KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Saulius Bira

# Magic Draw įrankio išplėtimas klasių diagramų ir būsenų mašinų derinimo galimybėmis

Magistro darbas

**Darbo vadovas**

**prof. dr. Lina Nemuraitė**

**KAUNAS, 2008**

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Saulius Bira

# Magic Draw įrankio išplėtimas klasių diagramų ir būsenų mašinų derinimo galimybėmis

Magistro darbas

Recenzentas

doc. dr. Vytautas Pilkauskas

data 2008.05.26

Darbo vadovas

prof. dr. Lina Nemuraitė

data 2008.05.26

Atliko

IFM-2/2 gr. stud.

Saulius Bira

**KAUNAS, 2008**

# Magic Draw įrankio išplėtimas klasių diagramų ir būsenų mašinų derinimo galimybėmis

## Santrauka

Modeliais paremtos architektūros (MDA) technologijos panaudojimo tikslas – automatizuoti kuriamos programų sistemos kuriamų modelių transformavimą ir kodo generavimą. Norint atlikti sukurtų modelių korektišką transformavimą, reikia užtikrinti modelių pilnumą ir suderinamumą tarpusavyje. Šie modeliai aprašomi UML modeliavimo kalba.

Magistriniame darbe nagrinėjama nuo platformos nepriklausančio modelio kūrimo stadija, modelio klasių ir būsenų mašinų suderimo galimybės ir būsenų mašinų korektiškumo ir išbaigtumo metodikos. Taip pat bus pateikiamas sprendimas atliktas MagicDraw aplinkoje įskiepio pagalba.

Raktažodžiai: UML, būsenų mašina, klasių diagrama, MagicDraw įskiepis, nuo platformos nepriklausančio modelio tikrinimas, būsenų mašinų tikrinimo metodikos, klasių ir būsenų mašinų suderinamumas.

# SUMMARY

EXTENSION OF MAGIC DRAW TOOL FOR RECONCILIATION OF CLASS DIAGRAMS AND STATE MACHINES

SUMMARY

The main goal of Model Driven Architecture (MDA) is the automation of software development process. According this technology, we need to create platform independent model (PIM), after that transform it to platform specific model (PSM); from PSM model we can generate program code. To do that, we need to ensure static and dynamic completeness of PIM. All these models are described using UML modeling language.

In this work correctness and completeness of PIM is achieved by analyzing compatibility of class diagrams and state machines, as well as correctness and completeness of state machines themselves. To solve this problem, algorithms were created and implemented in a plug-in for MagicDraw CASE tool.

Keywords: UML, state machine, class diagram, checking methodology, MDA, Magic Draw, plug-in.

# TURINYS

## PAVEIKSLŲ SĄRAŠAS

# LENTELIŲ SĄRAŠAS

# ĮVADAS

Šiandien sparčiai tobulėjant informacinėms technologijoms auga ir poreikis naujų IS ar esamų sistemų modifikavimui. Todėl nenuostabu, kad daugelis IS kūrėjų ieško būdų kaip paspartinti IS kūrimo darbus ir mažinti jų modernizavimo kaštus. Vienas iš būdų - kuo daugiau automatizuoti kūrimo procesą, stengiantis kuo dagiau klaidų eliminuoti programų sistemos kūrimo pradžioje. Kaip ir technologijų, taip ir IS sudėtingumas auga, todėl programų sistemos kūrimo procesas darosi vis sudėtingesnis ir painesnis. Šio proceso automatizavimas tampa taip pat sudėtingesnis. IS kūrėjai vis daugiau dėmesio telkia į programos kūrimo proceso pradinius etapus, nes juose padarytos klaidos įtakoja tolimesnių darbų vykdymo sklandumą. Pradiniuose etapuose sudaromas sistemos modelis. Sistemos modeliams aprašyti naudojama UML kalba.

Universali modeliavimo kalba UML (angl. *Unified Modeling Language*)– tai standartas, skirtas sistemos specifikavimui, vizualizavimui ir dokumentavimui. Šiandien UML yra kaip standartas, kuriuo remiantis projektuojamos objektiškai orientuotos sistemos. Modeliai, specifikuoti UML, gali būti vaizduojami įvairiomis diagramomis (klasių, būsenų, sekos, veiklos ir kt.), kurios tą pačią sistemą vaizduoja skirtingais aspektais (statinis, dinaminis, sąveikų). Projektuotojas, kurdamas skirtingas diagramas, gali jų nesuderinti tarpusavyje ir palikti kokius nors neatitikimus ar prieštaravimus.

