VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ KATEDRA

Programinės įrangos su kintamybėmis konfigūravimas

Configuration of software with variabilities

Magistro baigiamasis darbas

Atliko:	Danutė Pačebutaitė	(parašas)
Darbo vadovas:	Donatas Čiukšys	(parašas)
Recenzentas:	Viktoras Golubevas	(parašas)

Vilnius – 2010

Santrauka

Programų sistemų šeima, tai grupė programų sistemų turinčių tarpusavyje daug bendrybių, tačiau besiskiriančių viena nuo kitos tam tikrais iš anksto žinomais skirtumais, kurie yra modeliuojami kintamybėmis. Toks skirstymas į šeimas yra svarbus kalbant apie pakartotinį programų sistemų dalių panaudojimą išnaudojant programų sistemų panašumus. Tačiau kuo didesnė programų sistemų šeima, tuo daugiau joje skirtumų tarp šeimos narių, t.y. labai svarbiu tampa kintamybių suvaldymo klausimas. Konfigūravimas šiuo atveju yra procesas skirtas programų sistemų konstravimui iš turimų ypatybių (bendrybių ir kintamybių).

Šiame darbe yra aprašytas konfigūravimo metodas paremtas binarinėmis sprendimų diagramomis (BDD). Pirmiausiai programų sistemų šeima sumodeliuojama pagal Czarnecki-Eisenecker ypatybių modelį su kardinalumais, tada darbe aprašytu būdu verčiama į sumažintą surikiuotą binarinę sprendimų diagramą (ROBDD). Šiame darbe parodyta, kad turint ROBDD galime efektyviai konfigūruoti sumodeliuotą programų sistemų šeimą. Taip pat pateiktas konfigūravimo įrankio prototipas.

Raktiniai žodžiai: ypatybių modeliavimas, binarinės sprendimų diagramos, BDD, ROBDD, konfigūravimas, kintamybės, programinė įranga.

Summary

Software Systems family is a group of software systems that have great commonality among the systems, but differ from each other in some pre-known differences, which are modeled by variabilities. This distinction is important to families in terms of re-use of software systems by exploiting the similarities between software systems. However, the larger family of software systems, the more there are differences between family members, this means, managing variabilities becomes very important issue. In this case, configuration is a process of constructing new software systems from available features (commonalities and variabilities).

This paper describes the configuration method based on binary decision diagrams (BDD). First, the software is modeled according to the Czarnecki- Eisenecker feature model with cardinalities, subsequently in the described way it is transformed into reduced ordered binary decision diagram (ROBDD). This paper demonstrates effective configuration of modeled software with the use of ROBDD. In addition, the prototype of the configuration tool is presented.

Key words: feature modeling, binary decision diagrama, BDD, ROBDD, configuration, variabilities, software.

Turinys

Įvadas

Programų sistemų šeima, tai grupė programų sistemų turinčių tarpusavyje daug bendrybių, tačiau besiskiriančių viena nuo kitos tam tikrais iš anksto žinomais skirtumais, kurie yra modeliuojami kintamybėmis. Toks skirstymas į šeimas yra svarbus kalbant apie pakartotinį programų sistemų dalių (tiek kodo, tiek pavyzdžiui reikalavimų) panaudojimą išnaudojant programų sistemų panašumus (bendrybes). Tačiau kuo didesnė programų sistemų šeima, tuo daugiau joje skirtumų tarp šeimos narių, t.y. labai svarbiu tampa kintamybių suvaldymo klausimas. Tokioms programų sistemų šeimoms paprastai projektavimo metu būna nurodomos ypatybės (bendrybės ir kintamybės), kurios gali sudaryti šeimos narius, ir sąryšiai tarp jų. Pavyzdžiui, viena ypatybė gali reikalauti, kad kuriamoje programų sistemoje būtų pasirinkta ir kita ypatybė, arba atvirkščiai, pasirinkus vieną ypatybę yra draudžiama rinktis tam tikrą kitą. Konfigūravimas šiuo atveju yra procesas skirtas programų sistemų konstravimui iš turimų ypatybių (paprastai realizuojamų komponentais), o pati konfigūracija yra pasirinktų ypatybių aibė. Konfigūruojant yra labai svarbu sudaryti tik teisingas konfigūracijas, t.y. tokius ypatybių rinkinius, kuriuose nėra pažeistų priklausomybių. Pažeidus priklausomybes gausime neveikiančią programų sistemą.

Konfigūravimas paprastai yra interaktyvus procesas, kurio metu vartotojas (dažniausiai žmogus) renkasi norimas ypatybes.

Prieš pradedant konfigūravimo procesą, reikia atsakyti į klausimą:

1. Ar duotai programų sistemų šeimai egzistuoja bent viena teisinga konfigūracija?

Konfigūravimo proceso eigoje, turint aibę vartotojo pasirinktų ypatybių, svarbu atsakyti ir į šiuos klausimus:

2. Ar jau suformuota konfigūracija yra teisinga, t.y., ar yra pasirinktos visos būtinos ypatybės, ir ar nėra pažeistos priklausomybės tarp ypatybių?
3. Jei konfigūracija dar nėra teisinga, tai:
    a. Kokios priklausomybės buvo pažeistos, arba
    b. Kiek ir kokių pasirinkimų dar reikia atlikti vartotojui, kad gautų teisingą konfigūraciją?

Norint atsakyti į šiuos klausimus, ypatybių diagramą galime užrašyti teiginių logikos formule, kur kiekviena ypatybė žymima loginiu kintamuoju. Teigiama kintamojo reikšmė žymėtų pasirinktą ypatybę. Formulė būtų sudaryta tokiu būdu, kad įgytų teigiamą reikšmę tuo ir tik tuo atveju, kai pasirinktos ypatybės sudaro teisingą konfigūraciją. Turint tokią formulę, norint atsakyti į pirmą klausimą, užtektų surasti kintamųjų reikšmes, su kuria formulė įgyja teigiamą reikšmę. Jei tokių kintamųjų reikšmių neegzistuoja, seka, kad neegzistuoja ir teisingų

konfigūracijų ir ypatybių diagrama sudaryta nekorektiškai. Problema yra tai, kad nustatymas, su kuriomis kintamųjų reikšmėmis formulė įgyja teigiamą reikšmę (arba įrodymas, kad tokių reikšmių neegzistuoja), yra NP problema [Nor99], o tai reiškia, kad su realiuose projektuose sutinkamu ypatybių skaičiumi (tūkstančiai – dešimtys tūkstančių) atsakymo reikės laukti ilgai. Kadangi konfigūravimo procesas yra interaktyvus (t.y., pasirinkus variantą, norime iš karto sulaukti instrumento atsako, ar dėl mūsų pasirinkimo konfigūracija nepasidarė neteisinga), tai atsiranda poreikis efektyviems konfigūracijos tikrinimo algoritmams.

Vienas iš galimų sprendimų yra binarinės sprendimų diagramos. Tai yra specialus teiginių logikos formulių užrašymo būdas, kai formulėje naudojama tik viena loginė operacija `jei/taip/kitaip` [And97]. Konfigūravimo procesui daug geriau yra naudoti sumažintas surikiuotas binarines sprendimų diagramas, kurios bus aprašytos 2.1. poskyryje. Teiginių logikos formulės pavertimas į tokį specifinį pavidalą taip pat yra NP problema. Bet vieną kartą pavertus į tokį pavidalą (ir turint vartotojo jau pasirinktą ypatybių poaibį), atsakymą apie konfigūracijos teisingumą galime gauti per polinominį laiką [And97], taigi greitai.

Darbo tikslas yra sukurti efektyvų programinės įrangos su kintamybėmis konfigūravimo algoritmą, grindžiamą binarinėmis sprendimų diagramomis.

Siekiant apibrėžto tikslo, turi būti susipažinta su literatūra šiomis temomis:

1. programinės įrangos ypatybių modeliavimas;
2. binarinėmis sprendimų diagramomis.

Šiame darbe bus pasiūlyta, kaip:

1. kurti ypatybių modelius (naudojamus kintamybių modeliavimui);
2. ypatybių modelius užrašyti teiginių logikos formule;
3. teiginių logikos formulę užrašyti binarinės sprendimų diagramos pavidalu;
4. pasinaudojant binarinių sprendimų diagramų instrumentine priemone, gauti atsakymus į aukščiau suformuluotus klausimus.

1. Kintamybių modeliavimo būdų apžvalga

Plačiausiai naudojamas būdas modeliuoti kintamybes programinėje įrangoje yra ypatybių diagramos (angl. *feature diagrams*). Ypatybių modeliavimas yra neatsiejama dalykinių sričių inžinerijos (angl. *domain engineering*) dalis. Dalykinių sričių inžinerija, tai – procesas skirtas turimų dalykinės srities žinių pritaikymui, kuriant produktus skirtus pakartotiniam panaudojimui. Ypatybių modeliavimas ypač svarbiu tampa, jei norime sukurti pakartotinai panaudojamą produktą. Pakartotinai panaudojama programinė įranga neišvengiamai turi daugiau kintamybių, nei ištisinė, o ypatybių modeliavimas yra pagrindinė technika kintamybėms identifikuoti ir suvaldyti. Taip pat ypatybių modeliai padeda lengviau įsivaizduoti kintamybių mastą kuriamoje programinėje įrangoje.

Skirtinguose ypatybių modeliavimo būduose ypatybės apibrėžimas skiriasi. Bet paprastai ypatybė suvokiama, kaip tam tikros sąvokos (angl. *concept*) dalis, kuri tiksliai parodo apie kokią sąvokos realizaciją kalbame. Ypatybių modeliavimas yra procesas kuris nagrinėja bendras ir skirtingas sąvokų ypatybes bei jų tarpusavio priklausomybes (angl. *interdependency*) ir sudeda visa tai į vieną ypatybių modelį. Sąvokos šiuo atveju suprantamos kaip dalykinės srities elementai ar struktūros. Pavyzdžiui, modeliuojant ypatybes, kaip sąvoką galime laikyti programų sistemų šeimą arba tiksliau šeimos narį, nes tokiu atveju visi realizuoti programų šeimos nariai atitiks modeliuojamos sąvokos (šeimos nario) egzempliorius.

Ypatybių modelis kuriamas modeliuojant ypatybes. Jį paprastai sudaro ypatybių diagrama ir papildoma informacija apie kiekvieną ypatybę bei sąryšius su kitomis ypatybėmis. Ypatybių modelis iš esmės aprašo visą galimybių aibę, t.y. viską ką galima sukurti realizuojant kokį nors poaibį ypatybių, kuris neprieštarautų ypatybių diagramai ir tenkintų tam tikrus apribojimus (angl. *constraints*).

Ypatybių diagramos iš esmės yra medžiai, kuriais sąvokos yra išskaidomos į ypatybes, o ypatybės savo ruožtu skaidomos į sub-ypatybes, t.y. perkeliamos į žemesnį abstrakcijos lygį.

Šiame skyriuje bus aptarta 1990 metais S. G. Cohen, J. A. Hess, K. C. Kang, W.E. Novak ir S. A. Peterson pristatytą ypatybių modeliavimo būdą bei vėliau, 1998 metais K. Czarnecki ir 2005 metais K. Czarnecki bei Ch. H. P. Kim, pasiūlytus du šio metodo plėtinius. O taip pat bus aprašyti keletas alternatyvių ypatybių modelių užrašymo būdų.

1.1. FODA ypatybių modelis

FODA (angl. *Feature-Oriented Domain Analysis*) – tai ypatybėmis paremtas dalykinės srities analizavimo būdas pateiktas [CHK+90]. Toks požiūris pasirinktas, nes ypatybes supranta ne tik dalykinės srities analitikai, bet ir būsimos programinės įrangos vartotojai. Ši metodika apima tris žingsnius:

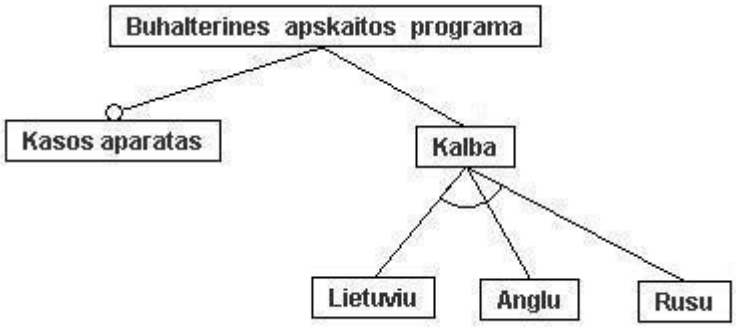
Konteksto analizė (angl. *context analysis*). Nagrinėjama dalykinė sritis, t.y. bandoma rasti problemų, kurias būtų galima spręsti kuriant programinės įrangos šeimą. Apibrėžiamos nagrinėjamos taikomosios srities ribos. Apibrėžiami išorinių elementų reikalavimai nagrinėjamai dalykinei sričiai.

Dalykinės srities modeliavimas (angl. *domain modeling*). Renkami programinės įrangos reikalavimai. Suformuluojamos norimos programinės įrangos ypatybės. Sudaromi ypatybių modeliai, t.y. ypatybių diagramos ir jų aprašai.

Architektūros modeliavimas (angl. *architecture modeling*). Daromi architektūriniai sprendimai, t.y. nusprendžiama, kaip ankstesniame žingsnyje aprašytos ypatybės bus realizuotos, kokiais komponentais ar jų dalimis.

Magistro darbo nagrinėjamoje problemoje svarbiausias yra antrasis žingsnis – srities modeliavimas ir jo metu sukuriamas ypatybių modelis, kuris iš esmės ir yra programų šeimos panašumų ir skirtumų (t.y. kintamybių) aprašas.

FODA ypatybių modelis susideda iš dviejų dalių: ypatybių diagramos ir jos aprašo. Ypatybių diagrama vaizduoja kokią nors sąvoką ir jos ypatybių dekompoziciją. Aprašas papildo diagramą sąryšiais tarp ypatybių bei papildoma naudinga informacija.



1 pav. FODA ypatybių diagrama

Pagal FODA ypatybių diagrama susideda iš mazgų (angl. *nodes*) ir kryptinių briaunų (angl. *directed edges*) bei briaunų dekoracijų. Šios briaunos ir mazgai sudaro medį. Briaunų dekoracijos aprašo mazgo sąryšį su briaunomis sujungtais kitais mazgais arba kitaip tariant aprašo mazgo dekompoziciją į sub-mazgus.

Medžio šaknis yra modeliuojama sąvoka, o visi kiti mazgai yra šios sąvokos ypatybių mazgai. Kiekvienos ypatybės mazgo tėvinis mazgas yra sąvokos mazgas arba kitos ypatybės mazgas. Tiesioginėmis (angl. *direct*) sub-ypatybėmis vadinsime sub-ypatybes, kurioms nagrinėjamas mazgas yra tėvinis mazgas, o visas kita sub-ypatybes vadinsime netiesioginėmis (angl. *indirect*). Pavyzdžiui, 1 pav. pavaizduotoje ypatybių diagramoje „Kasos aparatas“ yra tiesioginė sąvokos „Buhalterines apskaitos programa“ sub-ypatybė, o „Rusu“ – netiesioginė.

FODA naudoja tris mazgų rūšis: privalomi (angl. *mandatory*) mazgai, alternatyvūs (angl. *alternative*) mazgai ir nebūtini (angl. *optional*) mazgai. 1 pav. parodytas atitinkamų ypatybių mazgų vaizdavimas: „Kasos aparatas“ – nebūtinas mazgas; „Kalba“ – privalomas mazgas; „Rusu“, „Lietuviu“, „Anglu“ – alternatyvūs mazgai.

Sąvokos egzemplioriaus aprašas gaunamas iš privalomų ypatybių mazgų ir atitinkamai pasirenkamų ypatybių mazgų. Geri (validūs) aprašai gaunami einant per visą medį nuo šaknies ir pasirenkant visus privalomus mazgus, bei pasirenkant reikiamus neprivalomus mazgus (čia būtent žiūrima ar mazgas priklauso aprašomam sąvokos egzemplioriui) pagal taisykles:

- Medžio šakninis mazgas, t.y. pati sąvoka, visada įtraukiama į aprašą.
- Privalomi mazgai įtraukiami į sąvokos egzemplioriaus aprašą tada ir tik tada, kai tėvinis mazgas yra įtrauktas į aprašą. Bet jei pavyzdžiui tėvinis mazgas yra nebūtinas ir yra nepasirinktas, tai mazgas negali būti įtrauktas į aprašą.
- Nebūtini mazgai į aprašą gali būti įtraukti tik jei tėvinis mazgas yra įtrauktas. Žinoma, jei tėvinis mazgas įtrauktas, tai nebūtinas mazgas gali būti ir neįtraukiamas.
- Alternatyvūs mazgai gali būti įtraukti į aprašą, lygiai vienas iš kiekvienos grupės ir tik jei tėvinis mazgas yra pasirinktas.

Ypatybių diagramos aprašas paprastai susideda iš trijų dalių:

Kompozicijos taisyklės. Kompozicijos taisyklės iš esmės yra ribojimai (angl. *constraints*). FODA ypatybių modelyje yra dvi ribojimų rūšys. Pirmoji, nusako reikalavimo (angl. *requires*) ribojimą, t.y. nurodo, kad viena ypatybė negali būti pasirinkta be kokios nors kitos ypatybės. Antroji rūšis, nusako abipusio išskirtinumo (angl. *mutually exclusive*) ribojimą, t.y. nurodo, kad jei pasirinkta viena ypatybė, tai kita ypatybė negali būti pasirinkta. Kompozicijos taisyklės aprašomos pagal `<ypatybė1> („requires“ | „mutex-with“) <ypatybė2>` šabloną. Šios taisyklės paprastai gaunamos iš anksčiau kurtos nagrinėjamos dalykinės srities programinės įrangos. Pavyzdžiui, 1 pav. pavaizduotai ypatybių diagramai galima pritaikyti tokį ribojimą:

`„Kasos aparatas“ mutex-with „Rusu“.`

Šiuo ribojimu pasakoma, kad kasos aparate rusų kalba nėra palaikoma, todėl pasirinkus rusų kalbą prijungti kasos aparato negalima.

Problemos ir jų sprendimai. Šioje dalyje aprašomos ypatybių analizavimo metu iškilusios problemos ir pasirinkti jų sprendimo būdai. Būtent ši dalis padeda apsispręsti ar įtraukti į kuriamą programinę įrangą nebūtinas ypatybes ir kurias alternatyvias ypatybes pasirinkti iš alternatyvių ypatybių grupių.

Sistemos ypatybių katalogas. Šioje dalyje aprašomos jau egzistuojančios sistemos, kurtos pagal aprašytą ypatybių modelį, ir jose pasirinktos ypatybės, kitaip tariant būtent čia kaupiama patirtis.

