D; arba B ir C ir D) turi būti pasirinkti jeigu jo tėvas A yra pasirenkamas **Jei** A **tuomet** *bet kuris iš*(B,C,D)	
Ryšys (*XOR*)	Ryšys tarp dviejų atominių požymių. K *XOR* F. **Jei** F **tuomet** ne K ir **jei** ne F **tuomet** K	
Reikalavimas (*requires*)	Požymis K reikalauja požymio F. K **reikalauja** F	

Šie elementai yra dalis EFD metodo pasiūlyto Riebisch [32], su papildomu išplėtimu bei apjungiant kelis kitus variantus, kad atvaizduoti galimą požymių diagramos sintaksę. Vaizdavimas papildytas grafiniais elementais. Šis bei kiti vaizdavimo metodai bus aptarti vėliau prie požymių diagramų sintaksės variantų palyginimo.

2.2. Požymių diagramų sintaksės variantų palyginimas

Atsiradus požymių diagramoms, joms vaizduoti buvo pasiūlyti keli variantai. Kiekvienas iš jų turi savų pliusų ir minusų. Šiuo metu yra labiausiai paplitusi FODA pasiūlyta elementų notacija.

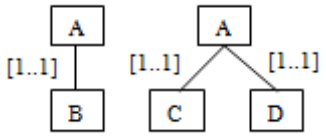
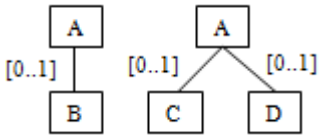
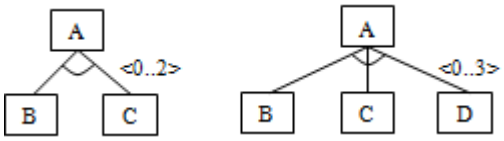
2.2.1. Požymių diagramos pagal Gomaa

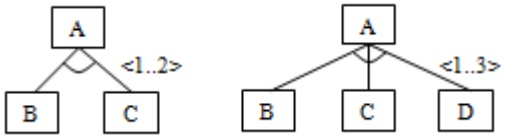
Gomaa (2000) pasiūlė vaizduojant požymius UML diagramomis naudoti stereotipą _Kernel_ (_Branduolys_) ir _Optional_ (_Pasirenkamas_) apibūdinamiems UML elementams. Požymiai yra vaizduojami kaip paketai ir ryšiai tarp jų yra modeliuojami naudojant _include_ bei _exclude_ ryšį. Šis būdas padaro diagramą naudingą visam modeliui, kadangi kiekvienas elementas žino savo paketą, taigi jis žino kuriam požymiui jis priklauso. Tačiau UML mums leidžia modeliuoti hierarchijas tik viename lygmenyje [12].

2.2.2. Požymių diagramos pagal Riebisch

Riebisch pasiūlė požymių diagramose naudoti grupes. Grupuoti požymiai yra atvaizduojami su antraštėmis, pvz. [0..1] reiškia, kad požymis gali būti pasirenkamas arba ne, tai pat naudojami ir [0..0] arba [1..1] , tai parodo ar požymis yra pašalinamas ar būtinai pasirenkamas iš grupės (žr. 2 lentelė) [32].

2 lentelė. Požymių diagramos pagal Riebich

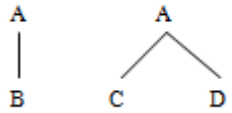
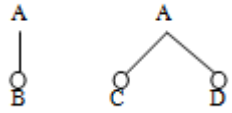
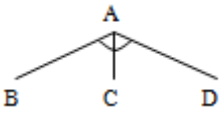
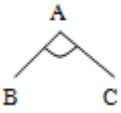
Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	
Pasirenkamas (case - pasirinkimas)	Jeigu A tuomet case iš(B,C,D)	

Pasirenkamas (*OR* - pasirinkimas)	Jei A tuomet bet kuris iš(B,C,D)	

2.2.3. OFT FODA Požymių medžio kalba pagal Kang (1990)

Tai buvo pirmoji požymių diagramų kalba, ji buvo apibrėžta kaip dalis FODA metodo. Jos pagrindinis tikslas buvo specifikuoti reikalavimuose panašumus ir kintamas dalis. Požymiai yra vaizduojami medžio tipo diagrama. Tačiau šiame metode nebuvo įtraukti *OR* tipo elementai, dėl šio trūkumo atsirasdavo problemų renkantis elementą, turintį daugiau nei vieną šaką iš kurių kelios būdavo nereikalingos, tačiau tekdavo pasirinkti visas (žr. 3 lentelė) [19].

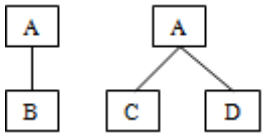
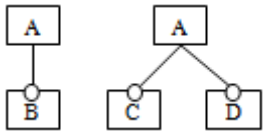
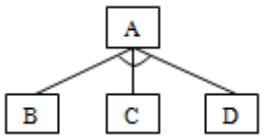
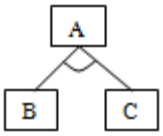
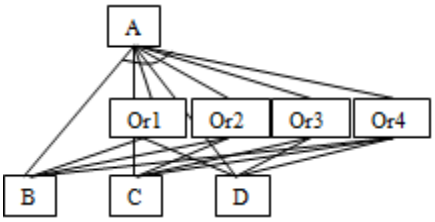
3 lentelė. OFT FODA Požymių medžio kalba pagal Kang (1990)

Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	
Pasirenkamas (case - pasirinkimas)	Jeigu A tuomet case iš(B,C,D)	
Pasirenkamas (*XOR* - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	

2.2.4. OFD FORM Požymių diagramų kalba pagal Kang (1998)

Kang, pasiūlė į požymius orientuoto pakartotinio panaudojimo metodą (FORM), kaip FODA metodo praplėtimą. Pagrindinis tikslas buvo padidinti požymių modeliavimo apimtį. Pagrindinis argumentas buvo kad požymių modeliavimas ne tik reikalavimų tyrimas, bet taip pat ir programinės įrangos modelis. Pagrindinis skirtumas nuo OFT modelio yra tas, kad požymiai vaizduojami dėžutėse. Taip pat šioje notacijoje atsirado OR pasirinkimas, kuriam reikalingi papildomi elementai padedantys atlikti pasirinkimą. Šį pasirinkimą buvo galima atlikti su visais pasirenkamo tipo elementais. Taip pat OR pasirinkimą buvo galima pritaikyti ir atlikti su XOR tipo elementais (žr. 4 lentelė) [18].

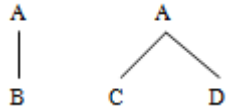
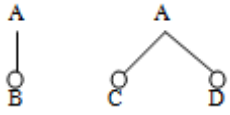
4 lentelė. OFD FORM Požymių diagramų kalba pagal Kang (1998)

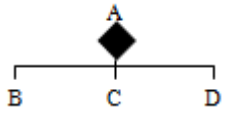
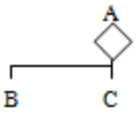
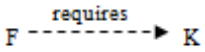
Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	
Pasirenkamas (case - pasirinkimas)	Jeigu A tuomet case iš(B,C,D)	
Pasirenkamas (XOR - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	
OR - skaidymas	Jei A tuomet bet kuris iš(B,C,D)	

2.2.5. RFD RSEB Požymių diagramų kalba pagal Griss (1998)

Šiame metode Griss paliko standartinį grafo tipo modelį, kuomet stačiakampiai elementai nėra naudojamos atvaizduoti požymiams. Pagrindiniai nauji dalykai buvo įtraukti naujo tipo elementai *OR* ir *XOR* ryšio dekompozicijos pakeitimui, kuomet *OR* ryšiui naudojamas pilnaviduris rombas prie tėvinio elemento, o *XOR* – tuščiaviduris. Taip pat atsirado ir naujas ryšio elementas vadinamas – *requires*. Jis parodo, kad atominis požymis gali reikalauti kito atominio požymio. Šis ryšys yra vaizduojamas punktyrine linija su užrašu „requires“. Pakeitimus galite pamatyti 5 lentelėje.

5 lentelė. RFD RSEB Požymių diagramų kalba pagal Griss (1998)

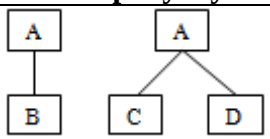
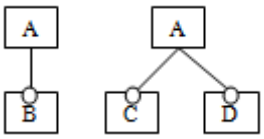
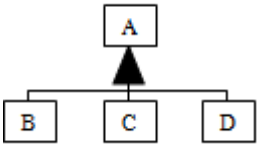
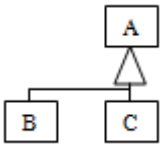
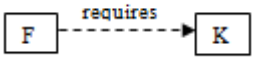
Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	

Pasirenkamas (OR - pasirinkimas)	Jei A tuomet bet kuris iš(B,C,D)	
Pasirenkamas (XOR - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	
Reikalavimas (requires)	K reikalauja F	

2.2.6. VBFD požymių diagramų kalba pagal van Gurp (2001)

Ši notacija yra RFD praplėtimas. Požymiai yra vaizduojami stačiakampiuose elementuose. XOR ir OR elemento vaizdavimui vietoje rombo naudojamas tuščiaviduris arba pilnaviduris trikampis. Išoriniai požymiai yra vaizduojami punktyriniais stačiakampiais. Tačiau šis ryšys nenulemia galutinių požymių derinio (žr. 6 lentelė).

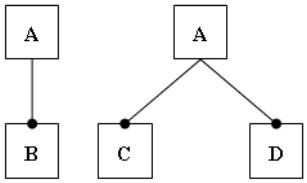
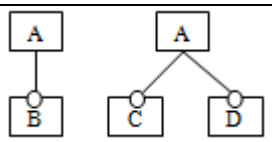
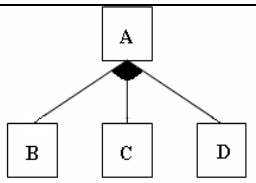
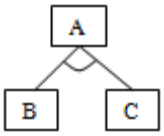
6 lentelė. VBFD požymių diagramų kalba pagal van Gurp (2001)

Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	
Pasirenkamas (OR - pasirinkimas)	Jei A tuomet bet kuris iš(B,C,D)	
Pasirenkamas (XOR - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	
Reikalavimas (requires)	K reikalauja F	

2.2.7. GPFT Eisenecker ir Czarnecki notacija (2000)

Ši notacija skiriasi nuo prieš tai aprašytų pagrinde tuo, jog būtinas ryšys yra žymimas ne tik linija, bet ir pilnaviduriu apskritimu virš vaiko požymio. Taip pat *OR* ir *XOR* vaizdavimui po tėviniu požymiu yra naudojamas lankas jungiantis visas šakas (žr. 7 lentelė).

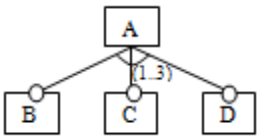
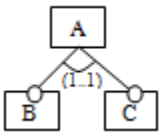
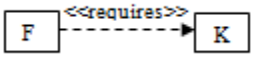
7 lentelė. GPFT Eisenecker ir Czarnecki notacija (2000)

Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Jei A tuomet B; Jei A tuomet B ir C	
Nebūtinas	Jei A tuomet B arba niekas; Jei A tuomet C arba D arba niekas	
Pasirenkamas (OR - pasirinkimas)	Jei A tuomet bet kuris iš(B,C,D)	
Pasirenkamas (XOR - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	

2.2.8. EFD Požymių diagramų išplėtimas pagal Riebisch (2002)

Šioje notacijoje pagrinde buvo pakeisti tik *OR* ir *XOR* elementai. Nebenaudojamas pilnaviduris lankas, jo viduje įrašomos pasirinkimo ribos kai (1..1) - XOR, (1..3) – OR, šiam praplėtimui yra naudojamas Riebisch metodas. Taip pat atsiranda apskritimai prie OR ir XOR vaiko elementų, kaip ir nebūtino požymio elemente. Requires laukui pridedami skliaustai <<requires >>. Šie elementai buvo pasiskolinti iš UML (žr. 8 lentelė).

8 lentelė. EFD Požymių diagramų išplėtimas pagal Riebisch (2002)

Elementas	Apibrėžimas	Grafinis pavyzdys
Pasirenkamas (OR - pasirinkimas)	Jei A tuomet betkuris iš(B,C,D)	
Pasirenkamas (XOR - pasirinkimas)	Jei A tuomet B bet ne C arba C bet ne B	
Reikalavimas (requires)	K reikalauja F	

Formaliai požymių diagramų sintaksė ir semantika aprašyta [27].

2.3. Bendra požymių diagramų modeliavimo įrankių apžvalga

2.3.1. Įrankis Feature IDE

Ši programa palaiko programų šeimynų kūrimą, pagrįstų AHEAD architektūros modeliu, kuris yra skirtas programų šeimynų kūrimui. AHEAD architektūroje laikoma, kad požymis, tai nedidelė sudedamoji sistemos dalis (klasė ar funkcija), kuri papildo programos funkcionalumą [36]. Šis įrankis leidžia braižyti požymių diagramas ir kurti programinį kodą, pagal sukurtas požymių diagramas.

Šiuo metu šis įrankis yra kuriamas Magdeburgo (Vokietija) universitete.

2.3.2. Įrankis XFeature

Tai požymių modeliavimo įrankis, kuris palaiko programų šeimynų modeliavimą. Šis įrankis leidžia vartotojams susikurti savo požymių meta modelį. Šis įrankis yra Eclipse programos priedas. Ši programa skirta pademonstruoti, kaip galima automatizuoti programų modeliavimą ir konfigūraciją pasitelkiant pakartotinai panaudojamas programos dalis [25].

Įrankį sukūrė Ondrej Rohlik ir Alessandro Pasetti, šiuo metu jis naudojamas bei tobulinamas ETH-Zürich (Automatikos institutas, Šveicarija).

2.3.3. Įrankis Feature modeling plugin

Šis įrankis yra Eclipse priedas, jis gali būti naudojamas vienas (leidžia modeliuoti požymių diagramas), arba kartu su fmp2rsm priedu, kuris sujungia šį įrankį su Rational Software Modeler (RSM) arba Rational Software Architect (RSA) modeliavimo įrankiais (Leidžia modeliuoti programų šeimynų UML diagramas). Šis įrankis skirtas modeliuoti reikalavimams produktų šeimai ir aprašyti priklausomybes tarp požymių. Savybės vaizduojamos hierarchiškai. Grafiniam vaizdavimui pasitelkiamas medžio vaizdavimo principas [22].

Šį įrankį sukūrė Kanados Waterloo universitetas ir šiuo metu jo kūrimas nebetęsiamas.

2.3.4. Įrankis Ecore Feature modeling plugin

Tai Eclipse priedas leidžiantis grafiškai modeliuoti požymių diagramas. Jis leidžia peržiūrėti bei modeliuoti EMF Ecore (Eclipse modeline framework) modelius. Šis įrankis susieja ECORE modelius taip, kad pakeitus ECORE modeli, pakeitimai matomi požymių diagramose ir atvirkščiai. Požymiai yra susiejami su ECORE modeliais [22].

2.3.5. Įrankis RequiLine

Tai reikalavimų surinkimo įrankis, sukurtas efektyviam produktų linijos valdymui. Šis įrankis leidžia modeliuoti produktus pasitelkiant požymių diagramomis ir reikalavimais. Iš sudarytų diagramų sukuriama produkto konfigūracija. Šis įrankis turi grafini požymių diagramų redaktorių leidžianti dirbti su FORM notacijos diagramomis, Klaidų tikrinimą, XML redaktorių [35].

2.3.6. Įrankis FAMA

Tai įrankis skirtas automatizuoti kintančių modelių analizavimą, panaudojant požymių modelius. Šis įrankis sudarytas iš kelių dalių: FAMA programa, FAMA karkasas ir FAMA Eclipse priedas. FAMA karkasas yra pagrindinis programos komponentas realizuojantis programos logiką, o pati programa ir Eclipse priedas realizuoja požymių diagramų vaizdavimą [4].

2.3.7. KumbangTools

Tai yra įrankių rinkinys konfigūruojamai programų šeimynai valdyti. Šiuo metu šį produktą sudaro šie įrankiai: „Kumbang Modeller“ tai įrankis programų šeimynai projektuoti; „Kumbang Configurator“ tai įrankis programų šeimynos architektūros modelių kūrimui [24].

2.3.8. pure::variants

Šis įrankis skirtas valdyti produktų šeimynos variantiškumą. Jis nėra priklausomas nuo programavimo kalbos, kuri yra naudojama, taigi jį galima įjungti į daugelį produktų, nepriklausomai nuo to, kokia kalba jie parašyti. Ši programa gali aptikti ir pašalinti konfliktus, modeliuojant produkto variantiškumą [7][24].

2.3.9. Požymių modeliavimo ir UML įrankiai

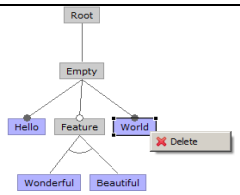
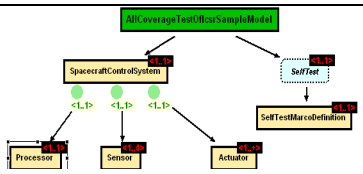
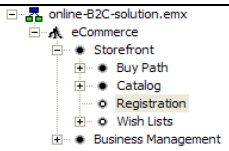
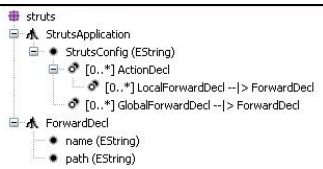
Peržvelgę eilę įrankių matome, kad įrankiai leidžiantys modeliuoti požymių diagramas, nesuteikia geros galimybės susieti šias diagramas su UML diagramomis. Peržvelgtame sąraše šią galimybę turi 2 įrankiai sukurti Waterloo universitete, „Feature modeling plugin” ir “Ecore Feature modeling plugin”. Įrankis “Feature modeling plugin” nesuteikia galimybės tiesiogiai susieti požymių modelių su UML modeliais, tam reikia naudoti brangų IBM Rational įrankį, o „Ecore Feature modeling plugin” dar nėra sukurtas.

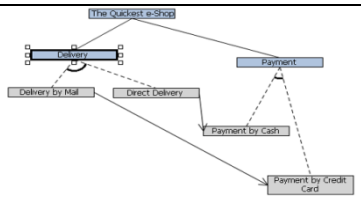
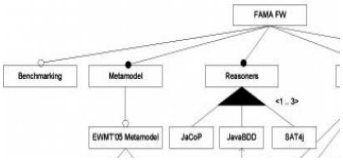
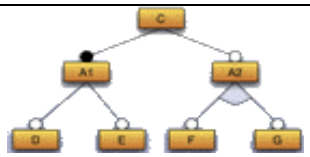
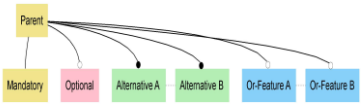
2.4. Požymių diagramų įrankių palyginimas

Šiuo metu yra sukurta keletas skirtingų požymių modeliavimo įrankių. Čia yra pateikiami keli išanalizuoti įrankiai: Feature IDE, XFeature, Ecore Feature modelling plugin,

RequiLine, FAMA, KumbangTools, pure::variants (žr. 9 lentelė). Dauguma paminėtų projektų yra sukurti kaip Eclipse įskiepiai su GPL licencija. Šie įrankiai turi labai skirtingas galimybes ir tikslus – kai kurie orientuojasi į programos kodo generavimą, kiti kuria konfigūracinius dokumentus arba net kitus modelius iš kurių galėtų būti sugeneruotas programos kodas arba leidžia sukurti savo meta modelius. Šie įrankiai naudoja skirtingas požymių vaizdavimo notacijas.

9 lentelė. Požymių diagramų braižymo įrankiai

Įrankio pavadinimas	**Notacija**	**Privalumai**	**Trūkumai**
Feature IDE		- Nemokamas; -galimybė išsaugoti XMI formatu; -validavimas; -programos kodo generavimas.	-viena notacija; -negali veikti kaip įskiepis. -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
XFeature		-Nemokamas; -turi kelias grafines notacijas; - galimybė išsaugoti XMI formatu; -validavimas; - programos kodo generavimas; -Eclipse įskiepis	-nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
Ecore / Feature Modelling Plug-in		-Nemokamas; - galimybė išsaugoti XMI formatu; - validavimas; - programos kodo generavimas; - Eclipse įskiepis	-viena notacija; -nėra grafinio vaizdavimo. -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
Ecore Feature modelling plug-in		-validavimas; - programos kodo generavimas; - Eclipse įskiepis.	- GPL licencija - viena notacija; -nėra grafinio vaizdavimo. -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;

RequiLine		-Nemokamas; - galimybė išsaugoti XMI formatu; - validavimas;	- viena notacija; - nėra programos kodo generavimo; -negali veikti kaip įskiepis; -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
FAMA		-Nemokamas; - galimybė išsaugoti XMI formatu; - validavimas; - programos kodo generavimas; -Eclipse įskiepis.	- viena notacija; -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
KumbangTools		-Nemokamas; - galimybė išsaugoti XMI formatu; - validavimas; - programos kodo generavimas; -Eclipse įskiepis.	- viena notacija; -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;
pure::variants		- galimybė išsaugoti XMI formatu; - validavimas; - programos kodo generavimas; -Eclipse įskiepis.	-nemokama tik ne komerciniam naudojimui; - viena notacija; -nėra galimybės konfigūruoti modelį; -nėra galimybės susieti su UML modeliu;

2.5. Įrankių pasirinkimo įvertinimas

Kadangi aiškiai apibrėžto standarto požymių diagramų vaizdavimui nėra, mes galime rinktis iš kelių variantų arba net apjungti kelis iš jų. Požymiams vaizduoti pasirinkta naudoti stačiakampius su viduje esančiu tekstu. Pagrindinė notacija pagal kurią kuriamas projektas yra Eisenecker ir Czarnecki notacijos požymių vaizdavimas. Kadangi jame aiškiai pateikiami tiek *OR*, tiek *XOR* elementai, be to papildomai pridedamas EFD išplėtimas reikalavimo ryšio vaizdavimui.

Požymių modeliavimo įrankių pasirinkta Eclipse platforma. Daugelis požymių modeliavimo įrankiu sukurti, kaip šios programos praplėtimai. Peržvelgus UML modeliavimo įrankius, taip pat Eclipse užima aukštą vietą. Taigi galima susidaryti įspūdį, kad ši platforma labiausiai tinkama požymių modeliavimo įrankio kūrimui. Taip pat Eclipse sistema yra

nemokama ir turi lengvą išplėčiamumo galimybę (įskiepius) ir leidžia lengvai apjungti ar išplėsti jau esamas jos dalis.

3. PROŽYMIŲ DIAGRAMŲ MODELIAVIMO ĮRANKIO KŪRIMO PROJEKTINĖ DALIS

3.1. Sistemos paskirtis

Sistema skirta programų (komponentų) šeimynų variantiškumo modeliavimui naudojant požymių diagramas produktų linijų metodologijos kontekste.

Požymių diagramos suteikia daug informacijos apie produkto variantiškumą. Jos palengvina įsisavinti sudėtingų sistemų galimybes labai greitai ir paprastai, kadangi jos tyrinėjamos aukštesniame abstrakcijos lygyje, kuomet nesigilinama į kiekvieno komponento savybes ar trūkumus.

Pasaulyje požymių diagramos taip pat yra nagrinėjamos, sukurta produktų, kuriais galima modeliuoti požymių diagramas. Tačiau daugeliui įrankių vis dar trūksta tam tikrų savybių.