Vienas iš automatizavimo uždavinių, užtikrinti modelio komponentų (atskirų diagramų) tarpusavio suderinamumą. Toks metodas, kai sudaromas sistemos pradinis abstraktus modelis ir toliau atliekamas jo transformavimas į žemesnį abstrakcijos lygį, vadinamas modeliais grindžiama architektūra. Šį modelį tikslinga apibrėžti kuo ankstesniame kūrimo etape, nepriklausomai ne tik nuo programinės realizacijos (kaip PIM), bet ir nuo loginės architektūros, kad jį būtų galima pritaikyti ne tik skirtingoms realizavimo platformoms, bet ir skirtingiems architektūriniams sprendimams. Toks modelis vadinamas DIM (angl. *Design Independent Model*). Organizacijos Object Management Group (OMG) sukurta Model Driven Architecture (MDA) sistemų kūrimo technologija koncentruojasi į nuo platformos nepriklausomų modelių PIM (angl. *Platform Independent Model*) transformavimą į konkrečių platformų modelius PSM (angl. *Platform Specific Model*) ir pastarųjų transformavimą į programos kodą, analizės dalyje plačiau apžvelgsime šių modelių poziciją kūrimo procese. Modeliais grindžiamo kūrimo metoduose pagrindinis dėmesys sutelktas į paskutinius kūrimo proceso etapus. Pradiniai kūrimo proceso etapai – reikalavimai ir jų analizė lieka nepaliesti, projektuotojas lieka atsakingas už PIM sudarymą, korektiškumą bei suderinimą. MDA siūlo pradiniame etape naudoti nuo skaičiavimų nepriklausomą modelį

CIM (angl. *Computation Independent Model*), bet šio modelio sudarymo ir naudojimo taisyklės nėra aiškiai apibrėžtos. Nėra apibrėžti reikalavimai ir projekto modeliui PIM.

Dabartiniuose CASE (angl. *Computer-Aided Software Engineering*) įrankiuose yra galimybės modelį ne tik pateikti vizualiai, piešiant skirtingas sistemos diagramas, bet ir susieti jas tarpusavyje per tam tikrus elementus. Įrankiai leidžia kurti modelį, kuris atitinka UML specifikaciją, o taip pat susieti diagramas tarpusavyje, t.y. naudoti vienose diagramose sukurtus elementus kuriant kitas diagramas. Tokiu būdu gali būti gaunamas vientisas modelis, tačiau jo suderinamumo užtikrinimas lieka projektuotojo rankose, o geros kokybės programos kodo generavimą gali užtikrinti tik išsamus, vientisas kompiuterizuojamos IS modelis, apimantis jos struktūrą ir elgseną.

Pradiniame etape analizuojami sistemos reikalavimai, juos galima apibrėžti naudojant panaudos atvejų (angl. *Use Case*) diagramą. Sudarant sistemos objektus, t.y. atvaizduojamas statinis sistemos vaizdas klasių diagramose (angl. *Class Diagrams*) klasių diagramoje remiantis OOP (angl. *Object-oriented programming*) paradigma, vienas sistemos elementas traktuojamas kaip objektas, atvaizduojamos esybės (angl. *Entities*) ir tų esybių valdikliai (angl. *Controllers*). Vieno objekto pilna elgsena gali būti atvaizduojama būsenų mašina (angl. *State Machine*). Šių abiejų tipų diagramos aprašo sistemą skirtingais aspektais (pirmoji – statinis sistemos vaizdas, antroji – dinaminis vaizdas). Kad būtų užtikrintas modelio vientisumas, šios diagramos turėtų derėti tarpusavyje. Diagramų suderinamumo patikrinimas gali būti atliekamas automatiškai.

**Darbe sprendžiamos problemos**:

- ✓ UML modelių klasių diagramų ir būsenų mašinų suderinimas informacinės sistemos kūrimo metu.
- ✓ Būsenų mašinos diagramos pilnumo ir korektiškumo patikrinimo galimybės ir būdai.
- ✓ Aukščiau paminėtų probleminių elementų užtikrinimas UML CASE įrankiuose.

