KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Paulius Žaliaduonis

POŽYMIŲ DIAGRAMŲ IR UML KLASIŲ DIAGRAMŲ INTEGRAVIMO TYRIMAS

Magistro darbas

Darbo vadovas:

Doc. Dr. Robertas Damaševičius

KAUNAS, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Paulius Žaliaduonis

POŽYMIŲ DIAGRAMŲ IR UML KLASIŲ DIAGRAMŲ INTEGRAVIMO TYRIMAS

Magistro darbas

Recenzentas

Doc. Dr. Eugenijus Toldinas

Vadovas

Doc. Dr. Robertas Damaševičius

Atliko

IFM-4/2 gr.studentas
Paulius Žaliaduonis

KAUNAS, 2010

TURINYS

LENTELĖS

PAVEIKSLAI

SUMMARY

Feature modeling is important approach to deal system variability at higher abstraction level. Variability models define the variability of a software product line. Unfortunately, it is not integrated into a modeling framework like the Unified Modeling Language (UML). To use it in conjunction with UML, it is important to integrate feature modeling into UML.

This thesis describes the way how feature variability models can be linked with existing UML models and how it is done in the feature modeling tool FD2. The feature modeling tool is described and the complete example provided.

Chapter 2 discusses the way of Feature model integration with UML model.

Chapter 3 describes the implementation of FD2 tool.

Chapter 4 discusses the advantages and disadvantages of FD2 tool.

Chapter 5 provides examples and discusses their results.

In conclusion this thesis propose feature modeling integration with UML modeling, discusses the program developed during master project, provides 2 examples and discusses their results, points out some issues requiring further work.

1. ĮVADAS

Programų sistemų kūrimas, kai yra daug užsakovų, kurių reikalavimai skiriasi, yra sudėtingas procesas ir reikalauja aprašyti galimus programų sistemos variantus. Programų variantiškumui aprašyti naudojami kuriamos sistemos požymių modeliai. Sistemos požymių modeliavimas yra svarbus variantiškumo aprašymo metodas. Sistemos požymių variantiškumo modeliai aprašo aibę programų sistemų, kurios dar vadinamos programų sistemų linija.

Programų sistemų linija yra eilė panašių programų kurios dalinasi bendrais atributais. Tiksliau apibūdinti programų sistemų linijai yra nustatomi sistemų atributai ir jų tarpusavio sąryšiai, jie yra pavaizduojami požymių diagramose. Požymis tai savitas, charakteringas sistemos atributas, kuris nusako matomus sistemos atributus, tačiau nesigilina į detalų sistemos apibūdinimą [9].

Greitam ir kokybiškam programų sistemos variantiškumo modeliavimui reikalingas geras įrankis. Tam skirtas požymių diagramų modeliavimo įrankis, nes sukurti požymių modeliai yra informatyvūs ir gali lengvai perteikti sistemos variantiškumo informaciją.

Tačiau programų sistemos požymių diagrama neturi techninės informacijos, kuri yra reikalinga programos kūrimui. Ši informacija yra saugoma UML modeliuose. Programos UML modelį galima išplėsti variantiškumo informacija, papildant jį sistemos požymių modelio informacija.

Magistrinio projekto metu buvo sukurtas įrankis (FD2), kuris įgyvendina požymių diagramos susiejimą su UML klasių diagrama.

Magistriniame darbe tiriamas sistemų variantiškumo modeliavimas ir galimybė keisti sistemų UML klasių diagramas pagal sudarytą programų sistemos požymių diagramą. Šis tyrimas aprašo požymių modelio sujungimą su UML klasių diagrama, paaiškina sprendimus ir sukurtos požymių modeliavimo programos įgyvendinimą.

Taip pat tiriamos FD2 programos galimybės ir realizacijos veiksmingumas, identifikuojamos klaidos ir sėkmingi problemų sprendimai. Įrankio bandymams buvo sukurti eksperimentiniai pavyzdžiai, mažos ir didesnės programų sistemos duomenų modeliai. Šie pavyzdžiai tiriami, lyginami ir analizuojami. Aprašomi pasiekti rezultatai.

2. POŽYMIŲ DIAGRAMŲ IR UML KLASIŲ DIAGRAMŲ INTEGRAVIMO ANALIZĖ

2.1. Požymių modeliavimo įrankis

Šiame darbe bus nagrinėjamas požymių modeliavimo įrankis FD2, kurį sukūrė Kauno Technologijos Universiteto studentai Paulius Žaliaduonis ir Deividas Kreivys, kaip magistrinio darbo projektą.

2.2. Pagrindiniai darbo tikslai ir uždaviniai

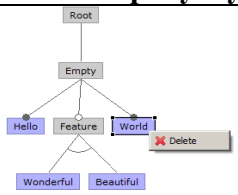
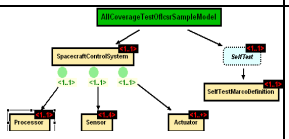
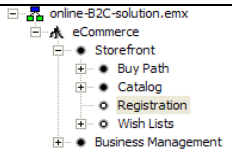
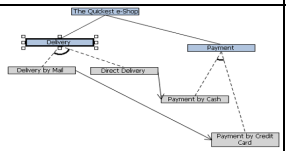
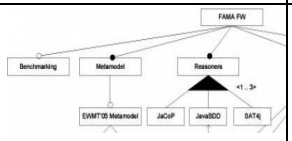
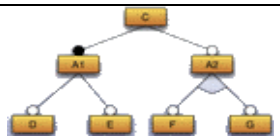
Išanalizuoti kuriamos sistemos požymių modeliavimą naudojant požymių diagramas susietas su UML klasių diagramomis, taip, kaip tai yra realizuota FD2 modeliavimo įrankyje. FD2 yra programa leidžianti kurti Požymių diagramas ir jas susieti su UML klasių diagramomis. Šio darbo tikslas yra:

1. Pagrįsti požymių diagramų ir UML klasių diagramų integravimo poreikį;
2. Aprašyti siūlomą integravimo meta modelį;
3. Išanalizuoti požymių diagramų ir UML klasių diagramų integravimo problemas kuriant požymių modeliavimo įrankį;
4. Ištirti UML klasių diagramų variantiškumo modelių konfigūravimo problemas;
5. Ištirti specializuotų UML klasių diagramų generavimo iš UML klasių diagramų variantiškumo modelių generavimo problemas;
6. Pateikti pavyzdį.

2.3. Egzistuojantys sprendimai pasaulyje

Atlikome kelių populiariausių požymių modeliavimo įrankių apžvalgą. Jos rezultatai pateikti 1-oje lentelėje. Detalus įrankių palyginimas pateiktas 9.1 skyriuje.

1 lentelė. Žinomi požymių modeliavimo įrankiai.

Požymių modeliavimo įrankis	**Požymių diagramos sintaksės pavyzdys**	**Privalumai**	**Minusai**
Feature IDE [14]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas	-Naudojama tik 1 grafine notacija -Neįlieta į kitas programas -Nesusieta su UML
Xfeature [15]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas -Galimos skirtingos diagramų sintaksės -Eclipse programos priedas	-Nesusieta su UML
Ecore / Feature Modelling Plug-in [16]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas -Eclipse programos priedas	-Diagramos vaizduojamos kaip įskleidžiamas medis. -Nesusieta su UML
RequiLine [17]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas	-nenumatytas programos kodo kūrimas -Neįlieta į kitas programas -Nesusieta su UML
FAMA [18]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas -Eclipse programos priedas	-Nesusieta su UML
KumbangTools [19]		-Nemokamas (atviro kodo) -XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas -Eclipse programos priedas	-Nesusieta su UML
pure::variants [20]		-XMI duomenų failas -diagramos klaidu tikrinimas -programinio kodo kūrimas -Eclipse programos priedas	-Komercinė licenzija -Nesusieta su UML

Iš 1-os lentelės matome, kad egzistuojantys požymių modeliavimo įrankiai nesusieja požymių su UML elementais, todėl mūsų sukurtas FD2 įrankis įgyvendins šią funkciją.

2.4. UML galimybės sistemų variantiškumui modeliuoti

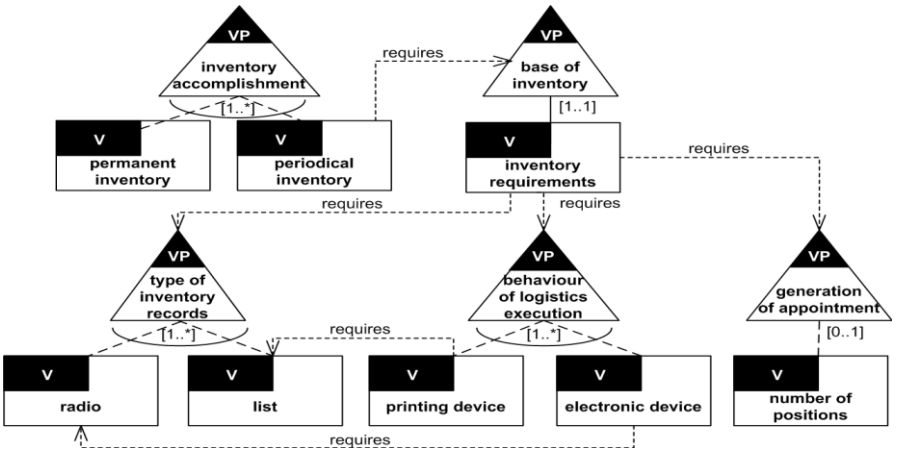
2.4.1. Įžanga

UML, tai modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus [13]. Šiuo metu UML yra labiausiai paplitęs programinės įrangos specifikavimo standartas. UML yra labai lanksti grafinio modeliavimo kalba.

Šiais laikais kuriamos vis didesnės programų sistemos ir pateikiamos rinkai vis platesnės programų sistemų šeimynos. Programų šeimynų variantiškumą galima aprašyti UML diagramose apibrėžiant tam tikrų elementų būtinumą sistemoje, nustatant jų paskirtį. Šioje dalyje apžvelgsime variantiškumo žymėjimo galimybes.

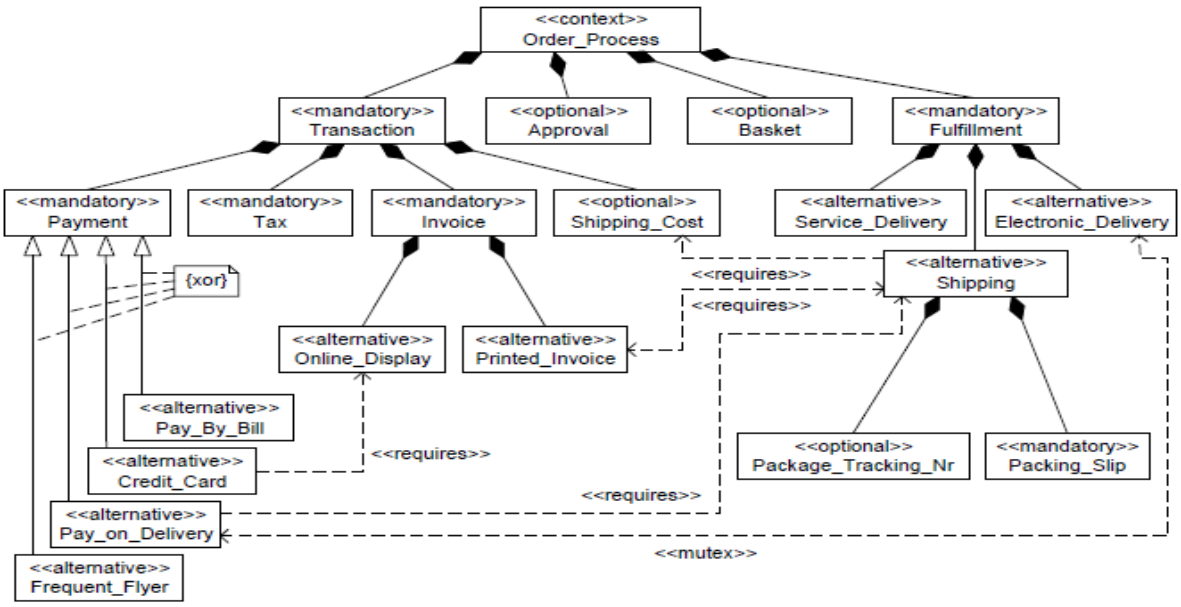
2.4.2. UML variantiškumo modeliavimo galimybės

UML grafinio modeliavimo kalba neturi galimybės modeliuoti produktų variantiškumo, tam buvo pasiūlytas UML modelių variantiškumo žymėjimo profilis [8]. Šis pasiūlytas UML profilis, truputi panašus į FD2 įrankyje naudojamą požymių diagramą. Sudarytos požymių diagramos pavyzdys pateiktas 1-ame paveiksle.



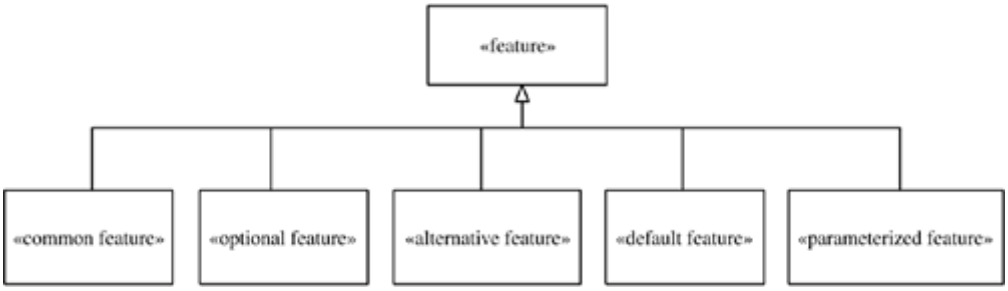
1 paveikslas. UML profilyje naudojamas požymių diagrama.

Taip pat Matthias Clauß siūlo žymėti modelių variantiškumą UML stereotipuose [6][7]. Šis variantiškumo sudarymo būdas yra labai detalus ir gilinasi į sistemos detales. Stereotipais žymimo variantiškumo pavyzdys pateiktas 2-ame paveiksle.



2 paveikslas. Stereotipais žymimas sistemos variantiškumas.

Požymių sistemų variantiškumo žymėjimas naudojant stereotipus yra glaustai aprašytas Hassan Gomaa knygoje [5]. Šioje knygoje detaliai aprašomos UML modeliavimo galimybės. Sistemos požymių diagramą siūlo sudaryti iš stereotipų, kurie bus naudojami UML elementuose. Požymių diagramos vaizduojamos stereotipų medžiu pavyzdys pateiktas 3-ame, 4-ame paveiksluose.

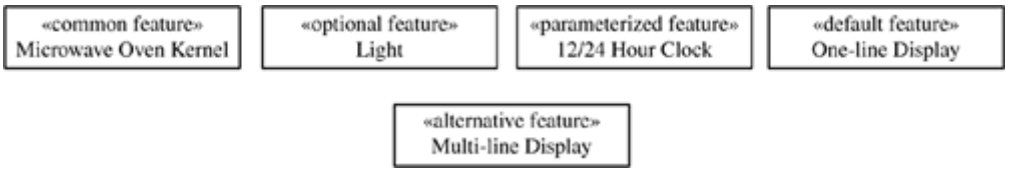


3 paveikslas. Galimi sistemos požymių kategorijos.

3 paveikslas vaizduoja UML stereotipus, kurie nurodo galimas sistemos požymių kategorijas. <<feature>> yra bendra abstrakti sistemos požymio kategorija, toliau iš eilės išvardintos sistemos požymių kategorijos paveldinčios abstrakčią požymių kategoriją. Detaliau aprašomi ir palyginami su požymių diagramų elementais 2 lentelėje.

2 lentelė. Požymių kategorijų reikšmės.

Stereotipas	**Jo reikšmė**	**Atitikmuo požymių diagramoje**
Feature	Bendra abstrakti sistemos požymio kategorija.	-
Common	Būtinas	Būtinas
Optional	Nebūtinas	Nebūtinas
Alternative	Pasirenkamas	Pasirenkamas
Default	Numatytas	-
Parametrized	Su parametru	-



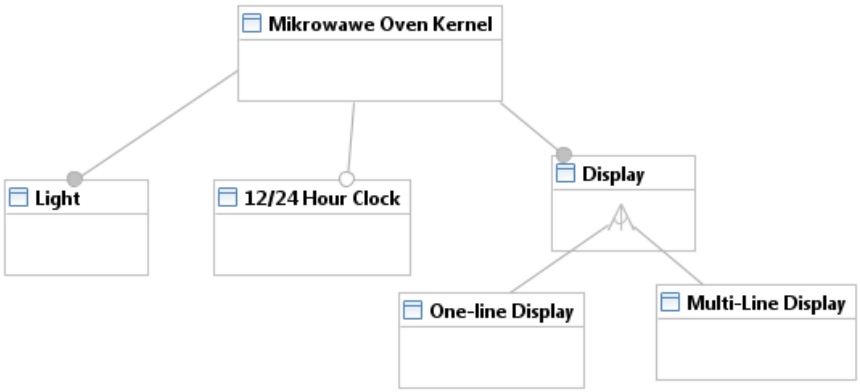
4 paveikslas. 5 paveikslas. Mikrobanginės krosnelės sistemos požymiai.

4 paveikslas nurodo nagrinėjamos sistemos požymius. Požymių aprašymas pateiktas 3 lentelėje.

3 lentelė. Sistemos požymių aprašymai.

Požymis	**Aprašymas**	**Būtinumas sistemoje**
Mikrowawe Oven Kernel	Mikrobangė	Būtinas
Light	Apšvietimas	Nebūtina
12/24 Hour Clock	12 arba 24 valandų laikrodis	Su parametru
One-line Display	Vienos linijos ekranas	Numatyta
Multi-Line Display	Daugelio linijų ekranas	Pasirenkamas

Taip pat 4-ame paveiksle pavaizduota sistema galėtų būti atvaizduota požymių diagrama iš 5-o paveikslo.



5 paveikslas. Mikrobanginės krosnelės požymių diagrama.

2.4.3. UML variantiškumo modeliavimo trūkumai

2.4.2 paragrafe buvo paminėti keli būdai sistemų variantiškumui modeliuoti. Šie variantiškumo modeliai vaizduojami tiesiogiai UML diagramose, todėl jos tampa perkrautos ir sunkiai skaitomos. Taip pat vienoje diagramoje nėra įmanoma peržiūrėti visos sistemos variantiškumo. Dėl to, didesnių sistemų variantiškumų modelis tampa neinformatyvus. Techninė sistemų variantiškumų dalis aiški, tačiau sistemos visumąf suprasti sunku.

Kita problema kiltų naudojant standartinius įrankius kurie kuria programų kodą iš UML klasių diagramos. Sunku atsekti variantiškumą kuriant programinį kodą. Tam reiktų kurti specialius kodo generatorius, kurie atsižvelgtų į tam tikrus stereotipus.

2.4.4. Požymių diagramos pranašumai

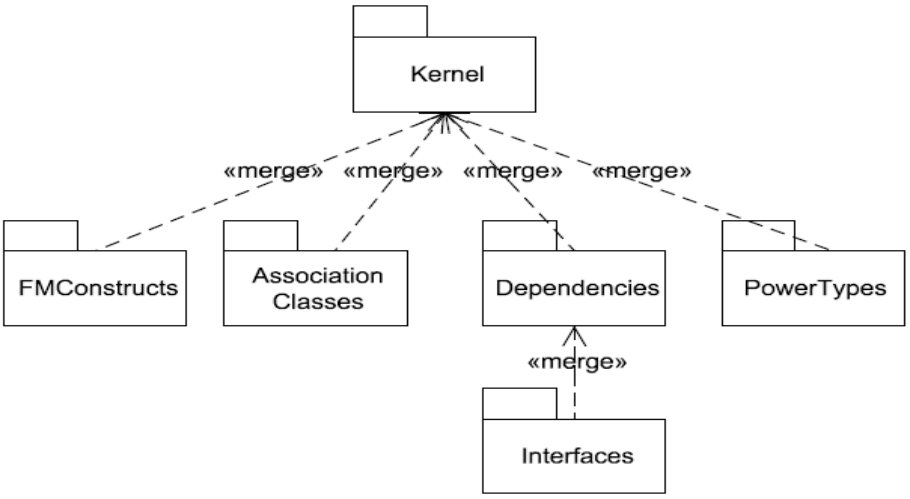
1. Požymių diagrama yra skirta sistemų variantiškumui modeliuoti, todėl ji yra informatyvi, lengvai suprantama ir skaitoma.
2. Požymių diagrama yra UML įrankio priedas, todėl neturėtų jokios įtakos UML diagramų vaizdavimui, UML modeliai liktų nepakitę.
3. Yra galimybė išsaugoti požymių modelio variantiškumą, todėl galima lengvai valdyti skirtingas sistemų konfigūracijas.
4. Modifikavus UML diagramą galima naudoti standartinį programinio kodo generatorių.

2.4.5. Išvados

Norime pasiūlyti vaizduoti sistemos variantiškumą požymių diagramoje ir neapkrauti UML diagramų pertekline informacija, nes UML modeliai negali patogiai atvaizduoti programų šeimynos variantiškumo galimybių.

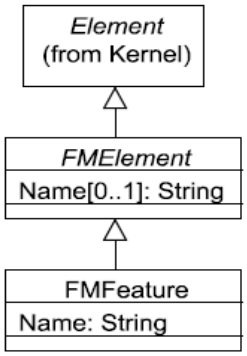
2.5. Valentino Vranić ir Ján Šnirc siūlomo UML meta modelio praplėtimo požymių meta modeliu apžvalga

Apžvelgtas Valentino Vranić ir Ján Šnirc metodas ([10] literatūros šaltinis). UML modeliavimo meta modelis buvo išplėstas papildant jį meta klasėmis ir meta ryšiais. Šis išplėtimas nekeičia UML meta modelyje esančių ryšių ir elemento, tik papildo papildomais. Požymių modeliavimo plėtinys yra apribotas pakete pavadinimu „FMConstructs“. Šis paketas sujungiamas su UML baziniu paketu, grafiškai pavaizduota 6-ame paveiksle.



6 paveikslas. FMConstructs komponento sujungimas su UML paketu.

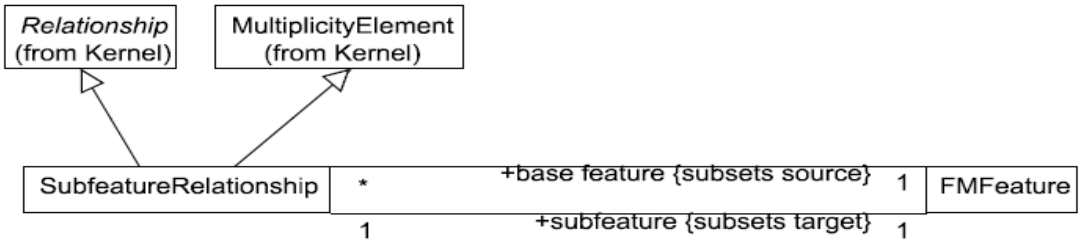
Bazinis šio plėtinio elementas yra „FMElement“, tai yra abstrakti meta klasė (7 paveikslas). Ji paveldi ir „UML Element“ klasę ir tuo pačių elementui būdingus atributus (duomenų laukus ir galimus ryšius su kitais UML elementais).



7 paveikslas. FMElement – meta klasė.

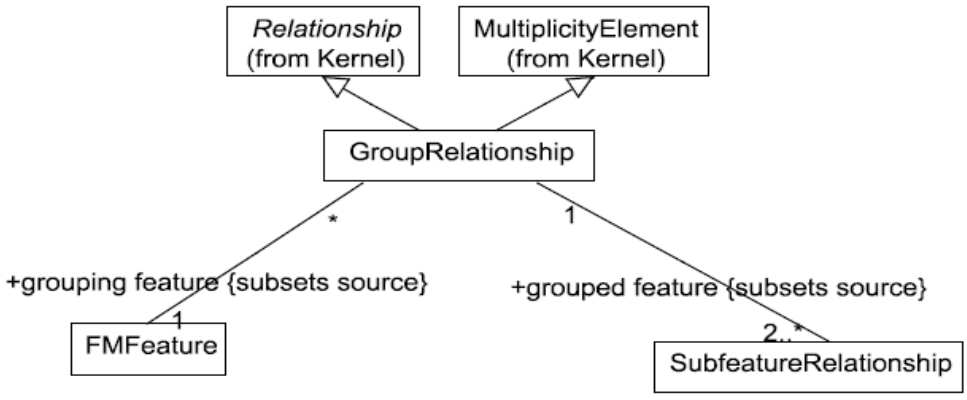
Naudodami toki modeli, jau galime kurti diagramas, tačiau yra reikalavimas, kad požymių modeliai butų kryptingo medžio formos. Tam įtraukiamas UML stereotipas <<concept>>, kuris nurodys, kad tai šakninis požymio modelio elementas.

Požymių diagramos ryšiai yra valdomi 8-ame paveiksle pavaizduota duomenų struktūra. „SubfeatureRelationship“ ryšio klasė paveldi „UML Relationship“ (ryšio funkcionalumas) ir „UML MultiplicityElement“ (ryšių kardinalumo funkcija). Taip pat turi ryšį su požymio klase.



8 paveikslas. Ryšio tarp požymių meta klasė.

Grupuojantis ryšys tarp požymių, yra sudaromas vieną požymį susiejant su keliais požymiais (pasirinkimo ryšys – tik vienas arba bent vienas iš visų ryšiu susietų elementų). Šis ryšys pavaizduotas 9-ame paveiksle. „GroupRelationship“ ryšio klasė paveldi „UML Relationship“ (ryšio funkcionalumas) ir „UML MultiplicityElement“ (ryšių kardinalumo funkcija). Taip pat turi ryšį su požymio klase (požymis gali turėti kelis grupuojančius ryšius) ir bent 2 ryšiais (ryšys yra pavaizduotas 8-ame paveiksle).



9 paveikslas. Grupuojančio ryšio tarp požymių meta klasė.

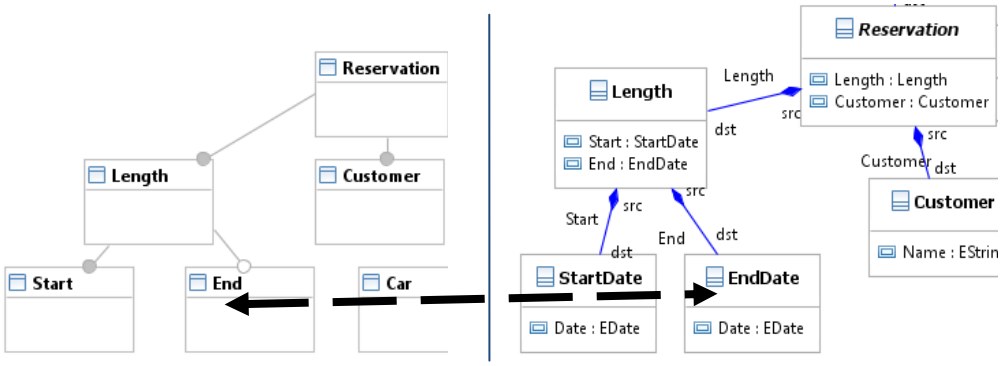
Ryšio klasės paveldi „UML MultiplicityElement‘, ši klasė turi kardinalumo funkciją. Autoriai požymių ryšių kardinalumą naudoja ryšio tipui nurodyti. Pavyzdžiui ryšio kardinalumas gali būti <0..1> (nebūtinas) arba <1..1> (būtinas), o grupuojančio ryšio kardinalumas gali būti <0..1> (vienas iš) arba <1..*> (bent vienas iš).