3.2. Kuriamos sistemos tikslai

Sistema turi įgalinti efektyviai specifikuoti ir modeliuoti gimininguų programų šeimynas naudojant *požymių diagramas*. Čia giminingomis vadinamos programos, kurios realizuoja tą patį uždavinį esant skirtingiems užduoties sprendimo kriterijams ar vartotojo reikalavimams.

3.3. Funkciniai reikalavimai

Žemiau pateikiamas pilnas funkciniu sistemos reikalavimu sąrašas:

1. Sistema turi leisti vartotojui grafiškai braižyti požymių diagramas;
2. Turi būti numatytas vartotojo pageidaujamo požymių diagramų dialekto pasirinkimas per meniu;
3. Požymių diagramų elementai turi būti braižomi laikantis pasirinkto požymių diagramos dialekto reikalavimų;
4. Vartotojas turi turėti galimybę bet kada darbo su sistema metu pasikeisti naudojamą požymių diagramos dialektą;
5. Požymių diagramų braižymas turi būti atliekamas kompiuterio pele pasirenkant standartinius (šabloninius) požymių diagramų elementus iš įrankių juostos;
6. Papildoma požymių diagramų elementų informacija įvedama klaviatūra užpildant dialoginio meniu laukelius;
7. Turi būti numatyta grafinių požymių diagramų elementų savybių (dydžio, aukščio, pločio, linijų storio, raidžių šrifto dydžio, spalvos) keitimo galimybė;

8. Turi būti numatyta požymių diagramos priartinimo/atitolinimo („zoom“) funkcija;
9. Sistema turi turėti požymių diagramos spausdinimo funkciją;
10. Sistema turi turėti požymių diagramos išsaugojimo grafiniu formatu (jpg, gif, bmp, png, pdf) funkciją;
11. Sistema turi turėti požymių diagramos eksportavimo žymėjimo kalbos (xmi) formatu galimybę.
12. Sistema turi turėti požymių diagramos, išsaugotos žymėjimo kalbos formatu, importavimo galimybę;
13. Sistema turi leisti vienu metu modeliuoti sistemą požymių diagramomis ir UML klasių diagramomis;
14. Sistema turi leisti vartotojui interaktyviai (pele) pasirinkti modeliuojamo produkto konfigūraciją pagal požymių diagramą spaudžiant atitinkamus požymius požymių diagramoje;
15. Vartotojui turi būti leidžiama pasirinkti tik teisingą produkto konfigūraciją pagal tai, kaip ją aprašo produkto požymių diagrama;
16. Sistema turi turėti galimybę generuoti specializuotą produkto UML klasių diagramą iš su požymių diagrama susietos UML klasių diagramos pagal vartotojo pasirinktą produkto konfigūracijos. (Jeigu požymių diagrama buvo susieta su UML klasių diagrama);
17. Sistema turi pateikti brėžiamos diagramos charakteristikas;
18. Sistema turi turėti galimybė generuoti aukšto lygio programavimo kalbos (C++ ir/arba Java) kodą iš specializuotos produkto UML klasių diagramos;
19. Sistema turi turėti pagalbos funkciją, kuria iškviečiamas sistemos funkcionalumo aprašymas.

3.4. Nefunkciniai reikalavimai

3.4.1. Reikalavimai sistemos išvaizdai

Konkrečių išvaizdos reikalavimų sistemai nėra. Sistemos išvaizda priklauso nuo jau sukurtos Eclipse sistemos. Požymių diagramų braižymo modulis, turi naudoti dialekte apibrėžtus elementus, kurių išvaizda yra griežtai priklausoma nuo taisyklių pateikiamų tam tikrame dialekte, tačiau vartotojui suteikiama galimybė pasikeisti spalvas, šrifto dydžius ir t.t.

Braižymo langas turi išlikti toks, koks jau sukurtas UML diagramų braižymui, jo elementus reikia tik papildyti požymių diagramų elementais. Pradiniai stiliai bus parinkti standartiniai, jau siūlomi naudojamos sistemos.

3.4.2. Reikalavimai panaudojamumui

Sistema turi būti kiek įmanoma labiau suprantama vartotojui, kad išvengti per didelio kiekio informacijos instrukcijoje. Kuriant sistemą reikia laikytis šių reikalavimu:

- Požymių braižymo elementai turi būti įrankių juostoje, vizualia ir tekstine forma, kas palengvintų darbą galutiniam vartotojui.
- Turi būti požymių diagramos priartinimo/atitolinimo („zoom“) funkcija, kuri padės lengviau peržiūrėti požymius esant didelėms diagramoms ar aukštoms rezoliucijoms.
- Požymių braižymas turi būti atliekamas naudojant kompiuterio pelę bei klaviatūrą, tai padės vartotojui iškart matyti jo naudojamą elementą.
- Pagalbos funkcija privaloma, kad vartotojas nežinodamas visų sistemos funkcijų galėtų lengvai rasti jam reikalingą informaciją.

3.4.3. Reikalavimai vykdymo charakteristikoms

Užduočių vykdymo greitis

Sistema turi greitai brėžti komponentus, juos šalinti, redaguoti. Jeigu vykdomas generavimas UML diagramų ar kodo, tuomet laikas priklausys nuo užduoties sudėtingumo ir generuojamų elementų kiekio.

Talpumas (DB apimtis)

Duomenys bus saugomi XML tipo dokumentais. Šio saugojimo metu talpumą apibrėš duomenų kiekis. Vaizduojama grafinė informacija bus užrašomas koordinatėmis bei spalvų kodais kad sutaupyti užimamą vietą.

Panaudojimo efektyvumas

Turi būti kiek įmanoma išnaudoti jau esami Eclipse sistemos algoritmai, bei bibliotekos. Taip pat algoritmai turi būti kuo efektyvesni. Turi būti naudojamas duomenų kešavimas, kad pasikeitus mažai daliai informacijos, nereikėtų perbrėžti viso grafiko. Skaitymas iš dokumentų negali trukti ilgai. Kadangi duomenys bus saugomi XML tipo dokumentuose, jų nuskaitymui yra parašyta daug algoritmų, kurie tai gali atlikti labai efektyviai.

Patikimumas

Sistema turi taisyklingai, pagal pasirinktą dialektą brėžti diagramas. Jeigu vykdomi neleistini veiksmai reikia informuoti vartotoją ir jų neleisti atlikti. Sistema turi veikti patikimai, nepriklausomai nuo pasirinktos operacinės sistemos (Windows ar Linux).

Išplėčiamumas

Sistema veiks kaip komponentas, tad jis turi būti lengvai tobulinamas papildomomis funkcijomis ir lengvai prieinamas kitiems komponentams.

3.4.4. Reikalavimai veikimo sąlygoms

Sistema turi sėkmingai veikti tiek Windows tie Linux operacinėse sistemose. Kadangi naudojama programavimo kalba palaikoma abiejose operacinėse sistemose problemų neturi iškilti.

Vartotojas prieš naudodamas sistema turi bent dalinai susipažinti su požymių diagramų elementais.

Prieš atliekant generavimus bus tikrinama ar diagrama išbaigta ir ar nekils problemų vykdant generavimą. Taip pat bus draudžiama brėžti klaidingus ryšius.

Komponentas bus diegiamas Eclipse sistemoje, todėl turės atitikti jos komponentų kūrimo taisykles. Bus apibrėžtos tam tikros griežtos klasės, kurių negalima bus keisti, nes tuomet komponentas bus nebeatpažįstamas Eclipse sistemai.

3.4.5. Reikalavimai sistemos priežiūrai

Komponentas bus kuriamas Eclipse sistemai, kuri yra pritaikyta veikti tiek Windows tiek Linux operacinėse sistemose. Taikant dokumentacijoje nurodytus metodus kūrimo metu komponentas turi sėkmingai susidoroti su braižymo, modeliavimo užduotimis abiejose operacinėse sistemose.

Eclipse yra pritaikytas daugiakalbystei, dėl to kuriant komponentą, reikia pritaikyti jį šiai funkcijai, nes komponentas ateityje bus naudojamas ne tik lietuvių kalba.

3.4.6. Reikalavimai saugumui

Sistema bus projektuojama ir sukurta kaip atskiras sistemos komponentas, todėl dalis saugumo priklausys nuo pasirinkto produkto saugumo sprendimų. Tačiau sistema turi apsisaugoti nuo klaidingų duomenų įvedimo. Jeigu kuriamas nuosavas duomenų saugojimo tipas, jis turi būti tikrinamas nuo klaidingos informacijos. Taip pat reikia išnaudoti visas jau siūlomas pagrindines sistemos kuriai bus kuriamas komponentas apsaugos galimybes.

Duomenys su kurias dirbs vartotojas nėra kritiniai, tad saugumo atžvilgiu nereikia persistengti, kad neapkrauti sistemos darbo papildomais šifravimais.

3.4.7. Kultūriniai-politiniai reikalavimai

Kuriamas komponentas turi būti pritaikytas Lietuvos rinkai. Dokumentacija turi būti pateikta Lietuvių kalba. Jeigu komponentai bus vadinami Anglų kalba, tuomet jų paaiškinimai turi būti randami dokumentacijoje arba turi būti galimybė keisti kalbą.

3.4.8. Teisiniai reikalavimai

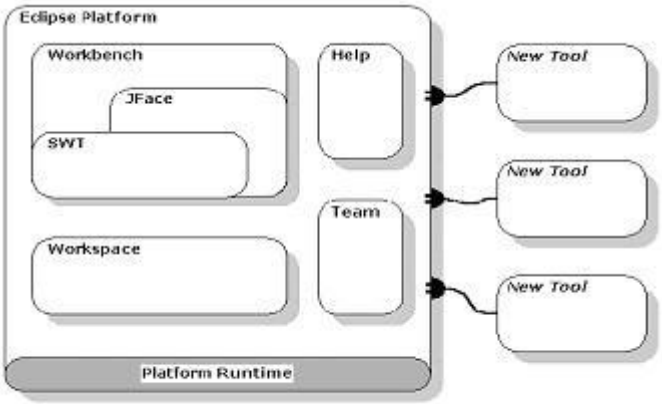
Sistema turi būti kuriama nemokama programavimo kalba (Java) ant atvirojo kodo sistemos Eclipse, kaip papildomas komponentas išplečiant UML diagramų braižymo komponentą. Kuriamas komponentas turi būti nemokamas, kaip ir sistema, kuriai jis kuriamas. Sistema, kuriai kuriamas komponentas yra saugomas EPL (The Eclipse Public License) teisių, kurios yra išplėtimas CPL (Common Public License) teisių. Tad kuriamas komponentas turi atitikti EPL teisių direktyvas, ir kūrimo procesas negali jų pažeisti.

3.5. Architektūros tikslai ir apribojimai

Šiuo metu nėra numatyta galimybė įdiegti kuriamą priedą į kitas programas, todėl apsiribojame, kad priedas veiks tik Eclipse platformoje.

Eclipse programa yra parašyta JAVA programavimo kalba. Ši programavimo kalba yra objektinė programavimo kalba ir tinkama akademinių projektų įgyvendinimui. Todėl siekiant išvengti suderinamumo su kitomis programavimo kalbomis klaidų, priedas bus įgyvendintas taip pat JAVA programavimo kalba.

Kuriamo komponento architektūra yra įtakojama programos, kuriai jis bus kuriamas. Šiuo atveju tai yra Eclipse. Eclipse tai programa, kuri pagaminta pagal specialią architektūrą, kuri leidžia, priedų pagalba, praplėsti jos veikimą. Viena Eclipse programa be priedų nebūtų taikomoji programa - visas jos veikimas pagrįstas jai parašytų priedų veikimu. Taigi priedo architektūra bus pritaikyta darbui su Eclipse aplinka.



1 pav. Eclipse – priedų platforma [28].

Taip pat mūsų priedas bus gaminamas pasitelkiant EMF bei GMF įrankius, kurie sukurs jiems būdingą programos architektūrą. Mūsų tikslas sukurti tinkamas duomenų struktūras, kurios leistų išvengti atsitiktinių klaidų.

GMF (the Graphical Modeling Framework) tai įrankių rinkinys, skirtas kurti modeliavimo įrankius, tokius kaip UML redaktoriai, duomenų srautų diagramas, ir panašiai.

Mūsų kuriamas Eclipse komponentas turi sietis su UML modeliavimo komponentu Eclipse sistemoje. Tam pasirenkamas UML2tools Eclipse komponentas. Šis komponentas dirba su naujausia UML 2.2 sintakse ir palaiko bazines diagramas.

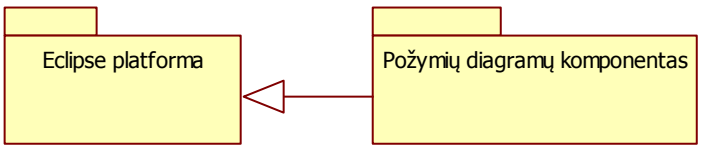
3.6. Apribojimai sprendimui

Bus sukurta ne atskira programa, bet jau egzistuojančios programos, programavimo aplinkos, priedas. Šis modulis bus kuriamas atviro kodo programavimo kalba JAVA. JAVA kalba suteiks galimybe veikti sistemai Windows bei Linux operacinėse sistemose. Pagrindiniai apribojimai, bus egzistuojančios sistemos architektūriniai sprendimai, kurie nulems sistemos kūrimo procesą.

3.7. Diegimo aplinka

Požymių diagramų braižymo įrankis bus diegiamas į Eclipse programavimo aplinką, kaip jos priedas. Kadangi Eclipse programavimo aplinka gali veikti įvairiose operacinėse sistemose, tai modulis turi atsižvelgti ir į tai, kad skirtingose operacinėse sistemose jo veikimas turi būti toks pats (Mes apsiribosime MS Windows ir Linux operacine sistema).

3.8. Sistemos statinis vaizdas

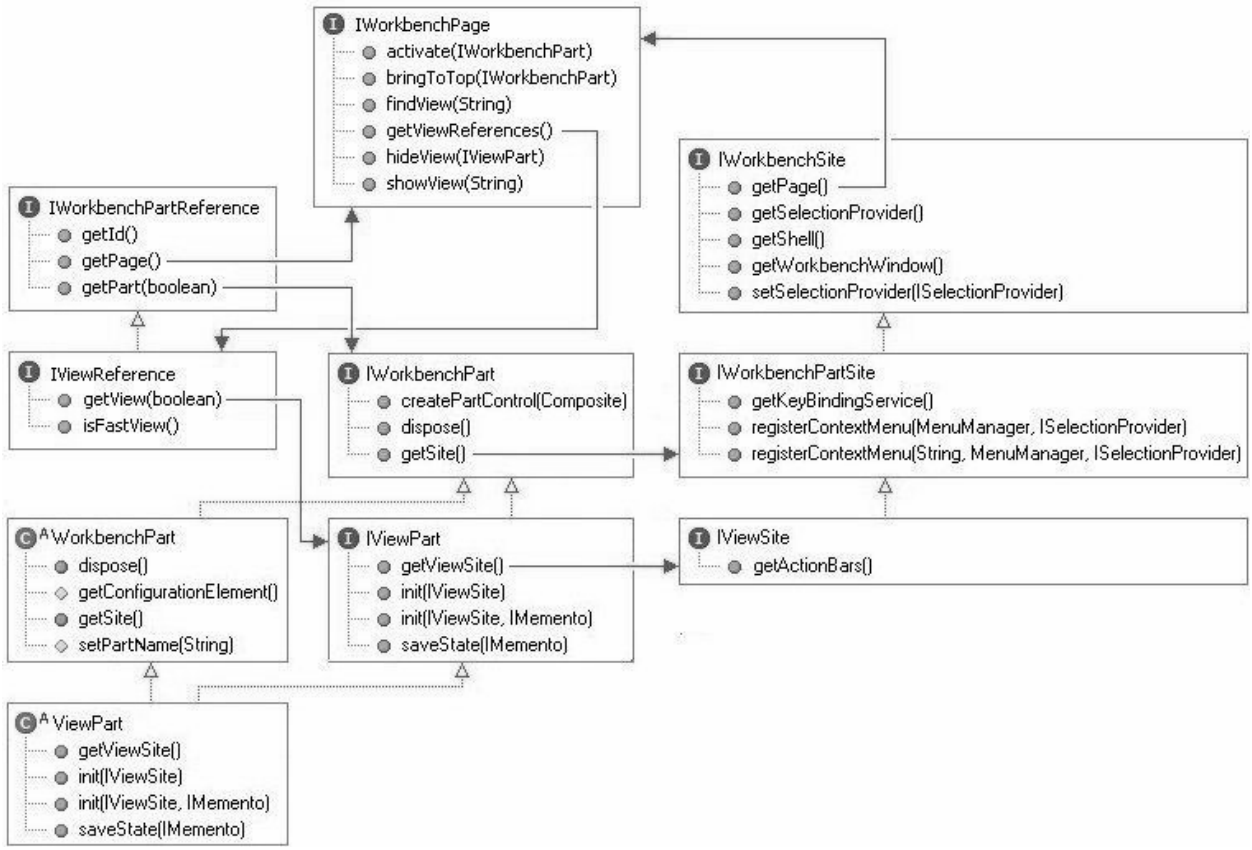


2 pav. Statinis sistemos vaizdas.

3.9. Paketas Eclipse sistema

Požymių diagramų projektui realizuoti buvo pasirinkta Eclipse sistema. Kadangi ji jau turi sukurtą UML diagramų braižymo komponentą, realizacijai bus galima jį praplėsti. Eclipse sistema yra sukurta kaip komponentų rinkinys. Kiekvienas iš jų kažkaip valdo tam tikrus vaizdus. Išvaizdos keitimas apibūdinamas kaip darbalaukio (*workbench*) keitimas.

Kuriant naują komponentą, jeigu naudojamas darbalaukis, jis turi paveldėti *org.eclipse.ui.IViewPart* sąsają. Paprastai kuriami nauji vaizdai yra klasės *org.eclipse.ui.part.ViewPart* poklasiai ir netiesiogiai klasės org.*eclipse.ui.part.WorkbenchPart* poklasiai. Taigi paveldint klasę *ViewPart* netiesiogiai yra paveldimi visi reikalingi komponentai naujo komponento vaizdo kūrimui. Šią struktūrą galima apibrėžti klasių diagrama (3 pav)



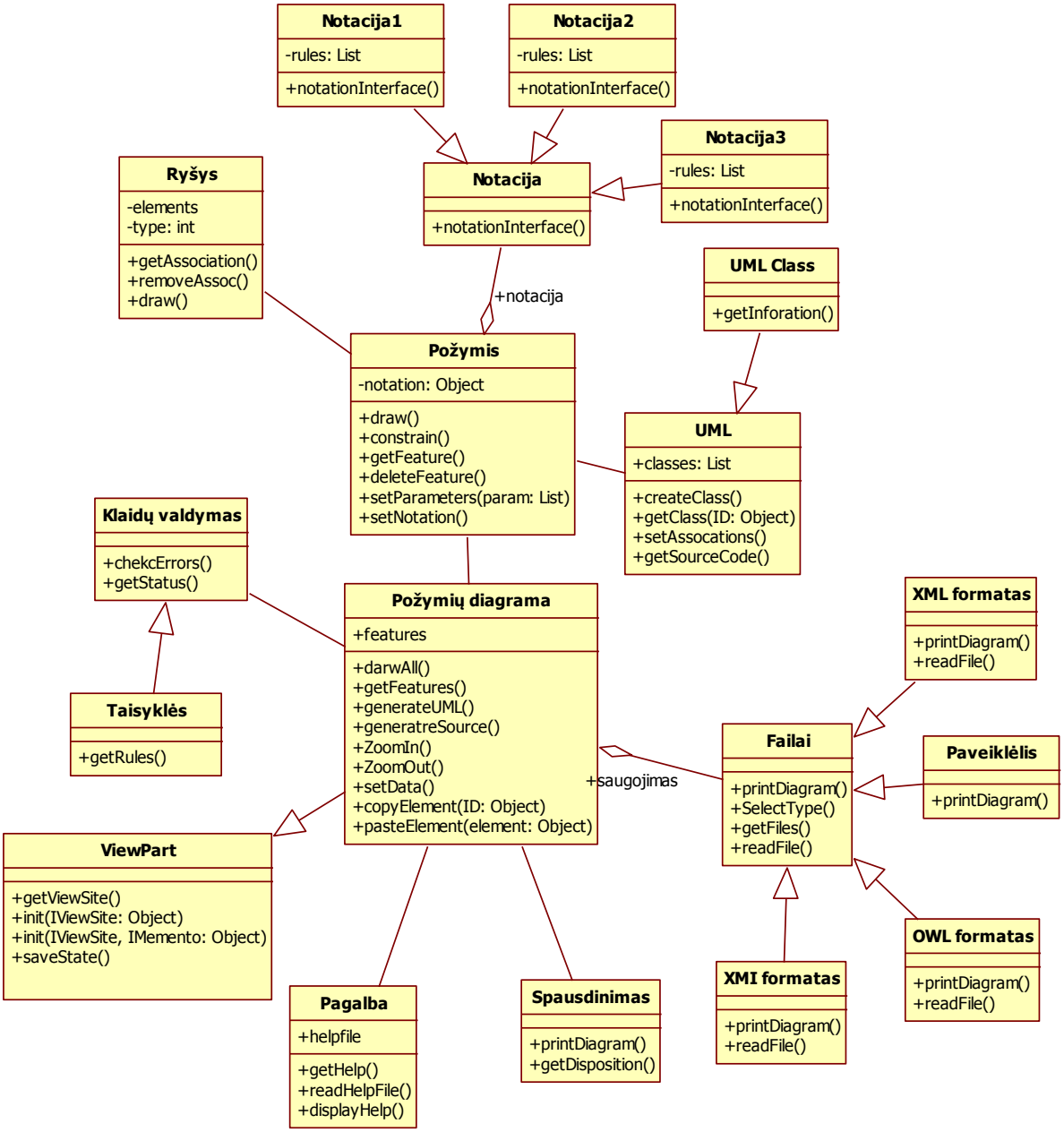
3 pav. Eclipse darbalaukio klasių diagrama [8].

Kuriant Požymių diagramų braižymo komponentą reikės paveldėti *ViewPart* klasę. Šios klasės dėka galėsime pakeisti darbalaukio išvaizdą. Darbalaukyje bus vaizduojamos diagramos ir visi jų ryšiai. Taip pat paveldint šią klasę galėsime kurti meniu juostas, iš kurių galima bus pasirinkti norimus komponentus brėžimui.

Naujo komponento įtraukimas į diagramą bus vaizduojamas grafiniame pavidale bei medžio tipo, projekto informacinėje dalyje. Kiekvienas naujas įtrauktas požymis turės savų savybių, kurias bus galima keisti, bus nurodoma ar jis privalomas produkto komplektacijoje ar ne, taip pat jam bus galima priskirti klasių diagramos elementą iš UML diagramos.

3.10. Paketas FD komponentas

Kadangi tai yra komponentas jis paveldėja dalį klasių iš Eclipse paketo. Pagrindinė klasė yra požymis, per kurią valdomas požymių, ryšių braižymas. Ji paveldi *ViewPart* klasę, kurios dėka diagramos yra atvaizduojamos Eclipse sistemos darbalaukyje (4 pav).