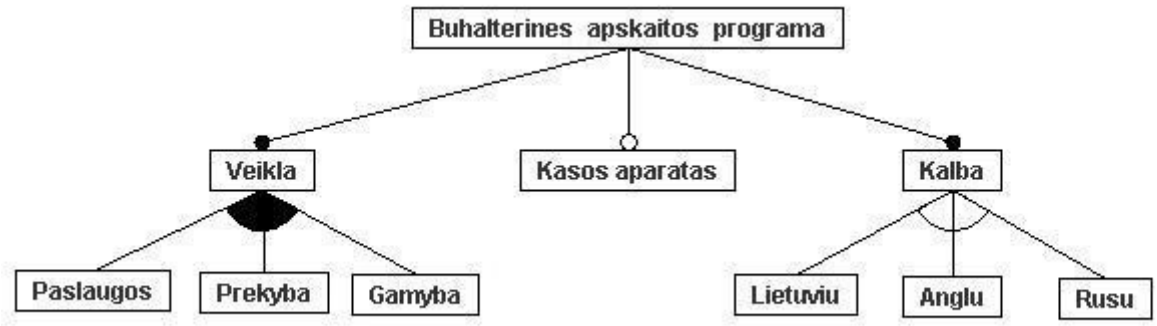
1.2. Czarnecki-Eisenecker ypatybių modelis

Czarnecki-Eisenecker aprašytas [Cza98] ypatybių modelis yra FODA modelis išplėstas arba-ypatybėmis (angl. *or-features*). Paprastai visi alternatyvių mazgų grupių ir arba-mazgų grupių nariai žymimi kaip privalomos ypatybės, tačiau jie gali būti ir nebūtini. Todėl atsiranda dvi galimos ypatybių rūšių kombinacijos. Pirmoji kombinacija, tai nebūtinos alternatyvios ypatybės. Antroji kombinacija, nebūtinos arba-ypatybės. Pastaroji nėra naudojama, normalizuojant ji suvedama į kitas rūšis.

Šis praplėstas modelis yra geras ta prasme, kad palieka galimybę keisti specifikaciją, t.y. pridėti ypatybes arba keisti jų savybes. Būtent tokiu atveju ir gali atsirasti ypatybių rūšių kombinacijos. Kad palengvėtų ypatybių modelio suprantamumas, ypatybių diagrama paprastai sutvarkoma – normalizuojama, panaikinamos atsiradusios kombinacijos.

Šis modelis normalizuojamas vykdant sekančius veiksmus:

- Grupė arba-mazgų, kurioje yra bent viena nebūtina arba-ypatybė keičiama į nebūtinas ypatybes. Tai iš esmės reiškia, kad galima nepasirinkti nei vienos ypatybės iš grupės.
- Grupė alternatyvių ypatybių su bent viena nebūtina ypatybe, keičiama taip: nebūtinos alternatyvios ypatybės „išmetamos“ iš grupės ir tampa nebūtinomis ypatybėmis, o likusi grupės dalis lieka nepakitusi.



2 pav. Papildyta ypatybių diagrama

Originaliai [Cza98] K. Czarnecki siūlė visoms būtinoms ypatybėms uždėti būtinumą reiškiančią briaunos dekoraciją, tačiau [CW07] jis atsisakė minties modeliuojant ypatybių grupes dėti būtinumo briaunos dekoraciją, tačiau ją paliko tik atskiroms ypatybėms. 2 pav. pavaizduota 1 pav. ypatybių diagrama su Czarnecki 1998 metų papildymais: arba-ypatybių grupe – ypatybės „Veikla“ vaikais ir būtinumo briaunų dekoracijomis.

Taip pat taikoma papildoma taisyklė. Arba-mazgai gali būti įtraukti vienas arba daugiau, jei pasirinktas tėvinis mazgas. Jei tėvinis mazgas nėra pasirinktas, tai arba-mazgai nėra traukiami į aprašą.

Papildyto modelio ypatybių diagramos aprašą sudaro šios dalys:

Semantinis aprašas (angl. *semantic description*). Kiekviena ypatybė turėtų turėti bent trumpą savo aprašą. Gali būti pridėtos kelios skirtingos UML diagramos, aprašytas ypatybės kategorija, t.y. kokiam programinės įrangos aspektui priklauso ypatybė.

Paaiškinimas (angl. *rationale*). Aprašas kodėl tokia ypatybė buvo įtraukta į modelį. Taip pat kiekvienai kintančiai ypatybei turi būti aprašytos sąlygos ir rekomendacijos, kada ši ypatybė turėtų būti įtraukta į sąvokos egzmplioriaus aprašą.

Dalininkai ir klientinės programos (angl. *Stakeholders and client programs*). Naudinga prie kiekvienos ypatybės turėti aprašą aprašantį, kas yra suinteresuotas duotąja ypatybe.

Pavyzdinės sistemos (angl. *exemplar systems*). Jei yra žinoma aprašomos egzistuojančios sistemos su aprašoma ypatybe.

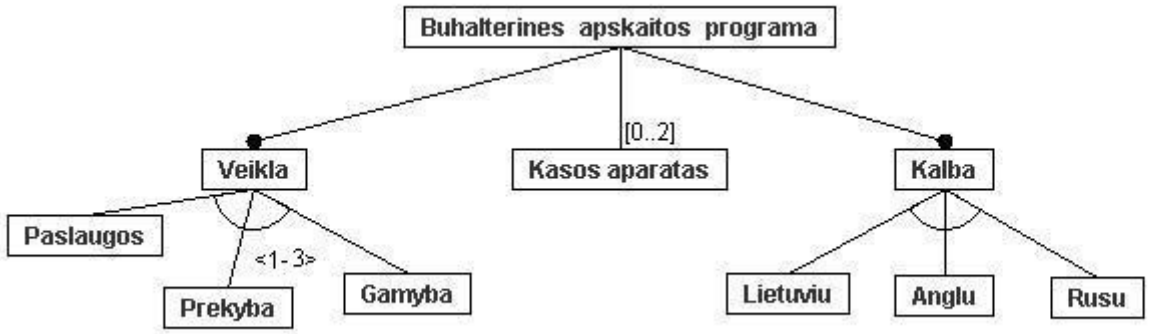
Apribojimai ir iš anksto numatytos priklausomybės (angl. *constraints and default rules*). Viena iš svarbiausių ypatybių modelio dalių. Aprašo sąryšius tarp skirtingų ypatybių diagramų. Pagrindinės apribojimų rūšys yra abipusio išskyrimo apribojimai ir būtinumo (angl. *requires*) apribojimai. Iš anksto numatytos priklausomybės nusako sąryšius tarp pasirinktų ir likusių ypatybių. Būtent ši dalis mums labai svarbi, nes tai labai palengvina ir automatizuoja konfigūravimo procesą.

Uždaryta/Atidaryta atributas (angl. *open/closed attribute*). Šiuo atributu patogu pasižymėti ar gali būti kitimo taškas dar praplėstas, t.y. ar galima pridėti papildomų tiesioginių ypatybių.

Prioritetai (angl. *priorities*). Prioritetai iš esmės parodo ypatybės svarbą projektui, pagal tai gali būti nuspręsta ypatybių realizavimo tvarka.

1.3. Czarnecki-Eisenecker ypatybių modelis su kardinalumais

2005 metais K. Czarnecki, Ch. H. P. Kim [CK05] pateiktas FODA ypatybių modelio papildymas atsirado bandant ypatybių modelį panaudoti ne tik specifikacijos, bet ir realizacijos lygmenyje. Pastebėta, kad realizuojant modeliuojamas sistemas, kartais prireikia ypatybių klonavimo, t.y. aprašant sąvokos egzempliorių prireikia dviejų ar daugiau vienodų ypatybių su skirtingai pasirinktomis sub-ypatybėmis. O taip pat, gali būti svarbu žinoti kiek ypatybių iš arba-ypatybių grupės galima pasirinkti. Dėl šių priežasčių [CK05] aprašomas ypatybių modelis yra papildytas ypatybių ir jų grupių kardinalumais bei dar keliais ypatybių modeliavimo elementais.



3 pav. Ypatybių diagrama su kardinalumais

Ypatybių modelis su kardinalumais yra ypatybių hierarchija, kurioje kiekviena ypatybė turi savo kardinalumą, t.y kiekvienai ypatybei yra nurodytas intervalas `[n..m]`, kur `n` ir `m` yra sveikieji skaičiai arba 0, arba * ir `n` yra mažesnis arba lygus `m`. Kardinalumas `[1..1]` atitinka privalomą ypatybę, o kardinalumas `[0..1]` atitinka nebūtiną ypatybę.

Ypatybių kardinalumas leidžia ypatybių diagramoje vaizduoti klonuojamas ypatybes bei ypatybes su nežinomu pasirinkimo kartų skaičiumi. Be to, kiekviena nuklonuota ypatybė gali turėti skirtingas sub-ypatybes.

Dar vienas papildymas, tai ypatybių grupių kardinalumas `<m-n>`, kur `m` ir `n` sveikieji skaičiai arba 0 bei `m` mažesnis arba lygus `n`. O taip pat `m` ir `n` negali viršyti ypatybių skaičiaus grupėje, t.y. šis kardinalumas parodo tik kiek ypatybių gali būti pasirinkta ir tos grupės, bet kiekviena pasirinkta ypatybė gali būti nuklonuota tiek kartų, kiek leidžia jos kardinalumas.

3 pav. pavaizduota ypatybių diagrama su aukščiau aptartais kardinalumais. Pavyzdžiui, ypatybės kardinalumas pavaizduotas ant ypatybės „Kasos aparatas“. Kardinalumas `[0..2]` reiškia, kad prie šios buhalterinės apskaitos programos negalima prijungti daugiau nei dviejų kasos aparatų. Grupės kardinalumas pavaizduotas ypatybės „Veikla“ sub-ypatybių grupėje. Toks kardinalumas reiškia, kad buhalterinės apskaitos programoje turi būti bent viena veikla, bet gali būti ir kelios arba visos veiklos.

Šiame papildyme taip pat pridėta ypatybės atributo sąvoka. Tačiau kiekviena ypatybė gali turėti tik vieną atributą bei atributai gali būti tik trijų rūšių: Integer (skaičius), String (simbolių eilutė) arba FRef (nuoroda į kitą ypatybę). Ypatybes su keliais atributais galima modeliuoti atitinkamu skaičiumi sub-ypatybių su atributais.

Paskutinis pridėtas dalykas yra ypatybės mazgas kaip nuoroda į kitą ypatybių diagramą. Būtent šis ypatybių diagramos elementas labai palengvina ypatybių modeliavimą, nes didelės diagramos gali būti išskaidytos į mažesnes, o taip pat ypatybių diagrama gali turėti kelias nuorodas į tą pačią sub-diagramą.

Pagal autorius modeliuojama dalykinė sritis turėtų gana retai naudoti visus naujus užrašus (angl. *notations*). Pavyzdžiui, ypatybių diagramos vaizduojančios aukšto abstrakcijos lygmens ypatybes niekada nenaudos ypatybių klonavimo, kuris labai naudingas aprašant konfigūracijas su vykdymo metu (angl. *runtime*) pasirinktais komponentais.

Papildyta notacija naudoja visas FODA ypatybių modelyje esančias ribojimų rūšis, tik skiriasi jų pavadinimai (`implies` vietoj `requires` ir `excludes` vietoj `mutex-with`). Dėl papildomų ypatybių diagramų elementų atsiranda poreikis papildomiems ribojimas su aibėmis ir jų elementais, todėl autoriai siūlo įvesti papildomas ribojimų rūšis ir jas aprašinėti kokia nors ribojimų kalba, kaip, pavyzdžiui, OCL. Šiuo atveju OCL kontekstas suprantamas ne kaip klasė, bet kaip ypatybė (paprastai ta, į kurią yra nuoroda), tačiau [PRS03] yra pateiktas būdas, kaip aprašytą ypatybių diagramą galima būtų užrašyti UML (angl. *Unified Modeling Language*) klasių diagrama ir panaudoti OCL su keliais, ten pat pasiūlytais, papildymais diagramos ribojimams aprašyti.

1.4. Kintamybių ir kitimo taškų vaizdavimas

Programinės įrangos panašumai šiose ypatybių diagramose vaizduojami privalomais mazgais. O programinės įrangos kintamybės (arba kintamos ypatybės) ypatybių diagramose vaizduojamos pasinaudojant visomis likusiomis mazgų rūšimis: nebūtinais, alternatyviais, nebūtinai alternatyviais ir arba-mazgais. Tokiais mazgais vaizduojamas ypatybes galime vadinti kintamomis ypatybėmis. Mazgai kurių sub-ypatybės yra kintamos ypatybės vadinami kitimo taškais (angl. *variation points*).

Kitimo taškų rūšys ypatybių diagramose:

- Dimensija. Visi tiesioginiai sub-mazgai yra alternatyvūs mazgai.
- Dimensija su nebūtinomis ypatybėmis. Visi tiesioginiai sub-mazgai yra nebūtini mazgai.
- Plėtimo taškas (angl. *extension point*). Turi bent vieną tiesioginį nebūtiną sub-mazgą arba bent vieną aibę tiesioginių arba-mazgų.

- Plėtimo taškas su nebūtinomis ypatybėmis. Visi tiesioginiai sub-mazgai yra nebūtini mazgai.
- Plėtimo taškas su arba-ypatybėmis. Visi tiesioginiai sub-mazgai yra arba-mazgai.

1.5. Alternatyvūs ypatybių diagramos užrašymo būdai

FDL (angl. *Feature Description Language*) aprašyta [DK01] šaltinyje yra viena iš konkretesnių DSL (angl. *Domain Specific Language*), dalykinės srities specifinių kalbų, skirtų ypatybių diagramoms užrašyti tekstiniu formatu. Svarbu ir tai, kad FDL turi normalizavimo taisykles, kurios yra skirtos ypatybių diagramos aprašo sumažinimui bei nenaudojamų ypatybių panaikinimui. Dar vienas naudingas šios kalbos bruožas yra palyginus lengvas aprašo pavertimas į normaliąją disjunkcinę formą [And97], kas gali būti labai patogu konstruojant 1.2. poskyryje aprašomas binarines sprendimų diagramas.

Be jau paminėtų alternatyvų FODA ir jos praplėtimų užrašymo būdams dar galima būtų paminėti M. Becker 2002 metais [BEC02] aprašytą XML kalba paremtą ypatybių modelio užrašymo būdą.

FDL kalbos notacija. FDL kalboje ypatybių diagrama aprašoma ypatybių apibrėžimais, kuriuos sudaro ypatybės vardas, `:` ir ypatybės išraiška. Ypatybės išraiška gali būti sudaryta iš:

- Atominių ypatybių, tokių ypatybių, kurios toliau neskaidomos į sub-ypatybes. Tokių ypatybių vardai pradedami mažąja raide.
- Sudėtinių ypatybių, tokių ypatybių, kurios turi sub-ypatybes ir turi savo apibrėžimus. Tokių ypatybių vardai pradedami didžiąja raide.
- Nebūtinų ypatybių. Žymima prie ypatybės pavadinimo pridedant `?` simbolį.
- Privalomų ypatybių. Tokių ypatybių sąrašas surašomas į `all()` išraišką.
- Alternatyvių ypatybių.Tokių ypatybių sąrašas surašomas į `one-of()` išraišką.
- Arba-ypatybių. Tokių ypatybių sąrašas surašomas į `more-of()` išraišką.
- Ypatybės su numatytąją reikšme. Tokios ypatybės aprašomos `default =` ir numatytojį vertė (t.y. kokia nors atominė ypatybė).
- Nebaigtų specifikuoti ypatybių. FDL kalba galima užrašyti ir neišbaigtas ypatybių diagramas, tokiu atveju planuojamos pridėti, bet nepridėtos ypatybės žymimos `...`.

Pavyzdžiui, 2 pav. esančios ypatybių diagrama ypatybės apibrėžtos FDL kalba atrodytų taip:

```
Buhalterines apskaitos programa : all(Veikla, kasos_aparatas?, Kalba)
Veikla : more-of(paslaugos, prekyba, gamyba)
Kalba : one-of(rusu, lietuvių, anglu).
```

O pati ypatybių diagrama apsirašytų taip:

```
all(more-of(paslaugos, prekyba, gamyba), kasos_aparatas?, one-of(rusu, lietuviu, anglu)).
```

2. Binarinių sprendimų diagramų apžvalga

Šiame darbe konfigūravimo procesu laikysime konfigūravimo procesą su dvejomis fazėmis (pasinaudosime [AHH+04] pasiūlytu dviejų fazių konfigūravimo procesu). Pirmojoje fazėje ruošiamasi konfigūravimui - apsirašoma teisingų konfigūracijų erdvė, o antrojoje fazėje atliekamas pats konfigūravimas naudojantis pirmosios fazės rezultatu. Šis požiūris yra geras tuo, kad pirmoji fazė vyksta tik vieną kartą ir paprastai nedalyvaujant žmonėms, todėl jos trukmė nėra tokia svarbi, kaip antrosios fazės, kai produktas yra interaktyviai, daug kartų konfigūruojamas, t.y. labai svarbus atsako greitis.

Pasinaudojant šiuo požiūriu, pirmoje konfigūravimo fazėje galima ypatybių modelį pervesti į teiginių logikos formulę, o pastarąją aprašyti binarinėmis sprendimų diagramomis. Antroje konfigūravimo fazėje pasinaudojant gauta binarine sprendimų diagrama galima atsakyti į visus konfigūravimo procesui svarbius klausimus per polinominį laiką, t.y. pakankamai greitai.

2.1. Binarinių sprendimų diagramų sudarymas

Norint atvaizduoti teiginių logikos funkciją pasinaudojant binarinėmis sprendimų diagramomis lengviausia yra ją pasivesti į tokią formą, kurioje būtų tik `jei/taip/kitaip` (žym. $\rightarrow$) loginis operatorius. Jis apibrėžiamas taip:

`(x→y0, y1)` atitinka $(x \wedge y0) \vee (\neg x \wedge y1)$.

Naudojant šį operatorių vietoje `y0` ir `y1` galima įrašyti tik kitą `jei/taip/kitaip` operatorių arba konstantas 0 arba 1, o vietoje `x` gali būti tik kintamasis, kuris paprastai vadinamas testu. Svarbu įsidėmėti, kad vietoje x negali būti kintamojo su neiginiu. Be to, šioje formoje negali būti tik kintamasis, be `jei/taip/kitaip` operatoriaus.

Visi kiti binarinių formulių operatoriai gali būti išreikšti per `jei/taip/kitaip` operatorių ir konstantas 0 ir 1 [And97], kitaip tariant, kiekviena binarinė formulė turi atitikmenį tik su `jei/taip/kitaip` operatoriumi. Pavyzdžiui, $\neg$`x` atitinka `(x→0, 1)`, o kintamasis `x` atitinka `(x→1, 0)`.