2.6. Požymio ir UML klasių diagramos elementų susiejimas

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. [V.Štuikys, R. Damaševičius]

2.6.1. Požymių ir UML elementų susiejimo įgyvendinimo galimybės

Požymius su UML klasių diagramų elementais apjungiame paprastu ryšiu, kurį sudaro unikalių raktų pora. Šį ryšį sudaro požymio pavadinimas ir UML elemento pavadinimas. Pavyzdinis ryšys vaizduojamas 10-ame paveiksle. Čia požymių diagramos fragmentas matomas kairėje, o klasių diagramos fragmentas matomas dešinėje. Rodyklė vaizduoja ryšį, kurį sudaro raktinių pavadinimų pora: Požymis „End“ atitinka UML diagramos klasę „EndDate“.



10 paveikslas. Požymio siejimas su UML elementu.

2.6.2. Kylančios problemos

Sujungiant požymį su UML klasės diagrama kyla kelios problemos:

1. Kaip nurodyti ryšį tarp sistemos požymių ir sistemos UML elementų;
2. Kur saugoti informaciją, kuri nurodytų, kurie UML klasių diagramų elementai atitinka tam tikrą požymį;

3. Kaip nurodyti konkrečios sistemos požymių variantiškumo konfigūraciją;
4. Kur išsaugoti konkrečios sistemos požymių variantiškumo konfigūraciją;
5. Kaip, pagal konkrečios sistemos požymių konfigūraciją, modifikuoti UML diagramas į specializuotas konkrečiai sistemai.

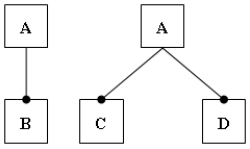
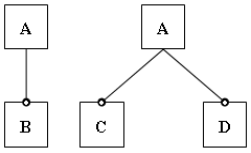
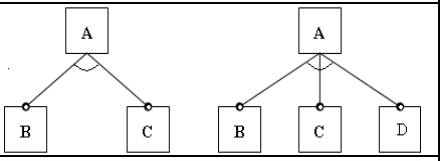
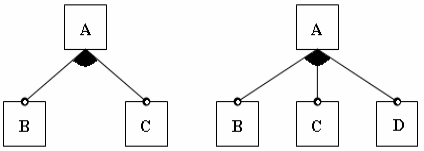
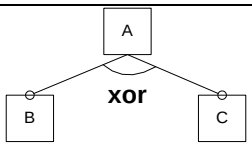
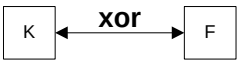
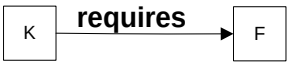
2.6.3. Problemų sprendimas

Kylančias problemas sprendžia FD2 įrankis. Šis įrankis buvo sukurtas tam, kad apjungtų programų sistemų požymių modeliavimą ir UML modeliavimą. Šiame įrankyje buvo sprendžiamos 2.6.2 paragrafe paminėtos problemos. Detaliau aprašysime kiekvienos iš problemų sprendimą.

2.6.3.1 Požymių modeliavimas

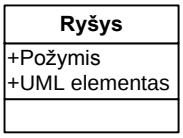
Požymių modeliavimui buvo sukurtas įrankis, galintis kurti grafines diagramas. Grafinių diagramų braižymas buvo įgyvendintas naudojantis V.Štuikio, R.Damaševičius ir J.Toldino [1] pasiūlyta požymių diagramų sintaksę, kuri pateikta 4 lentelėje.

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

Elementas	Apibrėžimas	Grafinis pavyzdys
Būtinas	Požymis B (C,D) yra įtrauktas, jeigu jo tėvas A yra įtrauktas Jei A tuomet B; Jei A tuomet D 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)	
Pasirenkamas (xor - pasirinkimas)	Tik vienas požymis galimas pasirinkti. Ryšys galimas tik tarp dviejų vaikų. Rašoma „xor“ antraštė po tėvynainiu požymiu Jei A tuomet B bet ne C arba C bet ne B	
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	

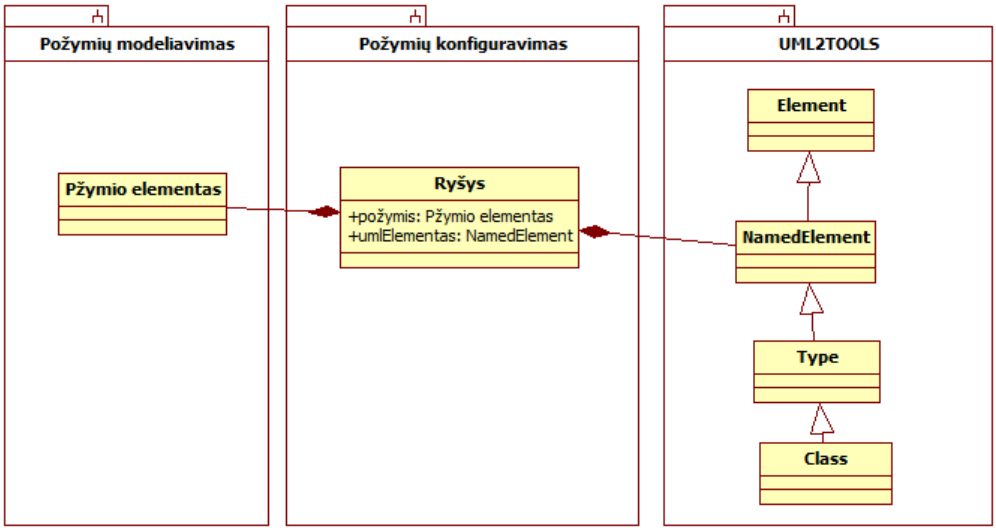
2.6.3.2 Ryšys tarp požymių ir UML klasių diagramų elementų

Ryšys tarp požymio ir UML elemento buvo įgyvendintas siejant elementų pavadinimus. Duomenų modelis pateiktas 11-ame Paveiksle. Šiam ryšiui tarp požymių ir UML modelių pakanka elementų pavadinimų, pagal kuriuos identifikuojami elementai (apsiribojame, kad UML elementų pavadinimai bus unikalūs). Šiame ryšio duomenų modelyje „Požymis“ reiškia požymio elemento pavadinimą, o „UML elementas“ atitinka UML modelyje saugomo elemento pavadinimą.



11 paveikslas. Ryšio tarp UML elemento ir požymio duomenų modelis.

Šis (11 paveikslas) meta modelis naudojamas tarp požymių modeliavimo ir UML modeliavimo komponentų. 12 paveikslas rodo, kaip siejami atskiri komponentai ir jų modeliai.



12 paveikslas. Požymių konfigūravimo meta modelio ryšys su sistema.

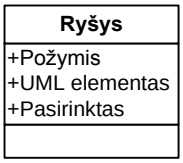
Realizacijoje bus įgyvendintas tik ryšys su UML klasių diagramos elementais, tai reiškia, kad susiejimo metu bus išfiltruojami visi UML elementai, kurie nėra klasių diagramos elementai.

Reikalavimai ir apribojimai šio ryšio įgyvendinimui:

1. Požymio ar UML elementas turi turėti pavadinimą;
2. Požymių elementų pavadinimai būti unikalūs visame požymių modelyje;
3. UML elementų pavadinimai turi būti unikalūs visame UML modelyje;
4. Ryšyje nurodyti požymio ar UML elementų pavadinimai turi egzistuoti.

2.6.3.3 Sistemos požymių variantiškumo konfigūracija

Konkrečios sistemos variantiškumui reikia saugoti papildoma konfigūruojamų požymių informaciją, kuri nurodo ar sistemos požymis yra numatytas konkrečioje sistemoje. Tam ryšio tarp požymio ir UML elemento esybė buvo papildyta pasirinkimo lauku (13 paveikslas).

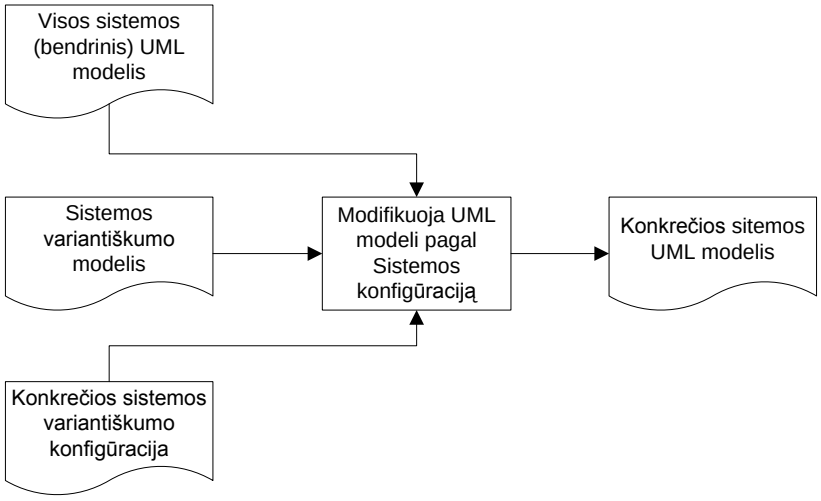


13 paveikslas. Ryšys papildytas pasirinkimo lauku.

Tokių ryšių sąrašas yra saugomas atskirame duomenų faile. Kiekvienai sistemos konfigūracijai sukuriamas atskiras failas. Tokių sistemos konfigūracijos failų gali būti be galo daug, kaip ir sistemos variacijų.

2.6.3.4 UML modelio modifikavimas pagal sistemos požymių variantiškumo konfigūraciją

Apjungiame sistemos variantiškumo ir variantiškumo konfigūravimo modelius. Gauname sistemos konfigūraciją, kurios teisingumas yra patikrintas variantiškumo modelyje (požymių diagrama). Panaudojame šio sistemos konfigūracijos failo duomenis ir modifikuojame UML modelį, taip, kad jame liktų tik tie UML elementai, kurie buvo pažymėti sistemos konfigūracijos faile. Šios procedūros schema pateikta 14-ame paveiksle.



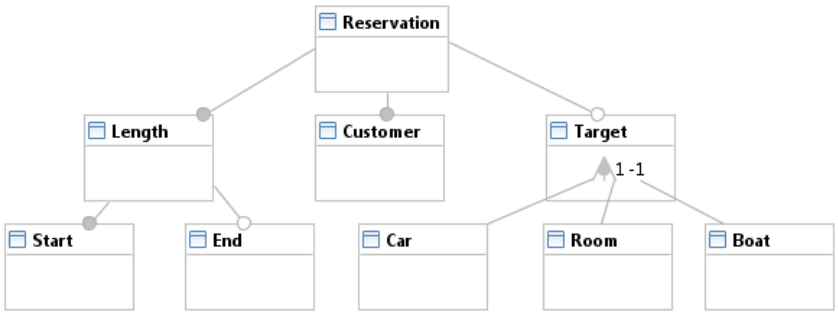
14 paveikslas. UML modelio modifikavimas pagal požymių variantiškumo konfigūraciją.

2.7. Požymių modeliavimo įrankis

Magistrinio projekto metu buvo sukurtas požymių modeliavimo įrankis. Šis įrankis skiriasi nuo egzistuojančių požymių modeliavimo įrankių tuo, kad konfigūruojamas objektas yra ne pati programa, o jos UML modelis. Galutinis šio įrankio produktas yra skirtingų konkrečių programų sistemų UML modelių aibė.

2.7.1. Požymių modeliavimas

FD2 programa leidžia vartotojui kurti programų sistemų variantiškumų modelius, kurie yra tikrinami, kad juose nebūtų loginių klaidų. Grafinis atvaizdavimas yra atliekamas pagal 2.6.3.1 skyriuje 4-oje lentelėje išvardintas sintaksės taisykles. Požymių modeliavimo produktas yra medžio formos sistemos požymių grafas su šakniniu elementu (šakninis elementas yra elementas turintis tik išeinančius ryšius. Pavyzdys pateiktas 15-ame paveiksle.



15 paveikslas. Požymių diagramos pavyzdys.

15-ame paveiksle apibūdinta rezervacijų sistema su keliais atributais:

1. Rezervacijoje būtinai nurodomas klientas;
2. Rezervacijoje būtinai nurodoma jos pradžia, bet pabaiga nebūtina;
3. Rezervacija gali būti 3 tipų: mašinos, kambario, laivo.

2.7.1.1 Variantiškumo modelio sudėtingumo įvertis

Taip pat galima peržiūrėti sukurto variantiškumo modelio sudėtingumo įvertinimus. Šie įvertinimai yra atliekami pagal Vytauto Štuikio ir Roberto Damaševičiaus [2] požymių diagramų įvertinimo metodus.

5 lentelė. Požymių diagramos sudėtingumo įvertinimų paaiškinimai.

Sudėtingumas	**Paaiškinimas**
Cognitive complexity – variation points	Rodo kiek yra pasirinkimo taškų diagramoje. Pasirinkimo taškas tai požymių diagramos elementas iš kurio išeina Nebūtinas, Pasirenkamas (or - pasirinkimas), Pasirenkamas (case - pasirinkimas) ryšiai.
Cognitive complexity – graph levels	Požymių diagramos grafo gylis.
Structural complexity	Rodo kiek požymių diagramos medyje yra vidinių medžių.
Compound complexity	Šis dydis parodo diagramos sudėtingumą. V. Štuikys, R. Damaševičius siūlo naudoti šį dydį [2].

Item	Result	Details
Cognitive complexity:		
Variation points	3	Low < 7 +/-2
Graph levels	3	Low < 7 +/-2
Structural complexity	7	Number of sub-trees
Compound complexity	85,89	Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3R^2)/9
Explanation:		
Features	9	F
Mandaroty	3	Rand
Optional	2	Ror
ExclusiveCase	3	Rcase
CaseOr	0	Ror
Constraints	0	R

16 paveikslas. Rezervacijos požymių modelio sudėtingumo įvertis.

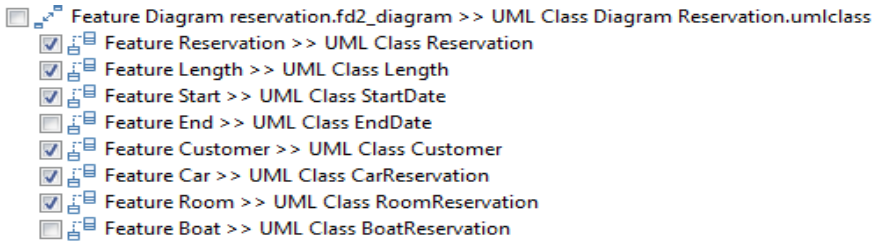
2.7.2. Požymių susiejimas su UML klasių diagrama

Ši funkcija nurodo, kaip sistemos požymiai susiję su UML klasių diagramos elementais. Susiejimas vykdomas diagramų pagalba, nurodant požymio elementą ir UML klasių diagramos elementą (pasirinkimas užtvirtinamas pelės kontekstinio meniu punktu).

2.7.3. Požymių konfigūravimas

Kai yra atliktas požymių susiejimas su UML klasių diagramų elementais, vykdomas požymių variantiškumo konfigūravimas. Jo metu pažymimi požymiai, kurie bus įtraukti į konkrečią programų sistemą. 17 paveikslas rodo pavyzdinį požymių konfigūracijos langą. Šiame lange vartotojas turi varnelėmis pažymėti sistemos požymius, kurie bus įtraukti į konfigūruojamą konkrečią sistemą.

Šiame etape programa neleis vartotojui atlikti klaidų, tai reiškia, kad nebus leidžiama pasirinkti negalimų arba nepasirinkti būtinų požymių.



17 paveikslas. Požymių konfigūravimo vaizdas.

Šią požymių konfigūraciją galima išsaugoti į failą, kurį vėliau bus galima atidaryti. Požymių konfigūracijos faile saugomas požymių ryšių sąrašas (sąrašo elementų duomenų modelis 13-ame paveiksle).

2.7.4. Specializuotos UML diagramos generavimas

Šis žingsnis sukuria konkrečios sistemos UML modelį, kuriame nėra įtraukti nepasirinktų požymių elementai. Kurie UML elementai bus įtraukti į modelį apsprendžia požymių konfigūracijos failas (aprašytas 2.7.3 skyriuje).

Konkrečios programų sistemos UML modelis kuriamas tokiais žingsniais:

1. Nuskaitoma požymių konfigūracija,
2. Nuskaitomas visos sistemos UML modelis,
3. Sudaromas sąrašas UML elementų, kurie bus šalinami (konfigūracijoje neparinkti elementai),
4. Iš visos sistemos UML modelio šalinami su šalinamais elementais susiję elementai,
5. Šalinami numatyti elementai.

Sukurtas konkrečios sistemos UML modelis atitinka standartą aprašytą UML specifikacijoje [13]. Tai įgyvendinama naudojant UML2TOOLS komponento įrankius, kurie yra atsakingi už UML failų kūrimą, skaitymą, grafinį vaizdavimą ir kitas funkcijas.

2.8. Įvertinimas

Apžvelgta literatūra:

1. Egzistuojantys sprendimai pasaulyje,
2. UML galimybės sistemų variantiškumui modeliuoti,
3. UML meta modelio praplėtimo požymių meta modeliu.

Magistrinio projekto metu buvo sukurtas FD2 įrankis, kuris realizuoja programų sistemų variantiškumo požymių ir UML modelių integraciją. Pagrindinis FD2 įrankio funkcionalumas yra:

1. Požymių modeliavimas,
2. Variantiškumo modelio sudėtingumo įvertįs,
3. Požymių susiejimas su UML klasių diagrama,
4. Požymių konfigūravimas,
5. Specializuotos UML diagramos generavimas.

Buvo pasiūlytas ir aprašytas požymių ir UML modelių integravimo meta modelis, kuris įgyvendina:

1. Požymių modeliavimą,
2. Požymių siejimą su UML klasių diagramos elementais,

3. Požymių modelio konfigūravimą,
4. Konkrečios UML diagramos kūrimo procesą.

Pasiūlytas požymių modelio ir UML modelio integravimo būdas išsprendžia požymių susiejimą su UML elementais, požymių modelių konfigūraciją ir aprašo apribojimus. Taip pat pasiektas modelių nepriklausomumas (modeliai nėra griežtai susiejami. angl. loose coupling) ir lengvai keičiami nepriklausomai vienas nuo kito.

3. POŽYMIŲ MODELIAVIMO ĮRANKIO PROJEKTINĖ DALIS

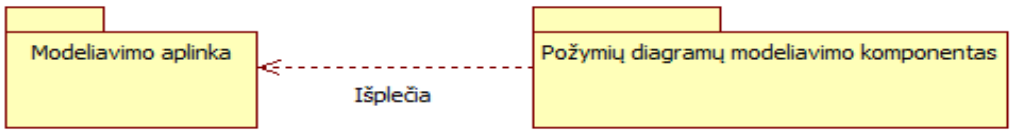
3.1. Įvadas

Šiame skyriuje bus pateiktas išsamus architektūrinis kuriamos sistemos vaizdas, parodyta, kaip realizuota sistema, nurodoma kokios klasės bus sukurtos, su kokiais duomenimis dirbs sistema, kur bus saugojama informacija, kokios duomenų struktūros bus naudojamos. Taip pat bus pagrįsti programos architektūros pasirinkimai, nustatomi sistemos apribojimai.

Tikslas yra sukurti požymių diagramų komponentą praplečiantį UML modeliavimo įrankį. Šis komponentas turi panaudoti esamo UML diagramų braižymo komponento savybes. Jis turi UML modeliavimo įrankį papildyti požymių braižymo galimybe ir UML modelių konkrečiai sistemai sudarymu.

3.2. Įrankio architektūros apžvalga

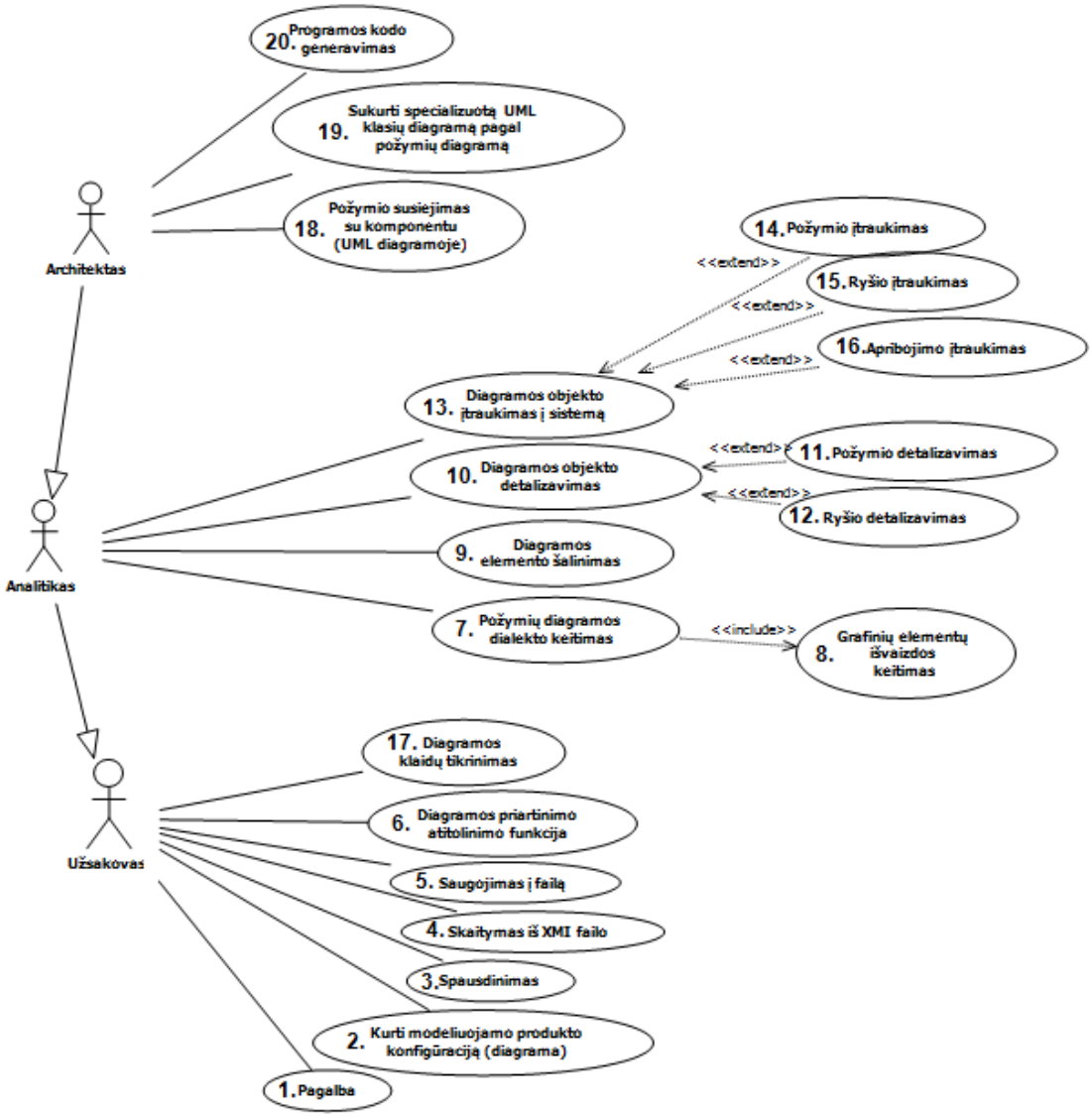
Požymių modeliavimo įrankis buvo kuriamas, kaip jau esančių UML modeliavimo įrankių plėtinys. Tam buvo analizuojami egzistuojantys požymių modeliavimo bei UML modeliavimo įrankiai ir pasirinktas palankiausias sprendimo būdas. 18-ame paveiksle pateiktas vaizdas, kaip įrankis turi būti įlietas į egzistuojantį modeliavimo įrankį.



18 paveikslas. Požymių modeliavimo įrankis modeliavimo įrankyje. Struktūrinis vaizdas.

3.3. Požymių modeliavimo projekto funkciniai reikalavimai

3.3.1. Panaudojimo atvejai



19 paveikslas. Panaudojimo atvejų diagrama.

6 lentelė. 19-o paveikslo panaudojimo atvejų aktorių aprašymai.

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.

7 lentelė. 19-o paveikslo panaudojimo atvejų aprašymai.

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: -
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.
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: -
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:

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.
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: -
7. Požymių diagramos dialekto keitimas	Tikslas: Galima pakeisti požymių diagramų išvaizdą. Aktoriai: Analitikas Ryšiai su kitais PA: Naudoja „grafinių elementų išvaizdos keitimas“ 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.
8. Grafinių elementų išvaizdos keitimas	Tikslas: Galima pakeisti diagramos elementų linijų storį, spalvą, šriftą. Aktoriai: Analitikas Ryšiai su kitais PA: Naudojamas „Požymių diagramos dialekto keitimas“. Nefunkciniai reikalavimai: - Prieš sąlygos: - Sužadinimo sąlyga: Spaudžiamas meniu mygtukas „keisti grafinę išvaizdą“. Po sąlyga: Pakeista grafinių elementų išvaizda. Pagrindinis scenarijus: Spaudžiamas meniu mygtukas „keisti grafinę išvaizdą“, pasirodžiusiame dialogo lange pasirenkami pageidaujami nustatymai. Alternatyvūs scenarijai: Operaciją galima atšaukti.