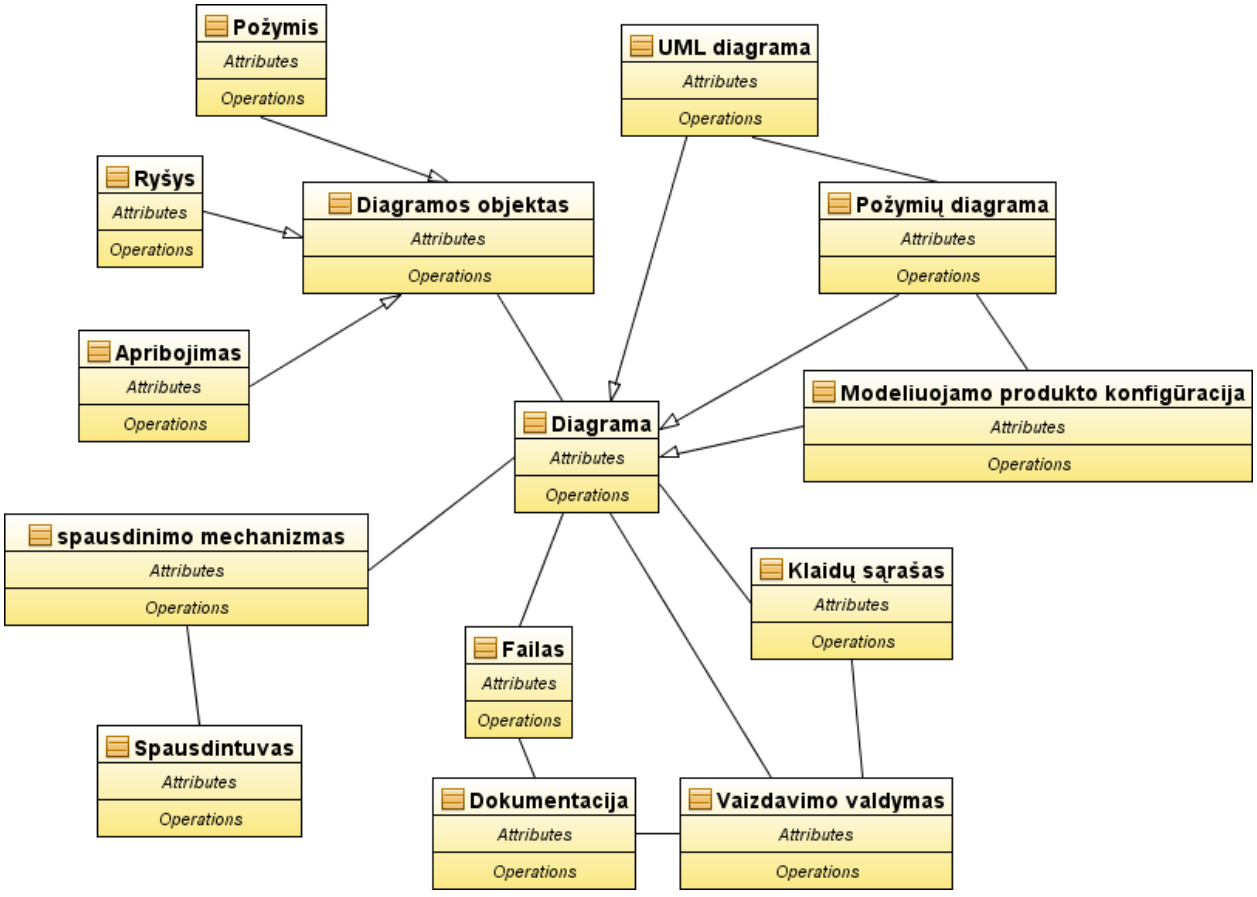
4 pav. Klasių diagrama FD komponentas.

3.11. Duomenų vaizdas

Kadangi sistema, kuriai kuriamas komponentas nėra veikianti serveryje, bei joje esami duomenys turi būti lengvai pernešami, o duomenų kiekiai nebus dideli, jų saugojimui buvo pasirinkta ne duomenų bazė, o failų sistema.

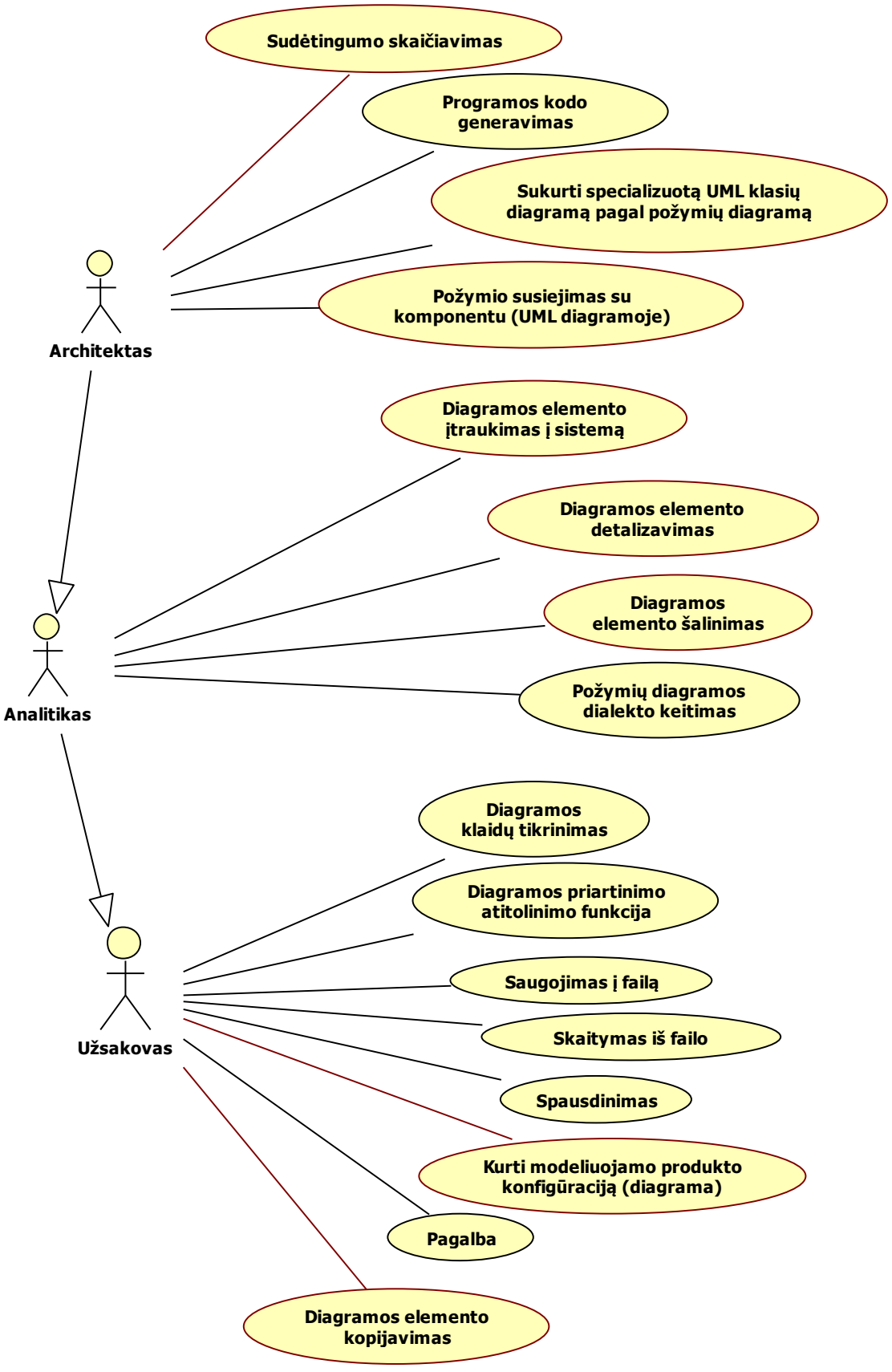
UML diagramų saugojimui yra paplitęs XML formatas. Šis formatas turi savų privalumų, lengvai skaitomas, jame saugoma informacija lengvai apdorojama. Kadangi pats XML formatas konkrečios struktūros neturi, diagramoms saugoti yra apibrėžti keli standartai, kurie nustato konkrečią failo struktūrą. Pagrindiniai standartai yra XMI ir OWL.

Žemiau yra pateikiama duomenų diagrama. Šis duomenų modelis, vaizduojamas kaip klasių diagrama. Klasių diagrama rodo klasių tarpusavio priklausomybę (5 pav).



5 pav. Duomenų diagrama.

3.12. Sistemos sudėtis



6 pav. Sistemos panaudojimo atvejai

3.13. Panaudojimo atveju aprašymai

10 lentelė. Aktoriai

Aktorius	**Pagrindinė veikla**
Užsakovas	Šis aktorius sistemoje bus laikomas kaip bazinis. Jis turi galimybę išsaugoti diagramas su kuriomis dirba kaip failus, atidaryti išsaugotus failus, spausdinti diagramas, bei peržiūrėti programos pagalbą. Tačiau šis aktorius gali modeliuoti tik produkto konfigūraciją.
Analitikas	Analitiko vaidmuo sistemoje – sudaryti tinkamą požymių diagramą, pagal kurią, bus atliekami tolimesni darbai (modeliuojama produkto konfigūracija ir projektuojama kuriamos programos architektūra).
Architektas	Šis aktorius pagal pateiktą požymių diagramą kurs UML klasių diagramą būsimai sistemai. Kiekvienas požymis bus siejamas su tam tikru komponentu, kuris bus įliejamas į bendrą UML klasių diagramą. Pagal klasių diagramą bus kuriamas pasirinktos programavimo kalbos kodas.

11 lentelė. Panaudos atvejai

Panaudos atvejis	Specifikacija
1. Pagalba	***Tikslas***: Atveriama programos pagalba vartotojui. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Langas atveriamas atskirai nuo kitų langų. ***Prieš sąlygos***: - ***Sužadinimo sąlyga***: Spaudžiamas menių punktas „Pagalba“. ***Po sąlyga***: Atvertas pagalbos langas ***Pagrindinis scenarijus***: - ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Pagalbos langas atidaromas Eclipse sistemos pagalbos priede. Duomenys saugomi HTML formate.
2. Kurti modeliuojamo produkto konfigūraciją (diagrama)	***Tikslas***: Pagal egzistuojančią požymių diagramą, kuriama konfigūracijos diagrama, kuri nurodo, kokie komponentai bus įtraukiami į naują produktą. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: - ***Sužadinimo sąlyga***: Spaudžiamas meniu punktas „Nauja programos konfigūracija. ***Po sąlyga***: ***Pagrindinis scenarijus***: Spaudžiamas meniu punktas „Nauja programos konfigūracija“, dialogo lange pasirenkama požymių diagrama, pagal kurią bus kuriama konfigūracija, pažymimos varneles ant elementų, kurie bus įtraukiami į naujai kuriamą produktą, failas išsaugomas. ***Alternatyvūs scenarijai***: Procedūra gali būti nutraukta. ***Architektūriniai ypatumai***: Atveriamas diagramos langas, kuriame galima pažymėti, kurie požymiai bus prieinami sumodeliuotoje sistemoje.

3. Spausdinimas	***Tikslas***: Peržiūrimos diagramos spausdinimas. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Spausdinama matoma programos sritis. ***Prieš sąlygos***: Atidaryti diagrama. ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „spausdinti“. ***Po sąlyga***: Atspausdinta diagrama. ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „spausdinti“. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Diagramų spausdinimas bus standartinis, kuris yra pateikiamas su GMF įrankiais. Individualus spausdinimo įgyvendinimas nebus kuriamas.
4. Skaitymas iš failo	***Tikslas***: Atidaromas diagramos failas. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: - ***Sužadinimo sąlyga***: Spaudžiamas meniu mygtukas „atidaryti“. ***Po sąlyga***: Atidaroma diagrama. ***Pagrindinis scenarijus***: Spaudžiamas meniu mygtukas „atidaryti“, nurodomas diagramos failas. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Skaitymui bus panaudotos standartines JAVA programavimo kalbos bibliotekos, galinčios apdoroti XML failus. Nuskaityti duomenys bus saugomi kompiuterio atmintyje, individualiose duomenų struktūrose. Tada duomenys atvaizduojami grafiškai.
5. Saugojimas į failą	***Tikslas***: Saugoma atidaryta diagrama į failą. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: Atidaryta diagrama. ***Sužadinimo sąlyga***: Spaudžiamas meniu mygtukas „saugoti“. ***Po sąlyga***: Diagrama išsaugoma kietame diske. ***Pagrindinis scenarijus***: Spaudžiamas meniu mygtukas „saugoti“, nurodoma vieta ir failo pavadinimas. ***Alternatyvūs scenarijai***: Galima atšaukti procesą. ***Architektūriniai ypatumai***: Iš individualių duomenų struktūrų, duomenys transformuojami ir išsaugomi XML faile. XML failų rašymui bus panaudotos standartinės JAVA programavimo kalbos bibliotekos.
6. Diagramos priartinimo atitolinimo funkcija	***Tikslas***: Leidžia didinti arba mažinti diagramos vaizdą. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: Atidaryta diagrama ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „artinti“ arba „tolinti“. ***Po sąlyga***: Norimo dydžio vaizdas ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „artinti“ arba „tolinti“, kol matomas vaizdas tenkina vartotoją.. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Ši funkcija yra įgyvendinta GMF

	įrankiuose, todėl jos įgyvendinimu programuotojams nereikės rūpintis.
7. Požymių diagramos dialekto keitimas	***Tikslas***: Galima pakeisti požymių diagramų išvaizdą. ***Aktoriai***: Analitikas ***Ryšiai su kitais PA***:- ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: - ***Sužadinimo sąlyga***: Spaudžiamas meniu mygtukas „Pasirinkti diagramos dialektą“. ***Po sąlyga***: Diagramos rodomas naudojant pasirinktą dialektą. ***Pagrindinis scenarijus***: Spaudžiamas meniu mygtukas „Pasirinkti diagramos dialektą“, iš dialogo lango pasirenkamas pageidaujamas dialektas. ***Alternatyvūs scenarijai***: Operaciją galima atšaukti. ***Architektūriniai ypatumai***: Diagramų vaizdavimas bus įgyvendintas panaudojant GMF bibliotekų rinkinius. Daugelis vaizdavimo galimybių bus priklausoma nuo GMF bibliotekų suteikiamų galimybių.
8. Diagramos elemento šalinimas	***Tikslas***: Pašalinamas pažymėtas diagramos elementas. ***Aktoriai***: Analitikas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: Atidaryta diagrama ir pažymėtas jos elementas. ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „trinti“ arba klaviatūros mygtukas „delete“. ***Po sąlyga***: Ištrintas diagramos elementas. ***Pagrindinis scenarijus***: Pažymimas diagramos elementas, spaudžiamas mygtukas „trinti“ arba klaviatūros mygtukas „delete“. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Šios operacijos metu atliekama daug veiksmų priklausančių vienas nuo kito. Visu pirma patikrinama ar galima elementą pašalinti iš sistemos, tada pašalinamas ir tikrinama ar atsiranda modelyje klaidų. Atsiradus klaidų, vartotojas gali atšaukti atlikta veiksmą.
9. Diagramos elemento detalizavimas	***Tikslas***: Galima pakeisti tam tikrus elementų požymius. ***Aktoriai***: Analitikas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Pažymėtas diagramos elementas. ***Prieš sąlygos***: Atidaryta diagrama ir pažymėtas diagramos elementas ***Sužadinimo sąlyga***: Pažymimas diagramos elementas ir požymių kortelėje keičiami nustatymai. ***Po sąlyga***: Pakeisti diagramos elemento parametrai. ***Pagrindinis scenarijus***: Pažymimas diagramos elementas ir požymių kortelėje keičiami nustatymai. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Diagramos elemento detalizavimas bus įgyvendintas pasinaudojant GMF bibliotekų rinkiniu. Kiekvienas grafinis elementas turi savo požymius.
10. Diagramos elemento įtraukimas į sistemą	***Tikslas***: Pasirinkto diagramos elemento įtraukimas į sistemą. ***Aktoriai***: Analitikas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: -

	Prieš sąlygos: Atidaryta diagrama. ***Sužadinimo sąlyga***: Pasirenkamas įrankis ir dedamas į diagramą. ***Po sąlyga***: Į diagramą įterptas pasirinktas elementas. ***Pagrindinis scenarijus***: Pasirenkamas elemento įrankis pelyte, paspaudus ant diagramos įtraukiamas pasirinktas elementas. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: GMF bibliotekoje yra įgyvendinti pagrindiniai įrankiai. Jie bus susieti su sukurtomis duomenų struktūromis, ir turės funkcionuoti, pagal numatytą logiką duomenų struktūrose.
11. Diagramos klaidų tikrinimas	***Tikslas***: Patikrinti ar diagramoje yra klaidų. ***Aktoriai***: Užsakovas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: - ***Prieš sąlygos***: Atidaryta diagrama ***Sužadinimo sąlyga***: Spaudžiamas meniu punktas „Tikrinti diagramos klaidas“. ***Po sąlyga***: Pateiktos diagramos klaidos. ***Pagrindinis scenarijus***: Spaudžiamas meniu punktas „Tikrinti diagramos klaidas“, tada pasirodo klaidų sąrašas. ***Alternatyvūs scenarijai***: - ***Architektūriniai ypatumai***: Klaidų tikrinimas bus atliekamas visoms diagramoms. Bus tikrinama ar teisingai suformuota diagramos sintaksė, ar tinkamai susieta požymių diagrama su klasių diagramomis, ... Diagramos, kuriose bus aptikta klaidų bus pažymimos klaidos simboliu, taip bus pranešama, kad reikia atkreipti dėmesį į tam tikrus elementus.
12. Požymio susiejimas su komponentu	***Tikslas***: Susieti požymį su konkrečiu programos komponentu. ***Aktoriai***: Architektas. ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Požymis gali turėti tik 1 komponentą ir atvirkščiai. ***Prieš sąlygos***: Į sistemą įtraukta UML komponentų diagrama ir nurodyta požymių diagrama. ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „Susieti požymį su komponentu“. ***Po sąlyga***: Susietas komponentas su požymiu. ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „Susieti požymį su komponentu“, iš pasirodžiusio dialogo pasirenkamas požymis, iš sekančio dialogo pasirenkamas komponentas. ***Alternatyvūs scenarijai***: Ši operacija gali bet kada būti nutraukta. ***Architektūriniai ypatumai***: Kiekvienam diagramos požymiui bus sukurta UML klasių diagrama, kaip komponentas. Klasių diagrama turės įgyvendinti tam tikras sąsajos klases. Sąsajų klasės, kurios turės įeiti į komponentą, bus aprašytos kuriamos programos specifikacijoje. Šios sąsajos klasės bus tikrinamos klaidų tikrinime. Sąsaja su požymiu bus įgyvendinta vedlio pagalba kiekvienam požymiui. Vedliui reikės nurodyti su kuriuo UML komponentu bus susietas požymis.
13. Sukurti specializuotą UML klasių	***Tikslas***: Sukuriama UML klasių diagrama. ***Aktoriai***: Architektas. ***Ryšiai su kitais PA***: -

diagramą pagal požymių diagramą	***Nefunkciniai reikalavimai***: Turi nebūti klaidų diagramose. ***Prieš sąlygos***: Į sistemą įtraukta UML komponentų diagrama, požymių diagrama ir modeliuojamo įrankio požymių konfigūracija. Požymiai susieti su komponentais. ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „Generuoti klasių diagramą“. ***Po sąlyga***: Sugeneruota klasių diagrama. ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „Generuoti klasių diagramą“, pasirodžiusiame dialoge pasirenkama požymių konfigūracija. ***Alternatyvūs scenarijai***: Ši operacija gali bet kada būti nutraukta. ***Architektūriniai ypatumai***: Kiekvienam diagramos požymiui bus sukurta UML klasių diagrama, kaip komponentas. Klasių diagrama turės įgyvendinti tam tikras sąsajos klases. Sąsajų klasės, kurios turės įeiti į komponentą, bus aprašytos kuriamos programos specifikacijoje. Šios sąsajos klasės bus tikrinamos klaidų tikrinime.
14. Programos kodo generavimas	***Tikslas***: Programinio kodo generavimas. ***Aktoriai***: Architektas. ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Turi nebūti klaidų diagramose. ***Prieš sąlygos***: Turi būti sukurta klasių diagrama. ***Sužadinimo sąlyga***: Spaudžiamas mygtukas „Generuoti kodą“. ***Po sąlyga***: Sugeneruojamas programinis kodas. ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „Generuoti kodą“, dialoge pasirenkama kalba, kuria bus generuojamas programinis kodas. ***Alternatyvūs scenarijai***: Ši operacija gali bet kada būti nutraukta. Nepilnai sukurtas kodas ištrinamas. ***Architektūriniai ypatumai***: Programos kodo generavimas bus galimas tik susiejus UML komponentų diagramas su požymių diagramomis. Programos kodas bus sugeneruotas pagal parinktas UML komponentų diagramas. Pradžioje sukuriama visos programos UML diagrama, o iš jos sugeneruojamas programinis kodas (JAVA programavimo kalba).
15. Diagramos elemento kopijavimas	***Tikslas***: Elemento kopijavimas ir tokio pačio įterpimas. ***Aktoriai***: Analitikas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Turi nebūti klaidų kopijuojamame elemente. ***Prieš sąlygos***: pažymimas 1 elementas, kurį norime nukopijuoti. ***Sužadinimo sąlyga***: Spaudžiama klaviatūros kombinacija, arba programos mygtukas su funkcija „Kopijuoti“. ***Po sąlyga***: Įterpiamas naujas diagramos elementas. ***Pagrindinis scenarijus***: Spaudžiamas mygtukas „Kopijuoti“, pažymima diagrama į kurią bus įdėtas kopijuojamas elementas ir kurioje vietoje jis prijungiamas. ***Alternatyvūs scenarijai***: Ši operacija gali bet kada būti nutraukta. ***Architektūriniai ypatumai***: Elementas bus kopijuojamas iš vienos diagramos į kitą. Kopijavimo metu bus nuskaitomas visas to elemento medis ir nukopijuojamas į kita diagramą, į nurodytą vietą. Tai bus atliekama vedlio pagalba.
16. Sudėtingumo	***Tikslas***: Suskaičiuoti ir pateikti charakteristikas apie nubrėžtą

skaičiavimas	diagramą. ***Aktoriai***: Architektas ***Ryšiai su kitais PA***: - ***Nefunkciniai reikalavimai***: Sistema turi pateikti brėžiamos diagramos charakteristikas. ***Prieš sąlygos***: Nubrėžiama požymių diagrama ir užsaugoma. ***Sužadinimo sąlyga***. Įjungus sudėtingumo peržiūros langą spaudžiama „Skaičiuoti“ ***Po sąlyga***: Sudėtingumo peržiūros lange pateikiamos požymių diagramos sudėtingumo informacija. ***Pagrindinis scenarijus***: Nubrėžiama diagrama, užsaugoma, atidaromas sudėtingumo peržiūros langas, spaudžiamas mygtukas „Skaičiuoti“, rezultatai pateikiami peržiūros lange. ***Alternatyvūs scenarijai***: Ši operacija gali bet kada būti nutraukta. ***Architektūriniai ypatumai***: Bus skaitomas diagramos medis ir atliekami keli sudėtingumo skaičiavimo algoritmai, bei surenkami ir pateikiami sąraše naudojami duomenys.