**Tyrimo sritis** – klasių ir būsenų mašinų suderinamumas, būsenų mašinų korektiškumo, pilnumo patikrinimas.

**Tyrimo objektas** – elektroninių paslaugų sistemų modeliai. Tai nereiškia, kad modelis yra specifiškai apribotas, tačiau išanalizuota eksperimentinių taikymų aibė yra orientuota paslaugų architektūra paremtoms sistemoms.

**Darbo tikslas** – pagerinti UML kalba naudojančius projektavimo procesus, papildant juos klasių ir būsenų diagramų derinimo ir būsenų mašinų patikros galimybėmis, praplėsti UML specifikaciją, kad būtų galima atlikti patikrą.

**Darbo uždaviniai**:

- ✓ Išanalizuoti būsenų mašinų ir klasių diagramas, ryšius tarp jų,
- ✓ Išanalizuoti būsenų mašinų tikrinimo metodus ir jų realizavimo galimybes CASE įrankiuose,
- ✓ Sudaryti būsenų mašinų patikros ir suderinimo su klasių diagramomis algoritmus,
- ✓ Suprojektuoti ir realizuoti sukurtus algoritmus,
- ✓ Ištestuoti realizuotus algoritmus ir atlikti sukurto įskiepio eksperimentinį išbandymą,
- ✓ Eksperimentiškai parodyti būsenų diagramų patikros ir suderinamumo su klasių diagrama rezultatus,
- ✓ Įvertinti sukurto įskiepio kokybę ir praktinio taikymo galimybes

**Praktinė darbo rezultatų reikšmė**

Sukurtas įrankis, kuris galie padėti užtikrinti klasių diagramos suderinamumą su būsenų mašinomis tai leistų kurti efektyvesnius ir korektiškus, nuo platformos nepriklausančius modelius (PIM), o tai užtikrintų sklandų šių modelių generavimą į platformai specifikuotus modelius (PSM). Nagrinėti šaltiniai susiję su UML standartu [9].

**Darbo struktūra**

Ši darbą sudaro įvadas, šeši pagrindiniai skyriai, išvados, literatūros sąrašas, santrumpų ir terminų sąrašas.

Pirmajame skyriuje analizuojamos egzistuojančių CASE įrankių funkcijos ir jų panaudojimas kuriant modeliais grindžiamos architektūros projektus. Taip pat analizuojami jau esantys sprendimai klasių ir būsenų mašinų suderinamumo užtikrinimui, bei būsenų mašinų patikros ir korektiškumo algoritmai.

Antrajame šio darbo skyriuje apžvelgsime atliktus tyrimai ir apibendrinami analizės rezultatai, ir kokias metodikas taikysime darbe keliamų uždavinių sprendimui.

Trečiajame skyriuje pateikiama projektinė darbo dalis, kuria sudaro sistemos reikalavimų aprašymas kuris atvaizduojamas panaudos atvejų diagramoje, kiekvieno panaudos atvejo detalus aprašymas. Pateikiami sistemai funkciniai ir nefunkciniai kelti reikalavimai, taip pat jie specifikuoti naudojant Volerje šablonus. Taip pat įtrauktas sistemos struktūros, jos parametrų vertinimas.

Ketvirtajame skyriuje aptarsime kaip ir kokios problemos iškilo kūrimo metu, kaip jos buvo sprendžiamos, kaip buvo atliekamas sistemos testavimas.

Penktame skyriuje pateikiami eksperimentinio taikymo pavyzdžiai.

Šeštame skyriuje pateikiami įrankio funkcionalumo įvertinimo rezultatai.

# 1 BŪSENŲ MAŠINŲ PATIKROS IR SUDERINIMO SU KLASIŲ DIAGRAMOMIS METODŲ CASE ĮRANKIUOSE ANALIZĖ

Šiame skyriuje apžvelgsime esamu CASE įrankius ir įvertinsim jų galimybes. Aptarsime esamus būsenų mašinų patikros ir suderinamumo su klasių diagrama įrankius, taip pat jų panaudojimo galimybes.

## *1.1 Informacinių sistemų modelių suderinamumas*

Nors galima atskirai nusakyti kiekvienos diagramos prasmę, kiekviena diagrama yra viso sistemos modelio projekcija. Sistemos modelis dažnai darosi painus dėl to, kad keliose diagramose pateikiama informacija iš dalies sutampa ir aprašo tuos pačius dalykus, tik skirtingais aspektais. Galimos situacijos, kai netinkamai naudojamos įvairios diagramos gali būti nenuoseklios ir nevienareikšmiškai aprašyti sistemą. Dabartiniai UML CASE įrankiai tokiais atvejais nedaug kuo gali padėti projektuotojui.