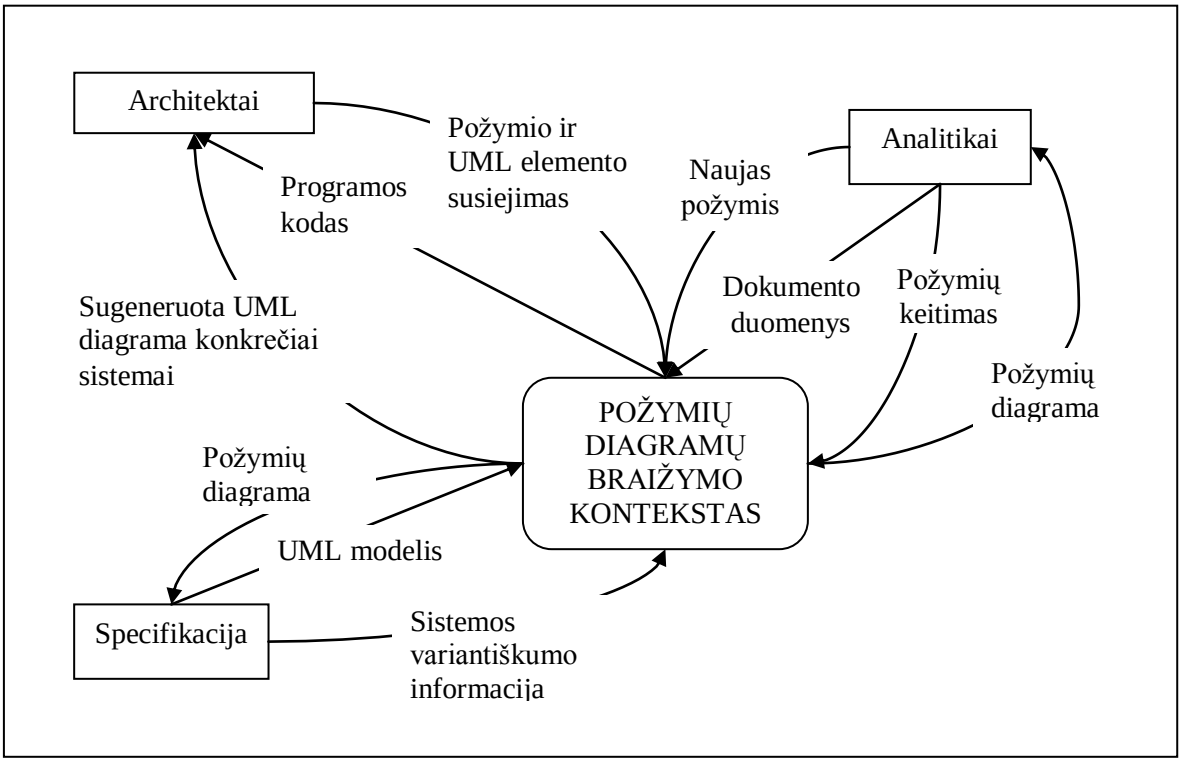
9. Diagramos elemento šalinimas	Tikslas: Pašalinamas pažymėtas diagramos elementas. Aktoriai: Analitikas Ryšiai su kitais PA: - Nefunkciniai reikalavimai: - Prieš sąlygos: Atidaryta diagrama ir pažymėtas jos elementas. Sužadinimo sąlyga: Spaudžiamas mygtukas „trinti“ arba klaviatūros mygtukas „debete“. Po sąlyga: Ištrintas diagramos elementas. Pagrindinis scenarijus: Pažymimas diagramos elementas, spaudžiamas mygtukas „trinti“ arba klaviatūros mygtukas „debete“. Alternatyvūs scenarijai: -
10. Diagramos elemento detalizavimas	Tikslas: Galima pakeisti tam tikrus elementų požymius. Aktoriai: Analitikas Ryšiai su kitais PA: Bazinis elementas, jį praplečia: „Požymio detalizavimas“, „Ryšio detalizavimas“. 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: -
11. Požymio detalizavimas	Tikslas: Galima nurodyti požymio pavadinimą bei komentarą. Aktoriai: Analitikas Ryšiai su kitais PA: Praplečia „Diagramos elemento detalizavimas“. Nefunkciniai reikalavimai: - Prieš sąlygos: - Sužadinimo sąlyga: - Po sąlyga: - Pagrindinis scenarijus: - Alternatyvūs scenarijai: -
12. Ryšio detalizavimas	Tikslas: Galima nurodyti ryšio kardinalumą, tipą, bei komentarą. Aktoriai: Analitikas Ryšiai su kitais PA: Praplečia „Diagramos elemento detalizavimas“. Nefunkciniai reikalavimai: - Prieš sąlygos: - Sužadinimo sąlyga: - Po sąlyga: - Pagrindinis scenarijus: - Alternatyvūs scenarijai: -

13. Diagramos elemento įtraukimas į sistemą	Tikslas: Pasirinkto diagramos elemento įtraukimas į sistemą. Aktoriai: Analitikas Ryšiai su kitais PA: Bazinis elementas, Jį praplečia: „Požymio įtraukimas“, „Ryšio įtraukimas“, „Apribojimo įtraukimas“. 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: -
14. Požymio įtraukimas	Tikslas: Naujo požymio įtraukimas į diagramą. Aktoriai: Analitikas Ryšiai su kitais PA: Praplečia „Diagramos elemento įtraukimas į sistemą“. Nefunkciniai reikalavimai: - Prieš sąlygos: Atidaryta diagrama. Sužadinimo sąlyga: Pasirenkamas įrankis ir dedamas į diagramą. Po sąlyga: Įdėtas požymis. Pagrindinis scenarijus: Pasirenkamas požymio elemento įrankis pelyte, paspaudus ant diagramos įtraukiamas, reikia nurodyti elemento pavadinimą. Alternatyvūs scenarijai: Operacija gali būti atšaukiama.
15. Ryšio įtraukimas	Tikslas: Naujo ryšio tarp požymių įtraukimas. Aktoriai: Analitikas Ryšiai su kitais PA: Praplečia „Diagramos elemento įtraukimas į sistemą“. Nefunkciniai reikalavimai: Diagramoje turi būti bent 2 požymiai. Prieš sąlygos: Atidaryta diagrama. Sužadinimo sąlyga: Pasirenkamas įrankis ir dedamas į diagramą. Po sąlyga: Įtrauktas ryšys tarp 2 požymių. Pagrindinis scenarijus: Pasirenkamas ryšio elemento įrankis pelyte, spaudžiama ant diagramoje esančio požymio, spaudžiama ant antro požymio. Alternatyvūs scenarijai: Operacija gali būti atšaukiama.

16. Apribojimo įtraukimas	Tikslas: Naujo apribojimo įtraukimas. Aktoriai: Analitikas Ryšiai su kitais PA: Praplečia „Diagramos elemento įtraukimas į sistemą“. Nefunkciniai reikalavimai: Diagramoje turi būti bent 2 požymiai. Prieš sąlygos: Atidaryta diagrama. Sužadinimo sąlyga: Pasirenkamas įrankis ir dedamas į diagramą. Po sąlyga: Įtrauktas apribojimas tarp 2 požymių. Pagrindinis scenarijus: Pasirenkamas apribojimo elemento įrankis pelyte, spaudžiama ant diagramoje esančio požymio, spaudžiama ant antro požymio. Alternatyvūs scenarijai: Operacija gali būti atšaukiama.
17. 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: -
18. 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.

19. Sukurti specializuotą UML klasių diagramą pagal požymių diagramą	Tikslas: Sukuriama UML klasių diagrama. Aktoriai: Architektas. Ryšiai su kitais PA: - 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.
20. 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.

3.3.2. Veiklos kontekstas



20 paveikslas. Sistemos konteksto diagrama.

Veiklos kontekste aprašomi požymių modeliavimo įrankio duomenų srautai, mainai. Detaliau kiekvienas srautas aprašytas 8 lentelėje.

8 lentelė. 20-o paveikslo veiklos įvykių sąrašas.

Eil. Nr.	**Įvykio pavadinimas**	**Įeinantys/Išeinantys informacijos srautai**
1	Architektui prireikia sugeneruoti programos kodą	Naujas programos kodas (išeinantis srautas)
2	Architektas susieja UML komponentą su požymiu	Naujas požymio susiejimas (įeinantis srautas)
3	Architektui prireikia UML diagramos pagal tam tikrus požymius	Naujos UML diagramos generavimas (išeinantis srautas)
4	Analitikas pateikia naują požymį	Naujas požymis (įeinantis srautas)
5	Analitikas modifikuoja požymius	Požymių keitimas (įeinantis srautas)
6	Specifikacijoje esančios UML diagramos	UML diagramos (in)
7	Specifikacijoje pateiktas programos komponentų variantiškumas	Sistemos variantiškumo informacija (įeinantis srautas)
8	Požymių diagramų sumodeliavimas	Požymių diagrama (išeinantis srautas)
9	Analitikas įkelia diagramos duomenis iš dokumento	Dokumento duomenys (įeinantis srautas)

3.4. Požymių modeliavimo projekto nefunkciniai reikalavimai

3.4.1. Reikalavimai sistemos išvaizdai

Sistemos išvaizda priklauso nuo Eclipse sistemos diagramų braižymo komponento. Požymių diagramų braižymo modulis, turi naudoti požymių diagramų sintaksės apibrėžtus elementus aprašytus 2.6.3.1 skyriuje, vartotojui suteikiama galimybė pasikeisti spalvas, šrifto dydžius, linijų storius.

Požymių diagramų braižymo langas turi būti panašus į UML diagramų braižymo langus. 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:

1. Požymių braižymo elementai turi būti įrankių juostoje, vizualia ir tekstine forma;
2. Turi būti požymių diagramos priartinimo/atitolinimo („zoom“) funkcija;
3. Požymių braižymas turi būti atliekamas naudojant kompiuterio pelę bei klaviatūrą;
4. Pagalbos funkcija.

3.4.3. Reikalavimai vykdymo charakteristikoms

3.4.3.1 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.

3.4.3.2 Talpumas (duomenų apimtis)

Duomenys bus saugomi XMI duomenų tipo dokumentais [26]. Šio saugojimo metu talpumą apibrėš duomenų kiekis. Vaizduojama grafinė informacija bus užrašomas koordinatėmis bei spalvų kodais.

3.4.3.3 Panaudojimo efektyvumas

Turi būti išnaudoti jau esami Eclipse sistemos algoritmai, bei bibliotekos. Taip pat algoritmai turi būti efektyvūs. Duomenų skaitymas iš XMI failų turi būti greitas.

3.4.3.4 Patikimumas

Sistema turi taisyklingai brėžti diagramas. Jeigu vykdomi neleistini veiksmai, programa turi informuoti vartotoją. Sistema turi veikti patikimai, nepriklausomai nuo pasirinktos operacinės sistemos (Windows ar Linux).

3.4.3.5 Išplečiamumas

Požymių modeliavimo įrankis veiks kaip Eclipse sistemos komponentas, todėl jis turi būti lengvai diegiamas.

3.4.3.6 Reikalavimai veikimo sąlygoms

Sistema turi sėkmingai veikti tiek Windows tie Linux operacinėse sistemose.

Vartotojas naudojantis požymių modeliavimo sistemą, turi būti susipažinęs su programų sistemų linijos modeliavimu.

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

3.4.3.7 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 gali būti naudojamas įvairiomis kalbomis.

3.4.4. Reikalavimai saugumui

Sistema bus projektuojama ir kuriama kaip atsiras sistemos komponentas, todėl dalis saugumo priklausys nuo Eclipse produkto saugumo sprendimų. Sistema turi apsisaugoti nuo klaidingų duomenų įvedimo.

3.4.5. 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.

3.4.6. Teisiniai reikalavimai

Sistema turi būti kuriama nemokama programavimo kalba (Java) ant atvirojo kodo sistemos Eclipse kaip komponentas išplečiantis UML diagramų braižymo komponentą. Kuriamas komponentas turi būti nemokamas, kaip ir Eclipse sistema. Sukurtas komponentas yra saugomas EPL licenzijos (The Eclipse Public License) ir atitinka EPL sąlygas.

3.5. Požymių modeliavimo projekto architektūros specifikacija

Požymių modeliavimo įrankis bus pateikiamas keliais vaizdais: panaudojimo atvejų vaizdu, procesų vaizdu, išdėstymo vaizdu ir realizavimo vaizdu. Šie vaizdai bus pateikti vieninga modeliavimo kalba (UML):

1. Panaudojimo atvejų vaizdą sudarys panaudojimo atvejų diagrama;
2. Statinį vaizdą sudarys klasių diagramos ir sistemos išskaidymas į paketus;
3. Dinaminį sistemos vaidą sudarys būsenų, sekų ir būsenų diagramos;
4. Išdėstymo vaizdą sudarys išdėstymo diagrama.

3.5.1. Priimti techniniai sprendimai įrankio kūrimui

Prieš kuriant požymių modeliavimo programą sujungtą su UML modeliavimo programa buvo analizuojami iki tol buvę sukurti požymių modeliavimo įrankiai. Nebuvo rastas tenkinantis sprendimas, todėl nuspręsta išplėsti funkcionalų UML modeliavimo įrankį požymių modeliavimo dalimi. Tam buvo sudaryti pradiniai reikalavimai kuriamai programai:

1. turi išplėsti UML modeliavimo programą;
2. turi būti atviro kodo;
3. nekomercinis sprendimas.

Šie reikalavimai realizuoti:

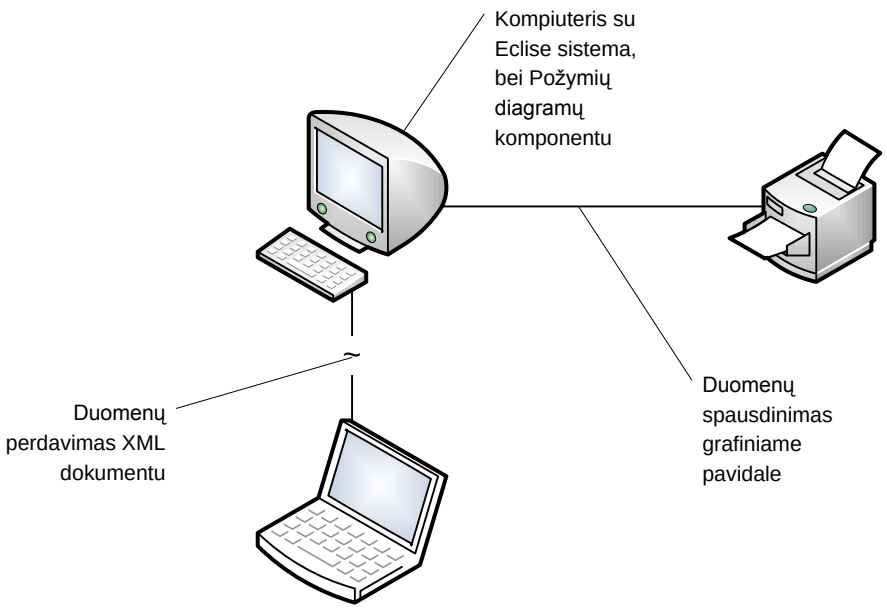
1. Pasirinktas „UML2TOOLS“ UML modeliavimo įrankis;
2. Pasirinkta atviro kodo programų platforma „Eclipse“;
3. „Eclipse“ programų platforma ir FD2 požymių modeliavimo įrankis sukurtas pagal Eclipse Public License (EPL) licenziją.

Sprendimas rinktis „UML2TOOLS“ įrankį ir kurti priedą Eclipse aplinkai buvo atliktas išanalizavus eilę įrankių. Detalus įrankių palyginimas pateiktas 9.1 priede.

3.5.2. Numatoma požymių modeliavimo įrankio eksploatacijos aplinka

Sistema veiks tiek Windows tiek Linux operacinėse sistemose. Kadangi naudojama JAVA programavimo kalba, kuri palaikoma abiejose operacinėse sistemose. Taip pat Eclipse sistema sukurta JAVA programavimo kalba ir yra nepriklausoma nuo operacinės sistemos.

Sistema veiks viename kompiuteryje, ji nereikalaus serverio ir papildomų techninės įrangos komponentų. Sistemos išdėstymas pateiktas 21-ame paveiksle.



21 paveikslas. Sistemos išsidėstymas techninėje įrangoje

Minimalūs kompiuterio duomenys turėtų būti:

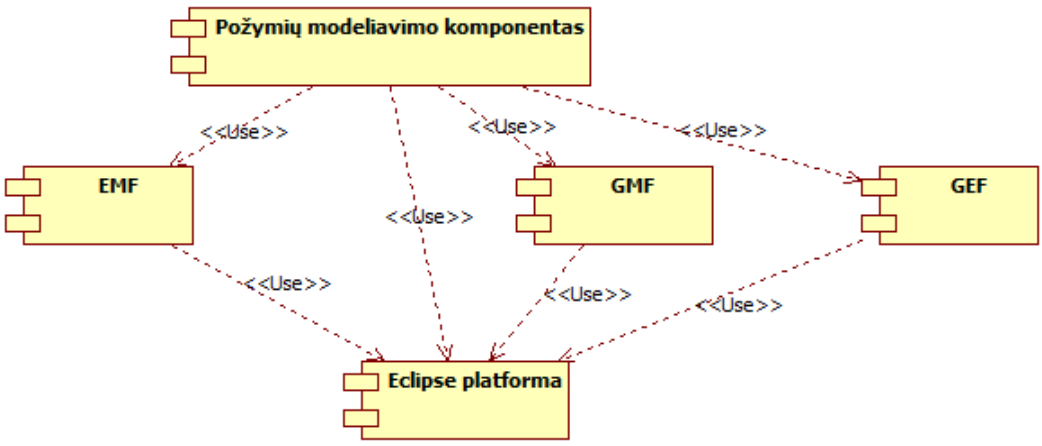
1. Procesorius: 400 MHz
2. Operatyvioji atmintis: 128 MB RAM
3. Vieta kietame diske: 400MB

Šie reikalavimai kompiuteriui keliami dėl Eclipse programų sistemai keliamų minimalių reikalavimų, papildomų techninės įrangos reikalavimų nėra.

3.5.3. Sprendimų įtaka požymių modeliavimo sistemos architektūrai

Požymių modeliavimo įrankio kūrimui buvo pasirinkta Eclipse programų sistema ir į ją integruotas UML modeliavimo įrankis UML2TOOLS. Daugelis programos architektūrinių sprendimų buvo orientuoti į įliejimo galimybes Eclipse sistemoje.

Nuosekliam komponento įliejimui i Eclipse sistemą buvo panaudoti Eclipse sistemoje pateikti komponentų rinkiniai. Šių komponentų priklausomybė pateikta 22-ame paveiksle. Taip pat panaudotos Eclipse sistemai kuriamų grafinių diagramų komponentų kūrimo rekomendacijos [25].



22 paveikslas. Požymių modeliavimo komponentų priklausomybė Eclipse programų sistemoje.

3.5.3.1 GMF komponentas

Tai komponentas, būtinas paleisti GMF įrankiais sukurtas programas, jis yra bazinis šių programų komponentas. Šiame komponente yra įgyvendinti grafinių diagramų vaizdavimo metodai.

GMF bibliotekų suteikiami privalumai:

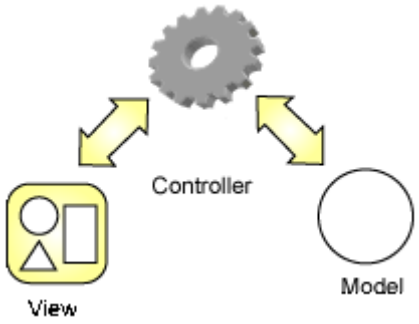
1. Šis komponentas suteikia, visiems GMF biblioteką naudojantiems įrankiams, panašią išvaizdą, kuri neblaško vartotojo.
2. Jis valdo diagramos paišymą, dėl to programuotojui nereikia rūpintis diagramos elementų paišymu ir išdėstymu. Programuotojui lieka tinkamai įgyvendinti biznio logiką.
3. Palieka galimybę sukurtam naujam komponentui, kuris naudoja GMF Runtime, būti išplėstam kitų komponentų. Taip pat leidžia lengvai integruotis į kitus komponentus, kurie sukurti pasitelkiant GMF bibliotekas.
4. GMF bibliotekos yra gerai ištestuotos ir plačiai naudojamos visame pasaulyje, dėl to sumažėja klaidų atsiradimo tikimybė.

3.5.3.2 EMF komponentas

EMF tai modeliavimo ir programinio kodo kūrimo biblioteka, skirta kurti programas ir įrankius pagal struktūrizuotą duomenų modelį. EMF įrankiai padeda aprašytus modelius transformuoti į JAVA programavimo kalbos kodą. EMF modeliai yra sukuriami iš UML klasių diagramų, EMF modelį būtų galima įsivaizduoti, kaip UML klasių diagramą. EMF modeliai naudoja XMI duomenų formatą.

3.5.3.3 GEF komponentas

Tai grafinių elementų piešimo biblioteka. Ji praplečia draw2d biblioteką ir suteikia diagramoms galimybę sietis su pelės ar klaviatūros veiksmais. GEF tarpininkauja tarp modelio ir modelio vaizdo (23 paveikslas).

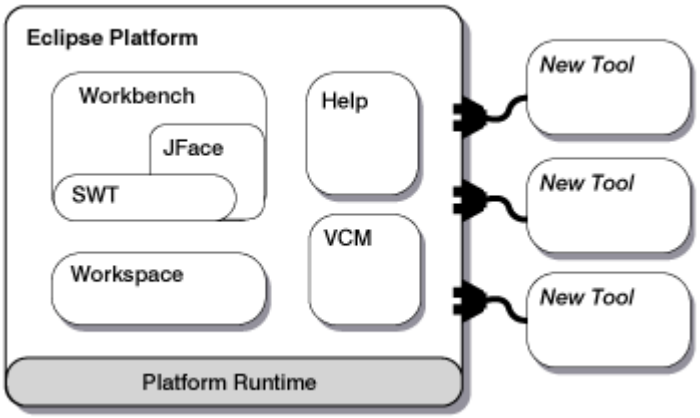


23 paveikslas. Modelis – valdymas – vaizdas

[http://www.ibm.com/developerworks/opensource/library/os-gef /, 2009]

3.5.3.4 Eclipse platforma

Eclipse platforma arba kitaip kamienas yra sudarytas iš daug komponentų. Dalis komponentų yra baziniai, kurie valdo kitų komponentų užkrovimą į atmintų, jų bendravimą su kitais komponentais. Kiti yra papildomi komponentai, kurie suteikia tam tikras galimybes Eclipse kamienui (grafinė vartotojo sąsaja). Geriausiai Eclipse kamienas pavaizduotas žemiau pateiktoje diagramoje (24 paveikslas):



24 paveikslas. Eclipse kamienas, ir kiti komponentai

[http://www.ibm.com/developerworks/library/os-plat/, 2009]

3.5.4. Požymių modeliavimo projekto apribojimai

Šis požymių modeliavimo įrankis sukurtas „Eclipse“ programų sistemų aplinkai, konkrečiai „Modeling Tools“ programos variantui. Šis Eclipse programos paketas turi visas pageidaujamas savybes programų modeliavimui ir programinio kodo kūrimui iš sukurtų UML diagramų.

3.5.4.1 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.

3.5.4.2 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).

3.5.4.3 Bendradarbiaujančios sistemos

Požymių diagramų modeliavimo modulis yra glaudžiai susijęs su Eclipse programų pakete naudojamas moduliais, taigi jo teisinga sąveika su skirtingomis Eclipse paketo dalimis nulems sėkmingą jo veikimą.

Požymių modeliavimo įrankis buvo sukurtas iš atskirų dalių. Dalys atskirai apibūdintos 9 lentelėje.

9 lentelė. FD2 įrankio komponentų aprašymai.

Komponentas	Aprašymas
1.com.ktu.fd2	Požymių diagramos duomenų modelis.
2.com.ktu.fd2.uml	Požymių diagramos sąsaja su UML klasių diagrama.
3.com.ktu.fd2.tests	Požymių modeliavimo įrankio vienetų testai.
4.com.ktu.fd2.help	Įrankio pagalbos failai.
5.com.ktu.fd2.edit	Požymių diagramos duomenų modelio redaktoriaus logika.
6.com.ktu.fd2.editor	Grafinis požymių diagramos duomenų modelio redaktorius.
7.com.ktu.fd2.diagram	Grafinis požymių diagramos duomenų modelio redaktorius įgalinantis požymių diagramos sintaksę, aprašytą 2.6.3.1 skyriuje.
8.com.ktu.fd2.custom	Įvairūs požymių modeliavimo įrankio papildymai.
9.com.ktu.fd2.complexity	Požymių modelio sudėtinguo skaičiavimo įrankis.

3.5.4.4 Komerciniai specializuoti programų paketai

Požymių diagramų modeliavimo modulio kūrime bus naudojamos tik nemokamos ir atviro kodo bibliotekos ir moduliai.

3.5.4.5 Numatoma darbo vietos aplinka

Darbo vieta kurioje dirbs vartotojas yra tipinė kontorinė darbo vieta. Darbo vietą būtinai sudarys kompiuteris su valdymo įtaisais (pelė, klaviatūra) ir didesnės įstrižainės monitorius. Ši darbo vieta turi atitikti visus ergonominius reikalavimus.

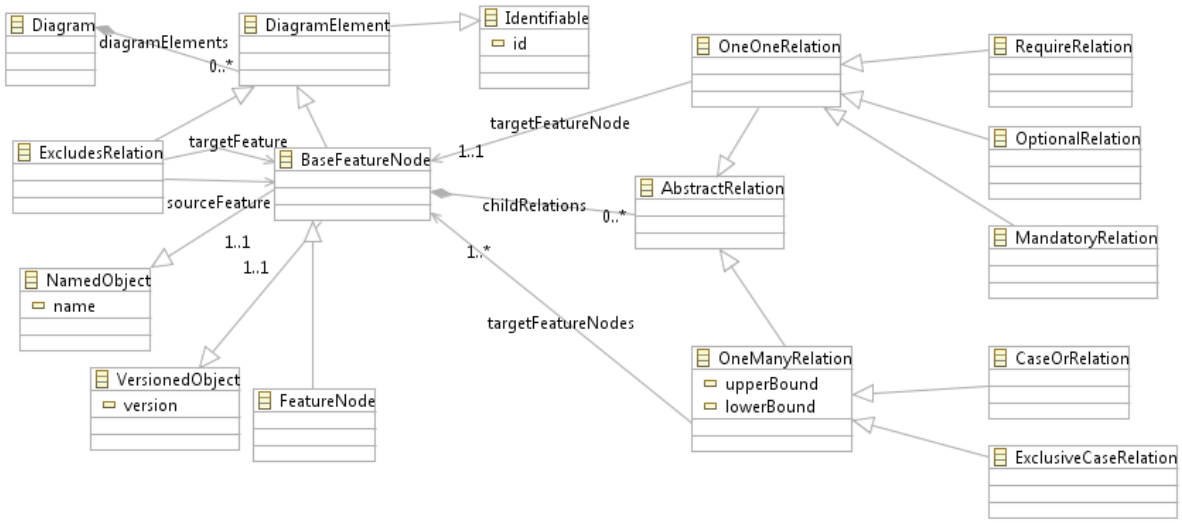
3.5.4.6 Svarbūs faktai ir prielaidos

Eclipse sistemoje naudojamų įrankių nesuderinamumas ar nepritaikymas dirbti su kitais įrankiais gali įtakoti įrankio veikimą. Sukurto sistemos priedo stabilus darbas gali būti užtikrintas tik su suderintais kitais priedais. Atsiradus kitiems Eclipse sistemos priedams, kurie turėtų įtakos sukurto priedo darbui, jis gali nustoti veikti, arba veikti neteisingai.