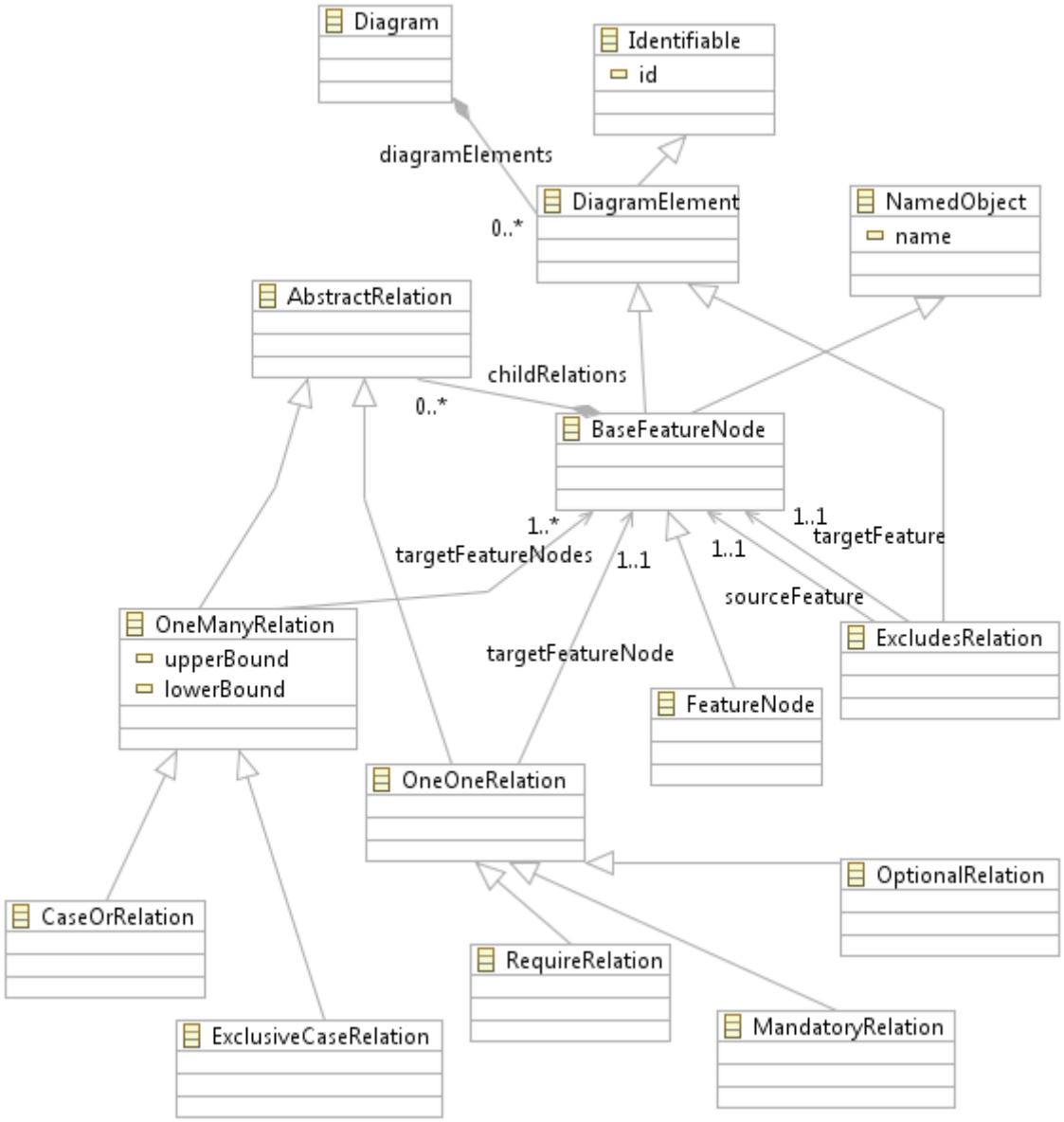
4. POŽYMIŲ DIAGRAMŲ METAMODELIS IR JO ELEMENTAI

4.1. Siūlomas požymių diagramų metamodelis

Požymių diagramos metamodelis naudojamas požymių diagramos sintaksei aprašyti. Literatūroje galima rasti kelis įvairių autorių pasiūlytus ir aprašytos požymių diagramų metamodelius [10] [2] [38]. Paprastai metamodeliai yra kuriami konkrečiam realizuojamam požymių diagramų sintaksės variantui. Toliau pateikiamas šiame darbe tiriamo požymių diagramos sintaksės varianto metamodelis.

Požymių diagrama susideda iš požymių ir juos jungiančių ryšių. Požymiai apibrėžia tam tikras sistemos dalis, komponentus, o ryšiai jų atributus, tarpusavio priklausomybes. Norint taisyklingai brėžti požymių diagramas reikia apsibrėžti tam tikrą metodologiją, pagal kurią galima būtų aiškiai ir taisyklingai modeliuoti diagramas.

Žemiau yra pateikiamas siūlomas metamodelis požymių diagramoms vaizduoti ir modeliuoti (žr. 7 pav).



7 pav. Požymių diagramos metamodelis

Toliau detaliau paaiškinami atskiri požymių diagramos metamodelio elementai ir pateikiamas jų grafinis žymėjimas naudojant standartinę notaciją. Notacija naudojama pasiūlyta Eisenecker ir Czarnecki notacijos požymių vaizdavimas (žr. 2.2.7), papildant ją EDF notacijos reikalavimo ryšiu (žr. 2.2.8).

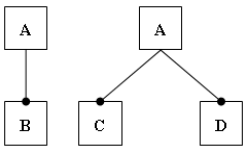
Požymiai identifikuojami pavadinimu ir vaizduojami kvadratėlyje (žr. 8 pav).

Požymis

8 pav. Požymis

Požymius gali jungti ryšiai. Ryšiai yra naudojami 6 tipų:

1. Būtinasis ryšys – jis apibrėžia kad elementas yra privalomas įtraukti į konfigūracija, jeigu jo tėvas su kuriuo yra būtinasis ryšys, yra naudojamas (žr. 9 pav).



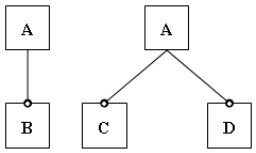
9 pav. Būtinasis ryšys

Požymis B (C,D) yra įtrauktas, jeigu jo tėvas A yra įtrauktas

***Jei** A **tuomet** B;*

***Jei** A **tuomet** B ir C*

2. Nebūtinas ryšys – jis apibrėžia kad elementas gali būti įtrauktas į konfigūraciją jeigu jo tėvas su kuriuo yra būtinasis ryšys, yra naudojamas (žr. 10 pav).



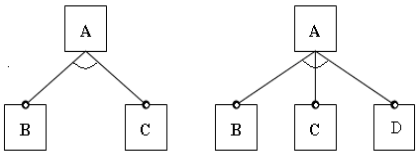
10 pav. Nebūtinas ryšys

Požymis B(C,D) gali būti įtrauktas jeigu jo tėvas A yra įtrauktas

***Jei** A **tuomet** B arba niekas;*

***Jei** A **tuomet** C arba niekas*

3. Pasirenkamas (*case* - pasirinkimas) – jis apibrėžia, kad iš grupės elementų galima įtraukti į konfigūraciją tik vieną elementą jeigu jų tėvas yra įtrauktas į konfigūraciją (žr. 11 pav).

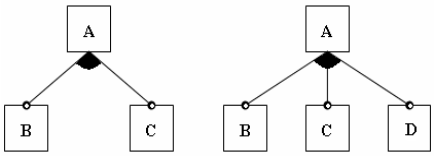


11 pav. Pasirenkamas (*case* - pasirinkimas)

Tik vienas požymis B arba C arba D turi būti pasirinktas jeigu jo tėvas A yra parinktas

***Jeigu** A **tuomet** case iš(B,C,D)*

4. Pasirenkamas (*OR* - pasirinkimas) – jis apibrėžia, kad iš grupės elementų gali būti į konfigūraciją įtrauktas ne mažiau kaip vienas elementas jeigu jų tėvas įtrauktas į konfigūraciją (žr. 12 pav).

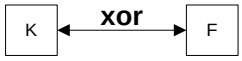


12 pav. Pasirenkamas (*OR* - pasirinkimas)

Mažiausiai vienas požymis (B,C arba D arba B ir C; arba B ir D; arba C ir D; arba B ir C ir D) turi būti pasirinkti jeigu jo tėvas A yra pasirenkamas

***Jei** A **tuomet** bet kuris iš(B,C,D)*

5. Ryšys *XOR* – jis apibrėžia ryši tarp dviejų atominių požymių, nurodydamas jog įtraukus vieną požymį negalima įtraukti nurodytojo požymio (žr. 13 pav).

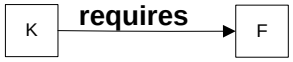


13 pav. Ryšys *XOR*

Ryšys tarp dviejų atominių požymių. K XOR F.

***Jei** F **tuomet** ne K ir **jei** ne F **tuomet** K*

1. Rekalavimas (*requiers)* – jis apibrėžia ryši tarp dviejų atominių požymių, nurodydamas jog įtrauktas vienas požymis reikalauja kad būtų įtrauktas nurodytas požymis (žr. 14 pav).



14 pav. Reikalavimas (*requires*)

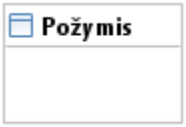
Požymis K reikalauja požymio F.

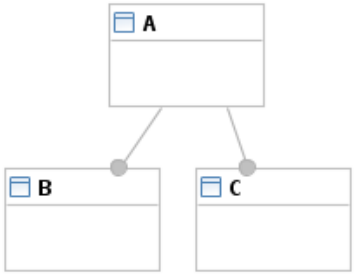
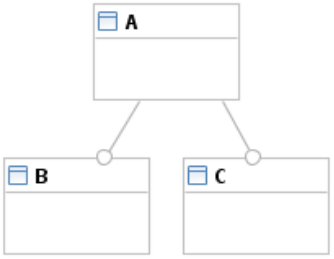
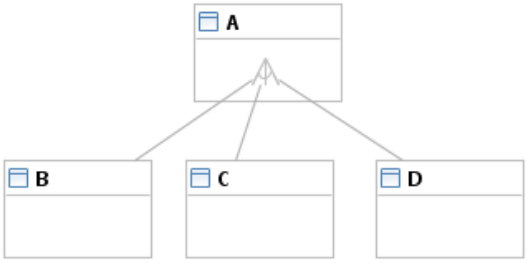
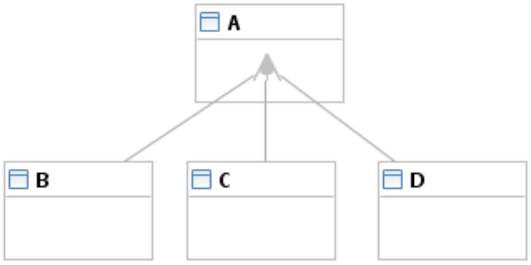
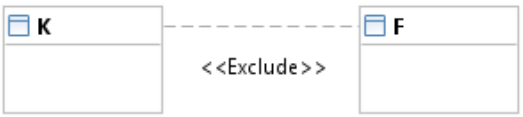
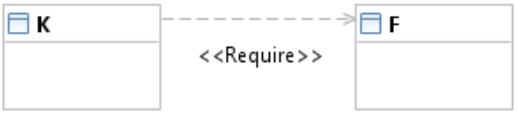
*K **reikalauja** F*

4.2. Realizuoto FD sintaksės varianto aprašymas

Požymių diagramos sintaksė buvo realizuota dviem būdais. Pirmas variantas buvo realizuotas pritaikant grafinį vaizdavimą, kuomet elementai yra laisvai dedami norimoje ekrano pozicijoje ir interaktyviai jungiami tarpusavyje ryšiais įtraukiant juos į braižymo aplinką iš įrankių juostos pelės pagalba. Žemiau yra pateikiama lentelė su realizuota požymių diagramų sintakse (žr. 12 lentelė).

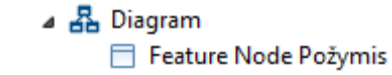
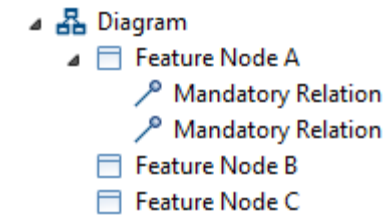
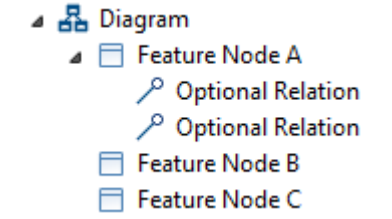
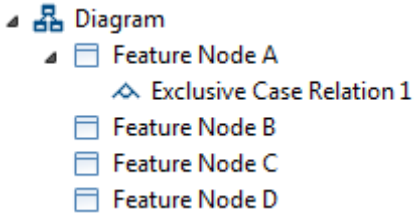
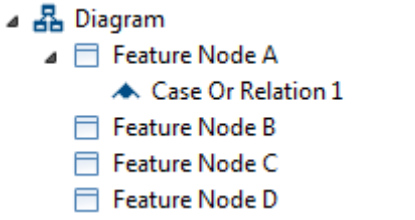
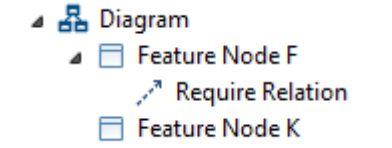
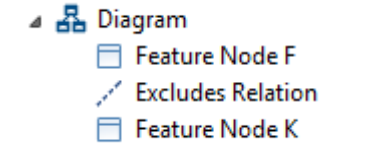
12 lentelė. Realizuoto FD sintaksės pirmo varianto aprašymas

Elementas	**Grafinės notacijos aprašas**	**Pavyzdys**
Požymis	Pilkas kvadratėlis su pavadinimu ir paveiksliuku. Po pavadinimu yra laukelis pasirenkamumo elementui.	

Būtinas	Ryšiai jungiami pilka vientisa linija, kurios gale yra pilnaviduris rutuliukas, parodantis, kad elementas yra privalomas.	
Nebūtinas	Ryšiai jungiami pilka vientisa linija, kurios gale yra tuščiaviduris rutuliukas, parodantis, kad elementas nėra privalomas, jį galima pasirinkti arba ne.	
Pasirenkamas (*case* - pasirinkimas)	Tėvo elementui dedamas *case* pasirinkimo elementas , kuris yra tuščiaviduris. Prie jo yra prijungiami požymiai vientisa pilka linija.	
Pasirenkamas (*OR* - pasirinkimas)	Tėvo elementui dedamas *OR* pasirinkimo elementas , kuris yra pilnaviduris. Prie jo yra prijungiami požymiai vientisa pilka linija.	
Ryšys (X*OR*)	Elementai yra jungiami pilka punktyrine linija, prie kurios pridedama antraštė <<Exclude>>	
Reikalavimas (*requires*)	Elementai yra jungiami pilka punktyrine rodykle, prie kurios pridedama antraštė <<Require>>	

Antrasis variantas buvo realizuotas, požymius vaizduojant medžio tipo grafiku. Ši notacija skirta peržiūrėti duomenų modelį ir yra mažiau informatyvi vizualiai. Elementai yra įtraukiami į medį ant pasirinktos šakos pasirinkus vaiko tipo elemento rūšį (žr. 13 lentelė).

13 lentelė. Realizuoto FD sintaksės antro varianto aprašymas

Elementas	**Grafinės notacijos aprašas**	**Pavyzdys**
Požymis	Požymis vaizduojamas su kvadratėlio ikonėle, jo tėvo elementas yra diagrama.	
Būtinas	Ryšys vaizduojamas su ikonėle, kurioje linija su pilnaviduriu rutuliuku ikonėle. Elementai, kurie priklauso šiam ryšiui yra matomi parametrų lange.	
Nebūtinas	Ryšys vaizduojamas su ikonėle, kurioje linija su tuščiaviduriu rutuliuku ikonėle. Elementai, kurie priklauso šiam ryšiui yra matomi parametrų lange.	
Pasirenkamas (*case* - pasirinkimas)	Ryšio elementas vaizduojamas tuščiaviduriu ketvirtadaliu apskritimo ikonėle. Elementai priklausantys šiam ryšiui išvardinami sąrašu parametrų lange.	
Pasirenkamas (*OR* - pasirinkimas)	Ryšio elementas vaizduojamas pilnaviduriu ketvirtadaliu apskritimo. Elementai priklausantys šiam ryšiui išvardinami sąrašu parametrų lange.	
Ryšys (*XOR*)	Ryšys vaizduojamas punktyrinės rodyklės ikonėle. Vaiko elementas nurodomi parametrų lange	
Reikalavimas (*requires*)	Ryšys vaizduojamas punktyrinės linijos ikonėle. Siejami elementai nurodomi parametrų lange.	

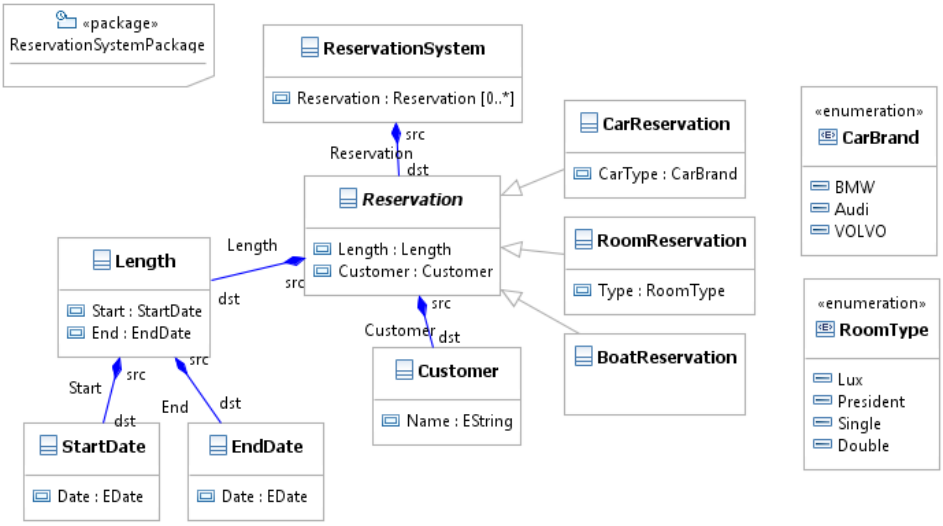
4.3. Siūloma produktų linijų projektavimo metodika naudojant požymių diagramas

Produktų linija gali būti apibrėžiama kaip tarpusavyje susijusių produktų rinkinys, kurie pateikia skirtingą funkcionalumą. Autoriaus pasiūlymu požymių diagramomis galima modeliuoti sistemos dalis, naudojant požymius. Sistemos elementas tai modulis, tam tikras sistemos komponentas su savo funkcionalumu.

Darbe aprašytoje produktų linijų projektavimo metodikoje siūloma išplėsti tradicinę objektinio sistemų projektavimo metodiką požymių modeliavimo ir sistemų konfigūravimo elementais.

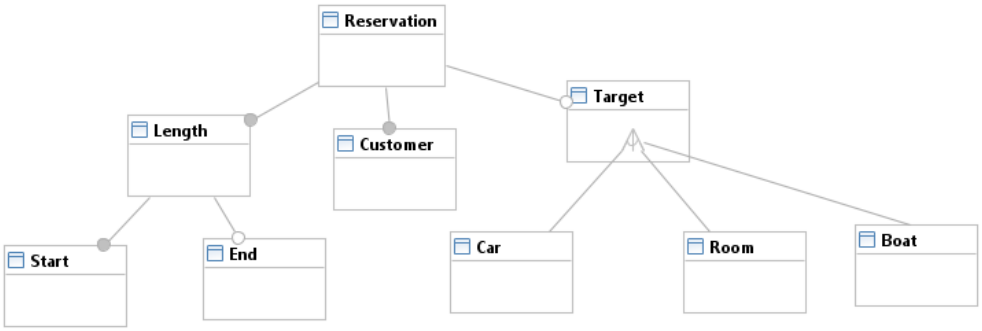
Aiškiausiai suprantama ir matoma produkto konfigūracija yra statiniame sistemos modelyje – UML klasių diagramoje. Požymius galima sieti su pavieniais UML klasių diagramos elementais ir pateikti tą patį sistemos vaizdą, aukštesniame abstrakcijos lygmenyje, nesigilinant i konkrečią komponento funkcijų realizaciją, sistemos modelį, kuriame būtų matoma informacija apie komponentų priklausomybę.

Žemiau yra pateikiama UML klasių diagrama nesudėtingos rezervavimo sistemos (žr. 15 pav), kurioje yra keli komponentai – tai užsakovas, kuris atlieka rezervaciją, rezervacijos trukmė bei rezervacijos objektas.



15 pav. Rezervacijų sistemos UML klasių diagrama

Susiejus atitinkamus sistemos komponentus, kurie gali kisti, gaunama požymių diagrama kurioje galima matyti elementų priklausomybė ir konfigūravimo galimybės (žr. 16 pav)



16 pav. Rezervacijų sistemos Požymių diagrama

Šiuo atveju siejami konkretūs sistemos elementai apibrėžti klase su požymiu, ir sujungiami ryšiu priklausomai nuo svarbos elemento tai sistemai.

Taip pat požymius galima sieti ir su UML panaudojimo atvejų modelį, tuomet požymį reikėtų apibrėžti kaip panaudojimo atvejį ir šiuo atveju būtų konfigūruojama naujos sistemos panaudojimo atvejų modelis.

Apibendrinant, siūloma tokia programų sistemų produktų linijų projektavimo metodika:

1. Srities analizė ir reikalavimų sistemai formulavimas. Rezultatas: sistemos reikalavimų dokumentas.
2. Sistemos požymių identifikavimas ir sistemos požymių sąrašo sudarymas. Rezultatas: sistemos požymių dokumentas.
3. Bendrinės sistemos, turinčios visus požymių variantus, projektavimas taikant objektinio projektavimo principus. Rezultatas: bendrinės sistemos objektiniai modeliai (UML diagramos).
4. Sistemos objektinių modelių variantiškumo analizė, sąryšių tarp sistemos požymių identifikavimas ir požymių modelio(-ių) sudarymas. Rezultatas: sistemos požymių diagrama(-os).
5. Sistemos konfigūravimas ir konkrečių produktų objektinių modelių generavimas. Rezultatas: konkretizuotis sistemos objektiniai modeliai.
6. Tolesnis sistemos kūrimas naudojant automatinio kodo generavimo ir objektinio programavimo metodus. Rezultatas: konkretus sistemos produktų linijos vienetas (egzempliorius).

Pastaba: sukurtame požymių modeliavimo įrankyje realizuotas ir darbe ištirtas tik produktų linijų kūrimas naudojant tik vieną objektinį modelį – UML klasių diagramas.

4.4. Atskiros produktų linijų projektavimo problemos ir siūlomi jų sprendimo metodai

4.4.1. Požymių modelių specifikavimas

Specifikuojant požymių diagramas kartais yra sunku suskirstyti sistemos dalis į tam tikrus požymius. Kartais nėra aišku ar požymis yra privalomas ar pasirenkamas duotoje sistemoje.

Požymį galima apibrėžti kaip matomą sistemos charakteristiką, dažniausiai tai būna komponentas, tam tikras modulis arba tam tikro sistemos funkcionalumo dalis. Tai iš esmės yra abstrakcija arba produkto charakteristika kurią abu, tiek užsakovas tiek ir programuotojas supranta [17].