Binarinę funkciją pervesti į `jei/taip/kitaip` forma labai lengva pasinaudojant Shannon plėtinio sąvoka. Tarkime turime binarinę funkciją `t` su kintamuoju `x`, tada galime laikyti, kad funkcija `t[0/x]` atitiktų funkciją, kuri būtų gauta iš `t` funkcijos vietoje `x` įrašius 0 (atitinkamai `t[1/x]` vietoje `x` į `t` įrašytume 1). Tokiu atveju funkciją

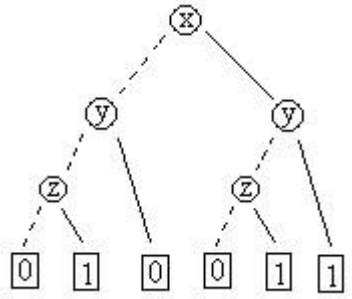
`t = x→t[1/x],t[0/x]`

vadiname funkcijos t Shannon plėtiniu x atžvilgiu. Kitaip tariant, sudarant jei/taip/kitaip formą pasirenkame kokią nors binarinės formulės kintamųjų tvarką ir pagal ją formulei taikome Shannon plėtinį pagal atitinkamą kintamąjį. Pavyzdžiui, turime formulę $(x \wedge y) \vee (z \wedge \neg y)$. Pasirenkame kintamųjų tvarką x, y, z. Gauname:

$x \rightarrow (y \vee (z \wedge \neg y)), (z \wedge \neg y);$

$x \rightarrow (y \rightarrow 1, z), (y \rightarrow 0, z);$

$x \rightarrow (y \rightarrow 1, (z \rightarrow 1, 0)), (y \rightarrow 0, (z \rightarrow 1, 0)).$



4 pav. Binarinė sprendimų diagrama

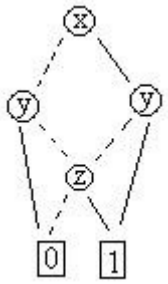
Kiekviena taip užrašyta funkcija gali būti atvaizduota kryptiniu grafu, be ciklų ir su šaknimi, į kurį įeina tik sprendimo mazgai (angl. *decision nodes*) ir du baigiamieji mazgai (angl. *terminal nodes*) – 0 ir 1. Kiekvienas sprendimo mazgas žymi binarinį kintamąjį ir turi du vaikinius mazgus: žemesnįjį (angl. *low*) vaiką ir aukštesnįjį (angl. *high*) vaiką su kuriais jis atitinkamai yra sujungtas žemesniąja ir aukštesniąja šakomis (arba trumpiau 0 ir 1 šakomis). Briauna jungianti mazgą su žemesniuoju vaiku vaizduoja sprendimą arba tikrinimą (angl. *test*), kuriame mazgo kintamajam priskiriama reikšmė 0 (žymima punktyru), o briauna jungianti su aukštesniuoju vaiku - kai mazgo kintamajam priskiriama reikšmė 1 (žymima linija). Jei Shannon plėtinį pažymėtume $(x \rightarrow t1, t0)$, tai braižant binarinę sprendimų diagramą x būtų mazgas, t1 sekantis pasirinktas kintamasis būtų aukštesnysis vaikas, o t0 – žemesnysis vaikas. 4 pav. pateikta tokiu būdu gauta binarinė sprendimų diagrama.

„Vaikštant“ binarine sprendimų diagrama pasirenkamos kintamųjų reikšmės. Jei galų gale prieinamas baigiamasis mazgas 0 su pasirinktomis kintamųjų reikšmėmis diagrama vaizduojama binarinė funkcija įgyja reikšmę 0, o jei prieinamas baigiamasis mazgas 1 – reikšmę 1. Jei priėjus baigiamąjį binarinės sprendimų diagramos baigiamąjį mazgą nebuvo „aplankyti“ kurie nors mazgai, galima daryti išvadą, kad su kitų kintamųjų pasirinktomis atitinkamomis reikšmėmis nuo kintamųjų su nepasirinktomis reikšmėmis binarinės formulės reikšmė nepriklauso.

Jei skirtingi kintamieji visuose keliuose nuo grafo viršūnės iki baigiamųjų mazgų yra išdėstyti vienoda tvarka, tai tokia binarinė sprendimų diagrama vadinama surikiuota (angl. *ordered*). 4 pav. esanti binarinė sprendimų diagrama yra surikiuota. Jos kintamųjų tvarka yra `x, y, z`.

Dar binarinės sprendimų diagramos gali būti sumažintos (angl. *reduced*), jei jos tenkina reikalavimus:

- Nėra dviejų mazgų vaizduojančių tą patį kintamąjį, kurių žemesnysis ir aukštesnysis vaikai atitinkamai yra vienodi. Mažinant jei du mazgai vaizduojantys tą patį kintamąjį turi tuos pačius vaikus, jie apjungiami į vieną.
- Nėra tokio mazgo, kurio žemesnysis ir aukštesnysis vaikai sutampa, t.y. jie yra tas pats mazgas. Jei mazgo briaunos rodo į tą patį mazgą, tai jis yra praleidžiamas, o visos į jį ėjusios briaunos perkeliamos į mazgo vienintelį vaiką.



5 pav. Sumažinta binarinė sprendimų diagrama

Praktikoje dažniausiai naudojami sumažintos ir surikiuotos binarinės sprendimų diagramos dar vadinamos ROBDD (angl. *Reduced Ordered BDD*), tokias binarines sprendimų diagramas ir turėsime omenyje, kai kalbėsime apie binarines sprendimų diagramas. 5 pav. pavaizduota sumažinta ir surikiuota binarinė sprendimų diagrama gauta iš 4 pav. diagramos, ją sumažinus pagal aukščiau aprašytus punktus.

Lema. Bet kuriai funkcijai $f: \mathbb{B}^n \to \mathbb{B}$ egzistuoja lygiai viena ROBDD su kintamųjų rikiavimu $x1 < x2 < \ldots < xn$, tokia kad $f^u = f(x1, \ldots, xn)$.

Įrodymas pateiktas šaltinyje [And97].

Surikiuotos ir sumažintos binarinės sprendimų diagramos labai lengvai parodo, jei jų vaizduojamos binarinės funkcijos yra visada teisingos arba visada neteisingos. Jei binarinė funkcija yra visada neteisinga, tai jos binarinė sprendimų diagrama visada bus baigiamasis mazgas 0, o jei binarinė funkcija yra visada teisinga – baigiamasis mazgas 1.

Sekančiame skyriuje aptarsime sudėtingesnes operacijas su binarinėmis sprendimų diagramomis.

2.2. Manipuliavimas binarinėmis sprendimų diagramomis

Labai svarbu kalbant apie binarines sprendimų diagramas paminėti, kad tai yra labai kompaktiškas būdas binarinių formulių reikšmių lentelėms atvaizduoti. Tačiau šis būdas patogus ne tik dėl mažesnės atminties naudojimo. Binarinėms sprendimų diagramos yra sugalvoti efektyvūs manipuliavimo algoritmai pirmiausia pateikti [Bry86]. Toliau trumpai pakalbėsime apie mums aktualius algoritmus arba veiksmus su binarinėmis sprendimų diagramomis, kurie patobulinti ir iš naujo aprašyti [And97]. Kaip matysime 1.2.3. punkte, tokie algoritmai jau yra realizuoti visuose binarinių sprendimų diagramų programiniuose paketuose. Todėl [And97] aprašyti algoritmai bus tik išvardinti ir trumpai aptarti, bet detaliau nenagrinėjami.

Vienas iš esminių algoritmų yra binarinės sprendimų diagramos sudarymo algoritmas. Lengviausias būdas užrašyti binarinės funkcijos sprendimų diagramą yra sudaryti pilną sprendimų medį ir pagal 1.2.1. punkte aprašytas mažinimo taisykles jį sumažinti. Tačiau realiam binarinių sprendimų diagramų panaudojimui toks būdas yra netinkamas, nes net ir nedidelio kintamųjų kiekio užtenka, kad sudarytas sprendimų medis būtų per didelis saugoti atmintyje. Todėl [And97] yra pristatytas algoritmas jau sumažintų binarinių sprendimų diagramų kūrimui - BUILD. Šis algoritmas, kaip ir didžioji dalis kitų naudoja funkciją autoriaus pavadintą MK, kuri nagrinėjamą mazgą paima iš atminties arba jį sukuria, jei tokio nėra. H. R. Andersen [And97] primygtinai siūlo naudoti funkciją MK visuose pateiktuose algoritmuose, kad būtų išvengta kintamųjų mazgų (su atitinkamais vaikiniais mazgais) pasikartojimo.

Kitas svarbus algoritmas yra APLLY. Šis algoritmas yra svarbus tuom, kad binarinės sprendimų diagramos paprastai nėra konstruojamos iš karto ir pilnos, ypač jei norima spręsti konfigūravimo problemas. Kaip pamatysime 1.2.4. punkte labai patogu binarines sprendimų diagramas konstruoti iš atskirų, mažesnių binarinių sprendimų diagramų. Algoritmas APLLY būtent tokiam konstravimui ir yra skirtas.

Kitas konfigūravime dažniausiai naudojamas algoritmas yra RESTRICT. Šis algoritmas panaikina binarinių sprendimų diagramų kintamųjų mazgus, kuriems yra priskiriamos interaktyviai konfigūruojant parinktos reikšmės. Gauta sumažinta binarinė sprendimų diagrama ir padeda atsakyti į klausimą ar gauta nepilna konfigūracija yra teisinga, t.y. ar nepažeisti kintamųjų ribojimai? Reikia tik pasižiūrėti ar po RESTRICT algoritmo gautoj binarinėje sprendimų diagramoje yra kelių vedančių į baigtinį mazgą 1.

Likę trys algoritmai SATCOUNT, ANYSAT, ALLSAT skirti galimų variantų nagrinėjimui. SATCOUNT pateikia galimų konfigūracijų skaičių. ANYSAT pasako ar su padarytais pasirinkimais dar yra gerų konfigūracijų. Ir ALLSAT binarinės sprendimų diagramos pilno apėjimo būdų atrenka visas galimas geras konfigūracijas.

1 lentelė. Algoritmų sudėtingumas

Algoritmas	Sudėtingumas
BUILD	$O(2^n)$
APLLY	$O(n^2)$
RESTRICT	O(n)
SATCOUNT	O(n)
ANYSAT	O(n)
ALLSAT	$O(n2^n)$

1 lentelėje pateikti paminėtų algoritmų sudėtingumai. Iš karto matyti, kad sudėtingiausia problema yra visų galimų sprendimų aibė. O ir binarinės sprendimų diagramos sudarymas nėra greitas. Tačiau kiti binarinių sprendimų diagramų manipuliavimo algoritmai yra pakankamai efektyvūs.

2.3. Binarinių sprendimų diagramų paketai

CUDD (angl. *CU Decision Diagram Package*), tai Kolorado Universiteto realizuotas paketas skirtas darbui su sprendimų diagramomis [Som09]. CUDD paketas apima ne tik binarines sprendimų diagramas, bet ir kitas sprendimų diagramas, kaip, pavyzdžiui, ADD (angl. *Algebraic Decision Diagrams*) bei ZDD (angl. *Zero-suppressed Binary Decision Diagrams*). Pagal šio paketo autorių binarinės sprendimų diagramos yra naudojamos atvaizduoti keitimo funkcijoms (angl. *switching functions*). Šis paketas pateikia nemažai funkcijų skirtų manipuliavimui su binarinėmis sprendimų diagramomis.

CUDD paketas gali būti panaudotas trimis būdais:

Kaip juodoji dėžė. Šiuo atveju programa, kuriai reikia manipuliuoti binarinėmis sprendimų diagramomis naudoja išorines CUDD bibliotekos funkcijas. Paketas palaiko automatinį kintamųjų tvarkos nustatymą, net kai programa yra užsiėmusi.

Kaip baltoji dėžė. CUDD paketa galima plėsti, jei neužtenka jame realizuotų primityvių funkcijų, taip pagerinant efektyvumą.

Per vartotojo sąsają. CUDD paketas turi C++ kalba parašytą vartotojo sąsają. Šiuo atveju yra realizuotas automatinis šiukšlių surinkėjas.

BuDDy, tai dano Jorn Lind-Nielsen realizuotas binarinių sprendimų diagramų paketas [Lin07]. Kaip ir CUDD jis parašytas C++ kalba bei realizuoja tas pačias funkcijas, kaip ir CUDD. Tačiau šis paketas skiriasi savo realizacija. CUDD paketas grįstas rodyklių (angl. *pointers*) naudojimu, o BuDDy naudoja maišos lentelėmis (angl. *hash table*) paremtą binarinių sprendimų diagramų saugojimo atmintyje būdą.

JDD (angl. *Java Decision Diagrams*) paketas [Vah09], tai BuDDy paketo įkvėptas projektas. Iš BuDDy paimta ne tik visa paketo struktūra, bet ir kai kurių funkcijų realizacija iš C++ pervesta į Java kalbą. Kaip ir CUDD, JDD neapsiriboja tik paprastomis binarinėmis sprendimų diagramomis, jis taip pat realizuoja ZDD. JDD yra nemokamas paketas, kuris savo efektyvumu prilygsta CUDD ir BuDDy. Pagrindinis šio paketo trūkumas yra atmintis. Nors binarinių sprendimų diagramų saugojimui JDD naudoja mažiau atminties nei BuDDy, bet dėl Java virtualios mašinos specifikos JDD dažnai negali panaudoti visos galimos atminties.

Šiame darbe naudosime JDD biblioteką.

BDD – pagrindinė bibliotekos klasė, kurioje realizuota visa binarinių sprendimų diagramų logika. Ji paveldi iš klasės `NodeTable`, todėl per ją galima naudoti metodus `getLow` ir `getHigh`, skirtus apeiti binarinę sprendimų diagramą. Be to, svarbu nepamiršti naudoti metodo `ref`, kad šiukšlių surinkėjas neištrintų diagramos.

2 lentelė. BDD klasės metodai

Metodas	**Aprašymas**
BDD(int nodesize)	Klasės konstruktorius. <nodesize> galimas mazgu skaičius binarinėje sprendimų diagramoje.
int **and**(int u1, int u2)	operacija - (u1 & u2), kur <u1> ir <u2> yra binarinės sprendimų diagramos arba binariniai kintamieji. Metodas gražina nuorodą į naują binarinę sprendimų diagramą, kuri yra operacijos rezultatas.
void **cleanup**()	metodas skirtas objekto išvalymui. Turi būti kviečiamas baigiant darbą. Jis atlaisvina atmintį.
int **createVar**()	metodas sukuria naują kintamąjį binarinėje sprendimų diagramoje.
long **getMemoryUsage**()	metodas grąžina užimtos atminties apimtį.
int **getOne**()	metodas grąžina simbolinį vienetą.
int **getZero**()	metodas grąžina simbolinį nulį.
int **imp**(int u1, int u2)	binarinė implikacija u1 $\rightarrow$ u2.
boolean **isVariable**(int bdd)	metodas gražina reikšmę „true“, jei kintamasis <bdd> yra binarinis kintamasis priklausantis šiam BDD klasės objektui.
int **ite**(int f, int then_, int else_)	operacija - (if f then then_ else else_), kur <f>,<then_> ir <else_> yra binarinės sprendimų diagramos arba binariniai kintamieji. Metodas gražina nuorodą į naują binarinę sprendimų diagramą, kuri yra operacijos (jei/taip/kitaip) rezultatas.

Metodas	Aprašymas
int **nodeCount**(int bdd)	metodas gražina mazgų skaičių binarinėje sprendimų diagramoje <bdd>. Metodas labai lėtas, todėl naudoti geriau tik būtinais atvejais.
int **not**(int u1)	kintamojo arba binarinės sprendimų diagramos neigimas.
int **numberOfVariables**()	metodas grąžina binarinėje sprendimų diagramoje esančių kintamųjų skaičių.
int **oneSat**(int bdd)	metodas grąžina vieną galimą binarinės sprendimų diagramos reikšmių kombinaciją su kuria ji yra teisinga.
int **or**(int u1, int u2)	operacija - (u1 | u2), kur <u1> ir <u2> yra binarinės sprendimų diagramos arba binariniai kintamieji. Metodas gražina nuorodą į naują binarinę sprendimų diagramą, kuri yra operacijos rezultatas.
void **printDot**(String filename, int bdd)	metodas sukuria DOT bylą <bdd> binarinei sprendimų diagramai ir naudodama AT&T DOT įrankį ją paverčia paveiksliuku. Labai naudingas metodas, norint pamatyti, kaip atrodo sukonstruota binarinė sprendimų diagrama.
int **restrict**(int u, int v)	metodas gražina binarinę sprendimų diagramą, gautą iš binarinės sprendimų diagramos <u> išmetus binarinį kintamąjį <v>. Metodas nepakeičia binarinės sprendimų diagramos <u>.
double **satCount**(int bdd)	metodas grąžina binarinės sprendimų diagramos <bdd> galimų konfigūracijų skaičių.

3 lentelė. BDD klasės metodai paveldėti iš NodeTable klasės

Metodas	Aprašymas
int **getHigh**(int bdd)	metodas skirtas gauti mazgo aukštesnįjį vaiką.
int **getLow**(int bdd)	metodas skirtas gauti mazgo žemesnįjį vaiką.
int **getVar**(int bdd)	metodas skirtas gauti mazgo kintamojo numerį.

BDDIO – klasė skirta saugoti ir užkrauti binarines sprendimų diagramas iš išorinių bylų.

4 lentelė. BDDIO klasės metodai

Metodas	Aprašymas
static int **load** (BDD manager, String filename)	Užkrauti binarinę sprendimų diagramą iš failo. <manager> BDD klasės objektas, į kurį krausime. <filename> bylos pavadinimas (arba kelias) iš kurios užkrausime. Metodas grąžina nuorodą į užkrautą binarinę sprendimų diagramą.
static void **save** (BDD manager, int bdd, String filename)	Išsaugoti binarinę sprendimų diagramą faile. <manager> BDD klasės objektas, iš kurio išsaugosime binarinę sprendimų diagramą. <bdd> nuoroda į binarinę sprendimų diagramą. <filename> bylos į kurią saugosime pavadinimas.

Šią klasę naudosime saugoti pradinę binarinę sprendimų diagramą, nes konfigūravimo metu darant pasirinkimus ji bus modifikuojama, todėl norint iš naujo pradėti konfigūravimo procesą reikia turėti pradinę diagramos kopiją.

2.4. Kintamųjų rikiavimo problema

Konfigūracijos procesas paprastai susideda iš dviejų dalių: pirmoji kompiliavimo, kai sudaroma binarinė sprendimų diagrama ir antroji, kai ja pasinaudojama konfigūravimo uždaviniui spręsti. Kalbant apie binarinės sprendimų diagramos sudarymą, mums aktualesnis pirmasis žingsnis. Šio žingsnio metu paprastai sudaromi atskiros diagramos kiekvienam ribojimui, o po to tie ribojimai sudedami pasinaudojant APLLY operacija. Toks konstravimas turi savo trūkumų. Paprastai taip konstruojant galutinę binarinę sprendimų diagramą antrajam žingsniui diagramos dydis labai šokinėja (tai padidėja, tai sumažėja). Šaltinyje [NW06] paminėtos dvi euristikos padedančios žymiai sumažinti dydžius. O tai yra labai svarbu, nes kuo mažesnė binarinė sprendimų diagrama tuo mažiau atminties reikia jai saugoti ir tuo greičiau galima atlikti operacijas su ja.

Apribojimų pridėjimo tvarkos euristika.