3.5.5. Duomenų vaizdas

Sistema kuriai kuriamas komponentas duomenis nuskaito iš kompiuterio kietojo disko, todėl buvo sukurtas požymių diagramos meta modelis. Šis modelis nusako kaip yra susiję požymių diagramos elementai vienas su kitu (25 paveikslas). Šis modelis naudojamas požymių modelio duomenų saugojimui faile. Į failą duomenys rašomi XMI formatu ir yra lengvai skaitomi ir pernešami.

Šis duomenų modelis buvo sudarytas vadovaujantis patirtimi, sukaupta tyrėjų, kurie mėgino įgyvendinti požymių modeliavimą. Miguel A. Laguna, José M. Marqués [12] darbas – diagrama turi matyti visus diagramoje matomus elementus. Valentino Vranić ir Ján Šnirc [10] straipsnyje apibūdintas meta modelis aprašė ryšius tarp elementų.



25 paveikslas. Požymių diagramos duomenų modelis.

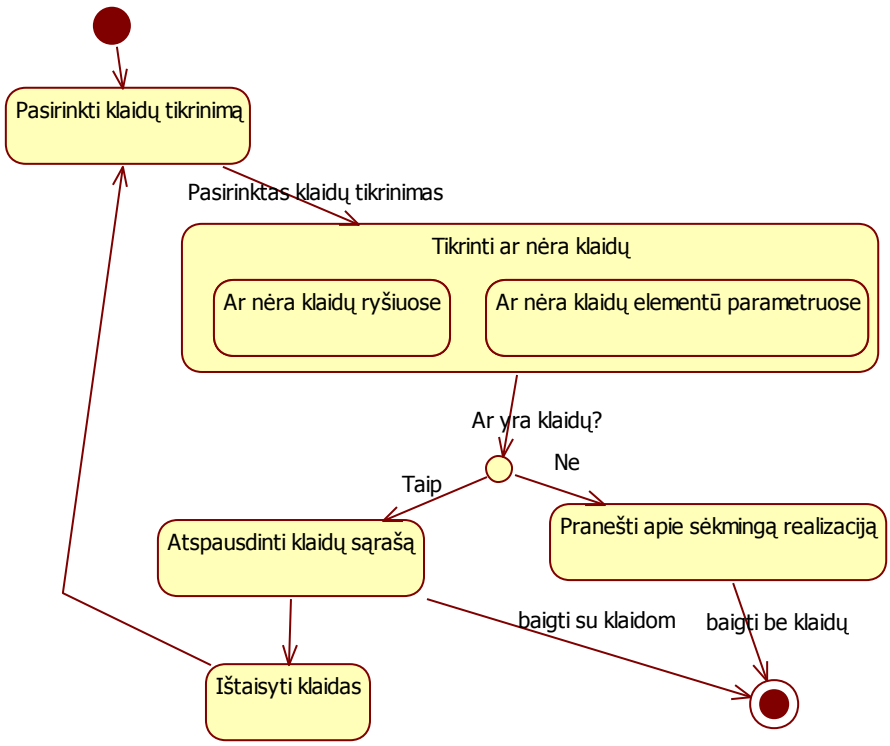
3.5.6. Sistemos dinaminis vaizdas

Šioje dalyje bus aprašytos svarbiausios ir sudėtingiausios požymių modeliavimo komponento funkcijos:

1. Diagramos klaidų tikrinimas (3.5.6.1 skyrelis),
2. Požymio susiejimas su komponentu (3.5.6.2 skyrelis),
3. Kurti modeliuojamo produkto konfigūraciją (3.5.6.3 skyrelis),
4. Sukurti specializuotą UML klasių diagramą pagal požymių diagramą (3.5.6.4 skyrelis).

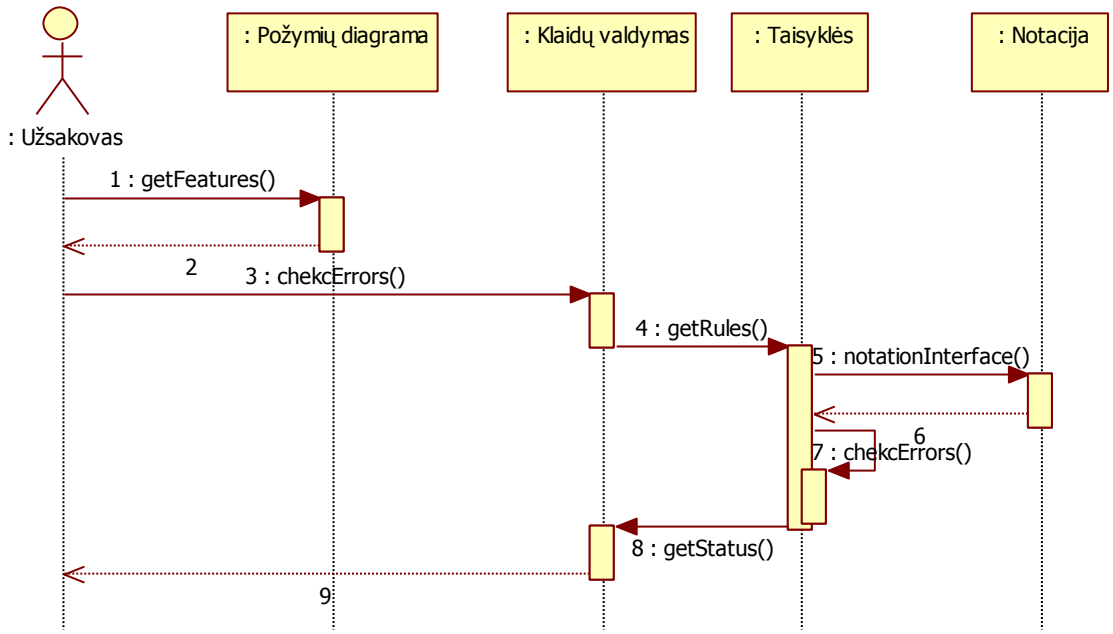
Šių funkcijų vaizdavimas atliekamas veiklos, sekų ir būsenų diagramomis.

3.5.6.1 Diagramos klaidų tikrinimas



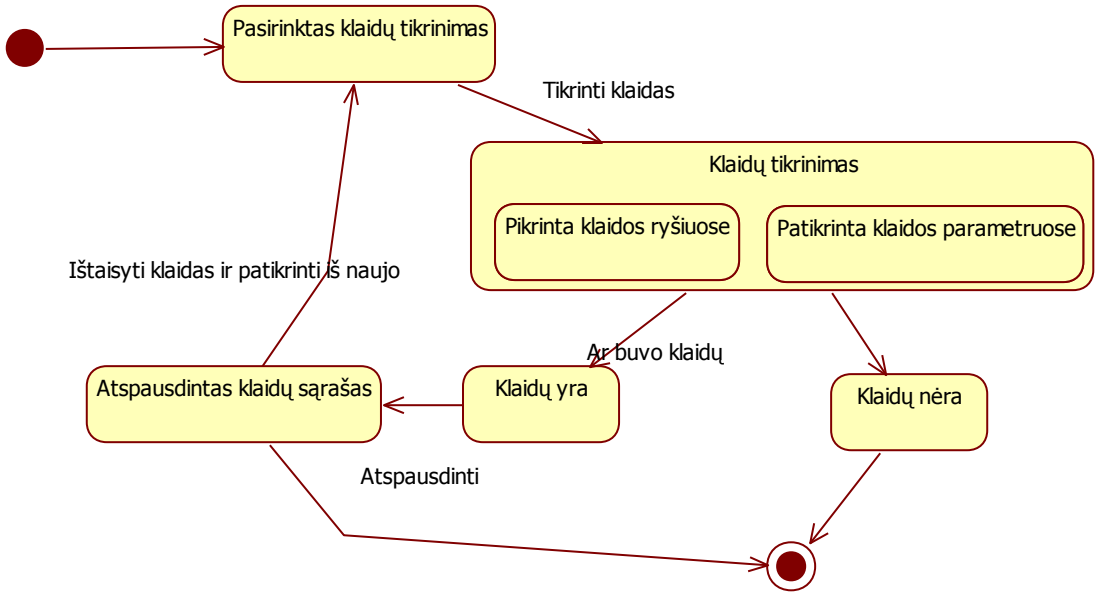
26 paveikslas. Diagramos klaidų tikrinimas. Veiklos diagrama.

Pasirenkamas diagramos klaidų tikrinimas, patikrinama ar nėra klaidų ryšiuose, ar nėra klaidų elementų parametruose jeigu buvo klaidų atspausdinamas klaidų sąrašas klaidos ištaisomos ir kartojami žingsniai nuo klaidų tikrinimo pasirinkimo arba baigiama ties klaidų sąrašo atspausdinimu. Jeigu atlikus patikrinimą, klaidų nebuvo pranešama apie sėkmingą požymių diagramos realizaciją (26 paveikslas).



27 paveikslas. Diagramos klaidų tikrinimas. Sekos diagrama.

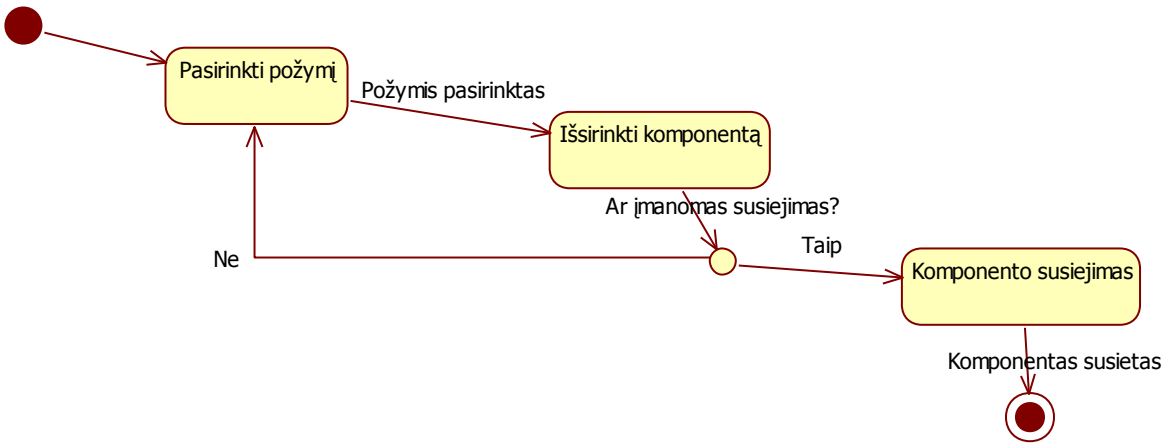
Pasirenkama požymių diagrama, tikrinamos klaidos, gaunamos taisyklės, patikrinamos klaidos ir perduodama būsena (27 paveikslas).



28 paveikslas. Diagramos klaidų tikrinimas. Būsenos diagrama.

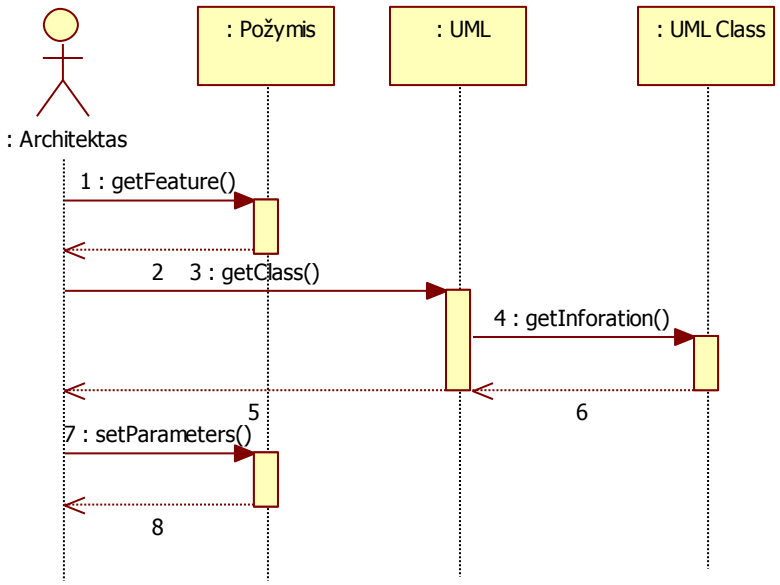
Iš būsenos „pasirinktas klaidų tikrinimas“ pereinama į būseną „klaidų tikrinimas“ atlikus patikrą pereinama į atitinkamas būsenas priklausomai ar buvo rasta klaidų ar ne. Klaidų sąrašo spausdinimo būsena pasiekiama tik radus klaidų (28 paveikslas).

3.5.6.2 Požymio susiejimas su komponentu



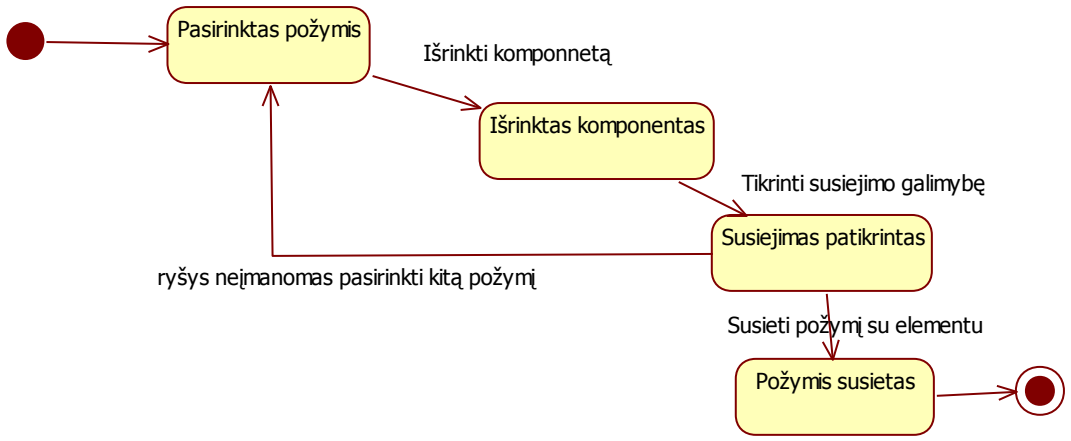
29 paveikslas. Požymio susiejimas su komponentu. Veiklos diagrama.

Pasirenkamas požymis, išsirenkamas norimas komponentas, jeigu susiejimas yra įmanomas komponentas susiejamas su požymiu, jei ne reikia pasirinkti kitą požymį (29 paveikslas).



30 paveikslas. Požymio susiejimas su komponentu. Sekos diagrama.

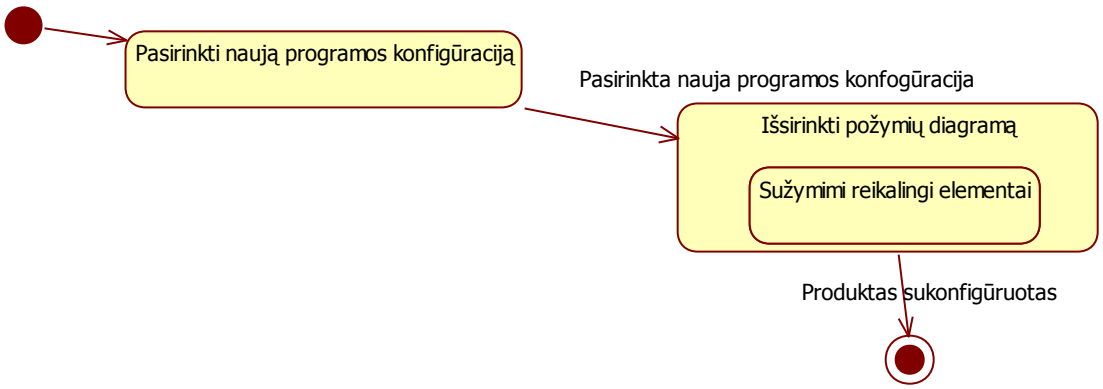
Pasirenkamas požymis, pasirenkama UML klasė, patikrinama ar įmanomas susiejimas ir nustatomi parametrai prie požymio (30 paveikslas).



31 paveikslas. Požymio susiejimas su komponentu. Būsenos diagrama.

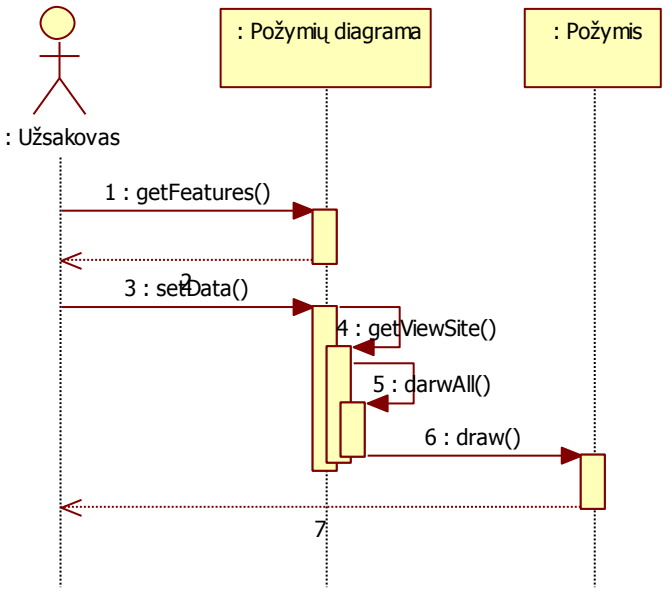
Iš būsenos „pasirinktas požymis“ pereinama į būseną „išrinktas komponentas po komponento išrinkimo“, atlikus tikrinimą ar įmanomas susiejimas pereinama į būseną „susiejimas patikrintas“ arba atgal į būseną „pasirinktas požymis“ jeigu susiejimas buvo neįmanomas. „Požymis susietas“ pasiekiamas po susiejimo veiksmo (31 paveikslas).

3.5.6.3 Kurti modeliuojamo produkto konfigūraciją



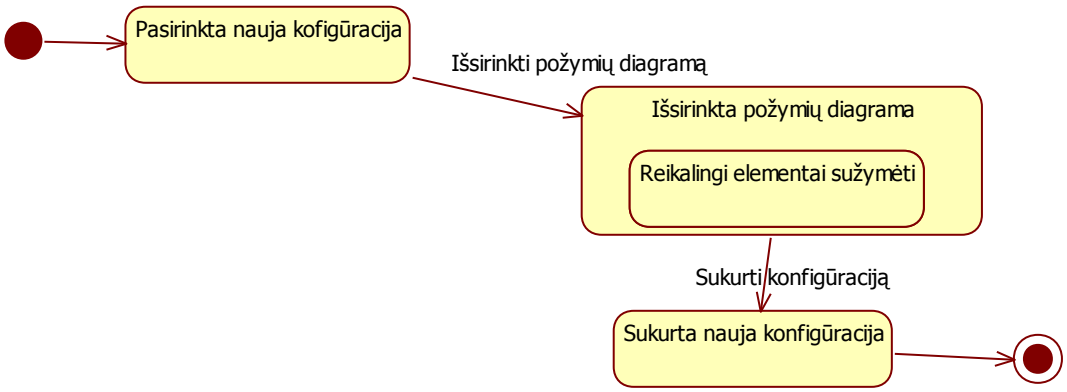
32 paveikslas. Kurti modeliuojamo produkto konfigūraciją. Veiklos diagrama.

Pasirenkama diagrama pagal kurią kuriama produkto konfigūracija, sužymimi reikiami elementai ir diagrama sukonfigūruojama (32 paveikslas).



33 paveikslas. Kurti modeliuojamo produkto konfigūraciją. Sekos diagrama.

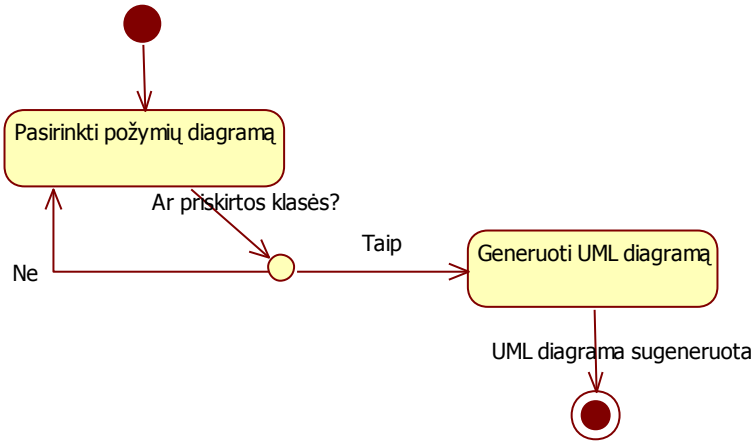
Pasirenkama požymių diagrama. Išsirenkami reikiami komponentai (setData nustatoma informacija) nubrėžiama diagrama (33 paveikslas).



34 paveikslas. Kurti modeliuojamo produkto konfigūraciją (diagrama)

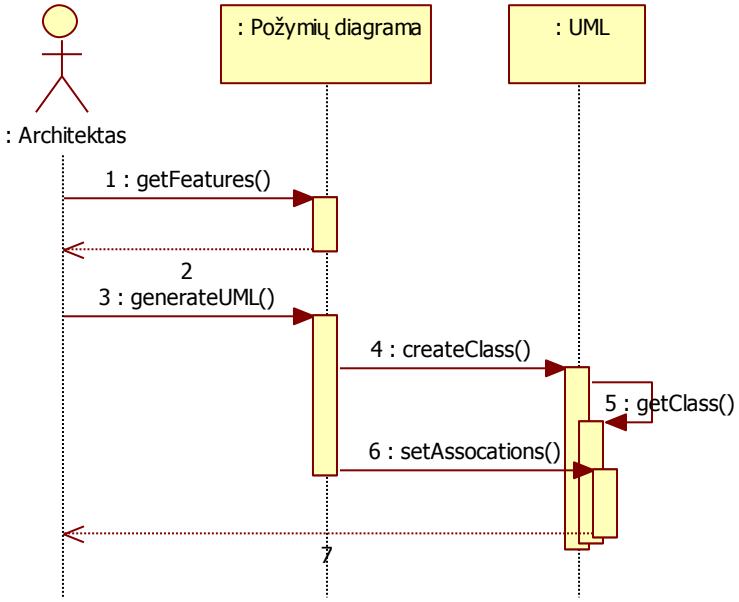
Visos būsenos pasiekiamos nuosekliai atlikus nurodytus veiksmus, grįžimo nėra. Būsena “Išsirinkta požymių diagrama” susideda iš vidinės būsenos “reikalingi elementai sužymėti” (34 paveikslas).

3.5.6.4 Sukurti specializuotą UML klasių diagramą pagal požymių diagramą



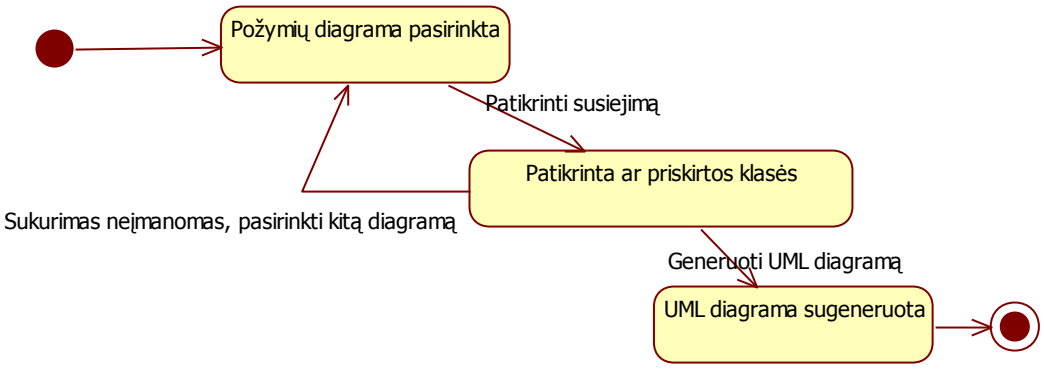
35 paveikslas. Sukurti specializuotą UML klasių diagramą pagal požymių diagramą. Veiklos diagrama.

Pasirenkama požymių diagrama, patikrinama ar yra priskirtos UML klasių diagramos elementas, jeigu ne pasirenkama kita diagrama arba pataisoma esama. Jei klasės priskirtos teisingai generuojama UML diagrama (35 paveikslas).



36 paveikslas. Sukurti specializuotą UML klasių diagramą pagal požymių diagramą. Sekos diagrama.

Pasirenkama požymių diagrama, pasirenkamas UML diagramos generavimas, sukuriamos klasės pagal požymių diagramos priskyrimą, nustatomi ryšiai (36 paveikslas).



37 paveikslas. Sukurti specializuotą UML klasių diagramą pagal požymių diagramą

Iš būsenos „pasirinkta požymių diagrama“ pereinama į būseną „patikrinta ar priskirtos klasės“ atlikus susiejimo tikrinimą. Jeigu neįmanomas susiejimas grįžtama į pradinę būseną, jei įmanomas į būseną „UML diagrama sugeneruota“ (37 paveikslas).

4. POŽYMIŲ MODELIAVIMO SUJUNGTO SU UML ĮRANKIO TYRIMAS

4.1. Įvadas

Šioje dalyje pateikiama informacija, kaip projektas atitinka užsakovo keliamus reikalavimus. Išrašomi svarbiausi programos trūkumai, teikiami trukumų šalinimo būdai. Siūloma tobulinti programą ir pateikiami siūlymo aprašymai.

4.2. Požymių modeliavimo įrankio kokybes analizė

4.2.1. Specifikacijos atitikimas

Požymių modeliavimo įrankis išpildo specifikacijoje išvardintus reikalavimus. Tačiau specifikaciją tik iš dalies atitinka reikalavimas požymių diagramoje keisti dialektą ir programos kodo generavimo galimybės. Dialekto keitimas programoje realizuotas skirtingais vaizdavimo būdais, o programinio kodo kūrimo galimybės jau buvo realizuotos Eclipse modeliavimo aplinkoje.

4.2.2. Požymių modeliavimo lango problemos

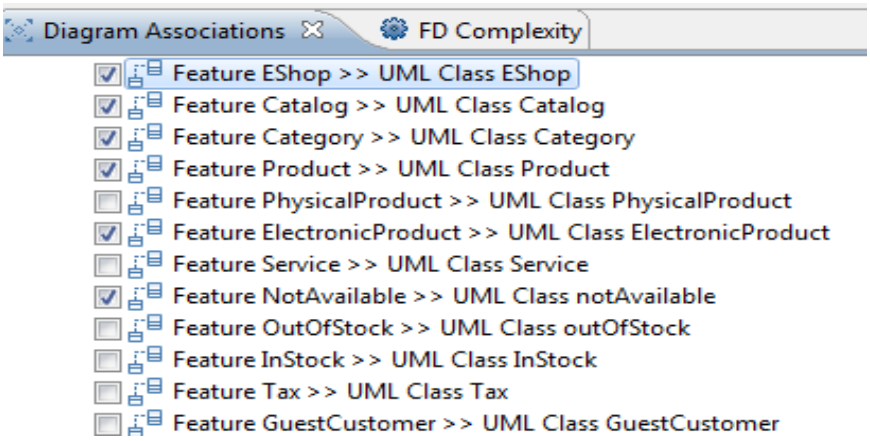
Kuriamoje požymių diagramoje sunku išskirti šakninį požymių sistemos elementą. Tai būtų galima lengviau atlikti taisyklingai išdėstant elementus diagramoje. Šiuo metu esančios elementų rikiavimo funkcijos netinka požymių elementų rikiavimui.