Svarbu apibrėžti kas yra geras požymis. Ne viskas kas galėtų būti požymis yra požymis. Požymio apibrėžimas turėtų būti aiškus ir išraiškus. Požymiai dažniausiai naudojami išskirti pasirinkimo galimybes sistemoje, bet ne funkcionalumo detalių apibrėžimui [11].

Jeigu sistema jau yra suprojektuota ir norima sumodeliuoti produktų linijos variantiškumą, atskirais požymiais galima naudoti UML klasių diagramos elementus, ko pasekoje sukonfigūravus produktą, nesudėtingai galima paruošti ir UML klasių diagramą pagal požymių diagramą.

4.4.2. Požymių modelių validavimas

Požymių diagramų sudarymo teisingumo, t.y. požymių modelių validavimo problema yra nauja ir aktuali tyrimų kryptis, kuri tik neseniai pradėta vystyti [15].

Pagrindinė problema yra kaip patikrinti iš sistemos požymio modelio pasirinktų sistemos požymių aibės teisingumą, t.y., ar požymis nėra pats sau tėvas, ar visi būtini sistemos požymiai yra pasirinkti, ar yra pasirinktas nors vienas pasirenkamas požymis, ar pasirinktas tik vienas alternatyvinis požymis, ar būtinai pasirinktas tik vienas elementas, susietas išskirtinumo (XOR) sąryšiu. Požymių modelių validavimas aprašomas formuluojant modelių validavimo taisyklių rinkinį ir pateikiant tos taisyklės teisingumo patikrinimo algoritmą (žr. žemiau).

Požymių diagramos vaizduojamos kaip aciklinis grafas. Kadangi grafas brėžiamas laisvai dėliojant šakas, susiduriama su keliomis problemomis, todėl yra apibrėžiamos kelios validavimo taisyklės, kurios apsaugo nuo galimų klaidų:

1. Taisyklė

   Požymis negali turėti ryšio pats su savimi.

   Algoritmas (pseudo kodas)

```
while (currentFeature has child){
     if(child equals currentFeature){
           exit "KLAIDA"
     }
}
```

   Algoritmas (OCL)

```
self <> oppositeEnd
```

Šiai taisyklei patikrinti yra naudojamas nesudėtingas algoritmas, kuris atlieka požymio vaikų elementų perrinkimą ir patikrina ar jie nėra identiški tėvo elementui.

2. Taisyklė

   Požymis negali turėti ryšio su aukštesnėje hierarchijoje esančiu elementu (jeigu tai ne XOR ar reikalavimo ryšys).

   Algoritmas (pseudo kodas)

```
validateCicle(Feature, firstElement){
    while (Feature has child){
          if(child equals firstElement)
             if(child has RequireConnector with firstElement OR
                       child has ExcludeConnector with firstElement){
                  return;
             } else {
                  return "KLAIDA";
             }
          } else {
                  return validateCicle(child, fistElement);
          }
    }
    return;
}
```

   Algoritmas (OCL)

```
self <> oppositeEnd and not self.eAllSuperTypes->
                                                    includes(oppositeEnd)
```

Šiai taisyklei patikrinti yra naudojama rekursija, kurios dėka perbėgama per visą šaką ir patikrinama ar nėra susidariusių ciklų medyje, kurių priežastis būtų ne *XOR* ar reikalavimo (*require*) ryšys.

3. Taisyklė

   Pasirenkami „case“ arba „OR“ ryšiai turi turėti bent vieną elementą.

   Algoritmas (pseudo kodas)

```
if(FeatureNodeContainmentElement is CaseOrRelation){
     if(FeatureNodeContainmentElement has not children)
          exit "KLAIDA"
}
if(FeatureNodeContainmentElement is ExclusiveCaseRelation){
     if(FeatureNodeContainmentElement has not children)
          exit "KLAIDA"
}
```

   Algortimas (OCL)

```
targetFeatureNodes->size()>0
```

Šiai taisyklei patikrinti naudojamas nesudėtingas algoritmas, patikrinama ar elementas „case“ arba „OR“ turi vaikų.

4.4.3. Požymių modelių sudėtingumo charakteristikų įvertinimas

Sistemos nuolat auga ir tampa vis sudėtingesnės, nes tobulėja technologijos, auga vartotojų reikalavimai, didėja vartojamumas. Sudėtingumo valdymas gali padėti padidinti programinės įrangos kokybę ir produktų suprantamumą taip pat sumažina projektavimo klaidų skaičių [5] ir trumpina jų kūrimo laiką. Valdyti sudėtingumą, reiškia visų pirma žinoti kaip jį išmatuoti. Sudėtingumo matavimas leidžia spręsti apie sistemos struktūrą, suprasti sistemos elgseną, lyginti bei vertinti sistemas arba įžvelgti jų evoliuciją.

4.4.3.1. Sudėtingumo vertinimo esminiai aspektai

Yra du požiūriai į sudėtingumą [31]: sudėtingumas kaip „sunku ištestuoti“ (pvz. skaičius testavimo atvejų reikalingų pilnam sistemos ištestavimui), ir sudėtingumas kaip „sunku suprasti modelį“. Dar vadinamas modelio *pažinimo sudėtingumu*.

Pažinimo sudėtingumas koncentruojasi analize kiek problema yra sudėtinga žiūrint iš perspektyvos asmens, kuris stengiasi išspręsti problemą. Pažinimo sudėtingumas susijęs su trumpalaikės atminties ribomis, kurios yra skirtingos kiekvienam žmogui ir priklauso nuo informacijos tipo [39].

Modelyje aukšto lygio pažinimo sudėtingumas veda prie didesnės rizikos padaryti klaidas, kas veda prie žemesnės programinės įrangos kokybės. Požymių diagramų modelis vaizduojamas kaip požymių medis ir jis turi pažinimo sudėtingumą.

Remiantis Miller ankstesniais darbais [9] apibrėžiant, kad žmogus gali vienu metu įsisavinti 7 (+/- 2) porcijas informacijos savo trumpalaikėje atmintyje. Taip pat naudojant Keating taisykle, kuri yra paremta Miller projektavimo sričiai: „Modulių kiekis kiekviename hierarchijos lygyje turi būti 7 +/- 2“ [21]. Buvo išvestos šios taisyklės, kurios apibrėžia požymių diagramų sudėtingumą:

1. Taisyklė.

Požymių diagramos pažinimo sudėtingumas yra skaičiuojamas kaip skaičius pasirinkimo taškų požymių diagramoje. Pasirinkimo taškų skaičius požymių diagramoje turi būti 7 +/- 2, jeigu projektuotojas nori išvengti aukšto pažinimo sudėtingumo pasekmių. Jeigu pasirinkimo taškų yra mažiau nei 5, modelis turėtų būti skaidomas į smulkesnes dalis, kadangi jis nėra pakankamai aiškus ir kai kurios svarbios dalys yra slepiamos. Jeigu pasirinkimo taškų yra daugiau nei 9, vartotojas turėtų modelį išskaidyti į smulkesnius modelius, kad tilptų į pažinimo sudėtingumo ribas.

2. Taisyklė.

Pažinimo sudėtingumas skaičiuojamas kaip maksimalus požymių diagramos lygių skaičius arba maksimalus skaičius dalių (grafo lygių) kiekviename hierarchijos lygyje.

3. Taisyklė.

Struktūrinis požymių diagramos sudėtingumas su pasirinkimo taškais gaunamas susumavus visus žemesnius medžio medžius (angl. sub-trees), kuomet kiekviename pasirinkimo taške pasirenkamas tik vienas variantas. Kiekvienas medis gaunamas iš pagrindinio medžio.

3 taisyklė apibrėžia struktūrinį požymių modelio sudėtingumą, t.y. galimybę apibrėžti skirtingas produkto dalis produktų linijoje. 3 taisyklė yra susijusi su ciklomatiniu skaičiumi, gerai žinomu matmeniu naudojamu programų sudėtingumui skaičiuoti [34].

Žemiau aprašoma 4 taisyklė apibrėžia kaip sudėtinis sudėtingumas ir struktūrinis sudėtingumas turėtų būti apjungiami. Jis paremtas Metcalfe empiriniu dėsniu ir Keating dėsnio pritaikymu projekto skaidymo sudėtingumo skaičiavimui [21].

Metcalfe dėsnis teigia, kad tinklo „galia“ yra lygi jame esančių šakų kvadratui, o „vertė“ lygi tinkle esančių briaunų kvadratui. Keating formulė:

$$C = M^2 + I^2$$

kur:

C – projektavimo sudėtingumas

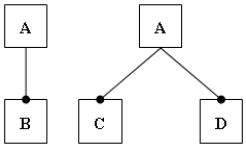
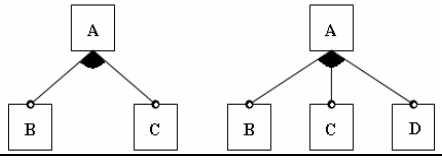
M – modulių skaičius projekte;

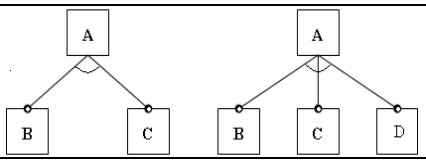
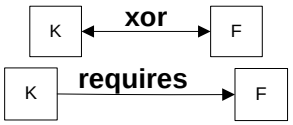
I – yra sąsajos tarp modulių.

Projektavimo sudėtingumas gali būti apibrėžtas kaip grafas kurio šakos apibrėžia modulius, o briaunos sąsajas (Keating modelis), mes galime pritaikyti šį sudėtingumo skaičiavimą požymių diagramoms. Pagrindinis skirtumas bus tik tas, kad mūsų grafas (požymių diagramoje) turi skirtingus parametrus: šakos ir briaunos turi skirtingas paskirtis (žr. 1 lentelė) ir turėtų turėti skirtingus pažinimo svorius.

Pažinimo svoris apibrėžiamas kaip sudėtingumo laipsnis arba laiko terminas reikalingas įsisavinti požymį. Bendras pažinimo svoris visos požymių diagramos yra suma viso grafo elementų pažinimo svorių. Shao ir Wang [14] apibrėžia elementų svorius taip:

14 lentelė. Požymių diagramos elementų pažinimo svoriai.

Požymių diagramos elementas	**Vaizdas**	**Pažinimo svoris**
Šaka (Požymis)		1
Privalomas ryšys		1
Nebūtinas ryšys ar pasirenkamas (*OR*)		2

Pasirenkamas (*case*)		3
Ryšių grupavimas (kardinalumas)	[1..*]	3
Reikalavimas arba *XOR*		3

4. **Taisyklė.**

Sudėtinis sudėtingumas požymių diagramoje skaičiuojamas pagal formulę:

$$Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3Rgr^2 + 3R^2)/9$$

kur:

Cm – sudėtinis sudėtingumas;

F – požymių skaičius;

Rand – būtini ryšiai;

Ror – nebūtini ryšiai arba pasirenkamas (*OR*);

Rcase – pasirenkamas ryšys (*case*);

Rgr – ryšių grupės (jeigu yra);

R – ryšių skaičius tarp šakų (*XOR* arba *reikalavimo*)

Konstanta 9 yra suma sudėtinių svorių, kad suvienodinti ryšius.

4.4.4. Požymių modelių konfigūravimas

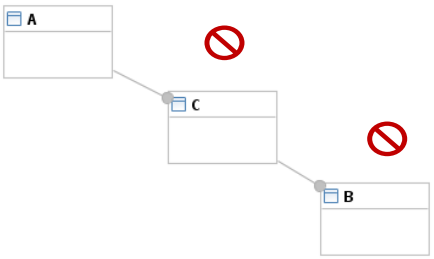
Požymių diagramos konfigūravimas tai galimų pasirinkimų požymių diagramoje automatizavimas, kuomet automatiškai sužymimos žemiau esančios grafo šakos, jeigu jas sieja privalomumo, reikalavimo, ar pasirenkamumo „*case*“ ryšiai. Atliekant automatinį požymių pasirinkimą susiduriama su tam tikromis problemomis, kurių metu nėra aišku kurią šaką rinkti. Žemiau yra pateikiamos kelios pagrindinės automatinio konfigūravimo problemos su pasiūlytais sprendimo būdais.

4.4.4.1. Privalomo ryšio konfigūravimas

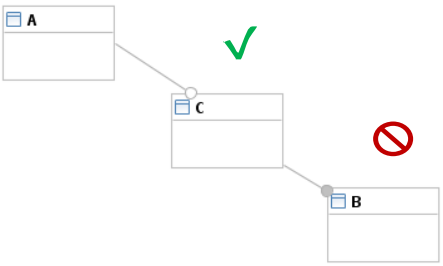
Realizuotoje sistemoje pateikiamas požymių sąrašas su galimybe pažymėti požymį arba ne (pasirinkti arba atsisakyti). Atsisakant elemento susiduriama su problema, jeigu elementą sieja aukštesnės hierarchijos elementai su privalomuoju ryšiu.

Siūlomas sprendimas: neleisti atsisakyti požymio, jeigu elementas turi privaloma ryšį su aukštesnės hierarchijos elementais ir sekoje nėra nei vieno pasirinkimo taško, jeigu hierarchijoje yra elementas turintis nebūtiną ryšį, atsisakant elemento, atsisakoma visa šaka požymių iki pasirenkamo elemento tėvo, pvz.:

Norima atsisakyti B elemento:



17 pav. Privalomų ryšių šaka



18 pav. Šaka su vienu neprivalomu ryšiu

Požymio B atsisakyti negalima, kadangi jo tėvas C turi privalomą ryšį su B požymiu, taip pat C elemento negalima atsisakyti, kadangi jo tėvas A turi privalomą ryši su C požymiu (žr. 17 pav).

Atsisakant B požymio bus atsisakoma taip pat ir C, kadangi jis turi privalomą ryšį su B požymiu, tačiau C yra pasirenkamas, todėl A požymio pasirinkimui tai neturės įtakos (žr. 18 pav).

Tikrinimas vykdomas lipant medžiu į viršų, jeigu randama pasirinkimo taškų, šaka išjungiama iš viršaus į apačią.

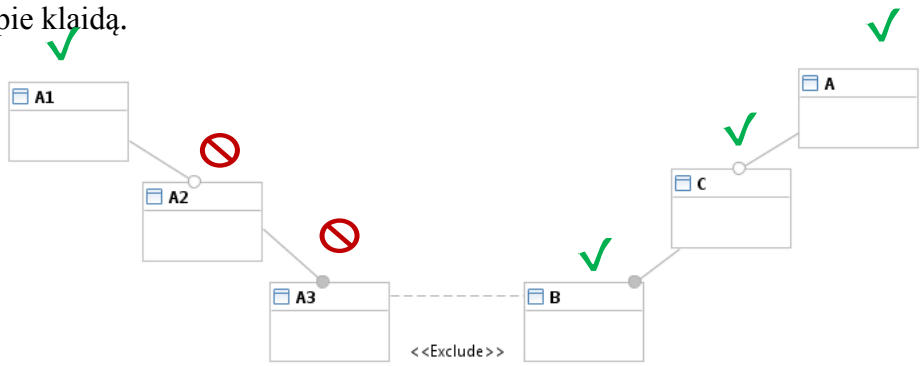
Algoritmas (pseudo kodas)

```
checkMandatory(selectedFeature){
     if(selectedFeature has MandatoryRelation
                                        with selectedFeature Parent){
               selectedFeatureParent.isChecked = false
               checkMandatory(selectedFeatureParent)
     }
}
```

4.4.4.2. XOR ryšio problema

Su XOR ryšio problema susiduriama, kuomet norima pasirinkti elementą, kuri kitoje šakoje jau yra pasirinktas.

Siūlomas sprendimas: neleisti pasirinkti elemento ar visos požymių šakos, informuoti vartotoją apie klaidą.



19 pav. XOR ryšio problemos pavyzdys

Tikrinimas vykdomas žemyn lipant medžiu ir tikrinant požymių ryšius, bei pasirinkimus.

Algorimtas (pseudo kodas)

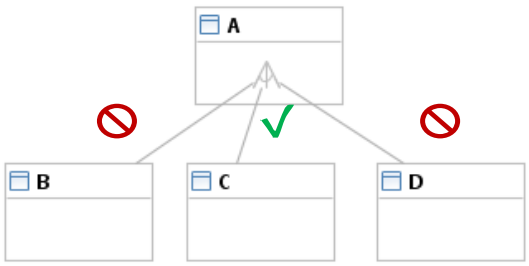
```
checkXORValidity(selectedElement){
     while(selectedElement has child){
           if(child has ExcludesRelation with
             selectedElement AND child.isChecked){
                exit Message("Elementas rodo į jau pasirinktą XOR
                              ryšiu sujungta elementą, atžymėkite jį
                              norėdami tęsti")

           }
           else checkXORValidity(child)
     }
}
```

4.4.4.3. Pasirenkamas elementas(case - pasirinkimas)

Pasirenkamas elementas „*case*“, turi savybę, kad iš tos grupės elementų būtų galima pasirinkti tik vieną elementą, problema iškyla kai reikia pasirinkti grupės požymį, kuomet iš tos grupės požymis jau buvo pasirinktas.

Siūlomas sprendimas: pasirenkant grupės elementą, kuomet iš tos grupės jau buvo pasirinktas požymis, atsisakyti anksčiau parinkto požymio ir leisti pasirinkti norimą požymį (žr. 20 pav).



20 pav. CASE pasirinkimo problemos pavyzdys

Tikrinimas vykdomas peržiūrint grupės požymius ir patikrinant ar jie nebuvo parinkti, jei buvo parinkti atsisakyti parinkimo ir leisti parinkti norimą požymį.

Algortimas (pseudo kodas)

```
while(selectedFeatureParent has child){
     if(child.isChecked)
           child.isChecked = false
}
selectedFeature.isChecked = true
```

5. SUKURTO POŽYMIŲ MODELIAVIMO ĮRANKIO KOKYBĖS ĮVERTINIMAS

5.1. Įvadas

Kuriama sistema veikia kaip komponentas, didžioji dalis architektūros yra paveldėta iš jau egzistuojančios sistemos. Galima išskirti kelias kuriamo komponento architektūros įtakojimus.

5.2. Sistemos testavimas

5.2.1. Testavimo tikslai ir objektai

Kuriant programinę įrangą vienas iš kūrimo tikslų yra pateikti programinę įrangą be jokių defektų. Deja tai nėra praktiškai pasiekiama, bet užtat galima pateikti programinę įrangą su kuo galima mažesniu defektų kiekiu.

Šiame dokumente pateikiamai testavimo atvejai, padedantys užtikrinti aukštą požymių diagramų modeliavimo įrankio programinės įrangos kokybė ir minimalų defektų kiekį.

5.2.2. Testavimo apimtis ir tipai

Kuriamas požymių diagramų braižymo ir modeliavimo įrankis bus testuojamas keliais aspektais:

- Testuojamas programinis kodas;
- Testuojama programos išvaizda;
- Testuojamos programos atliekamų funkcijų teisingumas.

5.2.3. Reikalavimai testavimui

Norint atlikti programinės įrangos testavimą reikalinga:

- Pilna programinės įrangos specifikacija išreikšta panaudojimo atvejų diagramomis ir panaudojimo scenarijais.
- Pilna ir testavimui parengta programinė įranga.
- Nustatyta procedūra, ką daryti su aptiktomis klaidomis.
- Klaidos ataskaitos šablonas.

5.2.4. Testavimo prioritetai

Atliekant sistemos testavimą pagrindinis dėmesys bus kreipiamas į šiuos faktorius. Jie pateikti svarbumo mažėjimo tvarka.

- Funkcionalumas – ar reikalingos programinės įrangos funkcijos yra ir atliekam joms priskirtas roles taip kaip reikia (atitinka specifikaciją)
- Saugumas – ar vartotojas gali prieiti ir dirbti tik su jam skirtais duomenimis.
- Panaudojamumas – kaip vartotoja sąsaja yra patogi vartotojui ir draugiška.

Našumas – ar programinė įranga dirba laikantis specifikacijoje nurodytais našumo reikalavimais.

5.3. Projektavimo komandos narių peržiūrų aprašymas.

Peržiūrų metu, buvo aptartos programos dalys, kurių kokybė kelia abejonių ar funkcionalumas nėra įgyvendintas.

Peržiūros metu, buvo aptariama galimybė iš požymių diagramos konfigūracijos kurti UML diagramą. Šis funkcionalumas veiks tokiu principu:

1. Sukuriamas visos sistemos požymių modelis
2. Sukuriamas visos sistemos UML modelis
3. Susiejamas UML modelis su požymių modeliu, visi failai išsaugomi.
4. Kuriama požymių modelio konfigūracija.
5. Sukurtai požymių modelio konfigūracijai generuojama nauja UML klasių diagrama iš FD2 Eclipse įskiepio įrankių rinkinio.
6. Iš sukurtos UML diagramos generuojamas programos kodo skeletas.

5.4. Interviu su užsakovu. Aptariami reikalingi patobulinimai. Sukuriamas pakeitimų sąrašas.

Interviu metu, buvo nutarta tęsti FD2 projekto vystymą. Interviu metu buvo sudarytas sąrašas darbų, kuriuos dar reikia įgyvendinti:

1. Realizuoti požymio modelio konfigūraciją
2. Kurti UML diagramą iš požymių modelio konfigūracijos.
3. Atlikti griežtesnį požymių modelio klaidų tikrinimą.

5.5. Formalios techninės peržiūros (ar korektiškai buvo sukurtas programų kodas)

Java programavimo kalba – objektinė programavimo kalba, todėl programoje naudojamos duomenų struktūros taip pat naudojamos objektiškai - projekte naudojamas Objektinis programavimas.

Įskiepis buvo tvarkingai parašytas, todėl, naudojant kodo analizės priemones, nebuvo rasta klaidų.

Naudojant XML programos konfigūraciją, buvo sugeneruotas diagramos vaizdavimo įskiepis. Šis programinis kodas nėra prižiūrimas programuotojų, todėl leistina, kad pasitaikys netvarkingai parašyto mašinos kodo.

5.6. Atlikto darbo kaina, darbo našumas

Buvo panaudotos CASE programų kūrimo priemonės, todėl darbo našumas buvo didesnis, išskyrus iškilusius nesklandumus projekto pradžioje, kada buvo mokomasi dirbti su naujomis CASE priemonėmis. Taip pat iškilo nenumatytų darbų, ir rezultatų, kuriuos reikėjo šalinti arba prisitaikyti prie jų.

Viena iš pagrindinių problemų prie kurių teko taikytis, tai kad GMF įrankiu nebuvo galima sukurti prilimpančio grupuojamojo ryšio, kadangi tokio tipo elementų nėra UML, todėl šis įrankis nebuvo pritaikytas kurti elementus kurie turėtų bendrą pražios piktogramą.

Šiai problemai išspęsti buvo panaudotas konteinerio tipo elementas, kuris apibrėžia grupuojamojo ryšio pradžia ir dedamas į požymio elementą iš kurio brėžiami sąlygos ryšiai. Šis sprendimas tenkino užsakovą, todėl kitų sprendimo būdų nebuvo ieškoma.