Ši euristika nusako, kokia tvarka apribojimus reiktų pridėti prie pagrindinės diagramos. Ši euristika paremta tuo, kad specifikacijose esantys apribojimai paprastai sudaro į medį panašias struktūras, kurių binarinės sprendimų diagramos yra palyginus nedidelės (nedaug išsišakojusios). Be to, apribojimai parastai sudaro klasterius (vietas kur jų yra daugiausiai), todėl teisingai pasirinkus tvarką ir pridedant apribojimus iš to paties klasterio, operuojame mažesne diagrama bei ji dar mažėja, dėl po apribojimų sumažėjusių taip vadinamų testų skaičiaus. Ir trečioji

priežastis yra tai, kad nuo kintamųjų skaičiaus smarkiai priklauso ir diagramos dydis, todėl geriausiai yra pridėti apribojimus su tais pačiais kintamaisiais.

Pateikta euristika aprašo algoritmą, pagal kurį parenkama geriausia apribojimų pridėjimo tvarka. Autoriai praktiškai išbandę su dviem gautom specifikacijom pastebėjo, kad ši euristika buvo su geriausia kintamųjų išdėstymo tvarka. Iš karto po jos sekė specifikacijoje surašyta tvarka, o trečioje vietoje liko atsitiktine tvarka išrikiuoti kintamieji. Pasak autorių, taip yra todėl, kad žmonės yra linkę logiškai išdėlioti apribojimus specifikacijoje.

Žingsnis 1. Pirmojo „centrinio“ kintamojo parinkimas. Iš visų sprendžiamos problemos kintamųjų reikia pasirinkti vieną, kuris turi daugiausiai apribojimų su kitais kintamaisiais.

Žingsnis 2. Pirmojo apribojimo parinkimas. Tarp visų apribojimų su „centriniu“ kintamuoju parinkti apribojimą su didžiausiu kiekiu kintamųjų.

Žingsnis 3. Pasirinkti sekantį ribojimą. Surenkami visi apribojimai su „centriniu“ kintamuoju. Iš jų pirmiausia pridedamas apribojimas su mažiausiu skaičiumi kintamųjų, kurių dar nėra konstruojamoje binarinėje sprendimų diagramoje. Po to, parenkami apribojimai su mažiausiu kintamųjų skaičiumi. Jei nebelieka apribojimų su „centriniu“ kintamuoju einama į 4 žingsnį.

Žingsnis 4. Kito „centrinio“ kintamojo parinkimas. Surenkami vis dabartinio „centrinio“ kintamojo gretimi kintamieji (sujungti per apribojimą). Atmetami kintamieji, kurie nėra naudojami apribojimuose dar neprijungtuose prie konstruojamos binarinės sprendimų diagramos. Kiekvienam iš likusių apskaičiuojama, kiek turi bendrų apribojimų su kitais kintamaisiais ir kintamasis su didžiausiu skaičiumi parenkamas kaip naujas „centrinis“ kintamasis. Jei žingsnyje gautų kintamųjų aibė yra tuščia, tai žingsnis kartojamas, tik vietoje dabartinio „centrinio“ kintamojo paimamas buvęs prieš tai arba dar senesnis.

Kintamųjų rikiavimo euristika yra gana naudinga, nes visi žinomi algoritmai optimaliam kintamųjų išrikiavimui rasti yra NP sunkumo problema [FS90]. Kintamųjų rikiavimo euristikas galima padalinti į dvi rūšis: statinio rikiavimo ir dinaminio rikiavimo euristikas.

Pirmajame konfigūracijos žingsnyje svarbiausios yra statinio rikiavimo euristikos, antrajame dinaminės. Kaip rašoma [MT98] optimali kintamųjų tvarka konfigūravimo pradžioje ir pabaigoje gali labai smarkai skirtis, todėl keičiantis specifikacijai natūraliai prisireikia dinaminio kintamųjų rikiavimo. Sekančios euristikos paremtos pastebėjimais, kad stipriai susiję kintamieji rikiuotėje turėtų būt šalia bei jei nuo vienų kintamųjų priklauso kiti kintamieji, tai ankstesnieji turėtų eiti pirmesni.

Statinis kintamųjų rikiavimas.

Žingsnis 1. Šiame žingsnyje sugrupuojami tarpusavyje priklausomi kintamieji. Grupavimas atliekamas pagal prielaidą, kad viename klasteryje (susikaupime) esantys kintamieji yra labiau susiję, nei esantys skirtinguose. Tai galima padaryti pasinaudojus bet kokiu klasterių radimo algoritmu, daugiau apie tai [NW06].

Žingsnis 2. Šiame žingsnyje grupuojami kintamieji kiekvienoje iš 1 žingsnio gautoje grupėje atskirai. Kiekvienam kintamajam apskaičiuojamas jo svoris, t.y. suskaičiuojama kiek apribojimų jis turi su kitais kintamaisiais. Pasirenkamas „centrinis“ kintamasis su didžiausiu svoriu. Jis ir bus laikomas mažiausiu. Po to, perskaičiuojami kintamųjų su bendrais apribojimais su „centriniu“ kintamuoju svoriai. Naujas svoris gaunamas suskaičiavus bendrus apribojimus su „centriniu“ kintamuoju. Šie kintamieji išrikiuojami iš eilės pagal perskaičiuotą svorį. Likę kintamieji eina po to, iš eilės, pagal savo svorius, jei jų svoriai vienodi, jie rikiuojami pagal susijusiu per apribojimus kintamųjų svorius.

Žingsnis 3. Šiame žingsnyje sudaroma tvarka tarp pirmu žingsniu gautų ir antru žingsniu surikiuotų grupių. Kiekvienai tokiai grupei suskaičiuojamas apribojimų skaičius su ne grupės viduje esančiais kintamaisiais. Pagal gautus skaičius, grupės išrikiuojamos mažėjimo tvarka.

Dinaminis kintamųjų rikiavimas. Šis metodas paprastai naudojamas, kai konstruojant binarinę sprendimų diagramą pasiekiama skaičiavimų riba ir reikia bet kokia kaina ją sumažinti. Paprastai naudojamas keitimo vietomis (angl. *swaping*) algoritmas aprašytas [MT98] ir [Rud93]. Jis naudojamas kelis kartus tam tikra tvarka, kol binarinės sprendimų diagramos dydis sumažėja. [NW06] autoriai pastebėjo, kad diagramos dydis paprastai būna mažesnis, kai stipriai susiję kintamieji yra šalia vienas kito. Todėl buvo pasiūlyta minėtą algoritmą naudoti stipriai susijusioms kintamųjų grupėms, o ne atskiriems kintamiesiems. Tokias grupes galima gauti statinio kintamųjų rikiavimo algoritmo pirmajame žingsnyje aprašytu būdu.

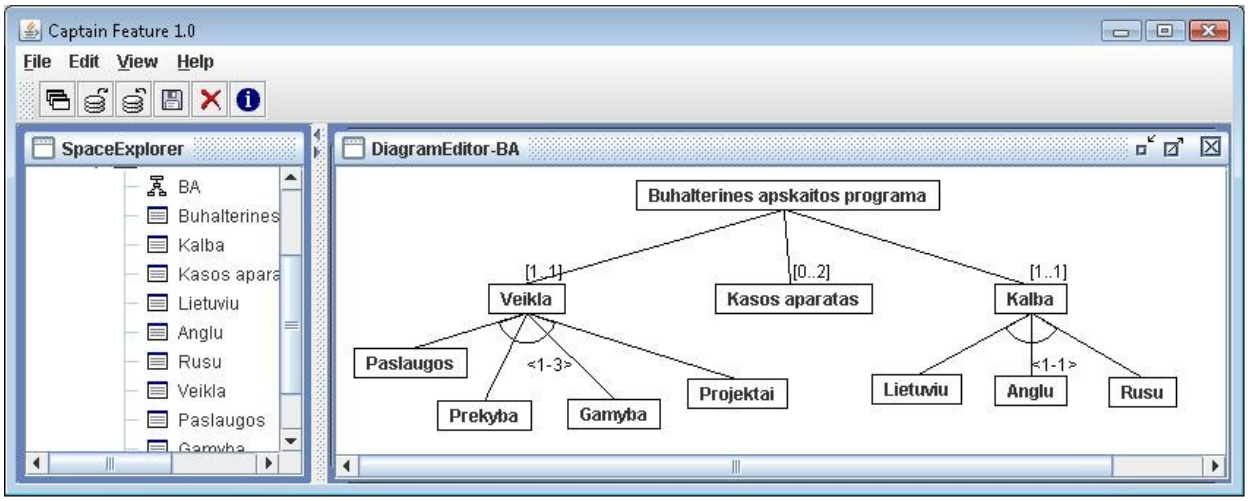
3. Konfigūravimo algoritmas, pagrįstas binarinėmis sprendimų diagramomis

3.1. Ypatybių modeliavimas

Plačiausiai naudojamas būdas modeliuoti kintamybes programinėje įrangoje yra ypatybių modeliai. Ypatybių modeliavimas yra procesas, kuris nagrinėja bendras ir skirtingas sąvokų (angl. *concept*) ypatybes bei jų tarpusavio priklausomybes (angl. *interdependency*) ir sudeda visa tai į vieną ypatybių modelį. Jį paprastai sudaro ypatybių diagrama ir papildoma informacija apie kiekvieną ypatybę bei sąryšius su kitomis ypatybėmis. Ypatybių modelis iš esmės aprašo visą galimybių aibę, t.y. viską ką galima sukurti realizuojant kokį nors poaibį ypatybių, kuris neprieštarautų ypatybių diagramai ir tenkintų tam tikrus apribojimus (angl. *constraints*).

Ypatybių diagramos iš esmės yra medžiai, kuriais sąvokos yra išskaidomos į ypatybes, o ypatybės savo ruožtu skaidomos į sub-ypatybes, t.y. perkeliamos į žemesnį abstrakcijos lygį.

Šiame darbe naudosime T.Bednesch, K.Charnecki, U.W.Eisenecker ir M.Lang sukurtą ypatybių modeliavimo įrankį – „Captain Feature“ versiją - 1.0.



6 pav. „Captain Feature“ grafinė vartotojo sąsaja

„Captain Feature“ yra ypatybių modeliavimo priemonė. Joje naudojamos ypatybių diagramos sudaromos pagal Krzystof Czarnecki ir Ulrich W. Eisenecker sukurtą notaciją bei yra naudojamoa CFCL (Captain Feature Constraint Language) apribojimų užrašymo kalba, sukurta Markus Lang. Tačiau dėl CFCL sudėtingumo bus naudojamas tik jos poaibis, kurį įmanoma atvaizduoti binarinėse sprendimų diagramose.

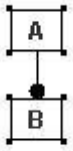
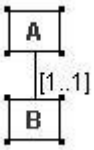
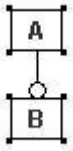
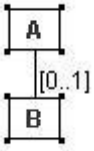
Šis įrankis pasirinktas dėl kelių priežasčių:

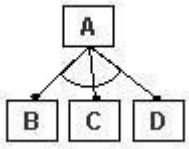
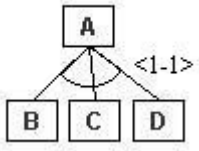
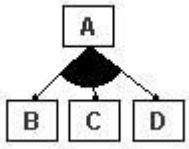
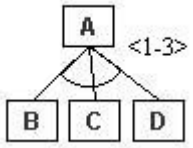
- turi grafinę vartotojo sąsają (6 pav.);
- turi pakankamai plačią dokumentaciją;
- ypatybių modelius saugo patogiu XML formatu (1 priedas.);
- yra atviro kodo ir nemokamas.

3.2. Ypatybių diagramos užrašymas logikos formulėmis

FDL (angl. *Feature Description Language*) aprašyta [DK01] šaltinyje yra viena iš konkretesnių DSL (angl. *Domain Specific Language*), dalykinės srities specifinių kalbų, skirtų ypatybių diagramoms užrašyti tekstiniu formatu. Šis užrašymo būdas pasirinktas, nes aprašius ypatybių diagramą šia kalba labai lengva ją paversti teiginių logikos formule. Tačiau pasirinktas ypatybių modeliavimo įrankis „Captain Feature“ naudoja platesnę ypatybių modelių notaciją nei FDL gali aprašyti. 5 lentelėje pateiktas palyginimas tarp FDL naudojamo ypatybių modeliavimo notacijos ir „Captain Feature“ naudojamos notacijos bei nurodyta, kokie notacijų elementai bus naudojami tolimesniame darbe.

5 lentelė. FDL naudojamos FODA notacijos plėtinio ir „Captain Feature“ naudojamos notacijos palyginimas

Ypatybės	**FDL naudojama ypatybių modelio notacija**	**Captain Feature notacija**	**Naudosime**
Privaloma ypatybė			Taip
Pasirenkama ypatybė			Taip
Privaloma klonuojama ypatybė	Nėra	`[n..m]`, kur `n` > 0 ir `m` >1	Taip
Pasirenkama klonuojama ypatybė	Nėra	`[0..m]`, kur `m` > 1	Taip
Ypatybės atributas	Nėra	f(*value* : T)	Ne

Ypatybės	FDL naudojama ypatybių modelio notacija	Captain Feature notacija	Naudosime
Alternatyvių ypatybių grupė			Taip
Arba ypatybių grupė		 <1-k>, kur k – grupės dydis	Taip
Ypatybių grupė su kardinalumu	Nėra	<i-j>, kur i > 1 ir j < k, o k – grupės dydis	Taip

Iš papildomos notacijos nenaudosime atributų, nes atributai gali būti sudėtingų tipų, todėl jų atvaizdavimas binarinėse sprendimų diagramose užimtų per daug atminties ir būtų mažai reikalingas. Taip pat, Krzystof Czarnecki yra rekomendavęs kuo mažiau naudoti naują notaciją, nes ypatybių klonavimas labai išaugina atminties sąnaudas, ypač aukščiausiuose abstrakcijos lygmenyse. Be to, net pats ypatybių modeliavimo įrankis nelabai supranta begalinį klonavimą, t.y. kai viršutinė riba žymima *. Tokiu atveju, kai modelis yra saugojamas yra įrašomas didžiausias galimas skaičius - 2147483647.

Dar vienas įrankio „Captain Feature“ ypatumas yra tai, kad jis neleidžia nurodyti kardinalumo ypatybėms esančioms grupėje, nors pagal notaciją tai turėtų būti leidžiama. Tačiau tai labai palankus apribojimas, nes tokioje situacijoje apimtis išauga labai greitai.

3.2.1. Papildyta FDL kalbos notacija

6 lentelėje yra pateikta naudojama FDL notacijos dalis bei mums reikalingi papildymai.

6 lentelė. Reikalingų žymėjimų ir FDL notacijos palyginimas

Žymėjimas	Aprašymas	FDL
A	atominė ypatybė	yra
A[n..m]	klonuojama atominė ypatybė	nėra
F	sudėtinė ypatybė (gali būti lygi atominei)	yra
F[n..m]	klonuojama sudėtinė ypatybė	nėra

Žymėjimas	Aprašymas	FDL
`...?`	pasirenkama ypatybė arba jos išraiška	yra
`all(...)`	privalomos ypatybės bei tarpiniuose skaičiavimuose gali būti ir nebūtinos	yra
`one-of(…)`	alternatyvios ypatybės	yra
`more-of(...)`	arba-ypatybės	yra
`<i-j>(...)`	ypatybių grupė su kardinalumu <i-j>	nėra
`F1 requires F2`	Jei pasirinkta F1, turi būti pasirinkta ir F2	yra
`F1 excludes F2`	Jei pasirinkta F1, negali būti pasirinkta F2	yra
`:`	Žymi apibrėžimą, t.y. kairėje pusėje rašomas sudėtinės ypatybės pavadinimas, o dešinėje išraiška atitinkanti tą ypatybę	yra

Prie FDL notacijos pridėti kardinalumai atominėms ir sudėtinėms ypatybėms. Kardinalumus panaikiname pagal formulę:

`F[n..m] = all(F.ind+1,.., F.ind+n, F.ind+n+1?,.., F.ind+m?),`

kur `ind` yra indeksas (gali būti visam modeliui arba tik ypatybei). `F.ind` sub-ypatybės irgi turi būti su indeksais. Be to, turi būti pridėti apribojimai:

`F.ind+n+2 requires F.ind+n+1,`

`F.ind+n+3 requires F.ind+n+2 ir t.t.`

Iš esmės su šia formule tiesiog nuklonavome ypatybę, kiekvienam jos klonui (ir jo sub-ypatybėms) suteikdami naują vardą. Apribojimai pridėti galimų konfigūracijų skaičiui sumažinti, nes be apribojimų, kurį ypatybės kloną bepasirinktume gautume naują konfigūraciją, o mums svarbus tik pasirinktų ypatybių kiekis.

3.2.2. FDL pavertimas į teiginių logikos formules

BDD bibliotekos palaiko ne tik binarinę operaciją jei/taip/kitaip, bet ir kitas dažniausiai naudojamas binarines operacijas, mūsų atveju naudosime: `ite` (if/then/else) – jei/taip/kitaip, `and` – konjunkcija, `imp` – implikacija, `or` - disjunkcija ir `not` - neigimas. Be to, net pats bibliotekos autorius nerekomenduoja sudarinėti binarinių sprendimų diagramų naudojantis tik `jei/taip/kitaip` operaciją. Dėl šių priežasčių mums užtenka ypatybių modelį suvesti į binarines logikos formules būtent su minėtomis operacijomis.

7 lentelė. FDL vertimas logikos formulėmis

Nr.	FDL notacija	Logikos formulės
1.	more-of(F1,..,Fk)	F1 or .. or Fk
2.	all(F1,..,Fk)	F1 and .. and Fk
3.	F:all(F1?,..,Fi?,Fj, ..,Fk)?	More-of(F1?,.., Fk) imp all(Fj, .., Fk)
4.	A?	true
5.	one-of(F1,..,Fk)	if F1 then not more-of(F2,..,Fk) else one-of(F2,..,Fk)
6.	<i-j>(F1,.., Fk)	if F1 then <min(0, i-1)-(j-1)>(F2,..,Fk) else <i-min(j,k-1)>(F2, .., Fk)

7 lentelėje pavaizduotas FDL notacijos suvedimas į teiginių logikos formules:

1. Jei reikia pasirinkti arba-ypatybes (grupę šiuo atveju laikome viena ypatybe), tai naudojame logine funkciją `or`.

2. Jei reikia pasirinkti visas sub-patybes (grupę šiuo atveju laikome viena sub-ypatybe), tai naudojame logine funkciją `and`.

3. Jei turime pasirenkamą ne atominę ypatybę, tai ji pati nedaro įtakos konfigūracijai. Tačiau reikia atsižvelgti į tai, kad pasirinkus bet kurią jos sub-ypatybę reikės pasirinkti ir visas jos privalomas sub-ypatybes.