4.2.3. Požymių modelio sudėtingumų skaičiavimo lango problemos

Požymių modelio sudėtingumo skaičiavimo langas nėra labai sudėtingas, tačiau sudarant duomenų reikalavimus, buvo praleista funkcija, kuri skaičiuotų visų galimų modeliuojamos sistemos variantų skaičių. Siūloma papildyti skaičiavimų langą papildomu rezultatu, kuris rodytų požymių modelio visų galimų variantų skaičių.

4.2.4. Požymių konfigūravimo lango problemos

Buvo realizuotas požymių modelio konfigūravimo langas, kuris tampa labai nepatogus, kai konfigūruojama sistema yra didelė. Sistemos požymiai yra pateikiami dideliame sąraše, kuriame sunku aptikti ieškomus sistemos požymius, jeigu tiksliai nežinomas jų pavadinimas. 38-ame paveiksle pateiktas požymių konfigūracijos sąrašo vaizdas.



38 paveikslas. Požymių konfigūravimo langas

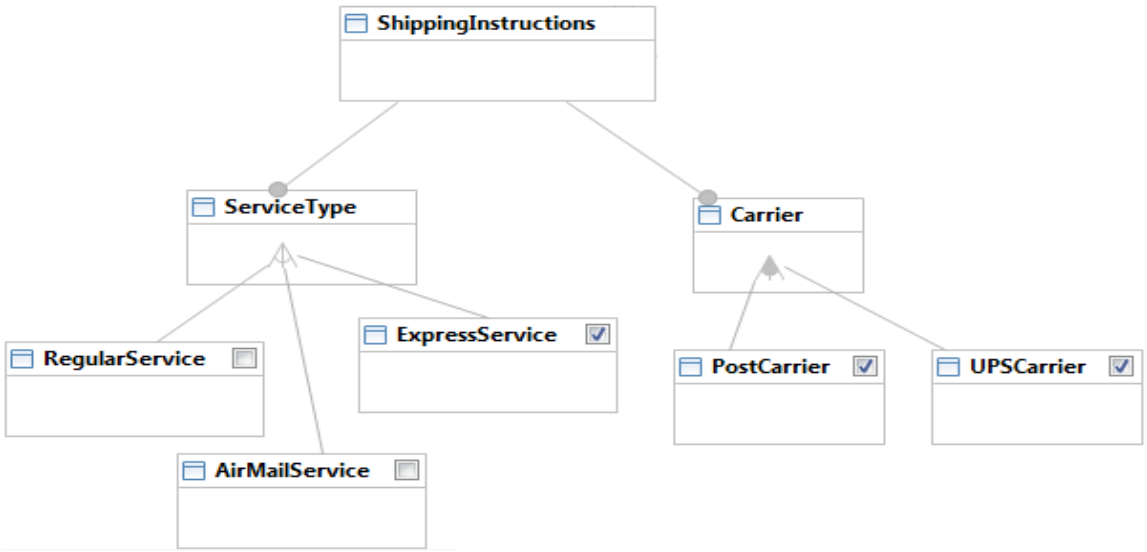
38-ame paveiksle matomo požymių konfigūravimo įrankio problemos:

1. Požymiai nėra rikiuojami pagal abėcėlę;
2. Sąrašo elementai neatsekami pagal diagramoje esančius elementus;

4.2.4.1 Siūlomas požymių konfigūravimo lango keitimas

Siūloma integruoti požymių konfigūravimą į patį požymių modeliavimo įrankį. Taip būtų išspręstos konfigūravimo įrankio problemos.

1. Nereikėtų ieškoti konkretaus požymio sąraše;
2. Vienoje vietoje būtų matoma požymių diagrama ir galimybė konfigūruoti matomą požymį;



39 paveikslas. Preliminarus požymių konfigūravimo įrankio vaizdas.

39-ame paveiksle pateiktas preliminarus vaizdas, kaip galėtų atrodyti požymių konfigūravimo langas. Toks konfigūruojamų požymių vaizdas būtų patogesnis vartotojui ir paspartintų konfigūravimo procesą.

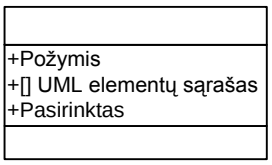
Taip pat nereikėtų rodyti vartotojui visų požymių konfigūracijos galimybių. Pavyzdžiui 39-ame paveiksle vaizduojamoje požymių diagramoje „ServiceType“ ir „Carrier“ požymiai ir taip visada bus pasirenkami, nes jie yra privalomi sistemoje (požymių diagramos ryšiai aprašyti 2.6.3.1 skyriuje), todėl jie būtų automatiškai parenkami programos. Taip vartotojas matytų mažiau informacijos ir jam būtų lengviau nuspręsti, kurie požymiai turi būti pasirinkti sistemoje.

4.3. Siūlymai tobulinti programą

4.3.1. Apjungimas su visais UML elementais

Sukurtas požymių modeliavimo įrankis yra sujungtas tik su UML klasių diagramų elementais. Ši funkcija realizuota tinkamai ir pateisina lūkesčius.

Ateityje siūloma sujungti 1 sistemos požymį su eile UML elementų. Tam reikėtų išplėsti požymių susiejimo elementą taip, kaip parodyta 40-ame paveiksle.



40 paveikslas. Požymio susiejimo su UML modelis.

Taip pat įgyvendinus šį patobulinimą, reiktų peržiūrėti konkrečių UML modelių kūrimo procesą, nes iš visos sistemos UML modelio reiktų šalinti su šalinamais elementais susijusius elementus.

4.4. Išvados

Realizuotos sistemos kokybės analizės metu buvo aptiktos kelios klaidos, jos aprašytos ir pasiūlytas jų sprendimo būdas. Pasiūlytas požymių konfigūravimo įrankio keitimas nauju, kuris paspartintų požymių konfigūravimo procesą.

Taip pat pasiūlyta ateityje išnagrinėti galimybę susieti sistemos požymius su visais UML elementais ir ištirti kitų (ne klasių diagramų) UML modelio keitimo galimybes.

5. EKSPERIMENTAI SU POŽYMIŲ MODELIAVIMO SUJUNGTO SU UML ĮRANKIU

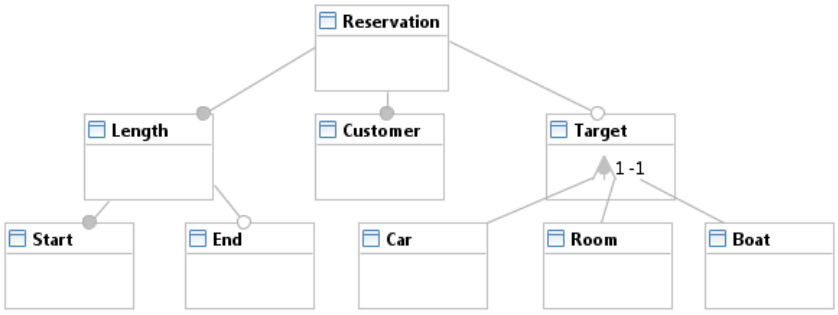
5.1. Įvadas

Šioje dalyje bus aprašomi atlikti eksperimentai su požymių modeliavimo įrankiu. Tam buvo sukurti 2 eksperimentiniai modeliai. Vienas modelis buvo skirtas funkcionalumo demonstravimui, kitas tirti sistemos galimybes esant didesniam duomenų kiekiui.

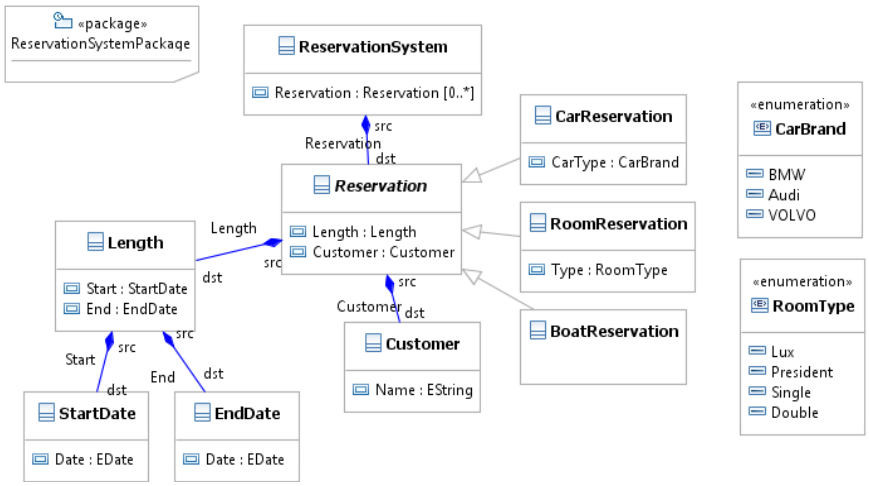
5.2. Rezervacijų sistema

Pirmo eksperimento metu buvo sukurtas mažas rezervacijų sistemos duomenų modelis (41 paveikslas, 42 paveikslas), kuriame galima nurodyti:

1. Rezervacijos trukmę – pradžią ir pabaigą,
2. Rezervacijos klientą,
3. Rezervacijos tipą – kambario, automobilio, laivo rezervacija.



41 paveikslas. Rezervacijų sistemos variantiškumo modelis.



42 paveikslas. Rezervacijų sistemos klasių diagrama.

5.2.1. Variantiškumo analizė

41-ame paveiksle vaizduojamoje požymių diagramoje yra saugoma informacija, kaip sistema gali būti keičiama. Kiekvieno elemento pasirinkimo galimybė aprašyta 10-oje lentelėje.

10 lentelė. Rezervacijų sistemos variantiškumo analizė.

Požymis	**Pasirinkimo galimybė**
Reservation	Būtinas
Length	Būtinas
Start	Būtinas
End	Nebūtinas
Customer	Būtinas
Target	Nebūtinas
Car	Pasirenkamas bent 1 iš grupės
Room	
Boat	

Matome, kad šios mažos rezervacijų sistemos variantų gali būti keli. Pagal požymių diagramos sudėtingumo skaičiavimo metodiką, aprašytą 2.7.1.1 skyriuje, apskaičiavome sudėtingumus ir gavome rezultatus pateiktus 43 paveiksle. Įvertinimo rezultatai rodo, kad variantiškumo modelis nėra sudėtingas, ir požymių medis negilus (11 lentelės rezultatai).

Item	Result	Details
Cognitive complexity:		
Variation points	3	Low < 7 +/-2
Graph levels	3	Low < 7 +/-2
Structural complexity	7	Number of sub-trees
Compound complexity	85,89	Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3R^2)/9
Explanation:		
Features	9	F
Mandaroty	3	Rand
Optional	2	Ror
ExclusiveCase	3	Rcase
CaseOr	0	Ror
Constraints	0	R

43 paveikslas. Rezervacijų sistemos sudėtingumo įvertis.

11 lentelė. Rezervaijų sistemos sudėtingumo įvertinimo aprašymas

Įvertinimas	**Reikšmė**	**Paaiškinimas**
Pasirinkimo taškai	3	Rodo, kad diagrama turi 3 pasirinkimo taškus.
Grafo gylis	3	Diagramoje giliausia šaka susideda iš 3 požymių.
Struktūrinis sudėtingumas	7	Vidinių medžių skaičius.
Sudėtinis sudėtingumas	86	Apskaičiuotas požymių diagramos sudėtingumo dydis. Ši reikšmė rodo, kad diagrama nėra sudėtinga [2].

Taip pat buvo palyginti sukonfigūruoti konkrečiai sistemai UML modeliai ir sukurto programos kodo kiekiai. Rezultatų apibendrinimas pateiktas 12-oje lentelėje, o sistemos darbo rezultatai pateikti 9.2 skyriuje.

12 lentelė. Rezervacijų sistemos skirtingų konfigūracijų palyginimas.

	1-as bandymas	2-as bandymas	3-as bandymas
Parinktų požymių kiekis	4	7	9
UML elementų kiekis	7	9	11
Programos kodo eilučių skaičius	3204	3698	4007
UML modelis	48 paveikslas	50 paveikslas	52 paveikslas
Požymių konfigūracija	47 paveikslas	49 paveikslas	51 paveikslas

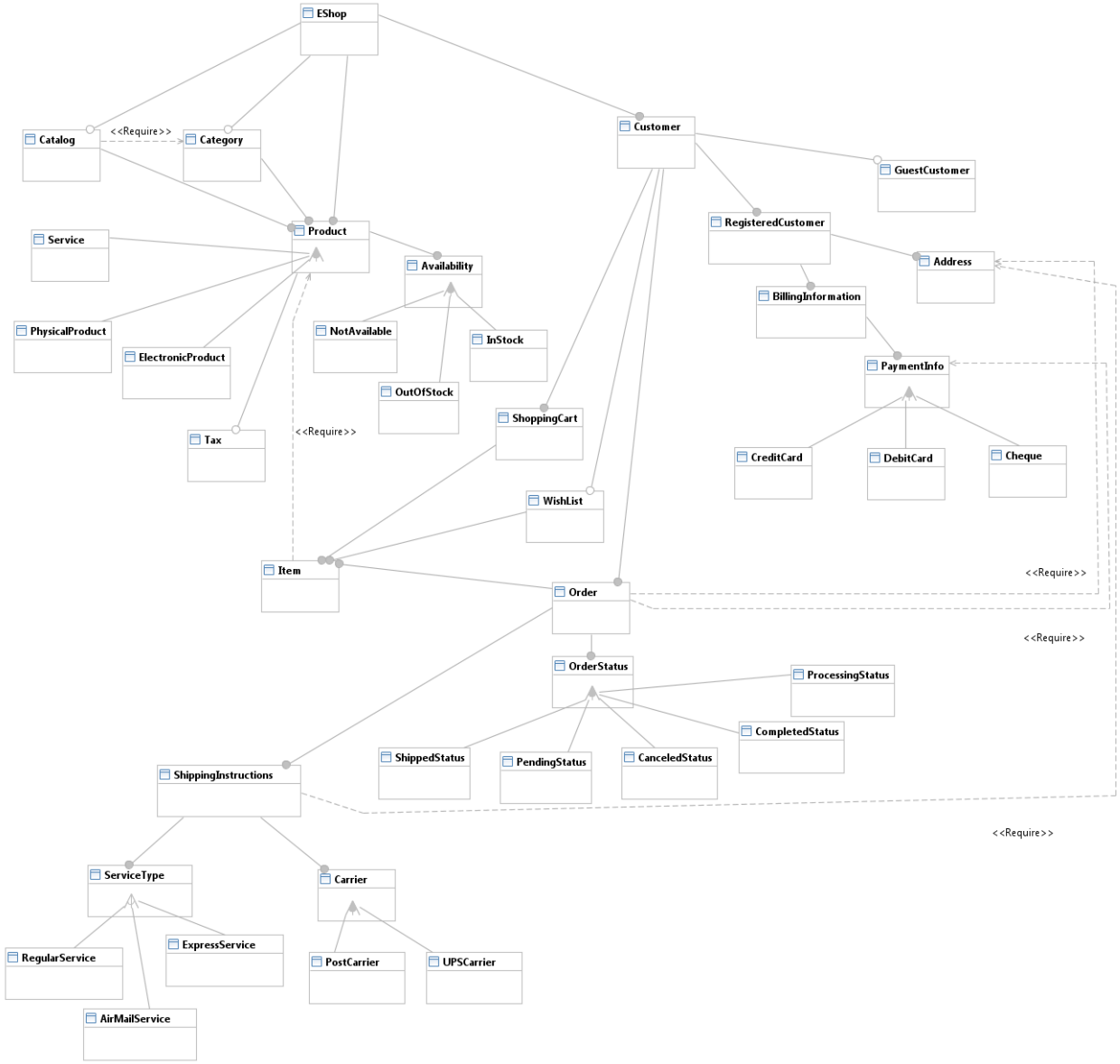
Šiam bandymui buvo konfigūruojamas rezervacijų sistemos variantiškumų modelis ir pagal sukurtą konfigūraciją modifikuojamas UML modelis. Modifikuotą UML modelį panaudojome programos kodo kūrimui. Apskaičiavome sukurto programinio kodo kiekį (programos kodo pavyzdys pateiktas 9.2.4 skyriuje).

5.3. Elektroninė parduotuvė

Antro eksperimento metu buvo sukurtas elektroninės parduotuvės duomenų (45 paveikslas) ir požymių (44 paveikslas) modeliai. Šio eksperimento modelis yra kur kas didesnis, matysime, kaip FD2 įrankis dirba su didesniais požymių modeliais.

Elektroninės parduotuvės požymių ir UML modeliai buvo sukurti vadovaujantis Sean Quan Lau teze „Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates“ [3], Krzysztof Czarnecki ir Micha l Antkiewicz straipsnyje pateiktais požymių bei klasių diagramų pavyzdžiais.

5.3.1. Elektroninės parduotuvės požymių modelis

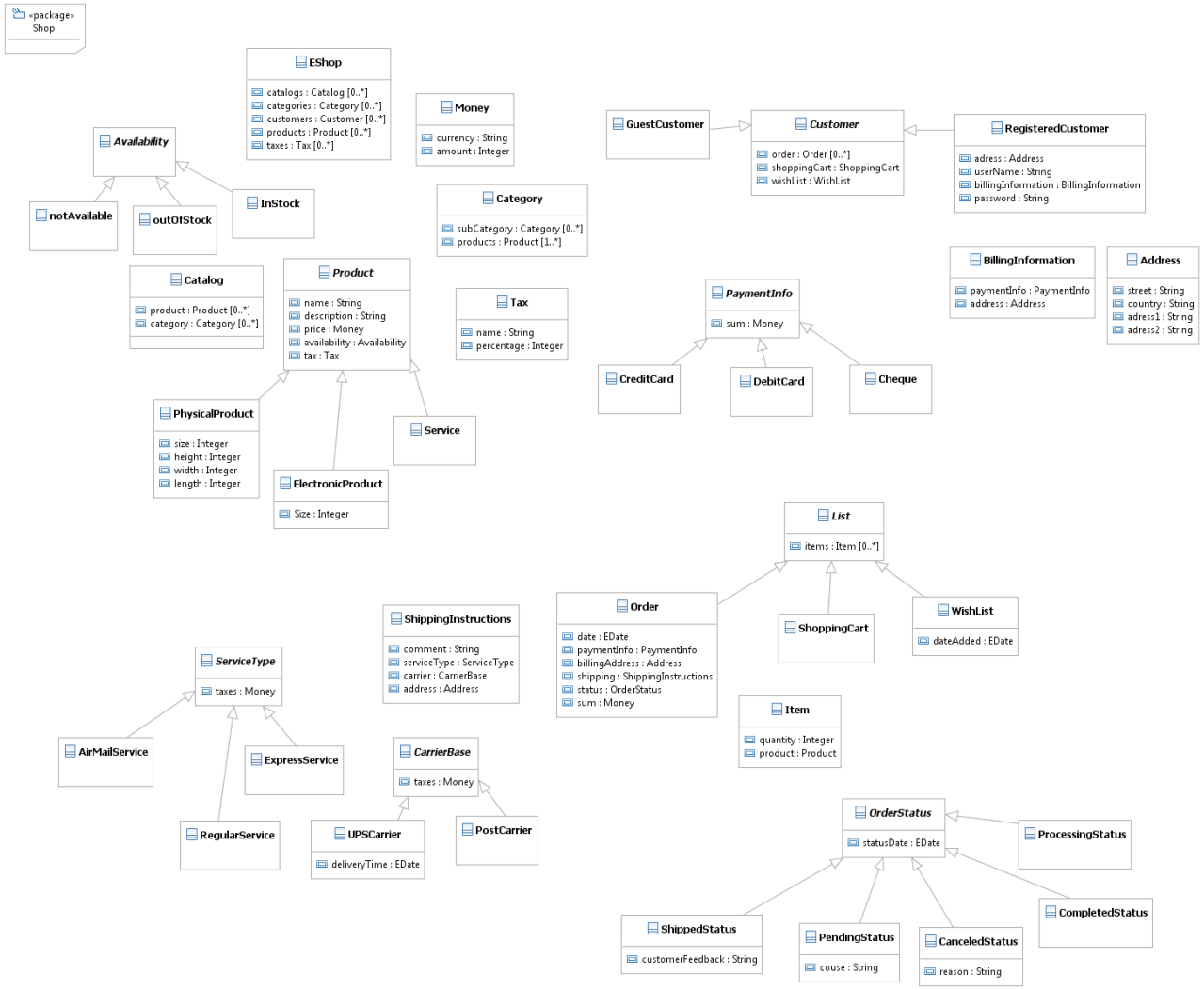


44 paveikslas. Elektroninės parduotuvės požymių modelis.

13 lentelė. Elektroninės parduotuvės sistemos požymių aprašymas.

Požymis	**Pasirinkimo galimybė**	**Aprašymas**
EShop	Būtinas	Elektroninė parduotuvė, šakninis elementas
Catalog		Prekių katalogas
Category	Nebūtinas, Būtinas kai pasirinktas ‚Catalog‘	Prekių kategorijos
Product	Būtinas	Abstraktus produktas
Psychical product	Pasirenkamas bent 1 iš ‚Product‘	Fizinis produktas
Electronic product		Elektroninio formato produktas
Tax	Nebūtinas	Mokesčiai
Availability	Būtinas	Abstraktus prekės prieinamumas
NotAvailable	Pasirenkamas bent 1 iš ‚Availability‘	Prekės nėra
OutOfStock		Išparduota
InStock		Prekyboje
ShoppingCart	Būtinas	Pirkinių krepšelis
WishList	Nebūtinas	Pageidavimų sąrašas
Order	Būtinas	Užsakymas
Item	Būtinas	Užsakymo eilutė
GuestCustomer	Nebūtinas	Klientas – svečias
RegisteredCustomer	Būtinas	Registruotas klientas
Address	Būtinas	Kliento adresas
BillingInformation	Būtinas	Kliento sąskaitos informacija
PaymentInfo	Būtinas	Mokėjimo informacina
CreditCard	Pasirenkamas bent 1 iš ‚PaymentInfo‘	Kreditinė mokėjimo kortelė
DebitCard		Debetinė mokėjimo kortelė
Cheque		Čekis
OrderStatus	Būtinas	Abstrakti užsakymo būsena
ShippedStatus	Pasirenkamas bent 1 iš ‚OrderStatus‘	Išsiųstas
PendingStatus		Numatytas apdoroti
CanceledStatus		Atšauktas
CompletedStatus		Užbaigtas
ProcessingStatus		Apdorojamas
ShippingInstructions		Prekės pristatymo duomenys
ServiceType		Prekės siuntimo būdas
RegularService	Pasirenkamas bent 1 iš ‚ServiceType‘	Įprastas
AirMailService		Oro paštu
ExpressService		Greitasis paštas
Carrier		Vykdytojas
PostCarrier	Pasirenkamas bent 1 iš ‚Carrier‘	Paštas
UPSCarrier		UPS pašto siuntų tarnyba

5.3.2. Elektroninės parduotuvės UML klasių diagrama



45 paveikslas. Elektroninės parduotuvės UML klasių diagrama

Elektroninės parduotuvės klasių diagrama (45 paveikslas) vaizduoja elektroninės parduotuvės naudojamą duomenų modelį. Klasių paaiškinimai pateikti 14-oje lentelėje.

14 lentelė. Elektroninės parduotuvės UML klasių diagramos aprašymas.

Klasės pavadinimas	**Paaiškinimas**
EShop	Elektroninės parduotuvės duomenų klasė.
Availability	Bazinė klasė nurodanti prekės pasiekiamumą.
NotAvailable	Napasiekiama.
InStock	Prekyboje.
OutOfStock	Išparduota.
Catalog	Prekių katalogas.
Money	Pinigų klasė nurodanti pinigų kiekį ir valiutą.
Category	Prekių kategorija.
Tax	Mokesčių klasė.
Product	Abstrakti produkto klasė.
PsychicalProdukt	Apčiuopiamas produktas siunčiamas paštu.
ElectronicProduct	Elektroninis produktas, kurį galima parsisiųsti internetu.
Service	Paslauga, kurią galima užsisakyti.
Customer	Klientas.
GuestCustomer	Svečias.
RegisteredCustomer	Registruotas parduotuvės klientas.
BillingInformation	Kliento atsiskaitymų informacija.
Address	Adresas
PaymentInfo	Mokėjimo informacija.
CreditCard	Kreditinė kortelė.
DebitCard	Debetinė kortelė
Cheque	Čekis
ServiceType	Pašto siuntimo paslaugos tipas.
AirMailService	Siuntimas oro paštu.
RegularService	Siuntimas įprastu paštu.
ExpressService	Skubi siunta.
ShippingInstructions	Užsakymo siuntimo informacija (Adresas ir siuntimo paslaugos tipas)
CarrierBase	Pašto tiekėjas.
UPSCarrier	UPS paštas.
PostCarrier	Įprastas paštas.
List	Krepšelis, kuriame laikomos prekės.
Order	Užsakymas.
ShoppingCart	Pirkinių krepšelis.
WishList	Pageidavimų sąrašas.
Item	Nurodo produkto kiekį krepšelyje.
OrderStatus	Užsakymo būsena.
Shipped	Išsiųstas užsakymas klientui.
Pending	Numatomas siuntimui.
Canceled	Atšauktas.
Processing	Apdorojamas.
Finished	Užbaigtas.

5.3.3. Variantiškumo analizė

Pagal požymių diagramos sudėtingumo skaičiavimo metodiką, aprašytą 2.7.1.1 skyriuje, apskaičiavome sudėtingumus ir gavome rezultatus pateiktus 46 paveiksle. Skaičiavimai rodo, kad modelis yra gerokai sudėtingesnis už pirmame eksperimente aprašytą modelį. Panaudota įvairesnių ryšių tarp sistemos požymių. 15 lentelė aprašo sudėtingumų reikšmes.

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
Mandaroty	18	Rand
Optional	5	Ror
ExclusiveCase	3	Rcase
CaseOr	16	Ror
Constraints	5	R

46 paveikslas. Elektroninės parduotuvės sistemos sudėtingumo įvertis.

15 lentelė. Elektroninės parduotuvės sudėtingumo įvertinimas.