5.7. Reikalavimų išpildymas

Panaudojimo atvejis	**Išpildymas**
Programos naudojimosi gidas	Išpildyta
Diagramos spausdinimas	Išpildyta
Požymių modelio skaitymas iš failo	Išpildyta
Požymių modelio saugojimas į failą	Išpildyta
Diagramos priartinimo atitolinimo funkcija	Išpildyta
Požymių diagramos dialekto keitimas	Sąlyginai. Požymių modelį galima keisti naudojantis 2 skirtingais redaktoriais. Viename redaktoriuje vaizduojama diagrama, kitame medžio formos požymių sąrašas.
Grafinių elementų išvaizdos keitimas	Išpildyta
Požymio elemento šalinimas iš diagramos	Išpildyta
Požymio elemento detalizavimas	Išpildyta
Ryšio detalizavimas	Išpildyta
Požymio įtraukimas į diagramą	Išpildyta
Ryšio įtraukimas į diagramą	Išpildyta
Apribojimo įtraukimas	Išpildyta
Diagramos klaidų tikrinimas	Išpildyta
Kurti modeliuojamo produkto konfigūraciją (diagrama)	Išpildyta
Požymio susiejimas su komponentu	Išpildyta, naudojant pavadinimus kaip identifikacinę informaciją, jeigu pavadinimo nėra – susiejimas negalimas.
Sukurti specializuotą UML klasių diagramą	Iš esamos klasių diagramos pašalinami

pagal požymių diagramą	nereikalingi elementai pagal FD konfigūraciją.
Programos kodo generavimas	FD2 Eclipse įskiepis programinio kodo generavimo galimybės neturi, tačiau ši funkcija išpildoma naudojantis papildomais įrankiais. Tam naudojamas įrankis iš UML klasių diagramos galintis sugeneruoti programos kodą.

5.8. Kokybės įvertinimas

Programinės įrangos kokybę įvertinta naudojant tokius parametrus (15 lentelė)

15 lentelė. Programos kokybės vertinimo kriterijai.

Parametras	**Aprašymas**
Saugumas	Programinės įrangos duomenų saugumas
Išplėčiamumas	Galimybė praplėsti programinės įrangos funkcijas. Naujų modulių kūrimo galimybė.
Pernešamumas	Galimybė perkelti programinę įrangą ant kitokios techninės įrangos, operacinės sistemos.
Sąsajos galimybės	Ar gali veikti su kitomis sistemomis.
Panaudojamumas	Ar lengva išmokti dirbti su programine įranga.
Patvarumas	Kiek tolerantiška sistema vartotojo klaidoms?
Atsparumas tinklo gedimams	Ar sugeba programinė įranga dirbti nesant tinklui, ar nepraras duomenų jam dingus.

Saugumas

Sistema suprojektuota ir sukurta kaip atsiras sistemos komponentas, todėl dalis saugumo priklauso nuo pasirinkto produkto saugumo sprendimų.

Išplėčiamumas

Sistema pagal vartotojo reikalavimus lengvai išplečiama, kadangi ji sukurta kaip atskiras pasirinktos sistemos įskiepis (komponentas), dėl šio privalumo naujų modulių kūrimas yra lengvai įgyvendinamas.

Pernešamumas

Sistema suprogramuota naudojant Java programavimo kalbą, taigi ji puikiai veikia tiek windows operacinėje sistemose tiek ir UNIX (kurios palaiko grafinę sąsają).

Sąsajos galimybės

Sistema leidžia išsaugoti diagramas keliais formatais, tarp kurių yra ir UML naudojamas XML bei XMI formatais, kurie suteikia galimybę panaudoti diagramas ant kitų panašaus tipo programų. Taip pat sistema veikia kartu su UML diagramų braižymo komponentu, išplėsdama jo galimybes požymių diagramų braižymu, beis įgalina apjungti diagramas ir atlikti konfigūravimą.

Panaudojamumas

Sistemoje diagramos braižomos naudojant grafinius elementus, juos valdant pelės pagalba, taip pat elementai turi “drag&drop“ galimybę iš įrankių juostos. Taip pat sistemoje yra integruota pagalbos funkcija, kurioje pateikiama visas sistemos vartotojo vadovas.

Patvarumas

Sistema informuoja vartotoją apie klaidingų ryšių ar elementų naudojimą diagramose, pateikdama detalesnę informaciją apie kiekvieno elemento naudojimo galimybes.

Atsparumas tinklo gedimams

Sistemai veikti interneto ryšys nėra būtinas. Diagramos braižomos lokaliame kompiuteryje, ir turi išsaugojimo į dokumentą galimybę.

5.9. Išvados

FD2 įskiepis, skirtas požymių modeliavimui tinkamas grafiniam požymių modelių vaizdavimui.

CASE priemonių naudojimas projekte paspartino įrankio sukūrimą, tačiau įvedė nenumatytų apribojimų su kuriais teko taikytis, tokių kaip prilimpančiojo grupuojamojo ryšio elemento galimybės nebuvimas, tam teko panaudoti konteinerio komponentą grupuojamiems ryšiams, kurio pagalba realizuoti panašų, tenkinantį vartotoją, veikimą.

6. POŽYMIŲ DIAGRAMŲ MODELIO EKSPERIMENTINĖ DALIS

6.1. E-parduotuvė

6.1.1. Problema

Reikia suskirstyti pasirinktos sistemos (elektroninės parduotuvės) dalis į požymius, pateikti požymių sąrašą, ir nubraižyti požymių diagramą.

Elektroninė parduotuvė požymių modeliavimui buvo pasirinkta, tam kad būtų lengviau suprasti požymių modeliavimo esmę, kadangi tiriama sritis yra visiems aiški ir lengvai suprantama, o realizuotas modelis yra pakankamo sudėtingumo kad pamatyti tam tikrus požymių modeliavimo atvejus.

6.1.2. Reikalavimai

Kaip ir kiekviena elektroninė parduotuvė, sistema turi atlikti pagrindines elektroninei parduotuvei skirtas funkcijas.

Keliami funkciniai reikalavimai:

1. Pateikti prekių katalogą;
2. Pateikti informaciją apie konkrečią prekę;
3. Leisti vartotojams užsiregistruoti;
4. Suteikti galimybę vartotojams įtraukti prekes į prekių krepšelį;
5. Pateikti užsakymą;
6. Galimybė pasirinkti pristatymo būdą;
7. Galimybė pasirinkti apmokėjimo būdą;
8. Peržiūrėti užsakytų prekių būseną;

6.1.3. Požymių sąrašas

Elektroninės parduotuvės požymių sąrašas sudarytas remiantis Lau [33].

EShop

Aprašymas: Tai yra šakninis požymis, nuo kurio pradedama brėžti diagrama, jis apibrėžia sistemos pagrindą.

Tipas: būtinas elementas.

Ctalog

Aprašymas: Tai yra prekių katalogas, kuriame saugomos prekės, bei prekių kategorijos. Prekių katalogas e-parduotuvėje nėra būtinas sistemos elementas, kadangi prekės gali būti vaizduojamos ir be katalogo.

Tipas: nebūtinas elementas.

Category

Aprašymas: Prekių kategorijos, kategorija savyje gali turėti kelias kategorijas arba konkrečias prekes. Kategorijos nėra būtinos sistemoje, kadangi prekės gali būti ir ne kategorizuojamos.

Tipas: nebūtinas elementas.

Product

Aprašymas: Tai yra abstraktus produkto elementas, kuris gali būti skirstomas į kelias rūšis – paslauga, fizinis produktas arba elektroninis produktas. Šis elementas kuriamais sistemai yra privalomas, kadangi be jo sistema nefunkcionuos sėkmingai. Elemento vaikai yra pasirenkami (*OR* pasirinkimas), kadangi galima prekiauti viena prekių rūšimi arba keliomis, tačiau turi bent viena būti pasirinkta.

Tipas: būtinas elementas.

Service

Aprašymas: Tai yra teikiamų paslaugų sąrašas ir įkainiai. Tai yra vienas iš galimų parduodamų prekių variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

PhysicalProduct

Aprašymas: Tai yra fizinis produktas, kuris gali būti sandėliuojamas ir pristatomas į vartotojo namus, jis gali turėti tam tikrus parametrus (dydi, spalvą ir t.t.). Tai yra vienas iš galimų parduodamų prekių variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

ElectronicProduct

Aprašymas: Tai gali būti produktas, kurį galima tiesiog atsisiųsti iš parduotuvės (programinė įranga, media). Tai yra vienas iš galimų parduodamų prekių variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

Tax

Aprašymas: Tai mokesčiai, kiekvienas produktas turi tam tikrus mokesčius, kurie taikomi jam, tai gali būti PVM, akcizas, muitai ir t.t. Kadangi visos prekės turi tam tikrus mokesčius, šis sistemos komponentas privalomas.

Tipas: būtinas elementas.

Availability

Aprašymas: Tai abstraktus tipas, kuris apibrėžia ar produktą galima įsigyti. Sistema turi turėti produktų būsenų informaciją, tačiau elemento vaikai gali būti ne visi įtraukiami į sistema, tai priklauso nuo kliento poreikių.

Tipas: būtinas elementas.

NotAvailable

Aprašymas: Produkto šiuo metu negalima įsigyti. Tai yra vienas iš galimų parduodamų prekių būsenos variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas). Tai yra vienas iš galimų parduodamų prekių būsenos variantų, jis gali būti įtrauktas arba ne.

OutOfStock

Aprašymas: Produktas buvo išparduotas. Tai yra vienas iš galimų parduodamų prekių būsenos variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

InStock

Aprašymas: Produktą galima įsigyti parduotuvėje. Tai yra vienas iš galimų parduodamų prekių būsenos variantų, jis gali būti įtrauktas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

Customer

Aprašymas: Tai abstraktus pirkėjo apibrėžimas, pirkėjas gali būti dviejų tipų, tai registruotas arba neregistruotas. Registruoti vartotojai gali pirkti prekes, o neregistruoti jas tik peržiūrėti arba dėt į prekių krepšelį. Šis abstraktus elementas yra būtinas, kadangi be jo sistema negali sekmingai egzistuoti, nes nebus vykdomas pagrindinis funkcinis procesas – prekių pardavimas.

*Tipas:*būtinas elementas.

RegisteredCustomer

Aprašymas: Tai yra registruotas vartotojas, asmuo kuris žinomas sistemoje ir jis gali atlikti prekių pirkimą. Registruoti vartotojai yra reikalingi tam kad sistemoje vyktų prekių užsakymas bei pardavimas.

Tipas: būtinas elementas.

GuestCustomer

Aprašymas: Tai vartotojas, kuris tik lankosi sistemoje ir peržiūri prekes. Šis vartotojas užsiregistravęs gali patapti registruotu vartotoju ir naudotis likusiomis funkcijomis. Tai elementas, kuris nėra privalomas kad sistema sėkmingai veiktų, tad jį galima įtraukti arba neįtraukti.

Tipas: nebūtinas elementas.

BillingInformation

Aprašymas: Tai informacija reikalinga išrašyti sąskaitas už prekes, tai asmens vardas pavardė, apmokėjimo galimybė ir t.t. Šis elementas būtinas kad sėkmingai įvykdyti užsakymą.

Tipas: būtinas elementas.

Address

Aprašymas: Tai adresas, kuriuo bus pristatomos prekės. Be jo negalima įvykdyti sėkmingo pirkimo žingsnių.

Tipas: būtinas elementas.

PaymentInfo

Aprašymas: Tai abstraktus elementas, kuri apibūdina apmokėjimo už prekes galimybės tipą. Norint parduoti prekes, už jas reikia sumokėti, o kad vartotojas tai galėtų padaryti, jis privalo žinoti apie galimus apmokėjimo variantus.

Tipas: būtinas elementas.

CreditCard

Aprašymas: Atsiskaitymo už prekes galimybė kreditine kortele. Tai vienas iš apmokėjimo už prekes variantų, jį galima pasirinkti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

DebitCard

Aprašymas: Atsiskaitymo už prekes galimybė debetine kortele. Tai vienas iš apmokėjimo už prekes variantų, jį galima pasirinkti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

Cheque

Aprašymas: Atsiskaitymo už prekes galimybė čekiu. Tai vienas iš apmokėjimo už prekes variantų, jį galima pasirinkti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

ShoppingCart

Aprašymas: Tai prekių krepšelis, kuriame gali būti laiko mos prekės, kurios vėliau yra nuperkamos. Tai būtinas sistemos elementas, kadangi vartojas kitu būti negalės informuoti sistemos, ką jis nori įsigyti ir atlikti sekančių pirkimo žingsnių.

Tipas: būtinas elementas.

WishList

Aprašymas: Tai prekių norų sąrašas, kuriame galima įsidėti prekes ir užsiregistravus perdėti į prekių krepšelį. Šis sąrašo elementas nėra būtinas, kadangi jis skirtas neregistruotiems vartotojams, kurie gali užsiregistruoti ir naudoti paprastą prekių krepšelį, šis

komponentas praplečia sistemos funkcionalumą, palengvindama naujiems vartotojams naudojimąsi sistema, nereikalaujant registracijos.

Tipas: nebūtinas elementas.

Item

Aprašymas: Tai daiktas, kuris dedamas į krepšelį ar norų sąrašą, ir tai kopija produkto elemento, kuris gali susidėti iš paslaugos, fizinio produkto ar elektroninio produkto. Jis yra privalomas, kadangi kitu atveju prekės nebus galima įraukti į prekių ar norų sąrašus.

Tipas: būtinas elementas.

Order

Aprašymas: Tai užsakymas, kurį gali sudaryti kelios prekės, taip pat turi tam tikras būsenas, pristatymo nurodymus, bei pristatymo adresą. Būtinas elementas kuris saugo užsakymo informaciją.

Tipas: būtinas elementas.

OrderStatus

Aprašymas: Tai abstraktus tipas kuris apibrėžia užsakymo būseną, jis gali būti pristatytas, laukia, atšauktas, atliktas, vykdomas. Tai būtinas elementas, kadangi norint sėkmingai įvykdyti užsakymą ir visą laiką informuoti vartotoją apie užsakymo padėtį, reikia saugoti bent vieną iš būsenų. Elemento vaikai yra pasirenkami.

Tipas: būtinas elementas.

ShippedStatus

Aprašymas: Ši būsena įgyjama kuomet prekė yra pristatyta. Tai viena iš užsakymo būsenų, ją galima naudoti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

PendingStatus

Aprašymas: Ši būsena būna kuomet prekė laukia eilėje. Tai viena iš užsakymo būsenų, ją galima naudoti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

CanceledStatus

Aprašymas: Ši būsena būna atšaukus užsakymo vykdymą. Tai viena iš užsakymo būsenų, ją galima naudoti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

CompletedStatus

Aprašymas: Ši būsena įgyjama atlikus užsakymą. Tai viena iš užsakymo būsenų, ją galima naudoti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

ProcessingStatus

Aprašymas: Ši būsena būna vykdant užsakymą. Tai viena iš užsakymo būsenų, ją galima naudoti arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

ShippingInstructions

Aprašymas: Tai objektas, kuris saugo informaciją apie užsakymo pristatymo būdą. Šis elementas yra būtinas kadangi jis saugo informacija apie prekės pristatymą.

Tipas: būtinas elementas.

ServiceType

Aprašymas: Tai abstraktus tipas paslaugoms. Šis elementas yra būtinas, tačiau jis gali turėti tik vieną vaiką pasirinktą.

Tipas: būtinas elementas.

RegularService

Aprašymas: Standartinė paslauga. Tai paslaugos pasirinkimo elementas, kuris gali būti įtrauktas jeigu kiti tėvo elemento vaikai nėra įtraukti.

Tipas: pasirenkamas elementas (*case* pasirinkimas).

AirMailService

Aprašymas: Oro pašto paslauga. Tai paslaugos pasirinkimo elementas, kuris gali būti įtrauktas jeigu kiti tėvo elemento vaikai nėra įtraukti.

Tipas: pasirenkamas elementas (*case* pasirinkimas).

ExpressService

Aprašymas: Express paslauga. Tai paslaugos pasirinkimo elementas, kuris gali būti įtrauktas jeigu kiti tėvo elemento vaikai nėra įtraukti.

Tipas: pasirenkamas elementas (*case* pasirinkimas).

Carrier

Aprašymas: Pristatymo įmonių abstraktus tipas. Tai pristatymo būdų pasirinkimas, kuris būtinas norint vartotojams leisti pasirinkti prekių pristatymo būdą. Elemento vaikai yra pasirenkami.

Tipas: būtinas elementas.

PostCarrier

Aprašymas: Pristatymas paprastu paštu. Tai pristatymo būdo pasirenkamas elementas, gali būti naudojamas arba ne.

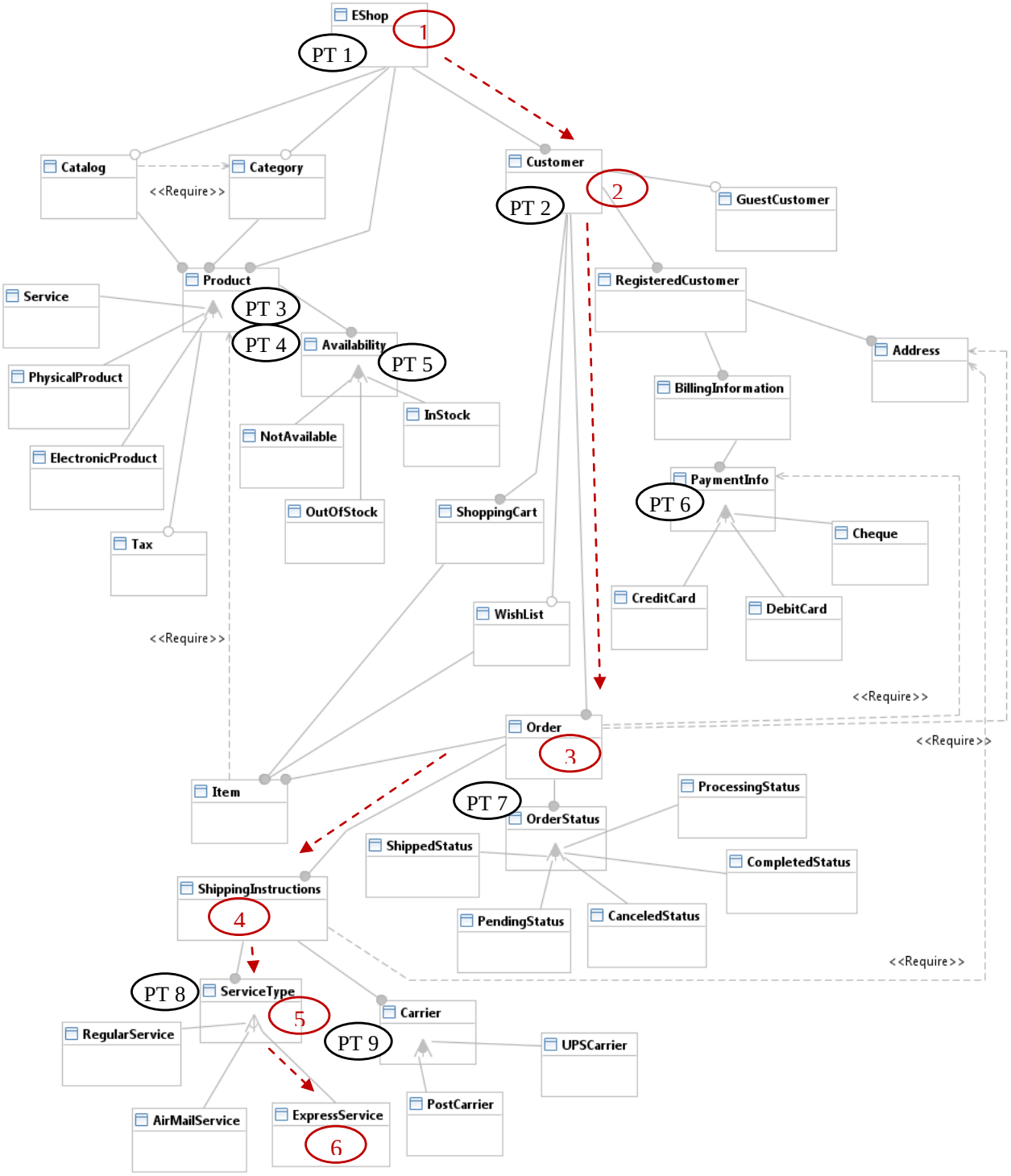
Tipas: pasirenkamas elementas (*OR* pasirinkimas).

UPSCarrier

Aprašymas: Pristatymas siuntų įmonės pagalba, šiuo metu UPS. Tai pristatymo būdo pasirenkamas elementas, gali būti naudojamas arba ne.

Tipas: pasirenkamas elementas (*OR* pasirinkimas).

6.1.4. E-parduotuvės požymių modelis (diagrama)



21 pav. E-parduotuvės požymių modelis

6.1.5. Požymių modelio charakteristikų tyrimas

E-parduotuvės modelio charakteristikų tyrimas atliekamas keliais etapais. Pagrindinė analizė atliekama skaičiuojant sudėtingumą pagal anksčiau apibrėžtas sudėtingumo vertinimo taisykles (žr. skyrių 4.4.3).

Visų pirma reikia nustatyti kiek egzistuoja požymių ir kokių tipų ryšiai juos sieja (taip pat reikalingas ryšių skaičius). Žemiau yra pateikiama E-parduotuvės modelio elementų skaičius (žr. 16 lentelė)

16 lentelė. Elementų kiekis požymių modelyje (pagal 21 pav)

Elementas	**Kiekis**	**Trumpinys**
Požymių	39	F
Ryšiai		
Būtini	18	Rand
Nebūtini	5	Ror
Pasirenkami (*case* - pasirinkimas)	3	Rcase
Pasirenkami (*OR* - pasirinkimas)	16	Ror
XOR ar reikalavimo (*require*)	5	R

Gautą informaciją galime panaudoti sudėtingumo skaičiavimui. Sudėtingumo skaičiavimas atliekamas pagal 4 taisykles (žr. skyrių 4.4.3).

Taisyklės Nr. 1 įvertinimas

Ši taisyklė nurodo, kad sudėtingumą galima apibrėžti pagal pasirinkimo taškų skaičių. Pasirinkimo taškai čia turima omenyje požymių diagramos šaka,s kuriose reikia atlikti pasirinkimą, kitaip tariant elementas turi nebūtinų ar pasirenkamų (*case, OR* - pasirinkimas) šakų. Kaip matome pasirinkimo taškų diagramoje yra 9 (žr. 21 pav, pasirinkimo taškas žymimas PT). Sudėtingumas yra tenkinamas, tačiau diagrama yra vadinama sudėtinga, nes 9 = 7 +/- 2.

Taisyklės Nr. 2 įvertinimas

Šioje taisyklėje sudėtingumas apibrėžiamas kaip grafo lygių skaičius. Šiam kriterijui įvertinti naudojamas ilgiausios šakos suradimo algoritmas, įvertinant kiek viršūnių pereita. Kaip matome diagramoje ilgiausia šaka susideda iš 6 viršūnių (žr. 21 pav, ilgiausias kelias žymimas punktyrine rodykle, o viršūnė numeriu). Vertinant pagal pirmos taisyklės sudėtingumo įvertinimą, diagrama taip pat yra sudėtinga 6 = 7 +/-2.

Taisyklės Nr. 3 įvertinimas

Šioje taisyklėje sudėtingumas vertinamas kaip atskirų medžio žemesnių medžių skaičius (angl. *sub-trees*). Kaip matome žemesnių medžių diagramoje yra 37 (žr. 21 pav).