4. Pasirenkama atominė ypatybė, nuo kurios nepriklauso konfigūracijos teisingumas, t.y. ji nenaudojama kitur.

5. Alternatyviom ypatybėm naudojame `ite` funkciją. Pasirenkame bet kurią ypatybę ir jei ji pasirenkama, tai visos kitos ypatybės negali būti pasirinktos, jei ne kartojame tą patį su likusiomis ypatybėmis. Kaip matome, FDL išraiškai „one-of“ reikalinga rekursija, kuri baigtųsi gavus tapatybę:

`one-of(F) = F.`

6. Darome labai panašiai kaip ketvirtuoju atveju. Tik pasirinkus ypatybę pamažiname kardinalumo ribas. Ypatybių grupei su kardinalumu, taip pat, turi būti taikoma rekursija. 8 lentelėje surašytos taisyklės, kuriomis užbaigiama rekursija.

8 lentelė. Suvedimo į FDL notaciją taisyklės

Aprašymas	Formulė
T1. Gaunama iš alternatyvių ypatybių grupės apibrėžimo.	`<1-1>(F1, .., Fk) = one-of(F1, .., Fk)`
T2. Gaunama iš arba-ypatybių grupės apibrėžimo.	`<1-k>(F1,..,Fk) = more-of(F1, ..,Fk)`
T3. Jei iš grupės reikia pasirinkti visas ypatybes, tai traktuojame, kad visos ypatybės yra privalomos.	`<k-k>(F1, ..,Fk) = all(F1, .., Fk)`
T4. Iš grupės negali būti pasirinkta nei viena ypatybe.	`<0-0>(F1, .., Fk) = not more-of(F1, .., Fk)`
T5. Pasirinkus bet ką (arba nieko) bus teisinga.	`<0-k>(F1, .., Fk) = true`

Praktiškai yra geriau turėti atskiras logikos formules kiekvienai sudėtinei ypatybei, nes binarinė sprendimų diagrama bus formuojama tik po vieną operaciją, tai nėra jokio privalumo turėti vieną formulę, kuri aprašytų visa ypatybių diagramą. Be to, nebūtinas atomines ypatybes galime išmesti iš logikos formulės, nes jos nedaro įtakos konfigūracijos teisingumui, jei nėra panaudotos kur nors kitur, pavyzdžiui, priklausomybėse.

Apribojimo taisyklės

Turint loginėmis formomis aprašytą ypatybių diagramą, prie jos dar reikia pridėti priklausomybes. Šiame darbe naudosime tik dvi priklausomybių rūšis:

- Jei ypatybė A pašalina (angl. *excludes*) ypatybę B, tai jei B priklauso konfigūracijai ir A priklauso tai pačiai konfigūracijai, tai konfigūracija yra neteisinga. Visais kitais atvejais konfigūracija teisinga. Šią priklausomybę galime užrašyti kaip - `not (A and B)`.
- Jei ypatybė A reikalauja (angl. *requires*) ypatybės B, tai jei A priklauso konfigūracijai, o B – nepriklauso, tai konfigūracija yra neteisinga. Visais kitais atvejais konfigūracija yra teisinga. Šią priklausomybę galime užrašyti kaip - `A imp B`.

Priklausomybės pridedamos prie ypatybių diagramos pasinaudojant logine operacija „and“.

Pagal ypatybių diagramos apibrėžimą, konfigūracija sudaroma iš privalomų ypatybių ir atitinkamai pasirenkamų ypatybių. Geros konfigūracijos gaunamos einant per visą diagramą nuo šakninės ypatybės ir pasirenkant visas privalomas ypatybes bei norimas neprivalomas ypatybes pagal taisykles:

- Diagramos šakninė ypatybė, visada įtraukiama į aprašą konfigūraciją.

- Privalomos ypatybės įtraukiamos į konfigūraciją tada ir tik tada, kai tėvinė ypatybė yra įtraukta į konfigūraciją. Bet jei pavyzdžiui tėvinė ypatybė yra nebūtina ir yra nepasirinkta, tai ypatybė negali būti įtraukta į konfigūraciją.
- Nebūtinos ypatybės į konfigūraciją gali būti įtrauktos tik jei tėvinė ypatybė yra įtraukta. Žinoma, jei tėvinė ypatybė yra įtraukta, tai nebūtina ypatybė gali būti ir neįtraukiama.
- Alternatyvios ypatybės gali būti įtrauktos į aprašą, lygiai viena iš kiekvienos grupės ir tik jei tėvinė ypatybė yra pasirinkta.
- Arba-ypatybės gali būti įtrauktos viena arba daugiau, jei pasirinkta tėvinė ypatybė. Jei tėvinė ypatybė nėra pasirinkta, tai arba-ypatybės nėra traukiamos į konfigūraciją.

Pasinaudojant šiomis taisyklėmis galime pastebėti, kad jei arba sub-ypatybė yra pasirinkta, tai turėjo būti pasirinktos ir visos tėvinės ypatybės. Kadangi konfigūracija vyksta tik atominių ypatybių lygmenyje, tai pagal šitą pastebėjimą priklausomybes galima rašyti ir sudėtinėms ypatybėms, o po to jas pervesti visoms jų vaikinėms atominėms ypatybėms.

Jei a1 ir a2 yra tiesioginės arba netiesioginės ypatybės A sub-ypatybės, o b1 ir b2 yra tiesioginės arba netiesioginės ypatybės B sub-ypatybės, tai apribojimą:

```
A requires B A imp B
```

galima užrašyti taip –

```
(a1 or a2) imp (b1 or b2).
```

Apribojimas `excludes` aprašomas analogiškai.

Papildykime mūsų ankstesniuose skyriuose nagrinėtą pavyzdį reikalavimu:

```
rusu excludes kasos_aparatas,
```

t.y. jei buhalterines apskaitos programos pasirinkta kalba yra rusu, tai prijungti kasos aparato negalima, nes kasos aparatas rusu kalbos nepalaiko. Čia pateikta priklausomybė pagal FDL notaciją, tačiau „Captain Feature“ ji apsirašo labai panašiai, t.y. vietoj ypatybių rašomi jų indeksai. O taip pat turime priklausomybę atsiradusią kardinalumo pašalinimo metu:

```
kasos_aparatas.2 requires kasos_aparatas.1.
```

3.3. Konfigūravimo procesas

Šiame skyriuje aprašysime antrąją fazę bandydami atsakyti į įvade iškeltus konfigūravimo klausimus.

Ar duotai programų sistemų šeimai (sumodeliuotai ypatybių diagrama) egzistuoja bent viena teisinga konfigūracija?

Tai yra svarbiausias klausimas, be atsakymo į jį nežinosime ar kiti klausimai turi prasmę. Su pasirinkta BDD biblioteka į jį atsakymą galime gauti pasinaudodami metodu `satCount`. Pavyzdžiui: `bdd.satCount(diagrama) > 0`.

Konfigūravimo interaktyviojoje fazėje, turint aibę vartotojo pasirinktų variantų, svarbu atsakyti ir į šiuos klausimus:

Ar jau suformuota konfigūracija yra teisinga, t.y., ar yra pasirinktos visos būtinos ypatybės, ir ar nėra pažeistos priklausomybės tarp variantų?

Į šį klausimą galime atsakyti pasinaudoję tuo pačiu metodu `satCount`. Taip pat gavus, kad liko tik viena teisinga konfigūracija su metodu `oneSat` galime gauti ir tą vienintelę gerą konfigūraciją su padarytais pasirinkimais. Ypatybių pasirinkimui realizuoti skirtas metodas `restrict`. Ypatybės atmetimui metodą teks realizuoti, nes JDD biblioteka tokio neturi, tačiau galima perrašyti būtent `restrict` metodą, nes jie yra analogiški. Pasirenkant ypatybę turi būti imamas aukštesnysis vaikas, o atmetant žemesnysis.

Jei konfigūracija dar nėra teisinga, tai:

Kokios priklausomybės buvo pažeistos?

Į šitą klausimą galima ir neatsakinėti, jei pasirinksime, kad neleisime vartotojui suklysti, t.y. po kiekvieno pasirinkimo už vartotoją pasirinksime arba atmesime ypatybes, kurios po pasirinkimo taps atitinkama būtinos (atmetus nebeliks teisingų konfigūracijų) arba negalimos (pasirinkus neliks teisingų konfigūracijų).

Kiek ir kokių pasirinkimų dar reikia atlikti vartotojui, kad gautų teisingą konfigūraciją?

Savaime aiškų jau atmestų arba pasirinktų ypatybių rinktis nebereikia. Ypatybes kurių nėra diagramoje reikia pasirinkti, nes nuo jų nepriklauso konfigūracijos teisingumas. Ypatybes, kurios yra diagramoje reikia pasirinkti tik tas, kurių žemesnysis arba aukštesnysis vaikas nerodo į 0. Reikia atsiminti, kad ypatybė gali turėti kelis žemesniuosius ir kelis aukštesniuosius vaikus. Šiuo atveju, jie turi sutapti. Be to, gali egzistuoti ir kelias be atitinkamos ypatybės iki mazgo 1.

6pav. pavaizduotos ypatybių diagramos užrašas papildyta FDL notacija būtų toks:

```
Buhalterines apskaitos programa : all(Veikla, kasos_aparatas[0..2], Kalba)
Veikla : <1-3>(paslaugos, prekyba, gamyba, projektai)
Kalba : one-of(rusu, lietuviu, anglu).
```

Panaikinę ypatybių kardinalumus gausime:

```
Buhalterines apskaitos programa : all(Veikla, all(kasos_aparatas.1?, kasos aparatas.2?), Kalba)
Veikla : <1-3>(paslaugos, prekyba, gamyba, projektai)
Kalba : one-of(rusu, lietuviu, anglu).
```

Be to gausime priklausomybes:

```
rusu excludes kasos_aparatas;
kasos_aparatas.2 requires kasos_aparatas.1.
```

Pasinaudoję 3.2.2. punkte aprašytais keitimais gautume, kad formulę:

```
if (paslaugos)
    then (if (prekyba)
        then (if(gamyba)
            then not (projektai)
            else (true))
        else (true))
    else (prekyba or gamyba or projektai)
and if (rusu)
     then (not (lietuviu or anglu))
     else (if (lietuviu) then (not anglu) else (anglu))
and not (rusu and (kasos_aparatas.1 or kasos_aparatas.2))
and (kasos_aparatas.2 imp kasos_aparatas.1).
```

kuri atitinka mūsų sukurtą ypatybių modelį.

Programos kodas, kuris sukurtų šia binarinę sprendimų diagramą atrodytų taip:

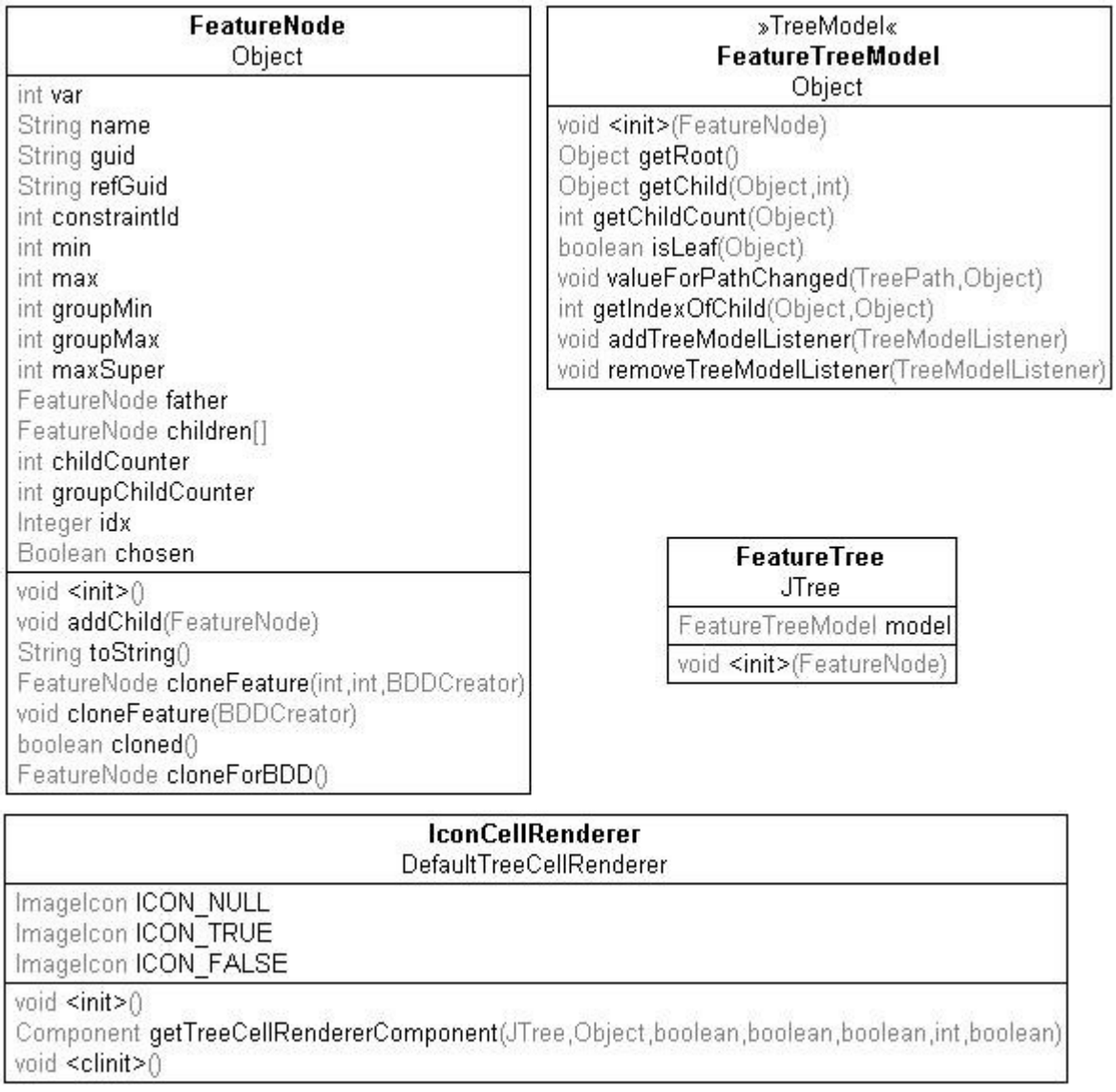
```
//inicijuojamas objektas su kurio pagalba konstruosime binarinę sprendimų diagramą
jdd.bdd.BDD bdd = new jdd.bdd.BDD(100);
//pavyzdžiui turėtų užtekti 100 mazgų
//susikuriam kintamuosius atitinkančius atomines ypatybes
int paslaugos = bdd.createVar();
```

```
    int prekyba = bdd.createVar();
    int gamyba = bdd.createVar();
    int projektai = bdd.createVar();
    int rusu = bdd.createVar();
    int lietuviu = bdd.createVar();
    int anglu = bdd.createVar();
    //kardinalumą vaizduojam keliais kintamaisiais
    int kasos_aparatas1 = bdd.createVar();
    int kasos_aparatas2 = bdd.createVar();
    //konstruojame binarinę sprendimų diagramą pradedami nuo žemiausio lygio
    int veikla = bdd.ref(bdd.ite(paslaugos,
       bdd.ref(bdd.ite(prekyba,
          bdd.ref(bdd.ite(gamyba,
             bdd.ref(bdd.not(projektai)),
             bdd.getOne())),
          bdd.getOne())),
       bdd.ref(bdd.or(bdd.ref(bdd.or(prekyba, gamyba)),
projektai))));
    int kalba = bdd.ref(bdd.ite(rusu,
       bdd.ref(bdd.not(bdd.ref(bdd.or(lietuviu, anglu)))),
       bdd.ref(bdd.ite(lietuviu,
          bdd.ref(bdd.not(anglu)),
          anglu))));
    //pasiekiamas aukščiausias lygis
    int buhalterines_apskaitos_programa = bdd.ref(bdd.and(veikla,
kalba));
    //priklausomybes
    int p1 = bdd.ref(bdd.not(bdd.ref(bdd.and(rusu,
bdd.ref(bdd.or(kasos_aparatas1, kasos_aparatas2))))));
    int p2 = bdd.ref(bdd.imp(kasos_aparatas1, kasos_aparatas2));
    int priklausomybes = bdd.ref(bdd.and(p1,p2));
    //visa binarinė sprendimų diagrama (3 priedas.)
     int diagrama =
     bdd.ref(bdd.and(buhalterines_apskaitos_programa,
     priklausomybes));
```

5. Prototipas

5.1. Ypatybių atvaizdavimas

Ypatybių modeliui atvaizduoti konfigūratoriaus prototipe buvo panaudotas grafinės sąsajos elementas `JTree`. Šis elementas ypatybių modelio vaizdavimui buvo pasirinktas dėl savo hierarchinės struktūros, kuri puikiai tinka atvaizduoti ypatybių diagramas.



7 pav. Ypatybių modelio atvaizdavimo JTree papildomos klasės

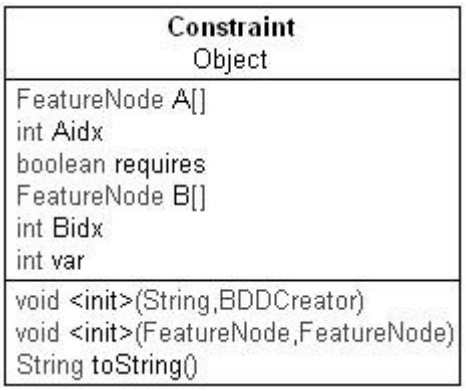
Buvo sukurtos keturios klasės (7 pav.):

`FeatureNode`. Klasė skirta ypatybės mazgo informacijai saugoti. Kintamasis `var` skirtas binarinės sprendimų diagramos kintamajam arba po medžiui saugoti. Turi nuorodas į tėvinę ypatybę bei visas sub-ypatybes. Gali save klonuoti, kartu klonuojant ir sub-ypatybes iki žemiausio lygio. Taip pat skirta saugoti grupių informacijai, todėl turi grupių klonavimą skirtą alternatyvių ir grupių su kardinalumų BDD konstravimui.

`FeatureTreeModel`. Sąsaja būtina automatiniam medžio piešimui.

`FeatureTree`. Medžio klasė.

`IconCellRenderer`. Klasė skirta skirtumams tarp pasirinktos, atmestos ir neaiškios ypatybės atvaizduoti.



8 pav. Apribojimų klasė

Apribojimai saugomi 8 pav. pavaizduotos klasės objektuose. Kiekvienas apribojimas saugo nuorodas į naudojamus ypatybių mazgus. Masyvai naudojami, nes apribojimas gali būti ne tik ypatybėms be sub-ypatybių, bet ir visoms tiesioginėms ir netiesioginėms ypatybės sub-ypatybėms.