Įvertinimas	**Reikšmė**	**Paaiškinimas**
Pasirinkimo taškai	9	Rodo, kad diagrama turi 9 pasirinkimo taškus.
Grafo gylis	6	Diagramoje giliausia šaka susideda iš 6 požymių.
Struktūrinis sudėtingumas	37	Vidinių medžių skaičius.
Sudėtinis sudėtingumas	1666	Apskaičiuotas požymių diagramos sudėtingumo dydis. Šis dydis rodo, kad diagrama yra vidutiniškai sudėtinga [2].

Taip pat buvo palyginti sukonfigūruoti konkrečiai sistemai UML modeliai ir sukurto programos kodo kiekiai. Apibendrinti rezultatai pateikti 12-oje lentelėje, o diagramų vaizdai ir sukurto programinio kodo pavyzdžiai pateikti 9.3 skyriuje.

16 lentelė. Elektroninės parduotuvės sistemos skirtingų konfigūracijų palyginimas.

	1-as bandymas	**2-as bandymas**	**3-as bandymas**
Parinktų požymių kiekis	23	30	37
UML elementų kiekis	25	31	39
Programos kodo eilučių skaičius	16000	18000	19000
UML modelis	54 paveikslas	56 paveikslas	58 paveikslas
Požymių konfigūracija	53 paveikslas	55 paveikslas	57 paveikslas

Šiam bandymui buvo konfigūruojamas rezervacijų sistemos variantiškumų modelis ir pagal sukurtą konfigūraciją modifikuojamas UML modelis. Modifikuotą UML modelį panaudojome programos kodo kūrimui. Apskaičiavome sukurto programinio kodo kiekį ir palyginome skirtingus bandymus.

5.4. Išvados

Buvo sukurti 2 eksperimentiniai požymių modeliai ir palygintos skirtingų jų konfigūracijų rezultatai. Rezultatai rodo, kad:

1. Požymių modelių sudėtingumas priklauso nuo požymių skaičiaus,
2. Požymių modelių sudėtingumas priklauso nuo ryšių tipų,
3. Sukurtos konkrečiai sistemai UML diagramos dydis priklauso nuo pasirinktų požymių skaičiaus,
4. Sukurtos konkrečiai sistemai UML diagramos dydis priklauso nuo su požymiu susietų UML elementų skaičiaus.

FD2 įrankis gali modifikuoti bendrinį UML modelį ir gauti konkrečios konfigūracijos UML modelį. Kuriant produktų liniją, tai būtų didelis privalumas, nes nereikėtų kurti atskirų UML modelių konkretiems produktams, o užtektų turėti vieną pagrindinį (bendrinį) ir iš jo sukurti konkrečių produktų UML modelius reikalui esant.

Skirtingus sistemos variantus (produktų linijos egzempliorius) galima patogiai suskirstyti į versijas. Bendrinį sistemos modeli galima sekti, kaip pagrindinę liniją, o skirtingus sistemos variantus, galima saugoti kaip požymių modelių konfigūracijas ir panaudojant jas konkrečių sukonfigūruotą UML modelių kūrimui, tai palengvintų skirtingų variantų atsekamumą ir palaikomumą.

6. TYRIMO IŠVADOS

Požymių modeliavimas iškelia programos variantiškumo valdymą į aukštesnį abstrakcijos lygį. Sistemos projektuotojas gali rūpintis sistemos variantiškumu nežinodamas kaip bus įgyvendinta pati sistema. O sukurtas požymių modelis vėliau gali būti naudojamas esamos programų sistemos variantiškumui valdyti.

1. Sukurtos sistemos variantiškumą galima valdyti UML modelio lygyje, kuris modifikuojamas pagal požymių modelį ir konkrečią jo konfigūraciją. Taip supaprastinamas programų sistemų variantiškumo valdymas.
2. Labai paprasta kurti skirtingus sistemos variantus. Buvo pateikti požymių modelių pavyzdžiai, iš kurių buvo galima sukurti aibę skirtingų UML diagramų variantų (1-ame eksperimente galima sukonfigūruoti 7 skirtingas sistemas, o 2-ame - 37). UML diagramų pritaikymas yra automatizuotas – UML modeliai keičiami automatiškai.
3. Dėka požymių modeliavimo įrankio integracijos į objektinio projektavimo ir modeliavimo srautą galimas tolesnis kodo generavimas iš sukonfigūruotų UML diagramų naudojant jau esamus Eclipse aplinkos įrankius.
4. Sistemos variantiškumo modeliavimas gali stipriai paspartinti projektuotojų darbą programos diegimo ir palaikymo stadijose. Sukurtas požymių modeliavimo įrankis FD2 automatiškai sukuria programų sistemų UML modelius ir leidžia prižiūrėti vieną bendrinį UML modelį, o konkrečius sistemos įgyvendinimus sugeneruoti pasinaudojant programų generavimo įrankiais.

Magistrinio projekto metu buvo sėkmingai sukurtas požymių modeliavimo įrankis sujungiantis požymių modelį su UML modeliu. Atlikti eksperimentai ir įvertinti jų rezultatai.

1. Pasiūlytas požymių variantiškumo modelių sujungimas su UML klasių diagramos elementais veiksmingas ir naudojamas sukurtame požymių modeliavimo įrankyje.
2. Sukurtas įrankis požymių modeliavimui sujungiantis požymių modeliavimą ir UML klasių diagramas. Įrankis modifikuoja UML modelius priklausomai nuo požymių diagramos pasirinktos konfigūracijos.
3. Požymių modeliavimo įrankio architektūros parinkimo sprendimai teisingi. Sukurtas įrankis yra lengvai prižiūrimas, keičiamas ir tobulinamas. Projekto įgyvendinimas buvo kuriamas etapais, kuriuose sukurti papildomi komponentai išplečiantys anksčiau sukurtus. Iš viso FD2 įrankį sudaro 9 komponentai.

4. Eksperimento metu buvo sėkmingai sukurti sistemų duomenų modeliai ir sudaryti jų variantiškumo modeliai. Pagal sistemos požymių variantiškumo diagramą buvo sukurta sistemų konfigūracija ir ją atitinkantis UML modelis. Iš modifikuoto UML modelio toliau sėkmingai sukurtas programos kodas.
5. Siūloma požymių modeliavimo sistemą praplėsti ir leisti požymių elementus susieti su daugeliu UML elementų (ne vien tik klasių diagramų).

7. LITERATŪRA

[1]ROBERTAS DAMAŠEVIČIUS, VYTAUTAS ŠTUIKYS, JEVGENIJUS TOLDINAS. *Domain Ontology-Based Generative Component Design Using Feature Diagrams and Meta-Programming Techniques*. Springer-Verlag Heidelberg, Berlin, 2002. Prieiga per internetą: http://portal.acm.org/citation.cfm?id=1434585, http://www.springerlink.com/content/54867gn2710406k2/,

[2]VYTAUTAS ŠTUIKYS, ROBERTAS DAMAŠEVIČIUS. *Measuring complexity of domain models represented by feature diagrams*. ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 3.

[3]SEAN QUAN LAU. *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates.* Waterloo, Ontario, Canada, 2006. p.23-89.

[4]KRZYSZTOF CZARNECKI AND MICHA L ANTKIEWICZ. *Mapping Features to Models: A Template Approach Based on Superimposed Variants*. Canada, 2005. p.3-7

[5]HASSAN GOMAA. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures.* ISBN: 0-201-77595-6. USA, 2004.

[6]MATTHIAS CLAUß. *Modeling variability with UML*. Vokietija, 2001. p.2-4.

[7]MATTHIAS CLAUß. *Generic Modeling using UML extensions for variability.* Vokietija, 2001. p.2-7.

[8]BIRGIT KORHERR, BEATE LIST. *A UML 2 Profile for Variability Models and their Dependency to Business Processes.*, Austrija, 2007.

[9]KYO C. KANG, SHOLOM G. COHEN, JAMES A. HESS, WILLIAM E. NOVAK, A. SPENCER PETERSON. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*.Technical Report number CMU/SEI-90-TR-21, ESD-90-TR-222. USA Pittsburgh, Pennsylvania, 1990.

[10] VALENTINO VRANIĆ, JÁN ŠNIRC. *Integrating Feature Modeling into UML*. Slovakia, 2006.

[11] VALENTINO VRANIĆ. *Reconciling Feature Modeling: A Feature Modeling Metamodel*. In Mathias Weske and Peter Liggesmeyer, editors, Proc. of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays 2004), Erfurt, Germany, September 2004.

[12] MIGUEL A. LAGUNA, JOSÉ M. MARQUÉS. *Feature Diagrams: a Formalization and extensible Meta-model Proposals.* GIRO Technical Report 2009/02-v2.0 2009-07-05. Ispanija, 2009.

[13] OBJECT MANAGEMENT GROUP. *OMG Unified Modeling LanguageTM (OMG UML) Infrastructure*. Version 2.2. 2009. Prieiga internete: http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/

[14] THOMAS LEICH, SVEN APEL, LAURA MARNITZ, 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.

[15] O.ROHLIK, A.PASETTI. *XFeature Modeling Tool. Automatic Control Laboratory*, ETH Zurich, 2005. Prieiga internete: http://www.pnp-software.com/XFeature/.

[16] MATTHEW STEPHAN, MICHAŁ ANTKIEWICZ. *Ecore.fmp: A tool for editing and instantiating class models as feature models*. Technical Report Tech. Rep. 2008-08, ECE, University of Waterloo, May 2008.

[17] T. VON DER MAßEN, H. LICHTER. *RequiLine: A Requirements Engineering Tool for Software Product Lines.* in Softw Product-Family Eng (PFE-5): Springer, 2004, pp. 168--180.

[18] D. BENAVIDES, S. SEGURA, P. TRINIDAD, A. RUIZ-CORTÉS. *FAMA: Tooling a framework for the automated analysis of feature models*. VAMOS First International Workshop on Variability Modelling of Software-intensive Systems. Ispanija, 2007. pages 129–134.

[19] V.MYLLÄRNIEMI, M.RAATIKAINEN, T.MÄNNISTÖ. *KumbangTools.* Software Product Line Conference (SPLC). Kyoto, Japan, 2007.

[20] DANILO BEUCHE. *Modeling and Building Software Product Lines with Pure: : Variants.* Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, 2008. p.358.

[21] MIGUEL A. LAGUNA, JOSÉ MANUEL MARQUÉS CORRAL. *Feature Diagrams and their Transformations: an Extensible Meta-model*. IEEE proceedings of the EUROMICRO SEEA. ISBN:978-0-7695-3784-9. 2009. p.97-104

[22] HASSAN GOMAA, MICHAEL EONSUK SHIN. *Multiple-view modelling and meta-modelling of software product lines*. IET Software 2(2): 94-122 (2008).

[23] TIMO ASIKAINEN, TOMI MÄNNISTÖ, TIMO SOININEN. *A unified conceptual foundation for feature modelling.* In: Liam O'Brien (editor). Proceedings of the 10th International Software Product Line Conference (SPLC 2006). Baltimore, Maryland, USA, August 2006.

[24] H.UNPHON.*A Comparison of Variability Modeling and Configuration Tools for Product Line Architecture*. Technical Report, Prieiga internete http://www.itu.dk/people/unphon/technical_notes/CVC_v2008-06-30.pdf

[25] THE ECLIPSE FOUNDATION. *Graphical Modeling Project (GMP)*. Žiūrėta 2010. Prieiga internete: http://wiki.eclipse.org/index.php/GMF_Tutorial.

[26] OBJECT MANAGEMENT GROUP. *XML Metadata Interchange (XMI) Specification.* Version 1.0, June2000, Prieiga internete: http://www.omg.org/technology/documents/formal/xml_metadata_interchange.htm

8. TERMINŲ IR SANTRUMPŲ ŽODYNAS

Eclipse (platforma) – programa, priedų platforma, kurią galima papildyti individualiais priedais.

Eclipse priedas – į Eclipse programą integruotas įrankis, kuris atlieką tam tikrą funkciją.

Kodo generavimas – Programinio kodo sukūrimas, pagal pasirinktą klasių diagramą.

XML - (angl. eXtensible Markup Language) yra W3C kompanijos rekomenduojama bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba.

XMI – (angl. XML Metadata Interchange) standartas meta duomenų keitimuisi tarp programų, panaudojant XML kalbą.

OWL (angl. Web Ontology Language) – Internetinių duomenų failų aprašymo panaudojant ontologijas kalba.

EPL - (angl. The Eclipse Public License) – Eclipse programos kūrėjų sukurta viešam naudojimui skirta licenzija. Pagal šią licenziją kuriamos atviro kodo programos.

CPL - (angl. Common Public License) – IBM korporacijos sukurta vieša licenzija leidžianti laisvai naudoti turinį. Gali būti vartojama atviro ir uždaro kodo programinei įrangai.

Diagramos dialektas – Tam tikras savitas diagramų vaizdavimas.

GMF – Grafinio modeliavimo įrankiai naudojami Eclipse programoje.

EMF – Modeliavimo įrankiai naudojami Eclipse programoje.

GEF – grafinių objektų braižymo biblioteka naudojama Eclipse programoje.

UML – (angl. Unified Modeling Language) Vieninga modeliavimo kalba. Tai modeliavimo ir specifikacijų kūrimo kalba, skirta specifikuoti, atvaizduoti ir konstruoti objektiškai orientuotų programų dokumentus [13].

Požymis - tai savitas, charakteringas sistemos atributas, kuris nusako matomus sistemos atributus, tačiau nesigilina į detalų sistemos apibūdinimą [9].

Požymių modelis – tai sistemos požymių sąrašas su informacija nusakančia kaip požymiai susieti tarpusavyje.

Požymių diagrama – grafinis požymių modelio vaizdavimo būdas.

Produktų linija – tai panašių produktų aibė turinti bendrų požymių, dalinasi bendrais atributais.

Variantiškumas – fiz. skaičius nepriklausomų fiz. kintamųjų, kuriuos galima keisti tam tikrose ribose, nekeičiant sistemos fazių skaičiaus [Tarptautinių žodžių žodynas, © Vyriausioji enciklopedijų redakcija, 1985].

Meta modelis – modeliavimo kalbos modelis, kuriame apibrėžtos esminės kalbos savybės.

Bendrinis UML modelis – tai modelis, kuris apima visą produktų liniją. Iš šio UML modelio kuriami produktai produktų linijai.

Modelių nepriklausomumas – modelių nepriklausomybė vienas nuo kito.

9. PRIEDAI

9.1. Požymių modeliavimo įrankių palyginimas

9.1.1. Bendra įrankių informacija

Kadangi daugelis požymių modeliavimo įrankių yra tyrimų objektas, tai jų įgyvendinimui pasirinktos nemokamos priemonės. 17-oje lentelėje matome, kad daugelis įrankių įgyvendinti atviro kodo programavimo kalba JAVA ir integruoti į atviro kodo programavimo aplinką Eclipse kaip pralėtimai.

17 lentelė. Bendra informacija apie įrankius, palaikančius požymių modeliavimą.

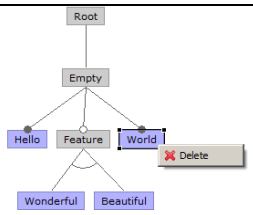
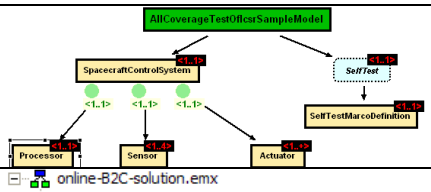
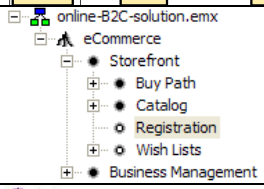
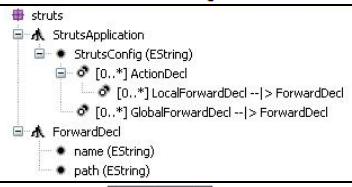
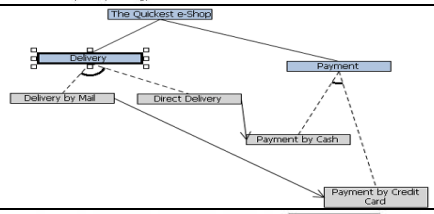
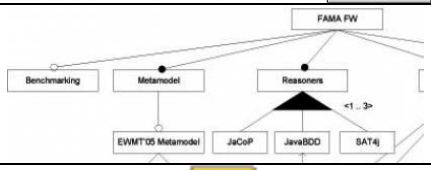
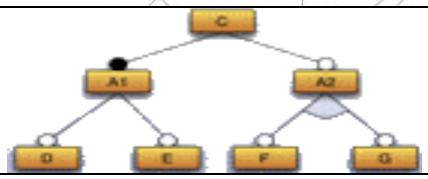
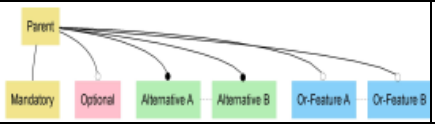
Pavadinimas	**Kūrėjas**	**Licenzija**	**Programos tipas**	**Programos veikimo platforma**
Feature IDE	Magdeburgo universitetas (Vokietija)	Atviras kodas	Eclipse priedas	JAVA
XFeature	P&P Software GmbH and ETH-Zürich	Atviras kodas	Eclipse priedas	JAVA
Feature modeling plugin	Generative software development lab, University of Waterloo, Canada	Atviras kodas	Eclipse priedas	JAVA
Ecore Feature modeling plugin	Generative software development lab, University of Waterloo, Canada	Atviras kodas	Eclipse priedas	JAVA
RequiLine	Research Group Software Construction, RWTH Aachen	Atviras kodas	Atskira programa	C#, .NET Windows
FAMA	University of Seville, Spain	Atviras kodas	Atskira programa arba Eclipse priedas	JAVA
Gears	Biglever software	Komercinė	Atskira programa arba Eclipse priedas	JAVA
KumbangTools	Helsinki University of Technology, Finland	General Public License	Eclipse priedas	JAVA
pure::variants	pure-systems GmbH	Komercinė	Eclipse priedas	C++ Windows

Dalis

9.1.2. Įrankiuose palaikomas požymių diagramų vaizdavimas

Daugelyje peržvelgtų požymių modeliavimo įrankių (žiūrėti 18-ą lentelę) naudojama grafinė požymių diagramų notacija. Kiekvienu atveju ji truputi skiriasi, tačiau pagrindiniai grafiniai žymėjimai išlieka.

18 lentelė. Požymių diagramų atvaizdavimas

Pavadinimas	**Pavyzdys**	**Ar palaiko skirtingus požymių diagramų vaizdavimo modelius?**
Feature IDE		Ne.
XFeature		Taip. Leidžia vartotojui nurodyti savo požymių meta modelius ir požymių vaizdavimo modelius.
Feature modeling plugin		Ne.
Ecore Feature modeling plugin		Ne.
RequiLine		Ne.
FAMA		Ne
KumbangTools		Ne
pure::variants		Ne

9.1.3. Įrankių galimybės

Daugelis programų sukurtas diagramas saugo kaip XML failus (19 lentelė). Šis pasirinkimas suprantamas, nes šiuo formatu lengva dirbti ir jį gali skaityti žmogus. Taip pat visi įrankiai turi diagramos klaidų aptikimo galimybę. Diagramos klaidų tikrinimas apsaugo vartotoją, nuo galimų tolimesnių klaidų, ši galimybė labai svarbi ir modeliavimo įrankis ją turi turėti. Dar viena patraukli vartotojui programos galimybė - programinio kodo kūrimas pagal parinktą požymių diagramą. Tinkamai įgyvendinta ši galimybė gali tapti pravarti programų šeimynų versijų leidimuose. [24]

19 lentelė. Įrankių įvedimo ir išvesties palyginimas.

Pavadinimas	**Ar yra galimybė išsaugoti bei skaityti XML / XMI failus**	**Požymių diagramos klaidų tikrinimas**	**Programinio kodo kūrimas**	**Įsilieja į programavimo programą**
Feature IDE	Yra	Yra	Yra	-
XFeature	Yra	Yra	Yra	Eclipse
Feature modeling plugin	Yra	Yra	Nėra	Eclipse
Ecore Feature modeling plugin	Yra	Yra	Yra	Eclipse
RequiLine	Yra	Yra	Yra	-
FAMA	Yra	Yra	Yra	Eclipse
Gears	Nėra	Yra	Yra	Eclipse
KumbangTools	Yra	Yra	Yra	Eclipse
pure::variants	Yra	Yra	Yra	Eclipse

9.1.3.1 Požymių modeliavimo įrankių apžvalgos rezultatai

Peržvelgę šiuos modeliavimo įrankius, matome, kad juose požymių diagramos nėra siejamos su UML diagramomis. Taigi naujas įrankis leisiantis susieti UML diagramas su požymių diagramomis, gali būti labai konkurencingas. Kad sustiprintumėm programos konkurencingumą, programa turi nenusileisti konkurentų pateiktiems produktams, taigi joje bus įdiegtą daugelis konkurentų produktuose esančių galimybių.

Kaip teko pastebėti 17–oje lentelėje, daugelis įrankių pagaminti kaip Eclipse programos praplėtimai, taigi susidaro įspūdis, kad Eclipse platforma yra labai palanki tokio tipo programinės įrangos kūrimui.

9.1.4. UML modeliavimo įrankių palyginimas

9.1.4.1 Tikslai

Kadangi UML modeliavimo įrankių rinkoje yra daug, tai nusprendėme sukurti vieno iš šių įrankių praplėtimą. Mums reikia pasirinkti įrankį, kuris:

- palaikytų naujausias UML diagramas;

- leistų lengvai praplėsti naujomis diagramomis ir funkcionalumu;
- turėtų jau įgyvendintus kai kuriuos mūsų užsibrėžtus tikslus.

9.1.4.2 Bendra UML modeliavimo įrankių informacija.

20-oje lentelėje pateikta bendra informacija apie nagrinėtus UML modeliavimo įrankius.

20 lentelė. Bendra informacija apie UML modeliavimo įrankius.

Pavadinimas	**Kūrėjas**	**Versija**	**Ar nemokama?**	**Licenzija**
ArgoUML	ArgoUML	0.26.1	nemokama	atviras kodas
Star UML	StarUML Development Group	5	nemokama	atviras kodas
Eclipse Model Development Tools (MDT)	Galileo Project	1	nemokama	atviras kodas
Magicdraw Comunity	No Magic, Inc.	15	bendruomenės	uždaras kodas
Magicdraw Pro	No Magic, Inc.	15	demonstracine	uždaras kodas
Visual Paradigm for UML free	Visual Paradigm International Ltd.	6	bendruomenės	uždaras kodas
Visual Paradigm for UML Pro	Visual Paradigm International Ltd. Change Vision, Inc.	6	demonstracine	uždaras kodas
Jude Pro	Change Vision, Inc.	5.4	demonstracine	uždaras kodas
Jude Community	Change Vision, Inc.	5.4	bendruomenės	uždaras kodas
DIA	Alexander Larsson, alla@lysator.liu.se	0.9	nemokama	atviras kodas
IBM Rational Rose	IBM	2008m. 10mėn	demonstracine	uždaras kodas
IBM Rational Software Architect	IBM	2008m. 10mėn	demonstracine	uždaras kodas
eUML2 free	Soyatec	2.1	nemokama	uždaras kodas

9.1.4.3 UML įrankių palyginimas

21-oje lentelėje pateikiama kokią UML versiją palaiko modeliavimo programa. Čia išsiskiria ArgoUML ir JudeCommunity UML modeliavimo įrankiai, nes jie palaiko sena UML 1.4. Įdomu pastebėti, kad MagicDrawPro palaiko SysML specifikavimo kalbą *[SysML.org, 2003-2008]*. Visi įrankiai gali modeliuoti standartines diagramas: UseCase, Class, State, Activity, Sequence, Collaboration, Component, Deployment, Composite Structure, Interaction Overview diagramas, kurios įeina į UML standartą.

21 lentelė. Modeliavimo įrankių palaikomos diagramos.

Pavadinimas	**UML standartas**	**SysML palaikymas**	**Ar galima išplėsti programą naujomis diagramomis?**	**UML profiliai**
ArgoUML	UML 1.4	-	+	+
Star UML	UML2.0 is dalies, o UML1.4 pilnai.	-	+	+
Eclipse Model Development Tools (MDT)	UML 2.x	-	Galima pergeneruoti UML2Tools pritaikius savo GMF sukurta versija.	
Magicdraw Comunity	UML 2	-	Open API	+
Magicdraw Pro	UML 2	+	Open API	+
Visual Paradigm for UML free	UML 2.1	-	Open API	-
Visual Paradigm for UML Pro	UML 2.1	-	Open API	+
Jude Pro	UML 2.0	+	Jude API	
Jude Community	UML 1.4	-	Jude API	
DIA	-	-	-	-
IBM Rational Rose	UML 1.4	-	?	+
IBM Rational Software Architect	UML 2.1	-	?	+
eUML2 free	UML 2.1	-	?	

MagicDraw, Jude bei Visual Paradigm for UML įrankiai palaiko daug diagramų ir yra lengvai praplečiami, tačiau nemokama šių įrankių versija yra stipriai apribota ir neturi daugelio galimybių. Dėl to jų nemokamų versijų naudojimas yra nepatogus.

Daugelis uždaro kodo UML modeliavimo įrankių, turi galimybę būti praplečiami naujomis diagramomis, tai dažniausiai įgyvendina „Open API“ pagalba („Open api“ aprašymas pateikiamas gamintojų svetainėse). Atviro kodo įrankiuose praplėtimai gali būti įgyvendinami ir kitaip(pavyzdžiui tiesiogiai keisti jų programinį kodą). Eclipse aplinkoje priedai kuriami truputi kitaip, ir praplėtimo galimybių yra daugiau, Eclipse platforma sukurta priedų rašymui.

Taip pat svarbu, kaip vartotojas jausis įsijungęs programą. Dabartiniai modeliavimo įrankiai labai įvairūs, vieni padaryti, kad kuo greičiau veiktų, kiti labai lėti, bet turi papildomų galimybių, dar kiti yra kitų programų priedai.

22 lentelė. Programos charakteristikos.

Pavadinimas	**Programos tipas**	**Programos veikimo platforma**	**Programos dydis**	**Integruojama i kitas programas**
ArgoUML	Programa	JAVA	Maža, bet dirbant su	Bandoma pritaikyti Eclipse.
Star UML	Programa	Windows	maža	-
Eclipse Model Development Tools (MDT)	Eclipse priedas	JAVA, Eclipse	didžiule	Eclipse
Magicdraw Comunity	Programa	JAVA		-
Magicdraw Pro	Programa	JAVA	didele	Eclipse, Sun Java Studio 8, IntelliJ IDEA, NetBeans, Borland's Jbuilder, IBM RAD
Visual Paradigm for UML free	Programa	JAVA	didele	-
Visual Paradigm for UML Pro	Programa	JAVA	didele	Visual Studio, Eclipse, NetBeans, IntelliJ IDEA, Borland Jbuilder, Oracle Jdeveloper,
Jude Pro	Programa	JAVA		-
Jude Community	Programa	JAVA	nedidele	-
DIA	Programa	Python	maža	-
IBM Rational Rose	Eclipse priedas	JAVA	didžiule	Visual Studio, Eclipse
IBM Rational Software Architect	Eclipse priedas	JAVA, Eclipse	didžiule	Eclipse
eUML2 free	Eclipse priedas	JAVA, Eclipse	didžiule	Eclipse UML2

22 lentelė nurodo, kaip pagamintas modeliavimo įrankis. Pastebima, kad daugelis įrankių pagaminti JAVA programavimo kalba ir dalinami į 2 grupes, Eclipse priedas ir atskira programa. Mūsų atveju būtų patogu sukurti Eclipse priedą, kuris galėtų jungtis su Eclipse UML modeliavimo įrankiu. Tačiau Eclipse yra labai didelė ir gremėzdiška programa, jai reikalingas galingas kompiuteris.