Taisyklės Nr. 4 įvertinimas

Naudojant šią taisyklę sudėtingumas skaičiuojamas pagal formulę:

$$Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3Rgr^2 + 3R^2)/9$$

Įstatome reikiamas reikšmes (iš 16 lentelė):

$$Cm = 39^2 + (18^2 + 2 * (5 + 16)^2 + 3 * 3^2 + 3 * 0^2 + 3 * 5^2)/9$$

Ir gauname kad sudėtinis sudėtingumas yra lygus:

$$Cm = 1666{,}3333$$

Žemiau paveikslėlyje galima matyti kokie rezultatai yra pateikiami realizuoto įrankio charakteristikų analizės lange (žr. 22 pav).

FD Complexity

Item	Result	Details
Cognitive complexity:		
Variation points	9	Complex = 7 +/-2
Graph levels	6	Complex = 7 +/-2
Structural complexity	37	Number of sub-trees
Compound complexity	1666,33	Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3R^2)/9
Explanation:		
Features	39	F
Mandatory	18	Rand
Optional	5	Ror
ExclusiveCase	3	Rcase
CaseOr	16	Ror
Constraints	5	R

22 pav. E-parduotuvės požymių modelio charakteristikos

6.1.6. Požymių modelio validavimo tyrimas

Požymių modelio validavimo tyrimas susideda iš validavimo taisyklių patikrinimo pasirinktame modelyje (taisyklės skyriuje 4.4.2). Tyrime naudojamas e-parduotuvės modelis.

Taisyklės Nr. 1 įvertinimas

Pagal taisyklę galima matyti modelyje nėra ryšių kurie rodytų į tą patį elementą iš kurio yra brėžiami (žr. 21 pav).

Taisyklės Nr. 2 įvertinimas

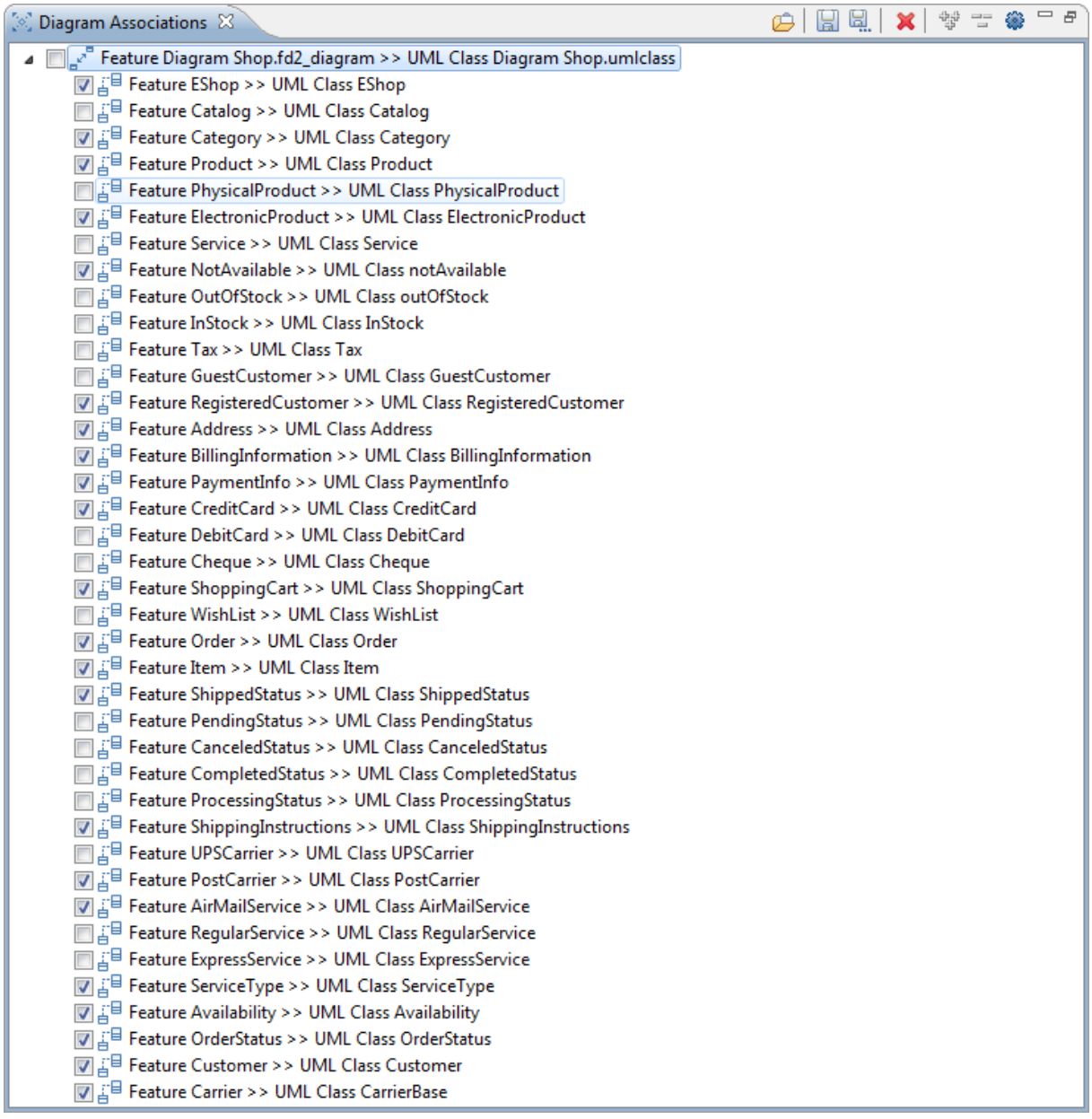
Pagal taisyklę matome jog visi elementai turi ryšius su žemiau hierarchijoje esančiais elementais, ir jeigu ryšys rodomas į aukštesnį hierarchijoje elementą, naudojamas reikalavimo ryšys (žr. 21 pav).

Taisyklės Nr. 3 įvertinimas

Pagal taisyklę matome jog visi elementai, kurie turi pasirenkamąjį ryšį *case* arba *OR* turi ryšį su kutais požymiais (žr. 21 pav).

6.1.7. Požymių modelio konfigūravimo tyrimas

E-parduotuvės konfigūravimas yra gana paprastas. Visoje diagramoje pagrinde vyrauja privalomi ryšiai, todėl pasirinkus pagrindinį elementą EShop yra automatiškai pažymimi privalomi jo vaikai, žemiau yra pateikiamas pavyzdys iš sistemos, pasirinkus EShop elementą (žr. 23 pav)



23 pav. E-parduotuvės konfigūravimo pavyzdys

Pasirinkimo taškuose jeigu tai neprivalomas ryšys požymiai nėra parenkami, nebent tai CASE arba OR pasirinkimo taškas, tuomet automatiškai yra parenkamas pirmas sąraše esantis elementas.

Privalomo ryšio atžymėjimo problemos pavyzdys (žr. skyrių 4.4.4.1) galėtų būti Availibility požymio atsisakymas, šio požymio atsisakius, bus atsisakoma visa šaka esanti

virš jo. Neįmanomas variantas būtų atsisakyti požymio OrderStatus kadangi visi aukščiau jo esantys požymiai sujungti privalomu ryšiu.

Pasirenkamo ryšio problema (žr. skyrių 4.4.4.3) būtų matoma požymių grupėje ServiceType, kuomet pasirenkant vieną iš paslaugų pvz., AirMailService kiti grupės požymiai yra atžymimi.

7. IŠVADOS

Šiame magistriniame darbe buvo:

1. išnagrinėti požymių diagramų sintaksės variantai,
2. ištirti egzistuojantys požymių diagramų braižymo ir požymių modeliavimo įrankiai, nustatyti jų privalumai ir trūkumai,
3. aprašytas požymių diagramų sintaksės varianto metamodelis,
4. remiantis aprašytu metamodeliu realizuotas požymių diagramų modeliavimo įrankis,
5. pasiūlyta programų sistemų produktų linijų projektavimo metodika.

Naudojant sukurtą požymių modeliavimo įrankį:

1. ištirtos požymių diagramų specifikavimo problemos,
2. identifikuotos požymių modelių validavimo problemos ir pasiūlytos taisyklės, leidžiančios suvaldyti ir neleisti įgyvendinti nekorektišką požymių modelį,
3. realizuotos ir ištirtos 4 požymių modelių sudėtingumo vertinimo taisyklės, pagrįstos žmogaus sugebėjimu suprasti sistemos požymių modelį,
4. išskirtos pagrindinės požymių modelių konfigūravimo problemos ir pasiūlyti jų sprendimo būdai siekiant kuo labiau automatizuoti šį procesą,
5. atliktas e-parduotuvės požymių modelio tyrimas (nagrinėtas modelio specifikavimas, validavimas, sudėtingumo valdymas, konfigūravimas).

8. LITERATŪRA

[1] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, Chin Kuan Ho. *Knowledge Based Method to Validate Feature Models. In Steffen Thiel, Klaus Pohl (Eds.).* Software Product Lines, Proc. of 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Second Volume (Workshops), University of Limerick, Ireland 2008, p. 217-225.

[2] Alexandre Bragança, Ricardo Jorge Machado. *Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines*. Proc. of Software Product Lines, 11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007, p. 3-12.

[3] C.W. Krueger, *Software reuse, ACM Computing Surveys (CSUR)*, v.24 n.2, June 1992 p. 131-183.

[4] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. *FAMA: Tooling a framework for the automated analysis of feature models*. VAMOS, 2007, p. 129–134

[5] D.A. Troy, S.H. Zweben. *Measuring the Quality of Structured Designs. Journal of Systems and Software, Vol. 2,* 1981, p. 113-120.

[6] D.M. Weiss, C.T.R. Lai. *Software Product-Line Engineering: A Family-Based Software Development Approach*. Addison-Wesley, Reading, MA, USA, 1999.

[7] Danilo Beuche. *Modeling and Building Software Product Lines with Pure: : Variants*. Software Product Lines, 12th International Conference, Limerick, Ireland, 2008, Proceedings. IEEE Computer Society 2008, p. 358.

[8] Eric Clyberg, Dan Rubel, Vasley, *Eclipse plug-ins, Third Editon, Addison Wesley*, 2008 p. 289-352.

[9] G. Miller. *The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information*. The Psychological Review, Vol. 63(2), 1956, p. 81-97.

[10] G. Perrouin, J. Klein, N. Guelfi, and J. M. Jezequel. *Reconciling Automation and Flexibility in Product Derivation*. 12th International Software Product Line Conference, SPLC '08. 2008, p. 339-348.

[11] Griss, M.L., Favaro, J., and d'Alessandro, M. *Integrating Feature Modeling with RSEB*, Proceedings of Fifth International on Software Reuse, Victoria, B.C. 1998, p. 9.

[12] H. Gomaa. *Object Oriented Analysis and Modeling for Families of Systems with UML*. InW. B. Frakes, editor, IEEE International Conference for Software Rreuse (ICSR6), 2000, pages 89–99.

[13] J. Coplien, D. Hoffman, D. Weiss. *Commonality and Variability in Software Engineering*. IEEE Software 15(6), 1998, p37–45.

[14] J. Shao, Y. Wang. *A New Measure of Software Complexity based on Cognitive Weights*. Canadian Journal of Electrical and Computer Engineering, 28(2), 2003, p. 69-74.

[15] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, A. Ruiz-Cortes. *Automated Diagnosis of Feature Model Configurations*. Journal of Systems and Software, in Press, 2010, p. 225-234.

[16] Jean-Christophe, *Quality of Feature Diagram Languages: Formal Evaluation and Compariso.* Evaluation and Formalization of Feature Modelling Languages, 2008.

[17] Kang, K.C., Lee, J. Donohoe, P. *Feature-Oriented Product Line Engineering*. In IEEE Software (July/August), 2002, p. 58-65

[18] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, Moonhang Huh. *FORM: A feature-oriented reuse method with domain-specific reference architectures*. Annals of Software Engineering, Springer, 1998, p. 143 - 168

[19] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report, 1990, p. 63-70.

[20] M. Anastasopoulos. *Increasing efficiency and effectiveness of software product line evolution: an infrastructure on top of configuration management*. Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol), Amsterdam, The Netherlands, 2009, p. 47-56.

[21] M. Keating. *Measuring Design Quality by Measuring Design Complexity*. Proc. of the 1st Int. Symp. On Quality of Electronic Design (ISQED 2000), San Jose, California, 2000, p103-108.

[22] Matthew Stephan, Michał Antkiewicz. *Ecore.fmp: A tool for editing and instantiating class models as feature models*. Technical Report Tech. Rep. ECE, University of Waterloo, 2008.

[23] McIlroy D. *Mass produced software components*. NATO Conference on Software Engineering, 1968, p. 55-69

[24] Myllärniemi, V., Raatikainen, M. and Männistö, T. *KumbangTools*. Proc. of Software Product Line Conference (SPLC), 2007, p. 1-2.

[25] O. Rohlik and A. Pasetti. XFeature Modeling Tool. Automatic Control Laboratory, ETH Zurich, 2005. Prieiga per internetą: <http://www.pnp-software.com/XFeature/>

[26] P. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston 2002.

[27] P.-Y. Schobbens, P. Heymans, J.-Ch. Trigaux, Y. Bontemps. *Feature Diagrams: A Survey and a Formal Semantics*. Proc. of 14th IEEE Int. Requirements Engineering Conference (RE'06), Minneapolis/St.Paul, Minnesota, USA, 2006, , p.136–145.

[28] *Part I: Eclipse Platform Technical Overview*, 2001, [žiūrėta 2009-04]. Prieiga per internetą: <http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html#N100B9>

[29] R. Damaševičius, V. Štuikys, E. Toldinas. *Domain Ontology-Based Generative Component Design Using Feature Diagrams and Meta-Programming Techniques*. Proc. of 2nd European Conference on Software Architecture ECSA 2008, September 29 - October 1, Paphos, Cyprus. In R. Morrison, D. Balasubramaniam, and K. Falkner (Eds.), Springer-Verlag, 2008, p. 338-341.

[30] R. Damaševičius. *Specification of Learning Content using Feature Diagrams*. To appear in Papadopoulos, G.A., Wojtkowski, W., Wojtkowski, G., Wrycza, S., and Zupancic, J. (Eds.), Information System Development: Design and Development. Springer, 2009, p. 1-2.

[31] R. Laue, V. Gruhn. *Complexity metrics for business process models*. In W. Abramowicz and H. C. Mayr (Eds.), Proc. of 9th Int. Conf. on Business Information Systems (BIS 2006), LNI 85, 2006, p. 1-12.

[32] Riebisch, M., B¨ollert, K., Streitferdt, D., Philippow, I. *Extending feature diagrams with UML multiplicities*. In: 6th Conference on Integrated Design & Process Technology (IDPT 2002), Pasadena, California, USA 2002, p. 2-7.

[33] Sean Quan Lau. Domain *Analysis of E-Commerce Systems Using Feature-Based Model Templates*. MsC Thesis, University of Waterloo, Ontario, Canada, 2006, p 23-81.

[34] Software Engineering Institute (SEI). *Cyclomatic Complexity*. In Software Technology Roadmap, 2006.

[35] T. von der Maßen and H. Lichter, *RequiLine: A Requirements Engineering Tool for Software Product Lines*. Softw Product-Family Eng (PFE-5): Springer, 2004, p. 168-180.

[36] Thomas Leich and Sven Apel and Laura Marnitz and Gunter Saake. Tool Support for Feature-Oriented Software Development - FeatureIDE: An Eclipse-Based Approach. In *Proceedings of OOPSLA Workshop on Eclipse Technology eXchange (ETX)*, San Diego, USA, October 2005.

[37] V. Štuikys, R. Damaševičius, I. Brauklytė, V. Limanauskienė. *Exploration of Learning Object Ontologies Using Feature Diagrams*. Proc. of World Conference on Educational Multimedia, Hypermedia & Telecommunications, Vienna, Austria 2008, p. 2144-2154, ISBN 1880094657.

[38] V. Vranic. *Reconciling Feature Modeling: A Feature Modeling Metamodel*. In Matias Weske and Peter Liggsmeyer, editors, Proc. of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays 2004), LNCS 3263, Erfurt, Germany, September 2004, p. 122–137.

[39] W. Kintsch. *Comprehension: a paradigm for cognition*. Cambridge University Press, 1998.

9. PRIEDAS

ADVANCED PRODUCT LINE FEATURE MODELLING USING FD2

Paulius Žaliaduonis, Deividas Kreivys, Robertas Damaševičius

Kaunas University of Technology, Software Engineering Department,
Studentų 50, Kaunas Lithuania,
bendras@gmail.com, deividas.kreivys@gmail.com, robertas.damasevicius@ktu.lt

Abstract. Product Line Engineering (PLE) is a methodology for creating a collection of similar software systems from a shared set of software assets. A key concept in PLE is feature modelling, which is used during entire product line (PL) development process to represent all PL products in terms of features, as well as to manage software feature variability and complexity, and to document the PL. Feature models are represented graphically by means of Feature Diagrams (FD). This paper analyzes FD notations and feature modelling tools, and proposes a new feature modelling environment FD2, which supports wide FD editing capabilities, configuration of FD models, as well as integration with UML and generation of UML class diagrams.

Keywords: Feature Diagram, Product Line Engineering, product configuration, model generation.

10. Introduction

Current approaches for architectural design of software systems predominantly use the product line (PL) concept. A software PL is a set of software systems that share a common, managed set of features satisfying the specific needs and are developed from a common set of core assets in a prescribed way [1]. The concept of PLs, if applied systematically, allows for the dramatic increase of software design quality, productivity, provides a capability for mass customization and leads to the 'industrial' software design [2]. The key for implementation of a PL is the use of domain analysis and domain modelling methods. A vast majority of the PL methods use feature modelling, which express domain content in the form of features and to model the domain well through the identification of commonality and variability of structural, functional and other characteristics (called features) and their relationships.

There are several definitions of a feature. The FODA approach defines a feature as a prominent and distinctive user-visible characteristic of a system [3]. When comparing to other conceptual abstractions (such as function, object, and aspect), features are externally visible characteristics, whereas functions, objects, and aspects have been mainly used to specify the internal details of a system. Therefore, feature modelling focuses on identifying external visible characteristics of products in terms of commonality and variability rather than describing all details of a system.

Feature modelling is used for variability management in the field of software PL development. Specifically, this activity can be seen as a part of the domain analysis process, for example, as it is described by the FODA method [3]. A feature model is an abstraction of a family of systems in a domain capturing the commonalities and variability among the members of the family. Feature models are used to specify the members of a product-line. They define *common* and *variable* features, their dependencies and relationships, and their usage constraints in product-lines. Feature models are easy to understand and provide a generic way to represent variability information.

Features are primarily used in order to discriminate between product instances (configuration choices). Common features among different products are modelled as mandatory features, while different features among them may be optional or alternative. Optional features represent selectable features for products of a given domain and alternative features indicate that no more than one feature can be selected for a product. Aggregation of features represents all variability within the scope of the domain and is thus the domain feature model, or feature model.

The derivation of a product consists of traversing the feature tree in an orderly manner and selecting the optional features. The result is a product description containing all the features in the product (a feature configuration). In product line terminology, a product is fully specified when all of its variation points are bound; that is the product specification is complete when all features have been selected. Feature diagrams provide a concise and explicit representation of variability. They guide the choices to be made for determining the features of specific products and facilitate the reuse of software components implementing these features.

In this paper we analyze concurrent projects developing feature modeling tools, discuss Feature Diagram syntax used in FD2 (Feature Diagramming Tool) tool, describe its architecture and present a concrete example. We demonstrate how to create different instances of software systems from the configuration of its

features: 1) analyze features; 2) create a feature model; 3) associate the feature model with the system's class diagram; 4) create the software system features configuration; and 5) finally, generate a new class diagram that will be used for creation of a reduced version of a software system (configured in the feature model).

The structure of the remaining parts of the paper is as follows. Related works are discussed in Section 2. Syntax of Feature Diagram is described in Section 3. The feature modelling environment FD2 is presented in Section 4. Case study is described in Section 5. Finally, the conclusions are presented and future work is described in Section 6.

11. Syntax of Feature Diagrams

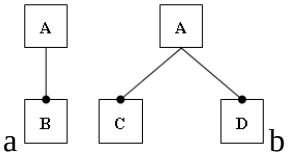
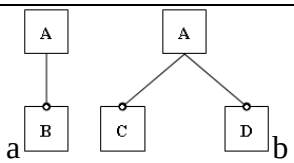
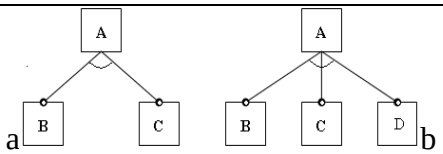
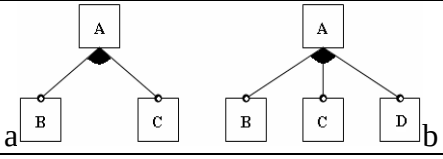
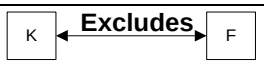
Feature models are specified using Feature Diagram (FD), a graphical notation for representing and modelling variability at a higher abstraction level. It was first was introduced in FODA [3] and since then has been a subject of active research and extension by numerous authors [4-7]. As a result, there are many similar notations of FD with different syntactic and semantic discrepancies. Here we use the notation of feature diagrams, adopted from [4-7] that is shown in Table 1.

A FD consists of a set of nodes, a set of directed edges and a set of edge decorations. The nodes and edges form a tree. The edge decorations are drawn as arcs connecting subsets or all of the edges originating from the same node. The root of a FD represents a concept. Features can be mandatory (and-features), optional, alternative or or-features. And-features denoted with a filled circle. Or-features are denoted with an empty circle. Extension points are features that has at least one optional sub-feature, an edge ending in an empty circle, or at least one set of (sub)features. Extension points with optional features are simply denoted as edges ending in an empty circle. Extension points with or-features use a filled decorated edge arc, the edges ending in a filled circle to denote the mandatory features.

There are three basic types of features: mandatory, optional and alternative. Mandatory features allow us to express common aspects of the concept (referred to as commonality), whereas optional and alternative features allow us to express variability. Moreover, features may have dependencies: the selection of one feature may rule out (mutual exclusion) or assume (requires, includes) the inclusion of another feature. FDs also can be supplemented by some additional information such as short semantic description of each feature, rationale for each feature, constraints, default dependency rules etc.

We have developed FD2 (Feature Diagramming Tool) that allows creation of graphical Feature Diagrams. The implementation syntax is presented in 0, which has been described in more detail in [8, 9].

Table 1. Main Feature diagram syntax elements.