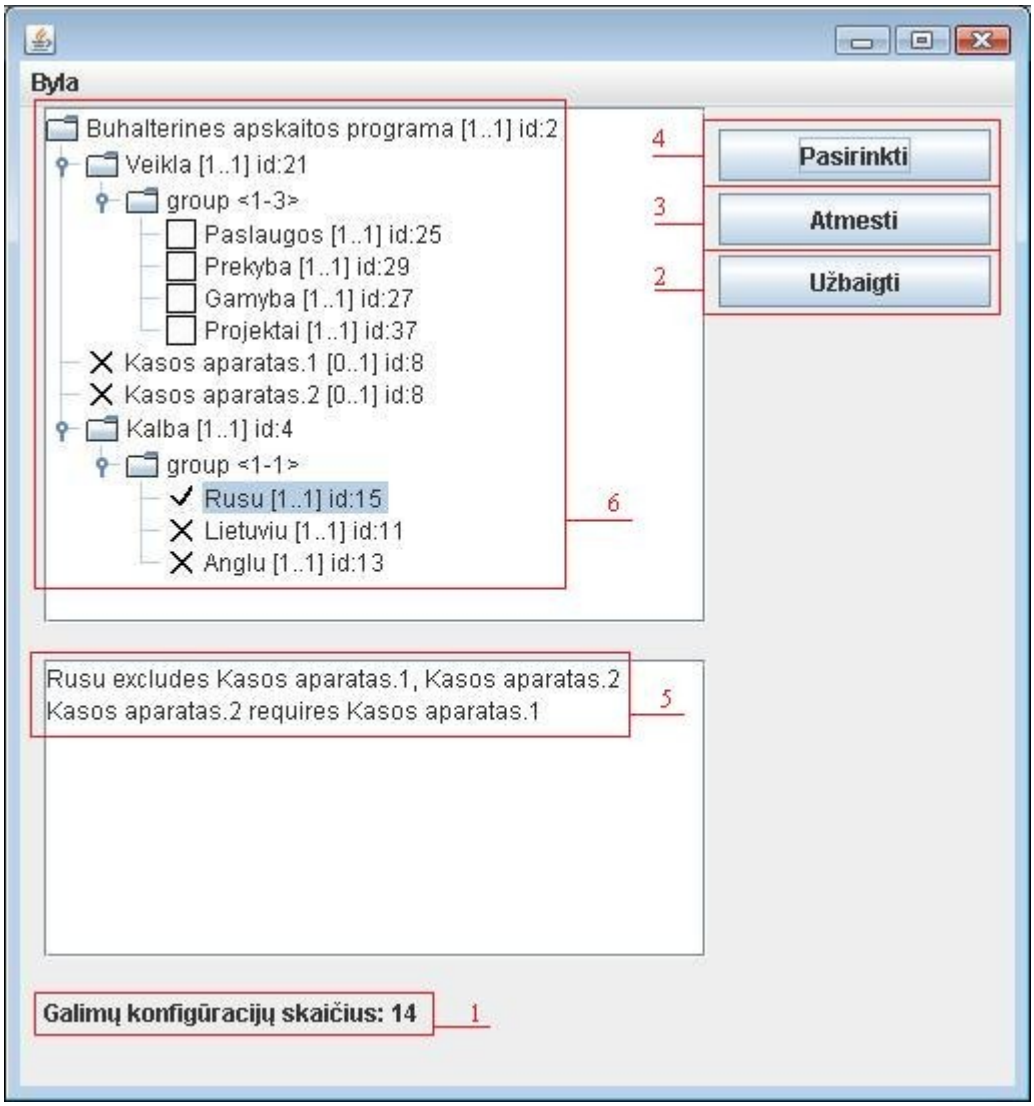
9 pav. 6 punktu parodytas gaunamas ypatybių medis, o 5 punktu ypatybių apribojimai. Su apribojimais jokių veiksmų atlikti negalima, o su ypatybėmis galima atlikti veiksmus aprašytus 3.3. poskyryje.

5.2. Binarinės sprendimų diagramos sudarymas

Pirmiausiai, prieš kuriant ypatybių modelį atitinkančią binarinę sprendimų diagramą, yra apibrėžiami binariniai kintamieji atitinkantys atomines ypatybes. Ir tik tada pradedama konstruoti pati binarinė sprendimų diagrama. Konstravimas pradedamas nuo šakninio ypatybių mazgo. Užsukama rekursija, kad pasiektume atomines ypatybes, o iš jų padarytume vis kitas sudėtines ypatybes, kol vėl grįšime į šakninį mazgą ir turėsime ypatybių diagramą atitinkančią binarinę sprendimų diagramą. Diagrama konstruojama pagal 4 ir 5 lentelėse esančias formules. Visas prototipo kodas yra 4 priede.

Binarinė sprendimų diagrama konstruojama pagal ypatybių diagramą turėtų gautis pakankamai nedidelė, nes ypatybių diagrama iš esmės yra ypatybių apribojimai aprašyti hierarchiškai. Todėl pats jos konstravimas iš dalies atitinka statinę kintamųjų rikiavimo euristiką, tik ypatybių klasterių išrikiavimas nėra optimizuotas, bei rikiuojant ypatybes neatsižvelgiama į ne ypatybių diagramoje esančius apribojimus, kurie prototipe pridedami tik jau sukonstravus binarinę sprendimų diagramą pagal ypatybių diagramą. Jie pridedami pasinaudojus operacija `and`.

5.3. Konfigūravimo galimybės



9 pav. Prototipo grafinė vartotojo sąsaja

Visą laiką matomas likusių konfigūracijų skaičius (1). Naudojamas `satCount` metodas, kurio sudėtingumas - O(n), t.y. einama per binarinę sprendimų diagramą ir skaičiuojami keliai vedantys į mazgą 1.

Galimas pilnas užpildymas bet kokia konfigūracija (2). Naudojamas `oneSat` metodas, kurio sudėtingumas - O(n). Taip pat einama per binarinę sprendimų diagramą, tik kol bus rastas pirmas kelias nuo šakninio mazgo iki 1 mazgo.

Ypatybės atmetimas (3). Realizuota pasinaudojant `discard` metodu, kuris yra identiškas `restrict`, todėl jo sudėtingumas – O(n). Atitinamai `and` ir `not` metodai yra O(n) sudėtingumo, todėl ypatybės atmetimas yra O(3n) sudėtingumo.

Ypatybės pasirinkimas (4). Realizuota pasinaudojant `restrict` metodu ir `and` operacija. `restrict` metodo sudėtingumas - O(n). `and` operacijos sudėtingumas – O(|u1|*|u2|) ~ $O(n^2)$, kur |u1| ir |u2| yra sudedamu binarinių sprendimų diagramų mazgų skaičiai. Tačiau šiuo atveju pridedamas tik vienas kintamasis, t.y. |u2| galime laikyti lygiu 1. Todėl bendras ypatybės pasirinkimo sudėtingumas – O(2n).

Automatinis modelio atnaujinimas pagal priklausomybes. Vykdomas sukonstravus binarinę sprendimų diagramą bei kiekvieną kartą pasirenkant arba atmetant ypatybę. Automatinis modelio atnaujinimas vykdomas einant per binarinę sprendimų diagrama ir tikrinant ar pasirinkimo šakos yra lygios 0 mazgui. Perėjimo per visus mazgus sudėtingumas - O(n). Perėjus per visus mazgus reikia atitinkamai atmesti arba pasirinkti ypatybes. Šis žingsnis kartojamas kol nebelieka atmetamų arba pasirenkamų ypatybių, blogiausiu atveju tiek kartų, kiek yra ypatybių (N), todėl šio funkcionalumo sudėtingumas - O(Nn) blogiausiu atveju. Toks sudėtingumas prie didelių binarinių sprendimų diagramų bus daug mažesnis už sudėtingumą – $O(n^2)$.

9 lentelė. Prototipo funkcijų sudėtingumai

Prototipo funkcijos	**Sudėtingumas**
Likusių galimų konfigūracijų skaičius	O(n)
Galimas pilnas užpildymas bet kokia konfigūracija	O(n)
Ypatybės pasirinkimas	O(n)
Ypatybės atmetimas	O(n)
Automatinis modelio atnaujinimas pagal priklausomybes	O(Nn) ~ O(n)

Kaip matyti iš 9 lentelės , sukurtas konfigūratoriaus prototipas yra pakankamai efektyvus, t.y. visos jo funkcijos yra įvykdomos per polinominį laiką. Funkcijų vykdymo laikas tiesiogiai priklauso nuo binarinės sprendimų diagramos mazgų skaičiaus. Prototipas naudojasi ypatybių modeliu, kuris yra medžio struktūros, t.y. beveik visi apribojimai yra išreikšti medyje (atskirai pridėtų apribojimų turėtų būti palyginus nedaug). Pagal tokią struktūrą konstruojant binarinę sprendimų diagramą visada bus vadovaujamasi pagrindiniu kintamųjų rikiavimo principu, kad susiję kintamieji būtų vienas šalia kito. Todėl su prototipu gautos binarinės sprendimų diagramos

mazgų skaičius turėtų būti pakankamai mažas, t.y. norint optimizuoti sukurta prototipą reikėtų pilnai įgyvendinti 2.4. poskyryje pateiktas kintamųjų rikiavimo euristikas.

Rezultatai ir išvados

Šiame darbe buvo pasiekti tokie rezultatai:

- Sukurtas ir aprašytas efektyvus programinės įrangos su kintamybėmis konfigūravimo algoritmas, pagrįstas binarinėmis sprendimų diagramomis.
- Pateiktas atskirų konfigūravimo algoritmo dalių efektyvumo įvertinimas (9 lentelė). Visų dalių sudėtingumas yra O(n), kur n – binarinės sprendimų diagramos mazgų skaičius.
- Sukurtas konfigūravimo priemonės prototipas.

Iš pasiektų rezultatų galime daryti išvadą, kad yra įmanoma efektyvi interaktyvi konfigūracija.

Šaltinių sąrašas

[AHH+04] H. R. Andersen, T. Hadzic, H. Hulgaard, R. M. Jensen, J. Moller, S. Subbarayan. Fast backtrack-free product configuration using a precompiled solution space representation. Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems, psl. 131-138, 2004.

[And97] H. R. Andersen. An Introduction to Binary Decision Diagrams. 1997.

[Bec02] M. Becker. XML-Enhanced Product Family Engineering. Integrated Design and Process Technology, 2002.

[Bry86] R. Bryant: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers. Vol. C-35, No. 8, 677 -691, 1986.

[BW96] B. Bollig, I. Wegener. Improving the Variable Ordering of OBDDs Is NP-Complete. IEEE Transactions on Computers, 45(9):993—1002, September 1996.

[CHK+90] S. G. Cohen, J. A. Hess, K. C. Kang, W.E. Novak, S. A. Peterson. Feature-Oriented Domain Analysis (FODA) - Feasibility Study. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University 1990.

[CK05] K. Czarnecki, Ch. H. P. Kim. Cardinality-Based Feature Modeling and Constraints - A Progress Report. University of Waterloo, 2005.

[CW07] K. Czarnecki, A. Wasowski. Feature Diagrams and Logics: There and Back Again. 2007.

[Cza98] K. Czarnecki. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models. PhThesis, Technical University of Ilmenau, Ilmanau, Germany 1998.

[DK01] A. van Deursen, P. Klint. Domain-Specific Language Design Requires Feature Descriptions. Journal of Computing and Information Technology, 2001.

[Lin07] J. Lind-Nielsen. BuDDy: A BDD package. 2007.
[žiūrėta 2009-06-15]. Prieiga per internetą:
http://buddy.sourceforge.net/manual/main.html

[MT98] Ch. Meinel, Th. Theobald. Algorithms and Data Structures in VLSI Design - OBDD - Foundations and Applications. Springer-Verlag, Berlin, 1998.

[Nor99] S. Norgėla. Matematinė logika. 1999.

[NW06] N. Narodytska, T. Walsh. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. Proc. 17th European Conf. Artificial Intelligence, Workshop on Configuration, 2006, pp. 2–7.

[PRS03] I. Philippow, M. Riebisch, D. Streitferdt. Details of Formalized Relations in Feature Models Using OCL. Proceedings of the 10 th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003.

[Rud93] R. Rudel. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. 1993.

[Som09] F. Somenzi. CUDD: CU Decision Diagram Package Release 2.4.2.. University of Colorado at Boulder, 2009.
[žiūrėta 2009-06-15]. Prieiga per internetą:
http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html

[Vah09] A. Vahidi. The JDD project. 2009.
[žiūrėta 2009-06-15]. Prieiga per internetą:
http://javaddlib.sourceforge.net/jdd/index.html

Priedai

1 priedas. „Captain Feature“ ypatybių modelių saugojimo DTD byla

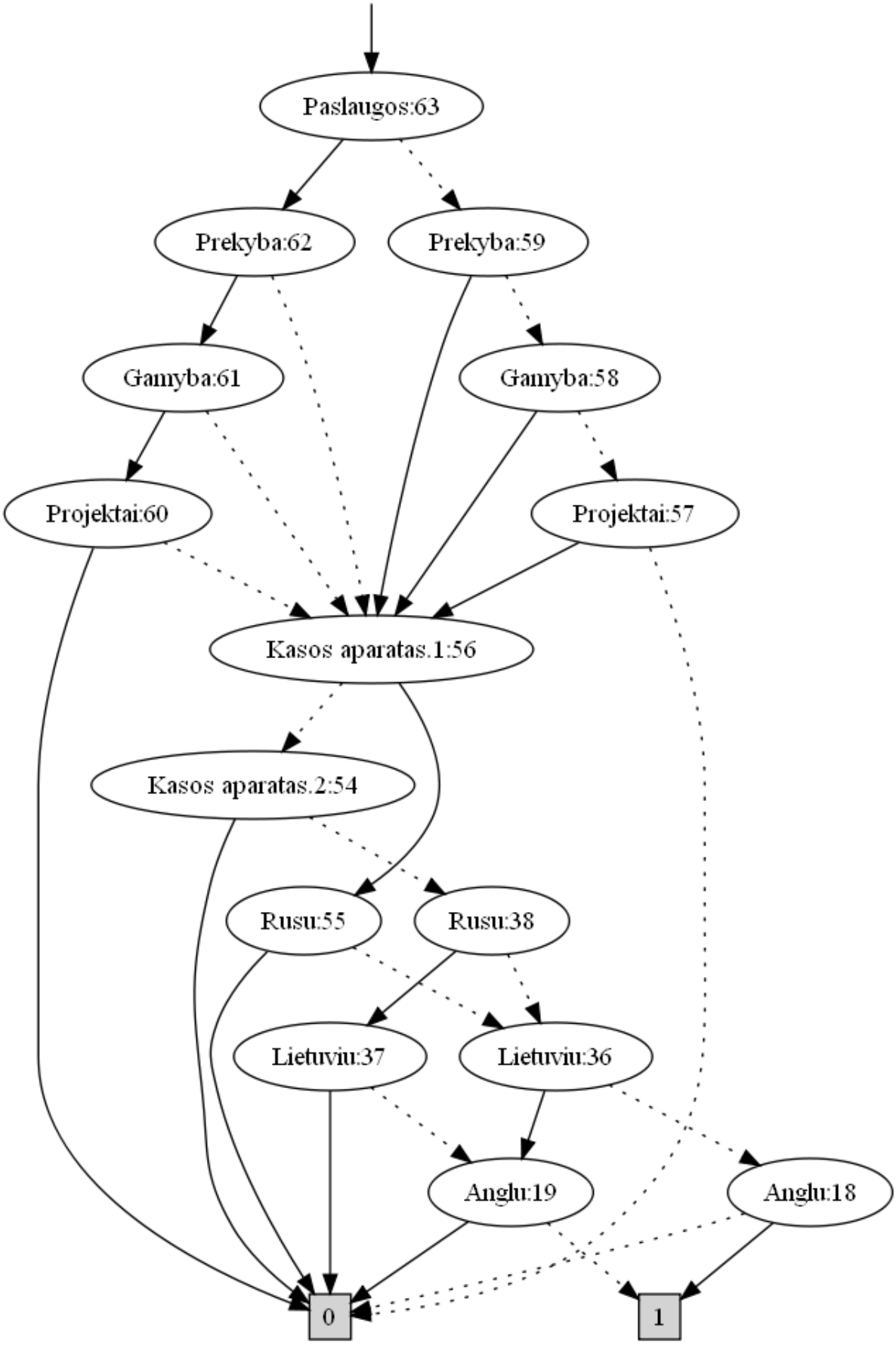
```
<!ELEMENT CFRepository (ModelSpace*)>
<!ELEMENT ModelSpace (MetaLevel,ModelLevel)>
<!ELEMENT MetaLevel (FeatureSpace*, Feature*, Diagram*, MetaFeature, MetaFeatureNode, MetaCardinality,
MetaGroup)>
<!ELEMENT MetaFeature (FeaturePlugIn*)>
<!ELEMENT FeaturePlugIn (Cardinality)>
<!ELEMENT MetaFeatureNode EMPTY>
<!ELEMENT FeatureNodePlugIn (Cardinality)>
<!ELEMENT MetaCardinality (CardinalityPlugIn*)>
<!ELEMENT CardinalityPlugIn (Cardinality)>
<!ELEMENT MetaGroup (GroupPlugIn*)>
<!ELEMENT GroupPlugIn (Cardinality)>
<!ELEMENT ModelLevel (FeatureSpace*, Feature*, Diagram*)>
<!ELEMENT FeatureSpace (FeatureSpace*, Feature*, Diagram*)>
<!ELEMENT Feature (SpecializationParameter?, PlugInRef*)>
<!ELEMENT PlugInRef EMPTY>
<!ELEMENT Diagram (FeatureNode?, FreeNodes?, Constraints?, Specialization*)>
<!ELEMENT FreeNodes (FeatureNode+)>
<!ELEMENT FeatureNode (SpecializationParameter?, (Group | Cardinality)*)>
<!ELEMENT Group (Limit, FeatureNode+, PlugInRef*)>
<!ELEMENT Cardinality (Limit+, FeatureNode, PlugInRef*)>
<!ELEMENT Limit EMPTY>
<!ELEMENT Specialization (FeatureNode)>
<!ELEMENT Constraints (Constraint*)>
<!ELEMENT Constraint EMPTY>
<!ELEMENT SpecializationParameter EMPTY>
<!ATTLIST CFRepository name CDATA #REQUIRED version CDATA #REQUIRED idCounter CDATA
#REQUIRED guid ID #REQUIRED>
<!ATTLIST ModelSpace name CDATA #REQUIRED guid ID #REQUIRED>
<!ATTLIST MetaLevel name CDATA #REQUIRED guid ID #REQUIRED>
<!ATTLIST MetaFeature guid ID #REQUIRED>
<!ATTLIST MetaFeatureNode guid ID #REQUIRED>
<!ATTLIST MetaCardinality guid ID #REQUIRED>
<!ATTLIST MetaGroup guid ID #REQUIRED>
<!ATTLIST ModelLevel name CDATA #REQUIRED guid ID #REQUIRED>
<!ATTLIST FeatureSpace name CDATA #REQUIRED guid ID #REQUIRED>
<!ATTLIST Feature name CDATA #REQUIRED description CDATA #REQUIRED premature (true|false)
#REQUIRED guid ID #REQUIRED constraintId CDATA #IMPLIED>
<!ATTLIST PlugInRef guidReference IDREF #REQUIRED>
<!ATTLIST FeaturePlugIn plugPosition (feature|name|description|premature|spectype|int|bool|string|float|pointer)
#REQUIRED>
<!ATTLIST CardinalityPlugIn plugPosition (cardinality|limit|minimum|maximum|featurenode) #REQUIRED>
<!ATTLIST GroupPlugIn plugPosition (group|limit|minimum|maximum|cardinality) #REQUIRED>
<!ATTLIST Diagram guid ID #REQUIRED name CDATA #REQUIRED xpos CDATA #REQUIRED ypos
CDATA #REQUIRED width CDATA #REQUIRED height CDATA #REQUIRED>
<!ATTLIST Specialization guid ID #REQUIRED name CDATA #REQUIRED xpos CDATA #REQUIRED ypos
CDATA #REQUIRED width CDATA #REQUIRED height CDATA #REQUIRED>
<!ATTLIST FeatureNode guid ID #REQUIRED refFeatureName CDATA #REQUIRED refGuidFeatureTree
IDREF #REQUIRED xpos CDATA #REQUIRED ypos CDATA #REQUIRED constraintId CDATA #IMPLIED
behaviour CDATA #IMPLIED sourceGuid CDATA #IMPLIED>
<!ATTLIST Cardinality guid ID #REQUIRED constraintId CDATA #IMPLIED sourceGuid CDATA #IMPLIED>
<!ATTLIST Group guid ID #REQUIRED constraintId CDATA #IMPLIED sourceGuid CDATA #IMPLIED>
<!ATTLIST Limit min CDATA #REQUIRED max CDATA #REQUIRED>
<!ATTLIST SpecializationParameter type CDATA #REQUIRED value CDATA #IMPLIED>
<!ATTLIST Constraint string CDATA #REQUIRED>
```