Trumpai apie palaikomus failų formatus (23 lentelė).

23 lentelė. Palaikomi failų formatai

Pavadinimas	**Sintaksės tikrinimas**	**XML(XMI) formatas**	**Palaiko kitus failų formatus**	**Galimybė išsaugoti diagramą paveikslėlio formatu**
ArgoUML	+	1.2	-	GIF, PNG, PostScript, Encapsulated PS, PGML and SVG.
Star UML	+	+	-	JPEG, BMP, Enhanced Meta File (*.emf), Metafile (*.wmf). Galima tiesiogiai kopijuoti į MS Word dokumentą.
Eclipse Model Development Tools (MDT)	+	2.x	-	
Magicdraw Comunity	+	2.1; 1.0;. 1.1; 1.2;	-	+
Magicdraw Pro	+	2.1; 1.0;. 1.1; 1.2;	EMF UML2	+
Visual Paradigm for UML free	+	1.0, 1.2, 2.0	-	+
Visual Paradigm for UML Pro	+	1.0, 1.2, 2.1	Import and export EMF based UML2 model	+
Jude Pro	+	+	EMF	+
Jude Community	+	-	-	+
DIA	-	Savitas xml	-	+
IBM Rational Rose	+	?	?	+
IBM Rational Software Architect	+	?	?	+
eUML2 free	+	+	-	+

Visi modeliavimo įrankiai gali patikrinti ar diagramose nėra klaidų, daugelis įrankių netgi neleidžia padaryti šių klaidų. Diagramos turi būti teisingos, kad vėliau jas būtų galima sėkmingai naudoti.

Išsaugoti failai turi būti lengvai pernešami ir vienodai suprantami skirtingų programų, taigi UML modeliai turi būti išsaugomi XMI formatu [26]. Dokumentus šiuo formatu gali išsaugoti daugelis įrankių. Saugoti failus kitais formatais tikimės, kad neprireiks, tačiau laikome kaip privalumą, jeigu programa leidžia vaizduojamą diagramą išsaugoti paveikslėlio formatu.

Labai paranki UML modeliavimo įrankio savybė yra programinio kodo generavimas. Kadangi siekiame susieti UML diagramas su požymių diagramomis, tai iš požymių ir UML modelių galėtumėm lengvai sukurti programinį kodą.

24 lentelė. Programinio kodo generavimo galimybė.

Pavadinimas	**Kuria programavimo kalbų kodą**	**Skaito programavimo kalbų kodą**
ArgoUML	Java, C++, C#, PHP4 and PHP5	java
Star UML	Java, C++, C#,	Java, C++, C#
Eclipse Model Development Tools (MDT)	java	java
Magicdraw Comunity	-	-
Magicdraw Pro	java, C++, .NET	java, C++, .NET
Visual Paradigm for UML free	-	-
Visual Paradigm for UML Pro	java, C++, .NET	java, C++, .NET
Jude Pro	java, C#	java, C#
Jude Community	java	java
DIA	-	-
IBM Rational Rose	java, C++, VB, Delfi	+
IBM Rational Software Architect	java, XSD, C++, CORBA	java, XSD, C++
eUML2 free	java	java

24 lentelė rodo, kad daugelis modeliavimo įrankių turi kodo generavimo galimybę.

9.1.5. UML modeliavimo įrankių apžvalgos rezultatai

Analizės metu, tolimesniam nagrinėjimui, buvo atrinkti keli įrankiai, kurie tenkino požymių modeliavimo įrankio kūrimui būtinas savybes. Pateiktas įrankių sąrašas prioritetų tvarka:

1. Eclipse Model Development Tools (UML2TOOLS)
2. Star UML
3. MagicDraw
4. Argo UML

Šis sąrašas buvo suformuotas atsižvelgiant į nagrinėtų požymių modeliavimo įrankių pasirinkimus (Eclipse yra populiariausia tarp nagrinėtų požymių modeliavimo įrankių), reikalavimus keliamus būsimam įrankiui (Star UML ir Argo UML įrankiai yra nemokami ir atviro kodo), kokybinius reikalavimus (MagicDraw yra vienas iš UML modeliavimo įrankių lyderių).

9.1.6. Išvados

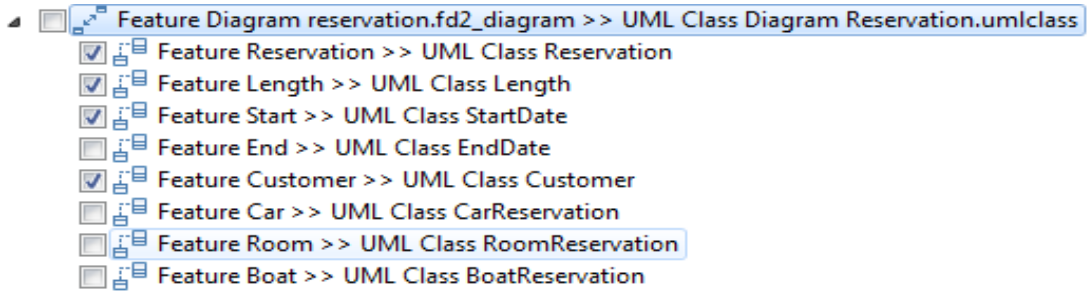
Peržvelgę požymių modeliavimo įrankius, pastebime, kad iki šiol požymių diagramos nebuvo glaudžiai siejamos su UML diagramomis, o tai galintys įrankiai vis dar kuriami *[Ecore Feature modeling plugin]*. Taigi požymių modeliavimo įrankis galintis susieti UML modelius su požymių modeliu, gali būti paklausus.

Taip pat tarp požymių modeliavimo įrankių išskiriama Eclipse platforma. Daugelis požymių modeliavimo įrankiu sukurti, kaip šios programos praplėtimai. Peržvelgus UML modeliavimo įrankius, Eclipse sukurtas UML2Tools priedas yra patrauklus savo išplečiamumo savybėmis. Taigi galima susidaryti įspūdį, kad ši platforma labiausiai tinkama požymių modeliavimo įrankio kūrimui.

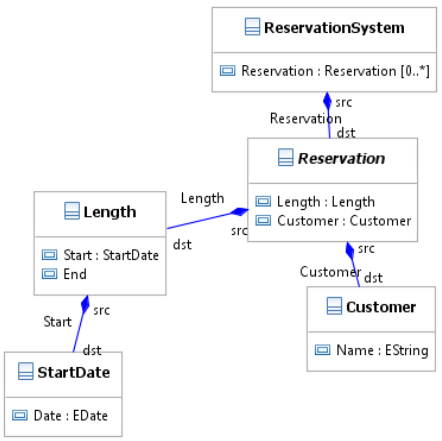
9.2. Rezervacijų sistemos eksperimentų rezultatai

Reikia paminėti, kad „Target“ požymis nėra įtrauktas į požymių konfigūraciją, todėl konfigūruojamų požymių sąraše matomas.

9.2.1. Bandymas pasirenkant 4 požymius

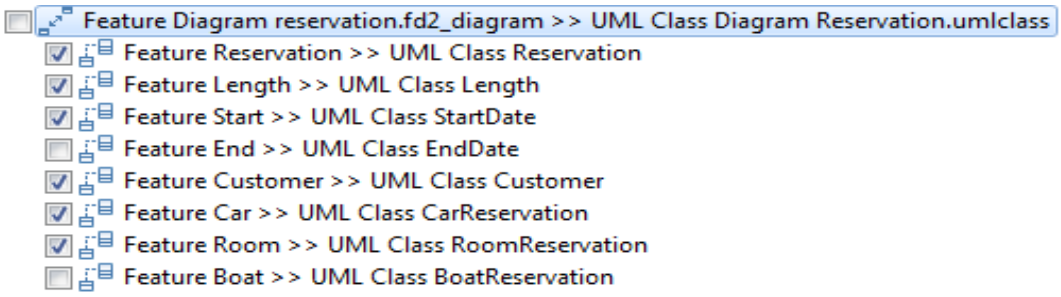


47 paveikslas. Bandymas pasirenkant 4 požymius. Požymių konfigūracija.

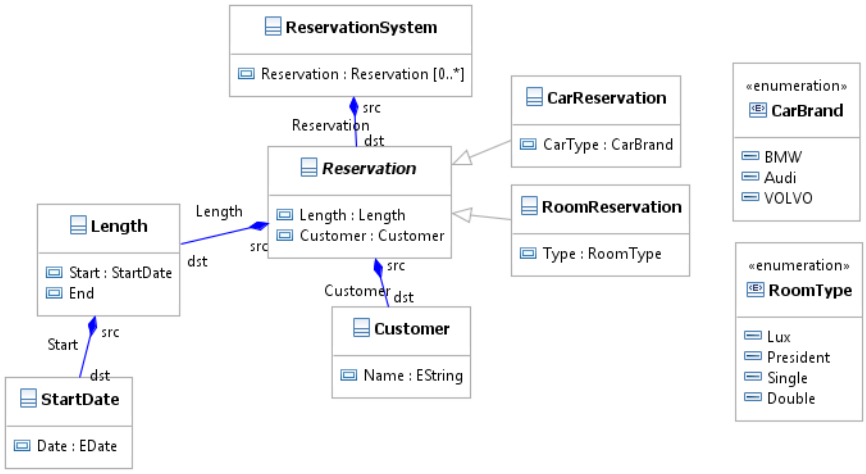


48 paveikslas. Bandymas pasirenkant 4 požymius. Klasių diagrama.

9.2.2. Bandymas pasirenkant 7 požymius

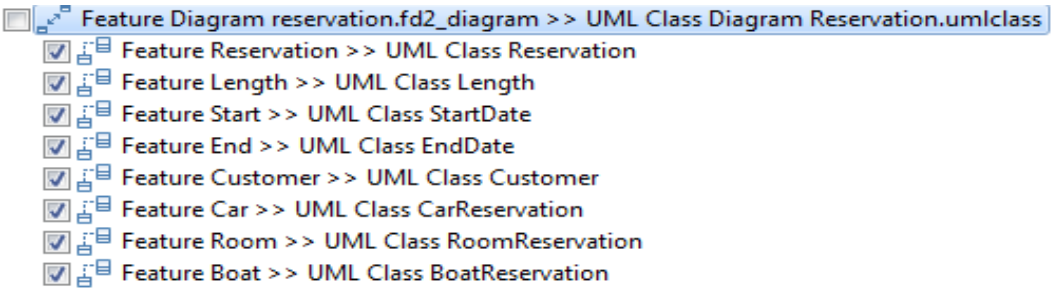


49 paveikslas. Bandymas pasirenkant 7 požymius. Požymių konfigūracija.

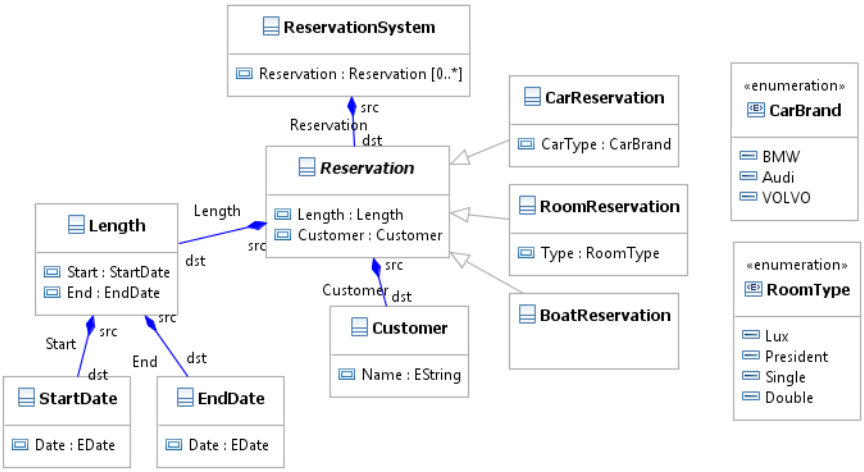


50 paveikslas. Bandymas pasirenkant 4 požymius. Klasių diagrama.

9.2.3. Bandymas pasirenkant 9 požymius



51 paveikslas. Bandymas pasirenkant 9 požymius. Požymių konfigūracija.



52 paveikslas. Bandymas pasirenkant 9 požymius. Klasių diagrama.

9.2.4. Sukurto programos kodo pavyzdys

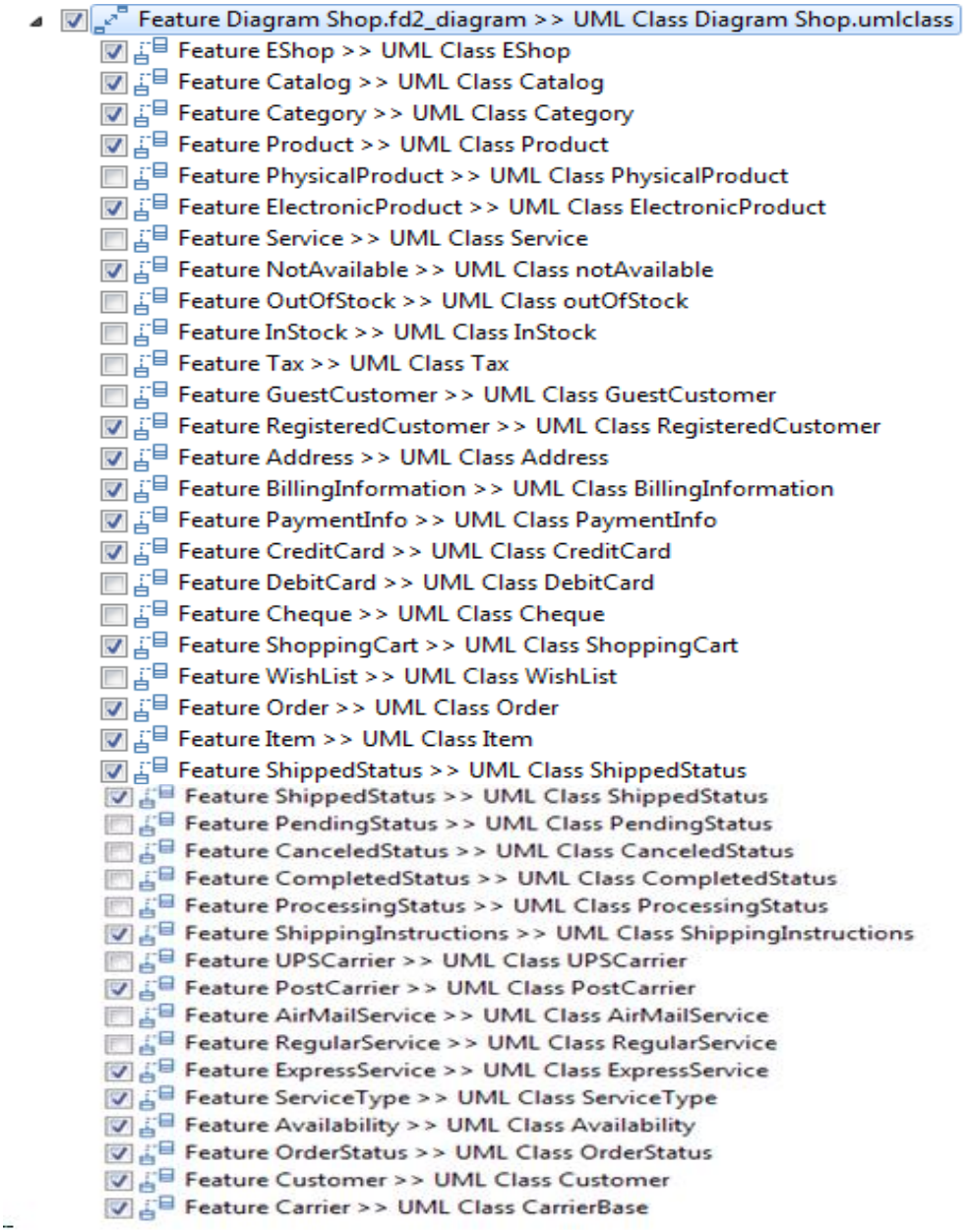
9.3. Elektroninės parduotuvės sistemos eksperimentų rezultatai

Eksperimento metu, buvo sukurtos 3 skirtingos elektroninės parduotuvės versijos. Kiekviena iš jų skiriasi konfigūracijoje parinktais požymiais. Šiame skyriuje aprašysime pagrindinius skirtumus tarp sukonfigūruotų sistemų ir bendrinės sistemos UML klasių diagramų. 25-oje lentelėje pateikta konfigūracijų informacija.

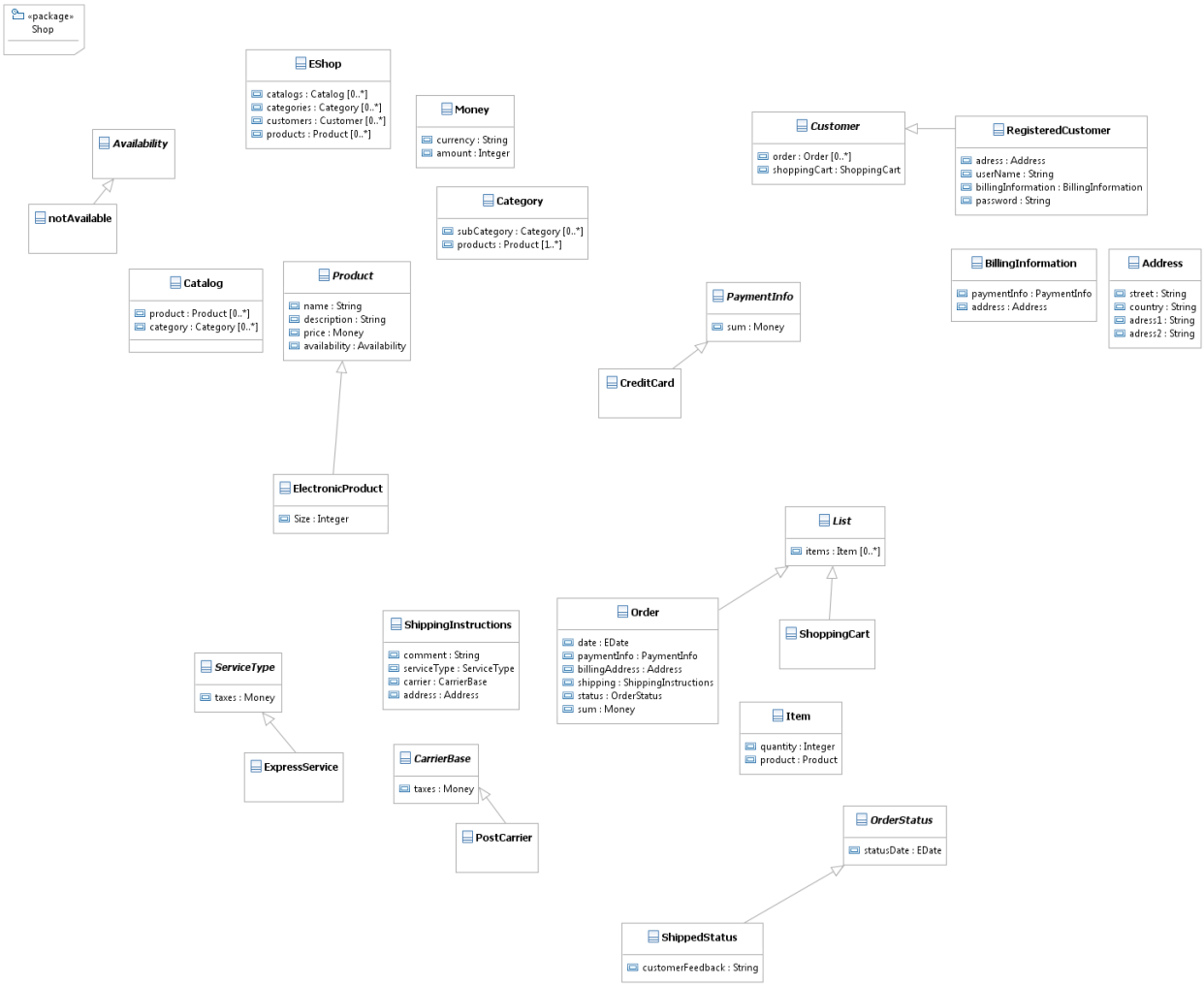
25 lentelė. Elektroninės parduotuvės variantų konfigūracijos.

Požymis	**1-as bandymas**	**2-as bandymas**	**3-as bandymas**
Address	+	+	+
AirMailService	-	-	-
Availability	+	+	+
BillingInformation	+	+	+
CanceledStatus	-	+	+
Carrier	+	+	+
Catalog	+	+	+
Category	+	+	+
Cheque	-	-	+
CompletedStatus	-	+	+
CreditCard	+	+	+
DebitCard	-	-	+
Electronic product	+	+	+
EShop	+	+	+
ExpressService	+	+	-
GuestCustomer	-	-	+
InStock	-	+	+
Item	+	+	+
NotAvailable	+	+	+
Order	+	+	+
OrderStatus	+	+	+
OutOfStock	-	-	+
PaymentInfo	+	+	+
PendingStatus	-	+	+
PostCarrier	+	+	+
ProcessingStatus	-	+	+
Product	+	+	+
Psychical product	-	-	+
RegisteredCustomer	+	+	+
RegularService	-	-	+
Service	-	-	+
ServiceType	+	+	+
ShippedStatus	+	+	+
ShippingInstructions	+	+	+
ShoppingCart	+	+	+
Tax	-	+	+
UPSCarrier	-	-	+
WishList	-	-	+

9.3.1. Bandymas pasirenkant 23 požymius



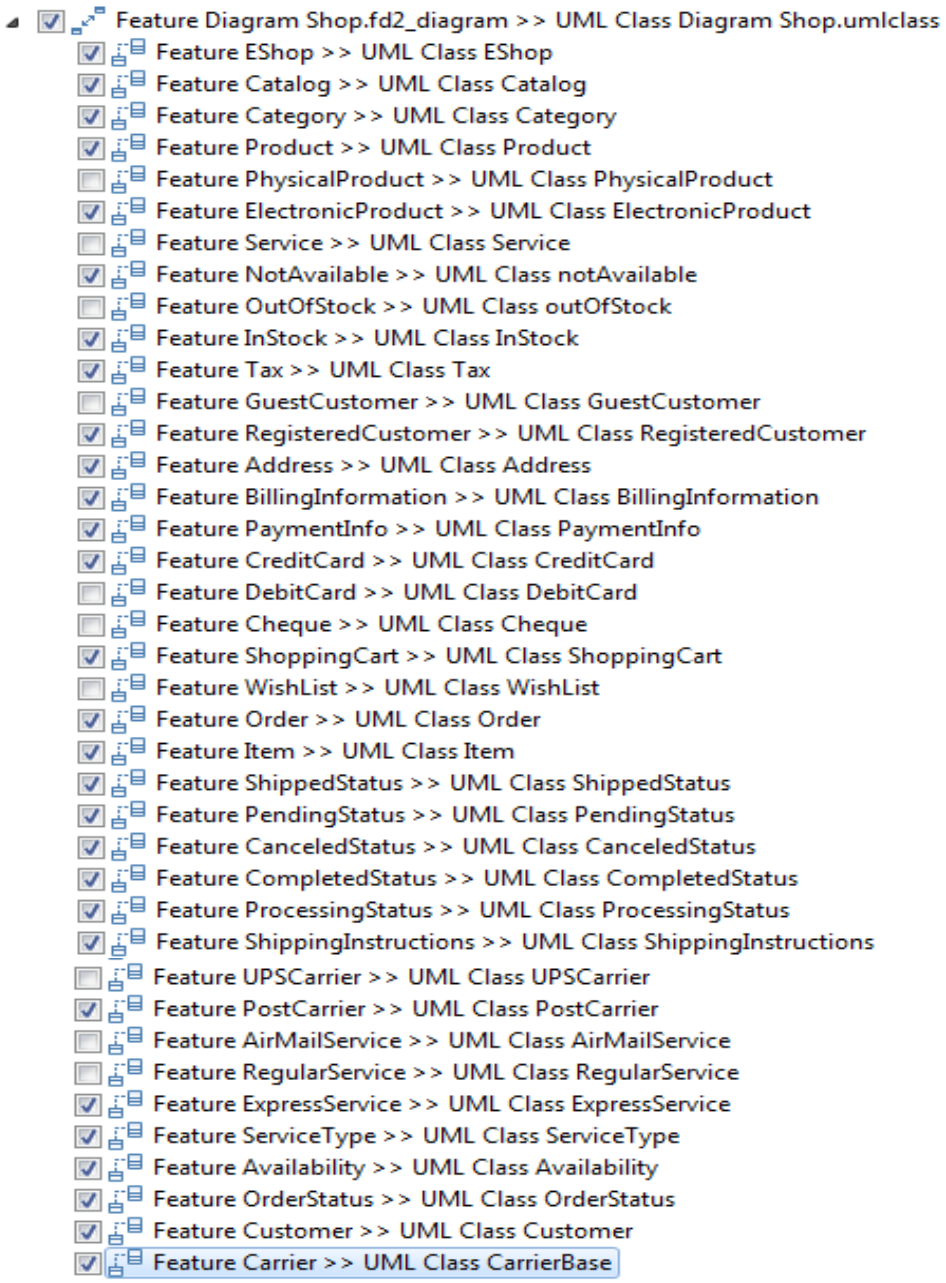
53 paveikslas. Bandymas pasirenkant 23 požymius. Požymių konfigūracija.