Relation element name	Definition	Graphical symbol
Mandatory	Feature B (C, D) is included if its parent A is included: a) if A then B; b) if A then C and D	
Optional	Feature B (C, D) may be included if its parent A is included: a) if A then B or <no feature> b) if A then C or D or <no feature>	
Exclusive case	Only (exactly) one feature has to be selected if their parent is selected: a) if A then case-of (B, C) b) if A then case-of (B, C, D)	
Case Or	At least one feature has to be selected if its parent A is selected: a) if A then any-of (B, C) b) if A then any-of (B, C, D)	
Exclude	Relation between 2 features. Both features cannot go together: If K then not F if F then not K	

Require	Feature K requires feature F: K requires F	

12. Related work

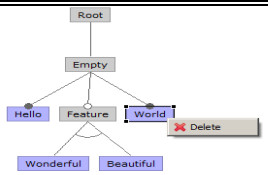
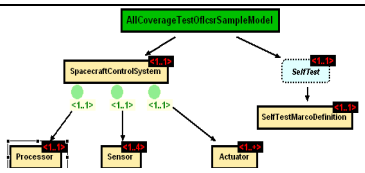
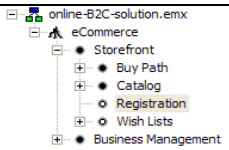
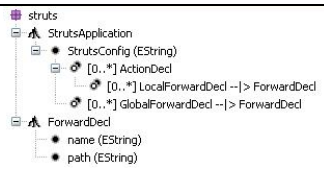
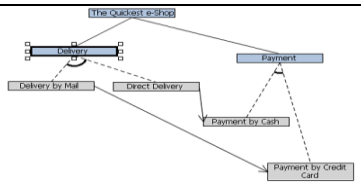
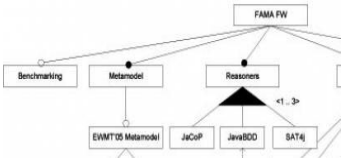
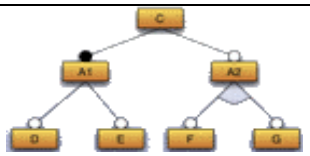
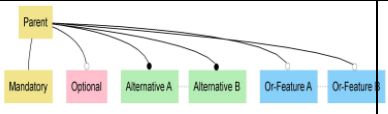
Currently, there are a number of feature modelling tools available. We have analyzed several feature modelling tools as follows: Feature IDE, XFeature, Ecore Feature modelling plugin, RequiLine, FAMA, KumbangTools, pure::variants (see Table 1). Most of projects mentioned are developed as Eclipse plug-ins and with GPL licenses. These tools have very different range of capabilities: some generate source code for programs, other make configuration files or even create other models, from which program source code can be created. These tools differ in their support of FD notation variants, visual appearance of FD as well as support for integration (interaction) with other types of modelling diagrams (UML) and support for code (or model) generation. These are summarized in 0 and 0.

Table 2. Other known feature modelling tools.

Tool Name	Creator	License	Source code generation	Reference
Feature IDE	University of Magdeburg	Open source	Source code	http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/
XFeature	P&P Software GmbH / ETH-Zürich	Open source	Configuration file	http://www.pnp-software.com/XFeature/Home.html
Ecore / Feature Modeling Plugin	University of Waterloo, Canada	Open source	Ecore model	http://gsd.uwaterloo.ca/projects/fmp-plugin/
RequiLine	Research Group Software Construction, RWTH Aachen	Open source	-	http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/index.php
FAMA	University of Seville, Spain	Open source	Custom model	http://www.isa.us.es/fama/
KumbangTools	Helsinki University of Technology, Finland	General Public License	Custom model	http://www.soberit.hut.fi/KumbangTools
pure::variants	pure-systems GmbH	Commercial	Source code	http://www.pure-systems.com/pv/

Table 3. Examples of different FD notations used in feature modelling tools.

Feature modelling tool	FD notation	Advantages	Disadvantages

Feature IDE		-Free (open source) -XMI files -diagram validation -source code generation	-only one graphical notation -can't be plugin
XFeature		-Free (open source) -different graphical notations -XMI files -diagram validation -source code generation -Eclipse plugin	
Ecore / Feature Modelling Plug-in		-Free (open source) -XMI files -diagram validation -source code generation -Eclipse plugin	-only one graphical notation
Ecore Feature Modelling Plug-in		-diagram validation -source code generation -Eclipse plugin	-Free (Genera public license) -only one graphical notation
RequiLine		-Free (open source) -XMI files -diagram validation	-only one graphical notation -no source code generation -can't be plugin
FAMA		-Free (open source) -XMI files -diagram validation -source code generation -Eclipse plugin	-only one graphical notation
KumbangTools		-Free (open source) -XMI files -diagram validation -source code generation -Eclipse plugin	-only one graphical notation
pure::variants		-XMI files -diagram validation -source code generation -Eclipse plugin	-Free for non-commercial use -only one graphical notation

The novelty of the developed feature modelling tool FD2 is its ability to associate Feature Model with UML models so that it could be configured and used further for object-oriented software modelling and code generation. Also it is Free to download and use, developed under GPL license, open source. Diagrams are created using graphical tool and validated during creation of it, uses XMI data format to store model data. FD2 from its nature is Eclipse plug-in.

13. FD2: A Feature Modelling Environment

The most important feature of FD2 tool is its ability to configure UML models. FD2 tool is an extended feature modelling environment, which besides its primary function as feature diagram drawing and editing tool also allows associating feature model elements with UML Class Diagram elements. This association defines how features in model represent classes in the modelled system. From this association, we can create a product configuration. The feature model configuration defines differences between products in a product line. Basically, configuration defines which elements have to be included in UML model. From this configuration we can create new UML models (class diagrams). Having UML models enables us further capabilities in product line development such as to generate program source code, documentation, etc.

13.1. Architecture of FD2

The FD2 tool has been built for Eclipse modelling platform. Eclipse is an open source integrated development environment that has grown into an extensible hosting platform for software development tools. FD2 relies on the Eclipse platform (Eclipse modelling edition) and other tools created for Eclipse. To implement the feature modelling plug-in, we have used the following technologies: Eclipse platform, GMF (Graphical Modelling Framework), EMF (Eclipse Modelling Framework), and GEF (Graphical Editing Framework) (see Table 4). All these frameworks give a lot of common look and feel for FD2 tool and UML2Tools.

Table 4. Technologies used for developing the FD2 tool.

Technology used	Description
Eclipse	Eclipse environment provides easy way to integrate new plug-in with already present plug-ins into a single environment. Eclipse modelling edition brings all the tools needed by FD2.
GMF	FD2 Feature diagramming component was entirely created with GMF. The GMF provides a generative component and runtime infrastructure for graphical editors based on EMF and GEF. GMF toolkit provides a powerful code generation facilities that implemented FD2 tool Feature model diagramming editor.
EMF	Provides convenient way to create feature modelling metamodel definition and model editor. This metamodel represents domain logic and is mapped to graphical user interface. EMF toolkit is used to create source code from UML models.
UML2Tools	This plug-in supports main UML diagrams and is connected to FD2. FD2 tool creates a customized UML model according to a configuration that the user creates.

13.2. Interface of FD2 tool

Standard view of FD2 interface is shown in Figure 1 and summarized in Table 5. However, the users can change the appearance of the interface after installation.

Table 5. Elements of the FD2 user interface.

Element	Description
1. Menu	Menu options
2. Toolbar	Tool bar
3. Project Explorer	Project view, including available model, diagram, relationship etc. files.
4. Outline	A scaled-down view of a feature diagram shown in Diagram Window.
5. Properties	A list of properties of a feature model.
6. Problems	A list of errors detected in a feature model.
7. Diagram Window	A window for creating, editing and representing of a feature model.
8. Tool box	Tools available when creating feature model.

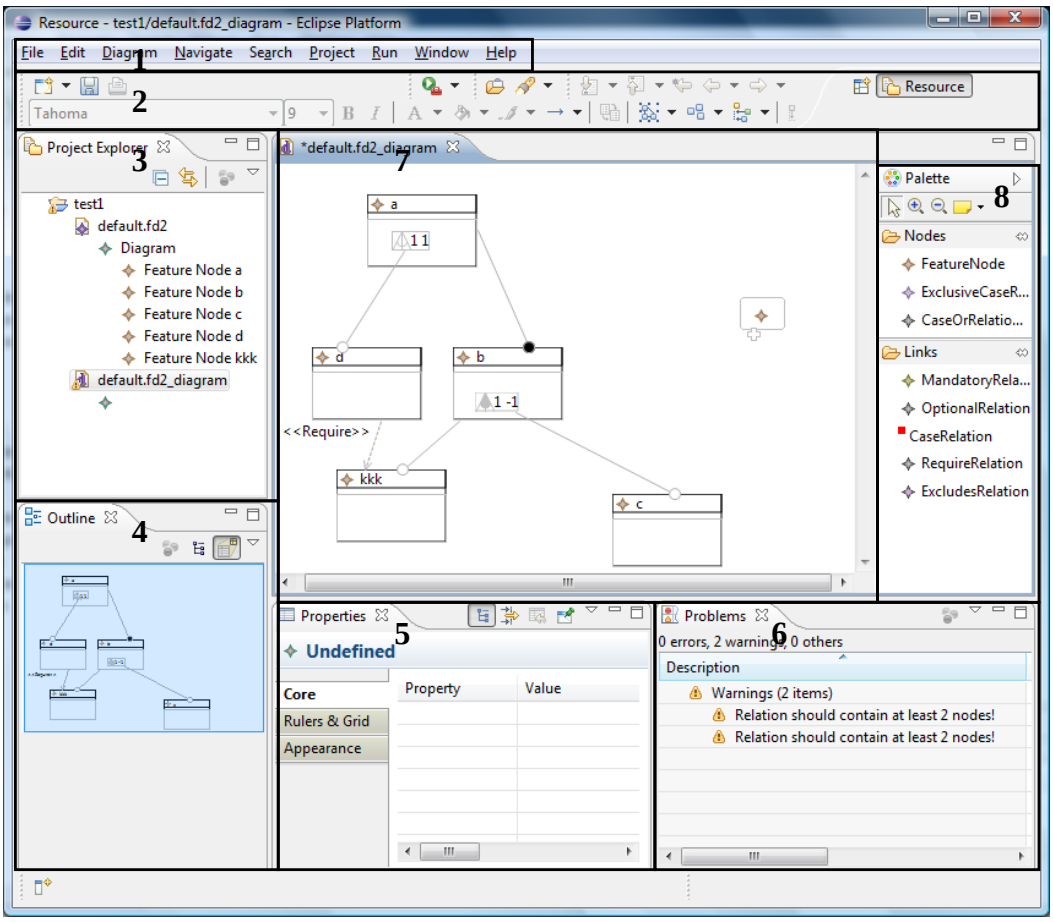


Figure 1. Interface of the FD2 modelling environment

13.3. Features of FD2

Main features of FD2 tool is graphical feature models creation, joining them with UML class diagrams and the most significant - "configure" UML models. This is the advantage of FD2 tool compared to other graphical Feature modelling tools. Other features such as XMI data format to store model data, graphical feature diagrams notation are not so important in our work and will not be discussed.

The features of FD2 are as follows:

1) The user you can create and edit graphical feature diagrams.
2) A unique feature of FD2 is the ability to associate feature model elements with UML class diagram elements. The association shows which class diagram elements are represented by system features. This is very useful when there is a need to show differences in different product line systems described using UML class diagrams. The associations show, how system can be configured and the feature diagram syntax add logic, which product components are to be included. The FD2 tool allows configuring association of features and classes. It shows which features are included in configured product. Deselecting a feature means excluding a component from a product.
3) FD2 can create a new configured UML model from the initial generic UML model. Using the existing feature configuration, the tool removes the unnecessary (unselected) elements of the UML class diagram and creates a new UML model. From now, the designer can use this model to generate code, documentation, etc.

13.4. Integration with Eclipse UML2Tools

The eclipse development environment is very comfortable as environment of modelling tool. Many modelling projects are developed for Eclipse platform, so we can have a lot of extensibility options. We have chosen few tools to use in Eclipse environment: Eclipse Modelling Framework, Graphical Editing Framework, UML2Tools, ECore modelling. UML2tools was developed on Eclipse platform, so integration of FD2 with UML diagrams is much simpler. Also because all components in eclipse are developed like plugins, they can easily communicate with each other. FD2 uses Eclipse UML2Tools plug-in and extends it by custom functions to enable ability to associate Feature model with UML model, which is easily done by Eclipse extension points. And finally, code generation is achieved by using Eclipse plugin Ecore Tools and UML2tools Ecore files.

14. Case study: reservation system example

In this example, we demonstrate the main steps of configurable software system creation using the FD2 tool. FD2 is used to create the configuration of software system's UML model that is used in later steps to create the concrete implementation of a software system.

14.1. System specification

We demonstrate as an example how to develop a reservation system that can mark different types (room, car, boat) of bookings. Bookings have different length (duration) and customer. The UML class model of the reservation system is defined in 0.

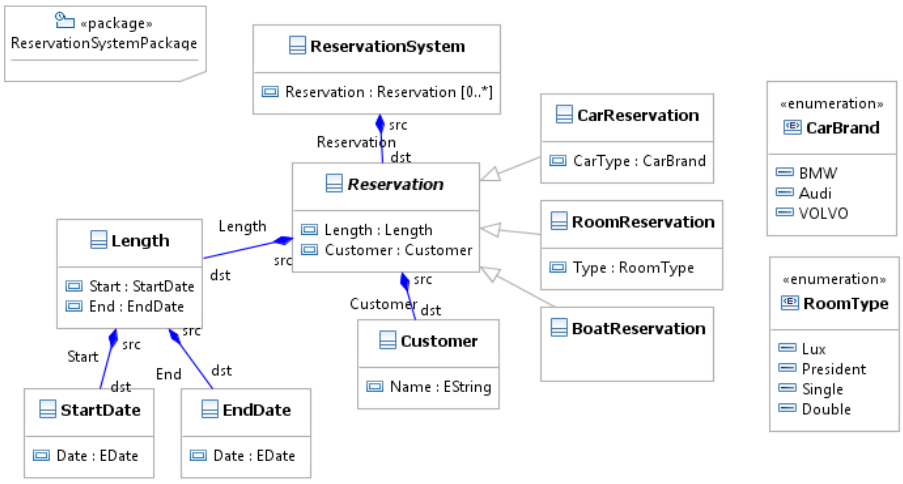


Figure 2.UML model of the reservation system.

A feature model of the reservation system is shown in 03. See a description of notation in Table 3.

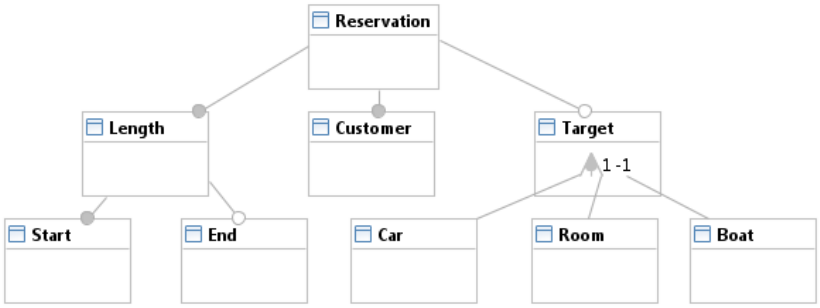


Figure 3. Feature model of the reservation system.

14.2. Association of features and classes, and configuration

Feature and class association is a process when feature is assigned to a class. This is done using context menu of FD2. An example process is as follows: Select feature element and right click it, from context menu select "Assign to UML Class", then class diagram is opened, select class element, right click it, from context menu select "Assign to Feature". After each assignment, a new entry in "Diagram Associations" (0) view is entered.

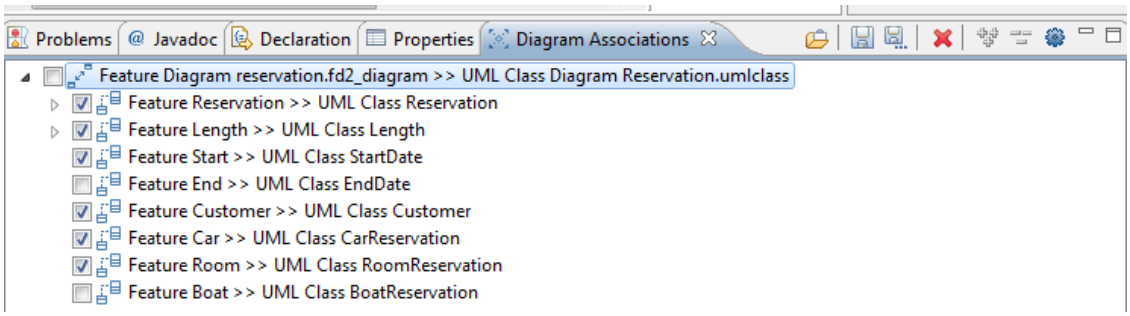


Figure 4. Feature configuration process.

14.3. Creation of configured UML model

When all associations are present, configuration can be made. It is simple process; you just select checkboxes near association elements. See example in 0.

Now it is possible to create a configured UML model by clicking model generation button. A new UML model is created with file ending "_configured.uml". This UML model does not have the configured and deselected class elements (see 0 and compare to Figure 2).

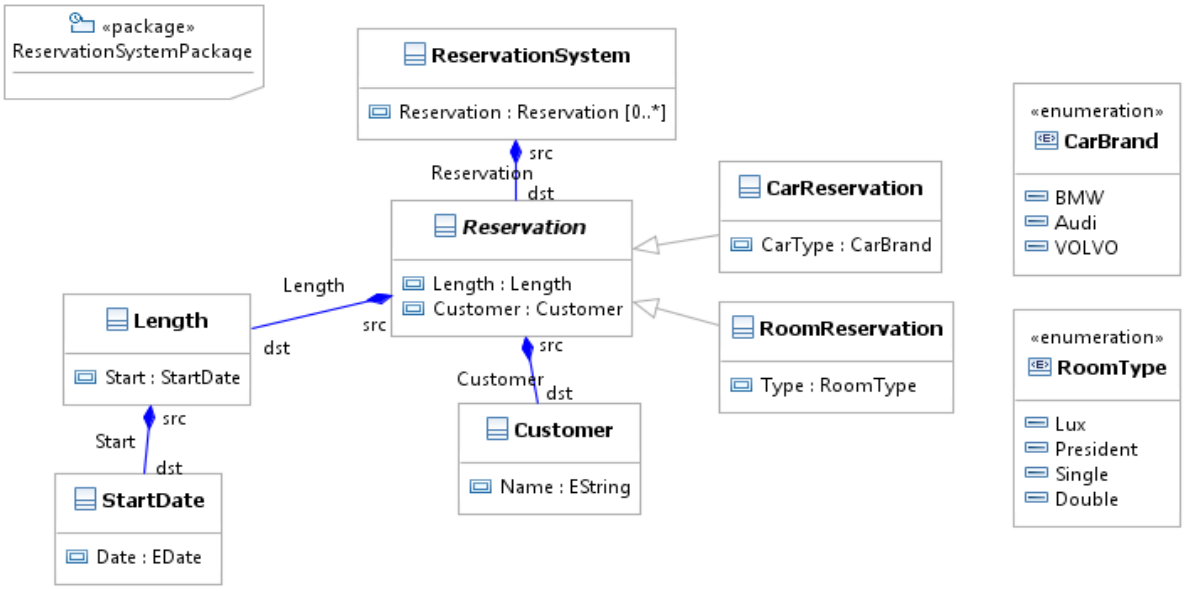


Figure 5. Configured class diagram of the reservation system.

14.4. Project realization from UML

And, finally, the last step in creating a new system implementation from configuration is as follows:

1. Create new EMF Project.
2. Select model importer as UML model and specify the configured UML model.
3. Take a look at what you have got in created ecore file (0).
4. Generate Model, Edit, Editor Code from ".genmodel" file.
5. Your project implementation is finished, now you can test your project.

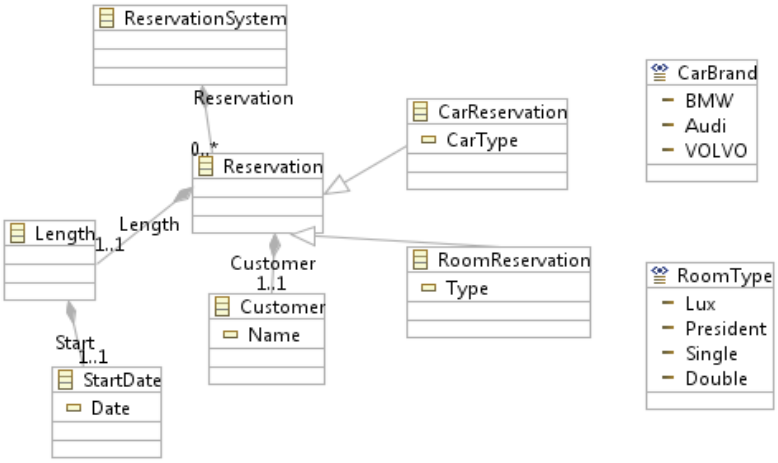


Figure 6. Configured ECore diagram of the reservation system.

15. Conclusion and future work

FD2 is a feature modelling and generation tool for the automated software product line development. Software designers usually have the design expertise and understanding of user requirements, but maybe lack the detailed coding skills. Using our tool, software designers can focus on the high-level design issues. A software designer can use our feature modelling tool FD2 to do high-level design work by selecting feature design choices in a product's feature model. A feature model contains information how to resolve variability in the product line. Feature models in FD2 have many different sources of variability: the existence of a feature, the structure and behaviour of a feature, dependencies between feature design choices, etc. Using FD2, a designer can control what variability is exposed without knowing information beforehand about possible design decisions a designer will make. After a software designer finishes a feature model, he can link it with the UML model (class diagram) and start configuration of a product instance model.

The FD2 tool allows a software system engineer to manage the architecture of a software product line using UML models rather than on a source code or configuration file level. Using the configured UML class diagram it is possible to create different implementations of the software product. In fact, the product line configuration process is brought into a higher abstraction level.

Future work will focus on developing a better product configuration manager, supporting associations with other UML elements for better documentation generation from UML, testing FD2 on real-world feature models and PL examples. Lessons and experiences learned during implementation of the FD2 project can contribute to the research field of automatic application development based on feature modelling. The FD2 tool also will promote a new paradigm of feature-oriented software development based on high-level modelling and automated software development technologies.

References

[1] **Bosch, J.** Design & Use of Software Architectures: Adopting and evolving a product-line approach. Addison-Wesley, 2000.

[2] **Clements, P., and Northrop, L.** *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2002.

[3] **Kang, K., Lee, J., and Donohoe, P.** Feature-Oriented Product Line Engineering. IEEE Software, Vol. 19, No.4, 58-65, 2002.

[4] **Czarnecki, K., Kim C.H.P., and Kalleberg K.T.** Feature Models are Views on Ontologies. Software Product Line Conference, SPLC 2006, Baltimore, USA, August 21-24, 2006, pp. 41-51.

[5] **Djebbi, O. and Salinesi C.** Criteria for Comparing Requirements Variability Modeling Notations for Product Lines, 4th Int. Workshop on Comparative Evaluation in Requirements Engineering. Minneapolis/St. Paul, MN, USA, 2006.

[6] **Schobbens, P.-Y., Heymans P., Trigaux J.-Ch. and Bontemps Y.** Feature Diagrams: A Survey and a Formal Semantics. Proc. of 14th IEEE Int. Requirements Engineering Conference (RE'06), 11-15 September, 2006, 139-148.

[7] **Sipka, M.** Exploring the Commonality in Feature Modeling Notations. M. Bieliková (Ed.), Slovak University of Technology, IIT.SRC 2005, April 27, 2005, 139-144.

[8] **Damaševičius, R., and Štuikys, V.** Design of Ontology-Based Generative Components Using Enriched Feature Diagrams and Meta-Programming. Information Technology & Control, Vol. 37, No., 4, 2008, 301-310.

[9] **Damaševičius, R., Štuikys, V., and Toldinas, E.** Domain Ontology-Based Generative Component Design Using Feature Diagrams and Meta-Programming Techniques. Proc. of 2nd European Conference on Software Architecture ECSA 2008, September 29 - October 1, Paphos, Cyprus. Springer, LNCS 5292, pp. 338-341.