2 priedas. „Captain Feature“ išsaugota pavyzdžio byla

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CFRepository SYSTEM "cfrepository1_0.dtd">
<CFRepository name="BA" version="1.0" idCounter="43" guid="ID_7F00000100000128C4953F6D00000001">
 <ModelSpace name="Model" guid="ID_7F00000100000128C4953F8C00000002">
  <MetaLevel name="META" guid="ID_7F00000100000128C4953FEA00000003">
   <MetaFeature guid="ID_7F00000100000128C495405700000004">
   </MetaFeature>
   <MetaFeatureNode guid="ID_7F00000100000128C495415000000021"/>
   <MetaCardinality guid="ID_7F00000100000128C49541600000002F">
   </MetaCardinality>
   <MetaGroup guid="ID_7F00000100000128C49541700000003E">
   </MetaGroup>
  </MetaLevel>
  <ModelLevel name="MODEL" guid="ID_7F00000100000128C495417F0000004D">
   <FeatureSpace name="FS" guid="ID_7F00000100000128C49561EA0000004E">
    <Feature name="Buhalterines apskaitos programa" description="" premature="false"
guid="ID_7F00000100000128C49761BB00000050" constraintId="1">
   </Feature>
   <Feature name="Kalba" description="" premature="false" guid="ID_7F00000100000128C498533A00000052"
constraintId="3">
   </Feature>
   <Feature name="Kasos aparatas" description="" premature="false"
guid="ID_7F00000100000128C49932DE00000056" constraintId="7">
   </Feature>
   <Feature name="Lietuviu" description="" premature="false" guid="ID_7F00000100000128C499B80400000059"
constraintId="10">
   </Feature>
   <Feature name="Anglu" description="" premature="false" guid="ID_7F00000100000128C499EAF50000005B"
constraintId="12">
   </Feature>
   <Feature name="Rusu" description="" premature="false" guid="ID_7F00000100000128C49A16E50000005D"
constraintId="14">
   </Feature>
   <Feature name="Veikla" description="" premature="false" guid="ID_7F00000100000128C49EFCCB00000063"
constraintId="20">
   </Feature>
   <Feature name="Paslaugos" description="" premature="false"
guid="ID_7F00000100000128C49FA0B700000067" constraintId="24">
   </Feature>
   <Feature name="Gamyba" description="" premature="false"
guid="ID_7F00000100000128C49FE7E400000069" constraintId="26">
   </Feature>
   <Feature name="Prekyba" description="" premature="false"
guid="ID_7F00000100000128C4A022060000006B" constraintId="28">
   </Feature>
   <Feature name="Projektai" description="" premature="false"
guid="ID_7F00000100000128C6F23B7B00000055" constraintId="36">
   </Feature>
   <Diagram name="BA" guid="ID_7F00000100000128C4959CB80000004F" xpos="-1" ypos="0" width="696"
height="439">
    <FeatureNode guid="ID_7F00000100000128C49761CA00000051" refFeatureName="Buhalterines apskaitos
programa" refGuidFeatureTree="ID_7F00000100000128C49761BB00000050" xpos="203" ypos="7"
constraintId="2">
     <Cardinality guid="ID_7F00000100000128C4A0EC470000006D" constraintId="30">
      <Limit min="1" max="1"/>
      <FeatureNode guid="ID_7F00000100000128C49EFCCB00000064" refFeatureName="Veikla"
refGuidFeatureTree="ID_7F00000100000128C49EFCCB00000063" xpos="102" ypos="78" constraintId="21">
       <Group guid="ID_7F00000100000128C6F2D9EB0000005B" constraintId="42">
        <Limit min="1" max="3"/>
        <FeatureNode guid="ID_7F00000100000128C49FA0B700000068" refFeatureName="Paslaugos"
refGuidFeatureTree="ID_7F00000100000128C49FA0B700000067" xpos="5" ypos="143" constraintId="25">
```

```
        </FeatureNode>
        <FeatureNode guid="ID_7F00000100000128C4A022060000006C" refFeatureName="Prekyba"
refGuidFeatureTree="ID_7F00000100000128C4A022060000006B" xpos="83" ypos="179" constraintId="29">
        </FeatureNode>
        <FeatureNode guid="ID_7F00000100000128C49FE7E40000006A" refFeatureName="Gamyba"
refGuidFeatureTree="ID_7F00000100000128C49FE7E400000069" xpos="180" ypos="173" constraintId="27">
        </FeatureNode>
        <FeatureNode guid="ID_7F00000100000128C6F23B7B00000056" refFeatureName="Projektai"
refGuidFeatureTree="ID_7F00000100000128C6F23B7B00000055" xpos="262" ypos="173" constraintId="37">
        </FeatureNode>
       </Group>
      </FeatureNode>
     </Cardinality>
     <Cardinality guid="ID_7F00000100000128C499501E00000058" constraintId="9">
      <Limit min="0" max="2"/>
      <FeatureNode guid="ID_7F00000100000128C49932DE00000057" refFeatureName="Kasos aparatas"
refGuidFeatureTree="ID_7F00000100000128C49932DE00000056" xpos="257" ypos="78" constraintId="8">
      </FeatureNode>
     </Cardinality>
     <Cardinality guid="ID_7F00000100000128C498C33A00000055" constraintId="6">
      <Limit min="1" max="1"/>
      <FeatureNode guid="ID_7F00000100000128C498533A00000053" refFeatureName="Kalba"
refGuidFeatureTree="ID_7F00000100000128C498533A00000052" xpos="448" ypos="78" constraintId="4">
       <Group guid="ID_7F00000100000128C49A85DC00000062" constraintId="19">
        <Limit min="1" max="1"/>
        <FeatureNode guid="ID_7F00000100000128C49A16E50000005E" refFeatureName="Rusu"
refGuidFeatureTree="ID_7F00000100000128C49A16E50000005D" xpos="526" ypos="142" constraintId="15">
        </FeatureNode>
                                        <FeatureNode guid="ID_7F00000100000128C499B8040000005A"
refFeatureName="Lietuviu" refGuidFeatureTree="ID_7F00000100000128C499B80400000059" xpos="364"
ypos="140" constraintId="11">
        </FeatureNode>
        <FeatureNode guid="ID_7F00000100000128C499EAF50000005C" refFeatureName="Anglu"
refGuidFeatureTree="ID_7F00000100000128C499EAF50000005B" xpos="448" ypos="141" constraintId="13">
        </FeatureNode>
       </Group>
      </FeatureNode>
     </Cardinality>
    </FeatureNode>
    <Constraints>
     <Constraint string="(id:14) excludes (id:7)"/>
    </Constraints>
   </Diagram>
  </FeatureSpace>
 </ModelLevel>
</ModelSpace>
</CFRepository>
```

3 priedas. Pavyzdžio binarinė sprendimų diagrama



4 priedas. Prototipo kodas

```
import jdd.bdd.*;

public class BDDCreator{
   public BDD bdd;
   public int root;
   public Feature[] features = new Feature[100];
   public FeatureNode[] featureNodes = new FeatureNode[100];
   public Constraint[] constraints = new Constraint[100];
   public BDDUserNames names;
   public int featuresIdx = 0;
   public int featureNodesIdx = 0;
   public int constraintsIdx = 0;
   public LH[] lh = new LH[100];

   private void updateTree(int var) {
      lh[bdd.getVar(var)].l = lh[bdd.getVar(var)].l && bdd.getLow(var) == 0;
      lh[bdd.getVar(var)].h = lh[bdd.getVar(var)].h && bdd.getHigh(var) == 0;
      int x = bdd.getVar(var) + 1;
      while(x < bdd.getVar(bdd.getLow(var)) && bdd.getLow(var) != 0){
         lh[x].l = false;
         lh[x].h = false;
         x++;
      }
      x = bdd.getVar(var) + 1;
      while(x < bdd.getVar(bdd.getHigh(var)) && bdd.getHigh(var) != 0){
         lh[x].l = false;
         lh[x].h = false;
         x++;
      }
      if(bdd.getLow(var) != 0 && bdd.getLow(var) != 1){
         updateTree(bdd.getLow(var));
      }
      if(bdd.getHigh(var) != 0 && bdd.getHigh(var) != 1){
         updateTree(bdd.getHigh(var));
      }
   }

   void anySat() {
      root = bdd.oneSat(root);
      updateTree();
   }

   public class LH{
      public boolean l = true;
      public boolean h = true;
   }

   public BDDCreator(int a, int b){
      bdd = new BDDwithDiscard(a, b);
   }
   public void addFeature(Feature f){
      features[featuresIdx] = f;
      featuresIdx++;
   }
   public void addFeatureNode(FeatureNode f){
      featureNodes[featureNodesIdx] = f;
      featureNodesIdx++;
   }
   public void addConstraint(Constraint c){
      constraints[constraintsIdx] = c;
      constraintsIdx++;
```

```
    }
    public void drawBDD(String str, int rootNode){
        bdd.printDot(str, rootNode);
    }
    public int getOrGroup(FeatureNode fn){
        int group = bdd.getZero();
        if(fn.childCounter == 0){
            return fn.var;
        }
        for(int i = 0; i < fn.childCounter; i++){
            group = bdd.ref(bdd.or(group, getOrGroup(fn.children[i])));
        }
        return group;
    }
    public int addConstraints(int bddiagram){
        int ret = bdd.getOne();
        int A = 0;
        int B = 0;
        for(int i = 0; i < this.constraintsIdx; i++){
            A = bdd.getZero();
            B = bdd.getZero();
            for(int a = 0; a < this.constraints[i].Aidx; a++){
                A = bdd.ref(bdd.or(A, getOrGroup(this.constraints[i].A[a])));
            }
            for(int b = 0; b < this.constraints[i].Bidx; b++){
                B = bdd.ref(bdd.or(B, getOrGroup(this.constraints[i].B[b])));
            }
            if(this.constraints[i].requires){
                this.constraints[i].var = bdd.ref(bdd.imp(A, B));
                /*((a1 or a2) imp (b1 or b2))*/
            }
            else{
                this.constraints[i].var = bdd.ref(bdd.not(bdd.ref(bdd.and(A, B))));
                 /*not((a1 or a2) and (b1 or b2))*/
            }
            ret = bdd.ref(bdd.and(ret, this.constraints[i].var));
        }
        ret = bdd.ref(bdd.and(ret, bddiagram));
        return ret;
    }
    public void createBDDVars(){
        for(int i = 0; i < featureNodesIdx; i++){
            if(featureNodes[i].childCounter == 0){
                featureNodes[i].var = bdd.createVar();
            }
        }
    }
    public void setNames(){
        names = new jdd.bdd.BDDUserNames();
        bdd.setNodeNames(names);
        int j = 0;
        for(int i = 0; i < featureNodesIdx; i++){
            if(featureNodes[i].childCounter == 0){
                names.setName(j, featureNodes[i].name +((featureNodes[i].idx == null)?(""):("."+ featureNodes[i].idx)));
                j++;
            }
        }
    }
    public int createBDD(FeatureNode rootNode){
        if(this.isLeaf(rootNode) && rootNode.min == 1){
            return rootNode.var;
        }
        else if(this.isLeaf(rootNode) && rootNode.min == 0){
```

```
        if(rootNode.father.min  == 1 && !rootNode.father.name.equals("group")){
            return bdd.getOne();
        }
        return rootNode.var;
    }
    else if(!this.isLeaf(rootNode) && !rootNode.name.equals("group")){
        rootNode.var = bdd.getOne();
        int and = bdd.getOne();
        int andAll = bdd.getOne();
        int all = bdd.getZero();
        int or = bdd.getZero();
        for(int i = 0; i < rootNode.childCounter; i++){
            if(rootNode.children[i].min  == 1){
                and =  bdd.ref(bdd.and(and, createBDD(rootNode.children[i])));
            }
            else{
                or =  bdd.ref(bdd.or(or, createBDD(rootNode.children[i])));
            }
            all =  bdd.ref(bdd.or(all, createBDD(rootNode.children[i])));
            andAll =  bdd.ref(bdd.and(andAll, createBDD(rootNode.children[i])));
        }
        if(rootNode.min == 1){
            return andAll;
        }
        if(and != bdd.getZero()){
            rootNode.var = bdd.ref(bdd.imp(all, and));
            return rootNode.var;
        }
        return and;
    }
    else if(!this.isLeaf(rootNode) && rootNode.name.equals("group")){
        rootNode.groupChildCounter = rootNode.childCounter;
        rootNode.groupMin = rootNode.min;
        rootNode.groupMax = rootNode.max;
        return this.createGroupBDD(rootNode);
    }
    return 0;
}


public String constraintString(){
    String ret = "";
    for(int i = 0; i < constraintsIdx; i++){
        ret += this.constraints[i].toString() + '\n';
    }
    return ret;
}
public void createCloneConstraints(FeatureNode root){
    FeatureNode saved = null;
    for(int i = 0; i < root.childCounter; i++){
        if(root.children[i].idx != null){
            if(saved == null){
                saved = root.children[i];
            }
            else if(saved.guid.equals(root.children[i].guid)){
                this.addConstraint(new Constraint(root.children[i], saved));
                saved = root.children[i];
            }
            else{
                saved = root.children[i];
            }
        }
        createCloneConstraints(root.children[i]);
```

```
        }
    }

    public FeatureNode getRoot() {
        for(int i = 0; i <= featureNodesIdx; i++ ){
            if(featureNodes[i].father == null){
                return featureNodes[i];
            }
        }
        return null;
    }

    public Object getChild(Object parent, int index) {
        return ((FeatureNode)parent).children[index];
    }

    public int getChildCount(Object parent) {
        return ((FeatureNode)parent).childCounter;
    }

    public boolean isLeaf(Object node) {
        return ((FeatureNode)node).childCounter == 0;
    }

    public int getIndexOfChild(Object parent, Object child) {
        FeatureNode fn = (FeatureNode)parent;
        for(int i = 0; i <= fn.childCounter; i++ ){
            if(fn.children[i] == (FeatureNode)child){
                return i;
            }
        }
        return -1;
    }


    private int createGroupBDD(FeatureNode rootNode) {
        if(rootNode.groupMin == 0 && rootNode.groupMax == rootNode.groupChildCounter){
            rootNode.var = bdd.getOne();
            return rootNode.var;
        }
        else if(rootNode.groupMin == 0 && rootNode.groupMax == 0){
            rootNode.var = bdd.getZero();
            for(int i = 0; i < rootNode.groupChildCounter; i++){
                rootNode.var = bdd.ref(bdd.or(rootNode.var, createBDD(rootNode.children[i])));
            }
            return bdd.ref(bdd.not(rootNode.var));
        }
        else if(rootNode.groupMin == rootNode.groupChildCounter && rootNode.groupMax ==
rootNode.groupChildCounter){
            rootNode.var = bdd.getOne();
            for(int i = 0; i < rootNode.groupChildCounter; i++){
                rootNode.var = bdd.ref(bdd.and(rootNode.var, createBDD(rootNode.children[i])));
            }
            return rootNode.var;
        }
        else if(rootNode.groupMin == 1 && rootNode.groupMax == rootNode.groupChildCounter){
            rootNode.var = bdd.getZero();
            for(int i = 0; i < rootNode.groupChildCounter; i++){
                rootNode.var = bdd.ref(bdd.or(rootNode.var, createBDD(rootNode.children[i])));
            }
            return rootNode.var;
        }
        else if(rootNode.groupMin == 1 && rootNode.groupMax == 1){
```

```
            rootNode.var = bdd.getZero();
            rootNode.groupChildCounter--;
            for(int i = 0; i < rootNode.groupChildCounter; i++){
               rootNode.var = bdd.ref(bdd.or(rootNode.var, createBDD(rootNode.children[i])));
            }
            rootNode.var = bdd.ref(bdd.not(rootNode.var));

            rootNode.var = bdd.ref(bdd.ite(createBDD(rootNode.children[rootNode.groupChildCounter]),
                  rootNode.var, createGroupBDD(rootNode)));
            /*if F1 then not(F2 or .. or Fk) else one-of(F2,..,Fk)*/
            return rootNode.var;
         }
         else{
            FeatureNode tmp = rootNode.cloneForBDD();
            rootNode.groupChildCounter--;
            rootNode.groupMin = rootNode.groupMin == 0? 0: rootNode.groupMin - 1;
            rootNode.groupMax--;
            tmp.groupChildCounter--;
            tmp.groupMax = tmp.groupMax > tmp.groupChildCounter? tmp.groupMax: tmp.groupChildCounter;
            rootNode.var = bdd.ref(bdd.ite(createBDD(rootNode.children[rootNode.groupChildCounter]),
                  createGroupBDD(rootNode), createGroupBDD(tmp)));
            /*if F1 then <min(0,(i-1))-(j-1)>(F2,..,Fk) else <i-min(j,k-1)>(F2, .., Fk)*/
            return rootNode.var;
         }
      }

      void createBDD() {
         this.createBDDVars();
         root = this.createBDD(this.getRoot());
         this.setNames();
         root = this.addConstraints(root);
         this.drawBDD("bdd", root);
      }
      void updateTree(){
         for(int i = 0; i <= bdd.getVar(0); i++){
            lh[i] = new LH();
         }
         updateTree(root);
         boolean updated = false;
         FeatureNode selected = null;
         for(int i = 0; i < bdd.getVar(0); i++){
            if(lh[i].l || lh[i].h){
               selected = findFeatureVar(i);
               if(selected == null) {continue;}
               if(selected.chosen == null && lh[i].l){
                  chooseNode(selected);
                  updated = true;
               }
               if(selected.chosen == null && lh[i].h){
                  this.discardNode(selected);
                  updated = true;
               }
            }
         }
         if(updated){
            updateTree();
         }
      }

      void chooseNode(FeatureNode selected) {
         selected.chosen = true;
         root = bdd.ref(bdd.restrict(root, selected.var));
         root = bdd.ref(bdd.and(root, selected.var));
```