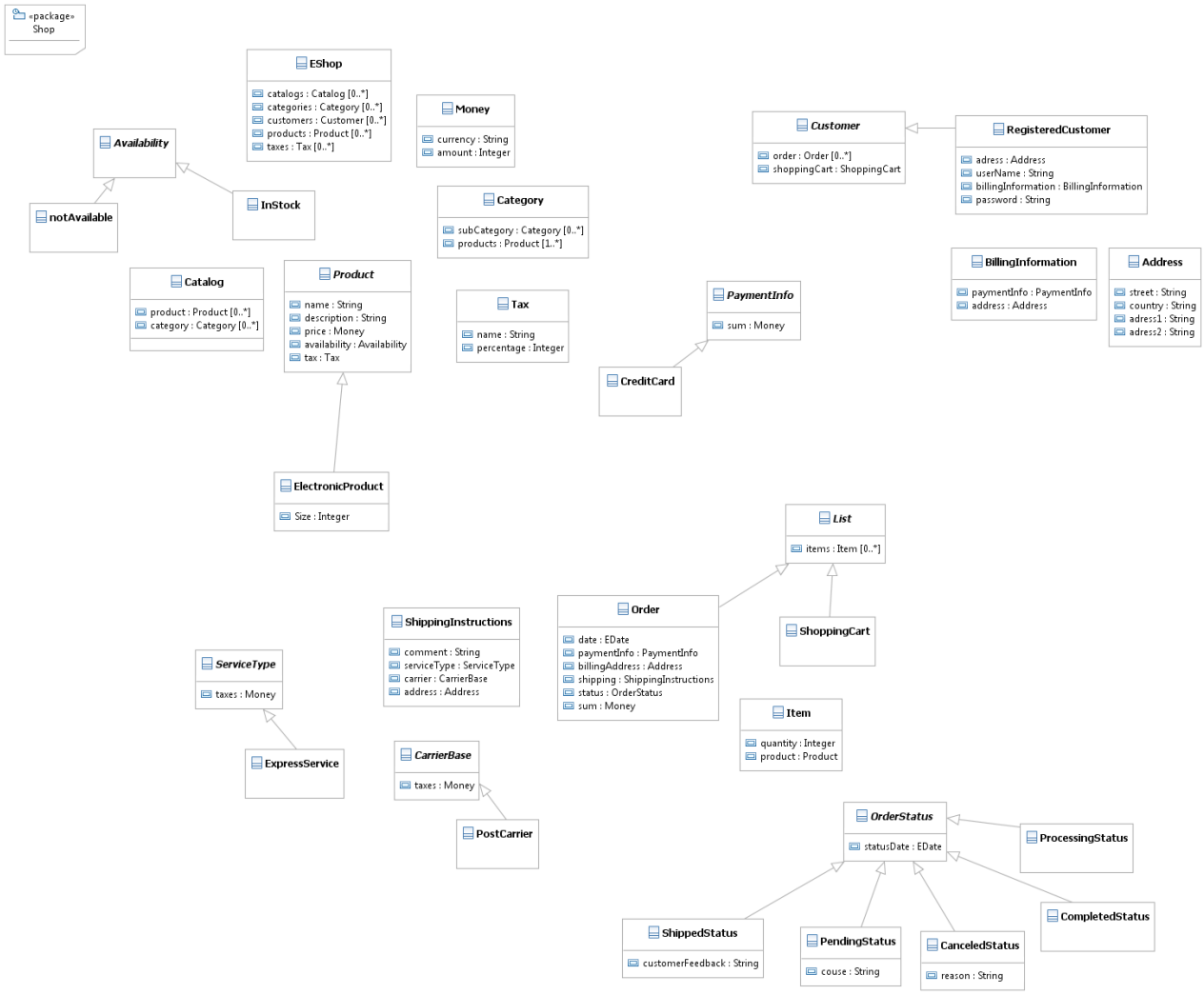
54 paveikslas. Bandymas pasirenkant 23 požymius. Klasių diagrama.

Sukurtas minimalus elektroninės parduotuvės variantas. Pasirinktas mažiausias galimas sistemos požymių skaičius. Ši sistema yra labai ribota ir neturi daugelio galimybių.

9.3.2. Bandymas pasirenkant 30 požymius



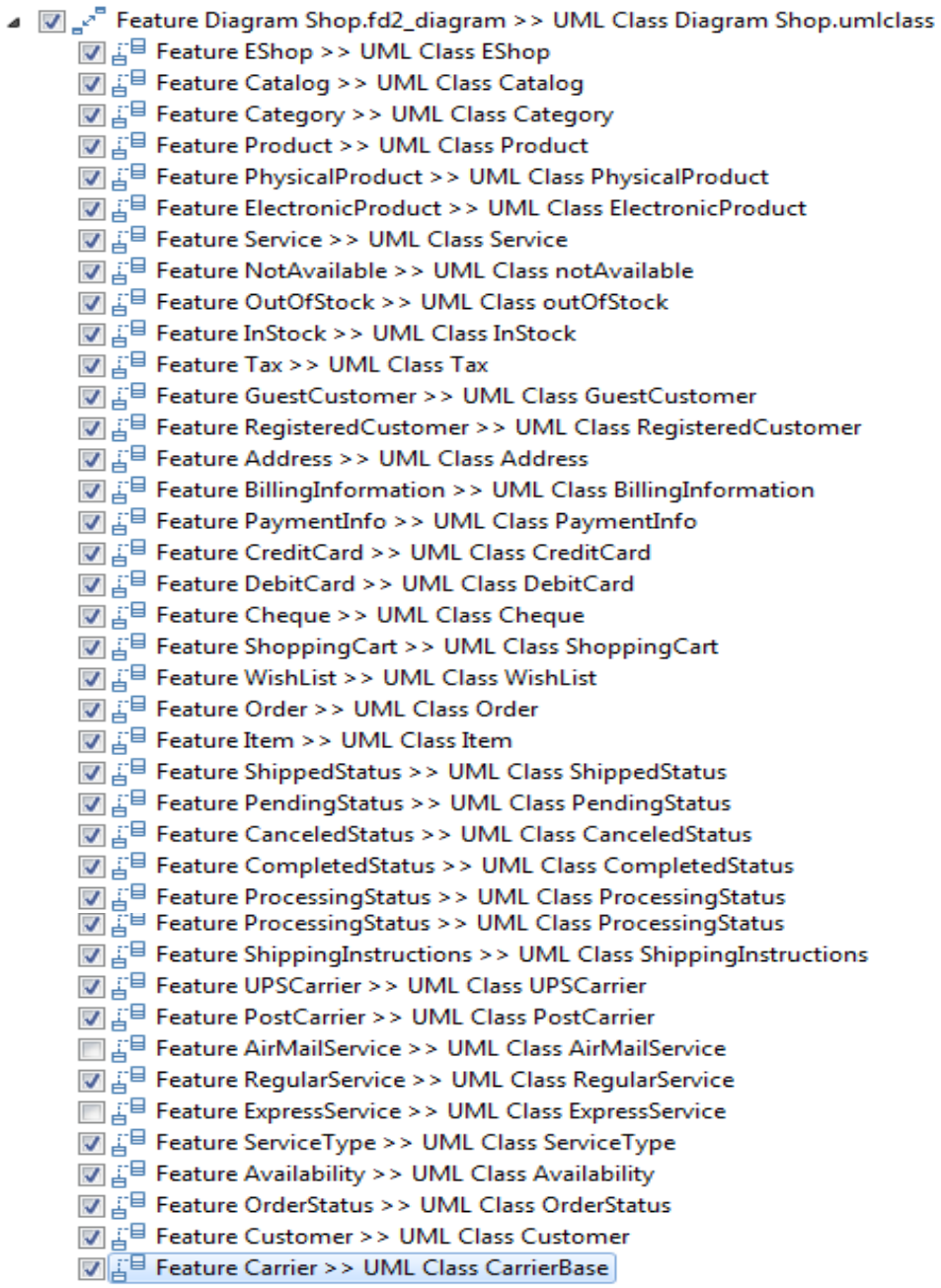
55 paveikslas. Bandymas pasirenkant 30 požymių. Požymių konfigūracija.



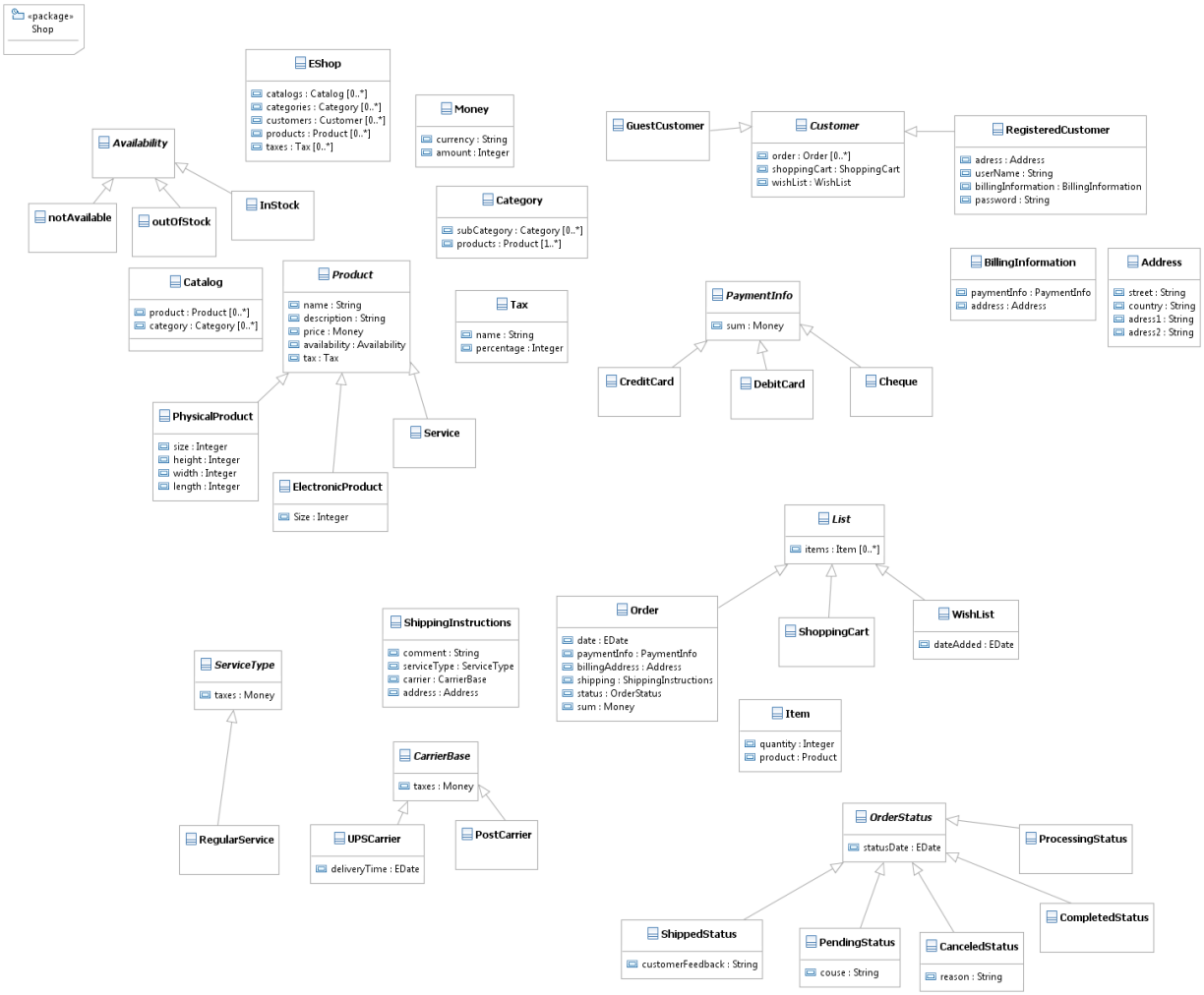
56 paveikslas. Bandymas pasirenkant 30 požymių. Klasių diagrama.

Ši sistema nuo bendrinės UML sistemos skiriasi eile požymių (AirMailService, Cheque, DebitCard, GuestCustomer, OutOfStock, Psychical product, RegularService, Service, UPSCarrier, WishList). Ir be šių komponentų sistema gali būti sėkmingai naudojama, tik bus jaučiamas kai kurių galimybių trūkumas.

9.3.3. Bandymas pasirenkant 37 požymius



57 paveikslas. Bandymas pasirenkant 37 požymius. Požymių konfigūracija.



58 paveikslas. Bandymas pasirenkant 37 požymius. Klasių diagrama.

Ši sistema nuo bendrinės UML sistemos skiriasi mažiausiai. Vienintelis skirtumas yra tas, kad nėra įtrauktos „AirMailService“ ir „ExpressService“ klasės, nes jų neleidžia įtraukti požymių diagramos konfigūravimo įrankis. Požymių diagrama nurodo, kad galima pasirinkti vieną iš 3 požymių: „AirMailService“, „ExpressService“, „RegularService“.

9.3.4. Sukurto programos kodo pavyzdys

Šiame priede pateikiamas vienos klasės pavyzdys, kuris parodo, kuo skiriasi programos kodas sukurtas iš bendrinio UML modelio (9.3.4.1 skyrius) ir modifikuoto UML modelio (9.3.4.2 skyrius).

9.3.4.1 Klasė ‚Customer‘ prieš keitimą

```
package Shop;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;
public interface Customer extends EObject {
	EList<Order> getOrder();
	ShoppingCart getShoppingCart();
	void setShoppingCart(ShoppingCart value);
	WishList getWishList();
	void setWishList(WishList value);
```

```
} // Customer
```

9.3.4.2 Klasė ‚Customer‘po keitimo

```
package Shop;
import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.ecore.EObject;
public interface Customer extends EObject {
      EList<Order> getOrder();
      ShoppingCart getShoppingCart();
      void setShoppingCart(ShoppingCart value);
      void setWishList(WishList value);
} // Customer
```

9.4. Advanced product line feature modelling using FD2

2010 metais Paulius Žaliaduonis, Deividas Kreivys ir Robertas Damaševičius dalyvavo tarptautinėje IT2010 konferencijoje, kurioje pristatė sukurtą požymių modeliavimo įrankį (FD2) ir parašė straipsnį požymių modeliavimo tema. Straipsnis pateiktas toliau Anglų kalba.

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.

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.

Syntax of Feature Diagrams

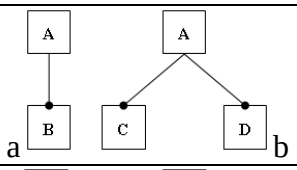
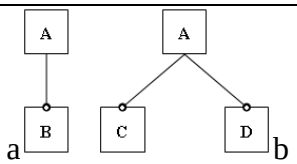
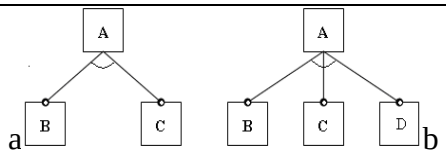
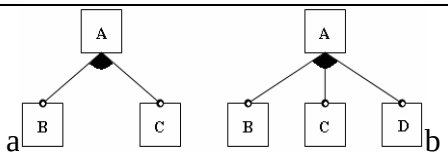
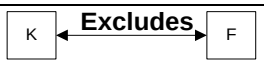
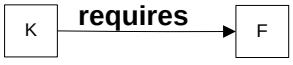
Feature models are specified using Feature Diagram (FD), a graphical notation for representing and modelling variability at a higher abstraction level. It was first was introduced in FODA [3] and since then has been a subject of active research and extension by numerous authors [4-7]. As a result, there 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].

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	

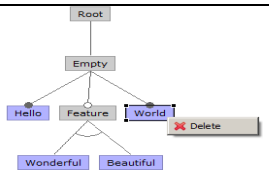
Related work

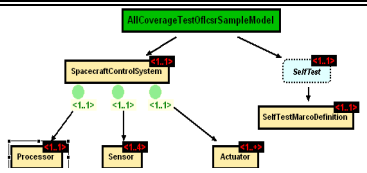
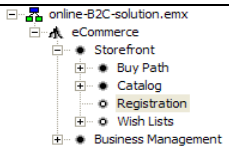
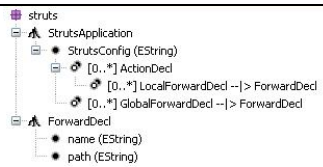
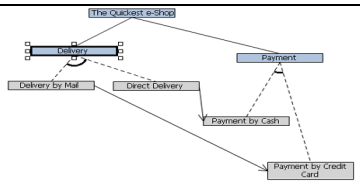
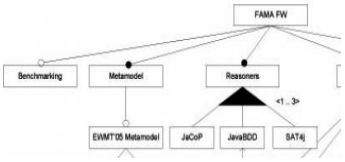
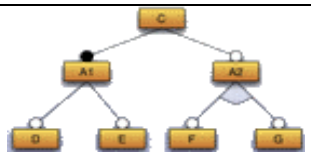
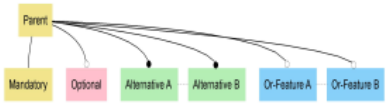
Currently, there are a number of feature modelling tools available. We have analyzed several feature modelling tools as follows: Feature IDE, XFeature, Ecore Feature modelling plugin, RequiLine, FAMA, KumbangTools, pure::variants (see Table 1). Most of projects mentioned are developed as Eclipse plug-ins 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.

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/

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.

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.

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.

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.

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.

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.

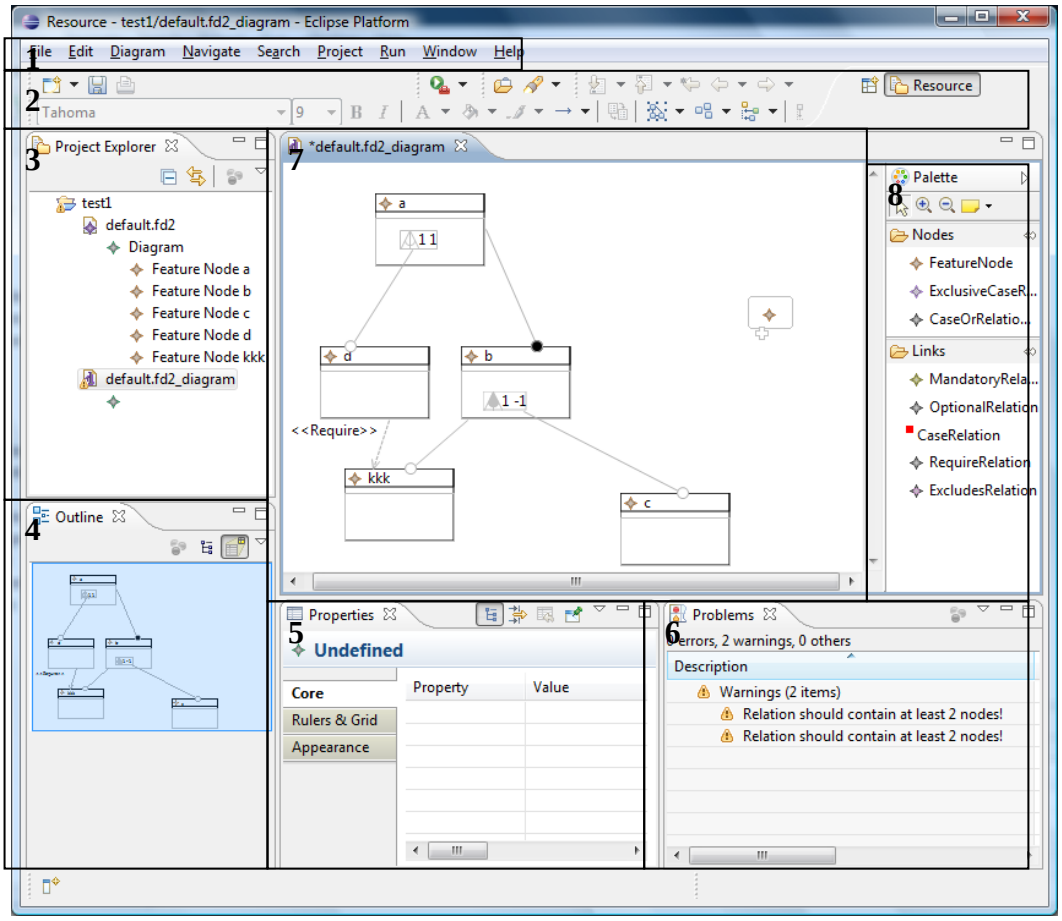


Figure 1. Interface of the FD2 modelling environment

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.

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.

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.

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.

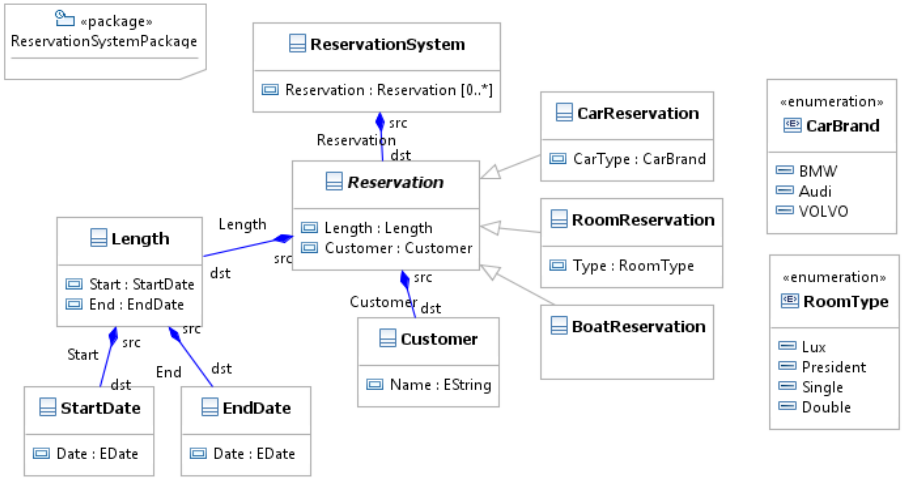


Figure 2.UML model of the reservation system.

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

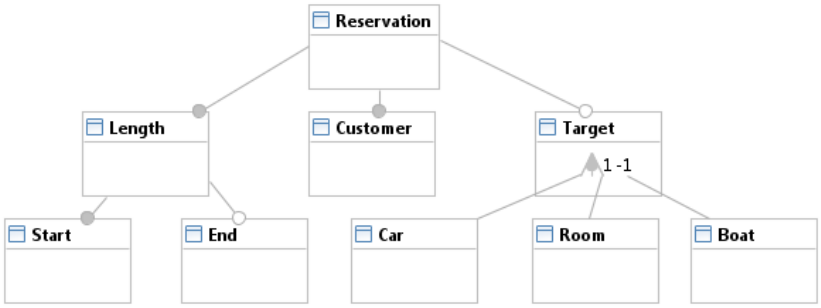


Figure 3. Feature model of the reservation system.

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.

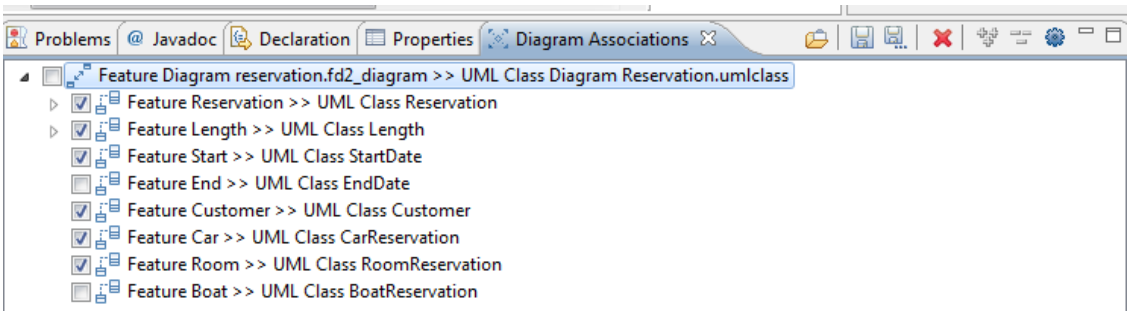


Figure 4. Feature configuration process.

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).

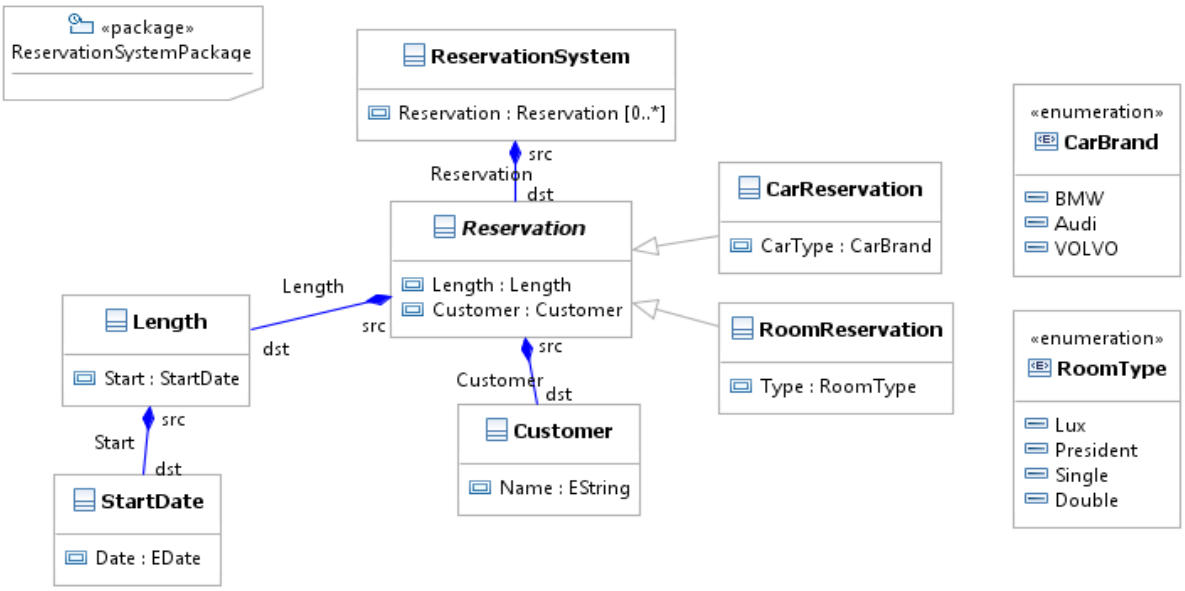


Figure 5. Configured class diagram of the reservation system.

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.

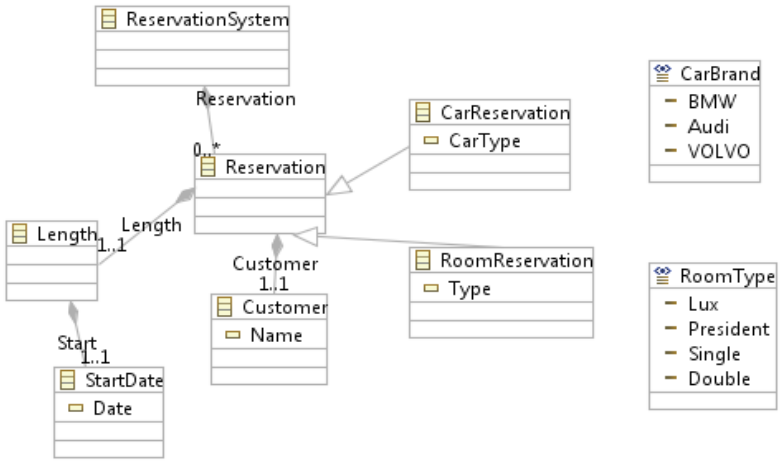


Figure 6. Configured ECore diagram of the reservation system.

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.