```
    }
    public String satCount(){
        return "Galimų konfigūracijų skaičius: " + ((int)(bdd.satCount(root)));
    }

    void discardNode(FeatureNode selected) {
        selected.chosen = false;
        root = bdd.ref(((BDDwithDiscard)bdd).discard(root, selected.var));
        root = bdd.ref(bdd.and(root, bdd.ref(bdd.not(selected.var))));
    }

    private FeatureNode findFeatureVar(int var) {
        for(int i = 0; i < featureNodesIdx; i++){
            if(featureNodes[i].childCounter == 0 && featureNodes[i].chosen == null  &&
bdd.getVar(featureNodes[i].var) == var){
                return featureNodes[i];
            }
        }
        return null;
    }
}
```

```
import jdd.bdd.BDD;

public class BDDwithDiscard extends BDD {

    public BDDwithDiscard(int nodesize, int cache_size){
        super(nodesize, cache_size);
    }

    public int discard(int u, int v) {
                                    if(v == 1) return u;
                                    varset_signed(v);
                                    restrict_careset = v;
                                    return discard_rec(u);
                    }

                    private void varset_signed(int bdd) {
                                    for(int i = 0; i < num_vars; i++) varset_vec[i] = false;
                                    while( bdd > 1) {
                                                    varset_last = getVar(bdd);
                                                    varset_vec[varset_last] = true;
                                                    sign_vec[varset_last] = (getLow(bdd) == 0);
                                                    bdd = getHigh(bdd);
                                    }
        }

                    private int discard_rec(int u) {
                                    if(u < 2 || getVar(u) > varset_last) return u;
                                    if(op_cache.lookup(u, restrict_careset, CACHE_RESTRICT)) return
op_cache.answer;
                                    int hash = op_cache.hash_value;
                                    int ret = 0;
                                    if(varset_vec[ getVar(u)] ) {
                                                                    ret = discard_rec( !sign_vec[getVar(u)] ?
getHigh(u) : getLow(u) );
                                    } else {
                                                    int l = work_stack[work_stack_tos++] = discard_rec( getLow(u) );
                                                    int h = work_stack[work_stack_tos++] = discard_rec( getHigh(u) );
                                                    ret = mk( getVar(u), l, h);
                                                    work_stack_tos -= 2;
                                    }
                                    op_cache.insert(hash, u, restrict_careset, CACHE_RESTRICT, ret );
                                    return ret;
```

```
                }
}
```

```
import org.xml.sax.*;

public class CFXReader implements ContentHandler {

   public BDDCreator BDDC;
   public Feature featureHandle = null;
   public FeatureNode featureNodeHandle = null;
   public FeatureNode featureNodePrev = null;
   private int min = 1;
   private int max = 1;

   CFXReader(BDDCreator BDDC) {
      this.BDDC = BDDC;
   }

  public void startDocument() throws SAXException
  {
   System.out.println("Pradeta");
  }

  public void startElement(String namespaceURI, String localName,
   String qualifiedName, Attributes atts) throws SAXException {
      if(localName.equals("Feature")){
         featureHandle = new Feature();
         featureHandle.name = atts.getValue("name");
         featureHandle.description = atts.getValue("description");
         featureHandle.guid = atts.getValue("guid");
         featureHandle.constraintId = Integer.parseInt(atts.getValue("constraintId"));
         BDDC.addFeature(featureHandle);
      }
      else if(localName.equals("FeatureNode")){
         featureNodeHandle = new FeatureNode();
         featureNodeHandle.name = atts.getValue("refFeatureName");
         featureNodeHandle.guid = atts.getValue("guid");
         featureNodeHandle.refGuid = atts.getValue("refGuidFeatureTree");
         featureNodeHandle.constraintId = Integer.parseInt(atts.getValue("constraintId"));
         featureNodeHandle.min = min;
         featureNodeHandle.max = max;
         BDDC.addFeatureNode(featureNodeHandle);
         if(featureNodePrev != null){
            featureNodePrev.addChild(featureNodeHandle);
         }
         featureNodePrev = featureNodeHandle;
      }
      else if(localName.equals("Group")){
         featureNodeHandle = new FeatureNode();
         featureNodeHandle.guid = atts.getValue("guid");
         featureNodeHandle.constraintId = Integer.parseInt(atts.getValue("constraintId"));
         BDDC.addFeatureNode(featureNodeHandle);
         if(featureNodePrev != null){
            featureNodePrev.addChild(featureNodeHandle);
         }
         featureNodePrev = featureNodeHandle;
      }
      else if(localName.equals("Limit")){
         if(featureNodePrev.name == null){
            featureNodePrev.min = Integer.parseInt(atts.getValue("min"));
            featureNodePrev.max = Integer.parseInt(atts.getValue("max"));
            featureNodePrev.name = "group";
            min = 1;
            max = 1;
```

```
            }
            else{
               min = Integer.parseInt(atts.getValue("min"));
               max = Integer.parseInt(atts.getValue("max"));
            }
         }
         else if(localName.equals("Constraint")){
            BDDC.addConstraint(new Constraint(atts.getValue("string"), BDDC));
         }
   }
   public void endElement(String namespaceURI, String localName,
    String qualifiedName) throws SAXException {
         if((localName.equals("FeatureNode") || localName.equals("Group")) && featureNodePrev != null &&
featureNodePrev.father != null){
            if(localName.equals("FeatureNode") && featureNodePrev.max > 1){
               featureNodePrev.cloneFeature(BDDC);
            }
            featureNodePrev = featureNodePrev.father;
         }
   }

   public void endDocument() throws SAXException
   {
      BDDC.createCloneConstraints(BDDC.getRoot());
      System.out.println("Baigta");
   }

   // Do-nothing methods
   public void setDocumentLocator(Locator l) {}
   public void startPrefixMapping(String p, String u)
    throws SAXException {}
   public void endPrefixMapping(String p) throws SAXException {}
   public void skippedEntity(String n) throws SAXException {}
   public void processingInstruction(String t, String d)
    throws SAXException {}
   public void characters(char[] t, int s, int l)
    throws SAXException {}
   public void ignorableWhitespace(char[] t, int s, int l)
    throws SAXException{}

}
```

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
import javax.swing.*;
import javax.swing.tree.*;

public class ConfigUtil extends javax.swing.JFrame {
    private XMLReader parser;
    FeatureTree tree;
    JTree   jTree;
    public BDDCreator BDDC = null;
    public int bdd;
    private FeatureNode selected;

    /** Creates new form ConfigUtil */
    public ConfigUtil() {
       initComponents();
       try {
          parser = XMLReaderFactory.createXMLReader();
       }
       catch (SAXException e) {
```

```
            System.err.println("createXMLReader failed.");
            return;
        }


        DefaultMutableTreeNode top =
        new DefaultMutableTreeNode("Empty");
        jTree1 = new JTree(top);
        jScrollPane1.setViewportView(jTree1);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTree1 = new javax.swing.JTree();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();
        msg = new javax.swing.JLabel();
        choose = new javax.swing.JButton();
        discard = new javax.swing.JButton();
        jButton1 = new javax.swing.JButton();
        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setName("ConfigUtil"); // NOI18N

        jTree1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jTree1MouseClicked(evt);
            }
        });
        jTree1.addTreeSelectionListener(new javax.swing.event.TreeSelectionListener() {
            public void valueChanged(javax.swing.event.TreeSelectionEvent evt) {
                jTree1ValueChanged(evt);
            }
        });
        jScrollPane1.setViewportView(jTree1);

        jTextArea1.setColumns(20);
        jTextArea1.setRows(5);
        jScrollPane2.setViewportView(jTextArea1);

        choose.setText("Pasirinkti");
        choose.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                chooseActionPerformed(evt);
            }
        });

        discard.setText("Atmesti");
        discard.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            discardActionPerformed(evt);
        }
    });

    jButton1.setText("Užbaigti");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(msg, javax.swing.GroupLayout.DEFAULT_SIZE, 473, Short.MAX_VALUE)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                        .addComponent(jScrollPane2, javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 331, Short.MAX_VALUE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jButton1, javax.swing.GroupLayout.DEFAULT_SIZE, 136,
Short.MAX_VALUE)
                        .addComponent(choose, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, 136, Short.MAX_VALUE)
                        .addComponent(discard, javax.swing.GroupLayout.DEFAULT_SIZE, 136,
Short.MAX_VALUE))))
            .addContainerGap())
    );
    jPanel1Layout.setVerticalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 258,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(18, 18, 18)
                    .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 149,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(11, 11, 11)
                    .addComponent(choose)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(discard)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jButton1)))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(msg, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(22, Short.MAX_VALUE))
    );

    msg.getAccessibleContext().setAccessibleName("Msg");

    jMenu1.setText("Byla");

    jMenuItem1.setText("Atidaryti");
```

```
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem1);

        jMenuBar1.add(jMenu1);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 493, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 488, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser fc = new JFileChooser();
        File file;
        BDDC = new BDDCreator(100, 100);
        parser.setContentHandler(new CFXReader(BDDC));
        int returnVal = fc.showOpenDialog(this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            try {
                parser.parse(file.getPath());
            }
            catch (Exception e) {
                System.err.println(e.toString());
            }
        }
        BDDC.createBDD();
        jTree1 = new FeatureTree(BDDC.getRoot());
        jTree1.addMouseListener(new java.awt.event.MouseAdapter() {
            @Override
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jTree1MouseClicked(evt);
            }
        });
        jScrollPane1.setViewportView(jTree1);
        jTextArea1.setText(BDDC.constraintString());
        BDDC.updateTree();
        msg.setText(BDDC.satCount());

    }

    private void jTree1MouseClicked(java.awt.event.MouseEvent evt) {
```

```
        TreePath path = jTree1.getSelectionPath();
        if(path != null){
            selected = (FeatureNode)path.getLastPathComponent();
        }
    }

    private void jTree1ValueChanged(javax.swing.event.TreeSelectionEvent evt) {
    }

    private void chooseActionPerformed(java.awt.event.ActionEvent evt) {
        BDDC.chooseNode(selected);
        BDDC.updateTree();
        jScrollPane1.setViewportView(jTree1);
        msg.setText(BDDC.satCount());

    }

    private void discardActionPerformed(java.awt.event.ActionEvent evt) {
        BDDC.discardNode(selected);
        BDDC.bdd.printDot("discard", BDDC.root);
        BDDC.updateTree();
        jScrollPane1.setViewportView(jTree1);
        msg.setText(BDDC.satCount());
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        BDDC.anySat();
        jScrollPane1.setViewportView(jTree1);
        msg.setText(BDDC.satCount());
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new ConfigUtil().setVisible(true);

            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton choose;
    private javax.swing.JButton discard;
    private javax.swing.JButton jButton1;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea jTextArea1;
    private javax.swing.JTree jTree1;
    private javax.swing.JLabel msg;
    // End of variables declaration

}
```

```
public class Constraint {
    public FeatureNode[] A = new FeatureNode[100];
    public int Aidx = 0;
    public boolean requires;
    public FeatureNode[] B = new FeatureNode[100];
    public int Bidx = 0;
    public int var = 0;
```

```
   Constraint(String value, BDDCreator BDDC) {
      String[] str;
      str = value.split(" ");
      str[0] = str[0].substring(4, str[0].length()-1);
      for(int i = 0; i < BDDC.featuresIdx; i++){
         if(BDDC.features[i].constraintId == Integer.parseInt(str[0])){
            for(int j = 0; j < BDDC.featureNodesIdx; j++){
               if(!BDDC.featureNodes[j].name.equals("group") &&
BDDC.featureNodes[j].refGuid.equals(BDDC.features[i].guid)){
                  A[Aidx] = BDDC.featureNodes[j];
                  Aidx++;
               }
            }

         }
      }
      for(int i = 0; i < BDDC.featureNodesIdx; i++){
         if(BDDC.featureNodes[i].constraintId == Integer.parseInt(str[0])){
            A[Aidx] = BDDC.featureNodes[i];
            Aidx++;
         }
      }
      requires = str[1].equals("requires");
      str[2] = str[2].substring(4, str[2].length()-1);
      for(int i = 0; i < BDDC.featuresIdx; i++){
         if(BDDC.features[i].constraintId == Integer.parseInt(str[2])){
            for(int j = 0; j < BDDC.featureNodesIdx; j++){
               if(!BDDC.featureNodes[j].name.equals("group") &&
BDDC.featureNodes[j].refGuid.equals(BDDC.features[i].guid)){
                  B[Bidx] = BDDC.featureNodes[j];
                  Bidx++;
               }
            }

         }
      }
      for(int i = 0; i < BDDC.featureNodesIdx; i++){
         if(BDDC.featureNodes[i].constraintId == Integer.parseInt(str[2])){
            B[Bidx] = BDDC.featureNodes[i];
            Bidx++;
         }
      }
   }
   Constraint(FeatureNode a, FeatureNode b){
      A[Aidx] = a;
      Aidx++;
      requires = true;
      B[Bidx] = b;
      Bidx++;
   }

   @Override
   public String toString(){
      String ret = "";
      for(int i = 0; i < Aidx; i++){
         ret += A[i].name +((A[i].idx == null)?(""):("."+ A[i].idx)) + (i+1 < Aidx?", ":" ");
      }
      ret += requires?"requires ":"excludes ";
      for(int j = 0; j < Bidx; j++){
         ret += B[j].name +((B[j].idx == null)?(""):("."+ B[j].idx)) +  (j+1 < Bidx?", ":"");
      }
      return ret;
```

```
    }

}
```

```
                public class Feature {
                    public int var;
                    public String name;
                    public String description;
                    public String guid;
                    public int constraintId;

                    @Override
                    public String toString(){
                        return name +  " id:" + constraintId;
                    }
                }
```

```
public class FeatureNode {
    public int var = 0;
    public String name;
    public String guid;
    public String refGuid;
    public int constraintId;
    public int min = 0;
    public int max = 0;
    public int groupMin = 0;
    public int groupMax = 0;
    public int maxSuper = 0;
    public FeatureNode father = null;
    public FeatureNode[] children = new FeatureNode[100];
    public int childCounter = 0;
    public int groupChildCounter = 0;
    public Integer idx = null;
    public Boolean chosen = null;

    void addChild(FeatureNode featureNodeHandle) {
        this.children[childCounter] = featureNodeHandle;
        childCounter++;
        featureNodeHandle.father = this;
    }
    @Override
    public String toString(){
        if(!name.equals("group")){
            return name +((idx == null)?(""):("."+ idx)) + " [" + min + ".." + max + "]" + " id:" + constraintId;
        }
        return name + " <" + min + "-" + max + ">";
    }
    public FeatureNode cloneFeature(int minF, int maxF, BDDCreator BDDC){
        FeatureNode newFN = null;
        if(this.idx ==null){
            this.idx = 1;
        }
        if(this.maxSuper == 0){
            this.maxSuper = this.max;
        }
        newFN = new FeatureNode();
        BDDC.addFeatureNode(newFN);
        newFN.name = this.name;
        newFN.guid = this.guid;
        newFN.refGuid = this.refGuid;
        newFN.constraintId = this.constraintId;
        newFN.min = minF;
        newFN.max = 1;
        newFN.maxSuper = this.maxSuper;
        newFN.idx = this.idx + maxF;
```

```
        for(int i = 0; i < this.childCounter; i++){
            newFN.addChild(this.children[i].cloneFeature(this.children[i].min,this.children[i].maxSuper, BDDC));
          }
        return newFN;
    }
    public void cloneFeature(BDDCreator BDDC){
        FeatureNode newFN = this;
        this.maxSuper = this.max;
        this.idx = new Integer(1);
        for(int n = 1; n < this.maxSuper; n++){
          newFN = newFN.cloneFeature(this.min <= 1? 0:1,1,BDDC);
          this.father.addChild(newFN);
          this.max--;
          this.min = this.min - 1 < 1? this.min: this.min -1;
        }
    }

    public boolean cloned() {
        return idx != null;
    }

    FeatureNode cloneForBDD() {
        FeatureNode newFN = new FeatureNode();
        newFN.groupMin = this.groupMin;
        newFN.groupMax = this.groupMax;
        newFN.groupChildCounter = this.groupChildCounter;
        newFN.children = this.children;
        return newFN;
    }
}
```

```
import javax.swing.JTree;
import javax.swing.tree.TreeSelectionModel;

public class FeatureTree extends JTree {
    FeatureTreeModel model;

    public FeatureTree(FeatureNode node) {
        super(new FeatureTreeModel(node));
        getSelectionModel().setSelectionMode(
                TreeSelectionModel.SINGLE_TREE_SELECTION);
        IconCellRenderer renderer = new IconCellRenderer();
        setCellRenderer(renderer);
    }
}
```

```
import javax.swing.event.TreeModelListener;
import javax.swing.tree.*;
import java.util.Vector;

public class FeatureTreeModel implements TreeModel {
    private FeatureNode root;
    private Vector<TreeModelListener> treeModelListeners =
        new Vector<TreeModelListener>();

    FeatureTreeModel(FeatureNode node) {
        root = node;
    }

    public Object getRoot() {
        return root;
    }

    public Object getChild(Object parent, int index) {
        return ((FeatureNode)parent).children[index];
```

```
    }

    public int getChildCount(Object parent) {
        return ((FeatureNode)parent).childCounter;
    }

    public boolean isLeaf(Object node) {
        return ((FeatureNode)node).childCounter == 0;
    }

    public void valueForPathChanged(TreePath path, Object newValue) {

    }

    public int getIndexOfChild(Object parent, Object child) {
        for(int i = 0; i < ((FeatureNode)parent).childCounter; i++){
            if(((FeatureNode)parent).children[i] == (FeatureNode)child);
            return i+1;
        }
        return 0;
    }

    public void addTreeModelListener(TreeModelListener l) {
        treeModelListeners.addElement(l);
    }

    public void removeTreeModelListener(TreeModelListener l) {
        treeModelListeners.removeElement(l);
    }

}
```

```
import java.awt.*;

import javax.swing.*;
import javax.swing.tree.*;

public class IconCellRenderer extends DefaultTreeCellRenderer {
    public static ImageIcon ICON_NULL = new ImageIcon("null.jpg");
    public static ImageIcon ICON_TRUE = new ImageIcon("true.jpg");
    public static ImageIcon ICON_FALSE = new ImageIcon("false.jpg");

    @Override
    public Component getTreeCellRendererComponent(
                        JTree tree,
                        Object value,
                        boolean sel,
                        boolean expanded,
                        boolean leaf,
                        int row,
                        boolean hasFocus) {

        super.getTreeCellRendererComponent(
                        tree, value, sel,
                        expanded, leaf, row,
                        hasFocus);
        if (leaf && ((FeatureNode)value).chosen == null) {
            setIcon(ICON_NULL);
        }
        else if (leaf && ((FeatureNode)value).chosen == true) {
            setIcon(ICON_TRUE);
        }
        else if (leaf && ((FeatureNode)value).chosen == false) {
            setIcon(ICON_FALSE);
```

```
        }
        return this;
    }
}
```