Yra nemaža bandymų aprašyti formalią UML ar jos dalių semantiką. Elgesį aprašanti UML dalis, t.y. aktyvios klasės ir būsenų mašinos, sudaro problemos branduolį (o tuo pačiu ir sprendimą). Būsenų mašinoms formalizuoti yra siūloma naudoti procesų algebrą [13], struktūrizuotą operacijų semantiką, bendrąją algebrinio specifikavimo kalbą CASL (angl. *Common Algebraic Specification Language*).

Daug žadančiu galima laikyti bandymą aprašyti UML formalias sistemas kaip apibendrintas perėjimų sistemas (angl. *Generalized Labeled Transition Systems – GLTS*) [13], aprašančias sistemos evoliucijos žingsnius kaip perėjimus. Sistema, kurioje vyksta nuoseklūs ar lygiagretūs procesai, susideda iš aktyvių ir pasyvių komponentų. Aktyvūs komponentai yra procesai ir jie gali pereiti iš vienos būsenos į kitą, o pasyvių komponentų (pvz., duomenų bazės objektų) būsena gali keistis tik aktyvaus komponento perėjimo į kitą būseną rezultate.

Apibendrintą perėjimų sistemą GLTS aprašo toks ketvertukas:

*(ST, LAB, INFO, →),*

kur ST, LAB ir INFO yra atitinkamai būsenų, perėjimų žymių ir papildomos perėjimų informacijos (stimulų) aibės, o $\rightarrow$ aprašo perėjimus ir yra tokių aibių Dekarto sandauga:

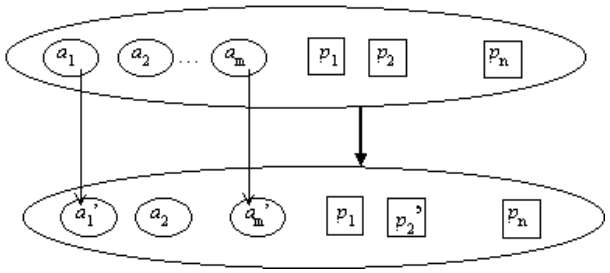
$$\rightarrow \subseteq INFO \times ST \times LAB \times ST$$

Kiekvieną GLTS perėjimą sudaro ketvertas $(i, s, l, s') \in \rightarrow$. Perėjimas dažniausiai yra užrašomas taip: $i: s \xrightarrow{l} s'$

Perėjimas žymimas lanku.

Modeliuodami procesą P, turime perėjimų medį, aprašytą naudojant GLTS, ir to medžio pradinę būseną $s_o \in$ ST. Tokiame medyje perėjimas (lankas) *i:* $s \xrightarrow{l} s'$ reiškia, kad procesas *P* situacijoje s turi galimybę pereiti į situaciją s', atlikdamas perėjimą, pažymėtą $l$ , kur $l$ aprašo aplinkos sąlygas, tinkamas perėjimui įvykti, ir tos aplinkos transformaciją po perėjimo.

1 paveiksle pateikiamas apibendrintas GLTS perėjimo vaizdas. Sistema turi m aktyvių komponentų (aprašytų elipsėmis) ir n pasyvių komponentų (aprašytų kvadratais). Aktyvūs komponentai yra modeliuojami GLTS. Paveikslėlyje vaizduojamame perėjime du aktyvūs komponentai $a_1$ ir $a_m$ pereina į naują būseną, o to rezultate pasikeičia pasyvaus komponento $p_2$ būsena.




**1 pav. Apibendrintas GLTS perėjimas**

Formali GLTS semantika yra aprašoma algebrine kalba CASL-LTL (CASL papildymas GLTS aprašymui), tačiau ši kalba yra sudėtinga, sunkiai skaitoma ir suprantama, specifikacija yra ilga ir nevaizdi.

Kiti bandymai formalizuoti UML susiduria su tokia pačia problema: formalūs modeliai yra labai sudėtingi, o jais remiantis sukurtos specifikacijos – labai ilgos.

Be formalių specifikacijų, yra dar vienas būdas formalizuoti UML semantiką – tai taikyti vaizdinį metamodeliavimą, kuris padėtų užtikrinti informacinės sistemos elgsenos suderinamumą.

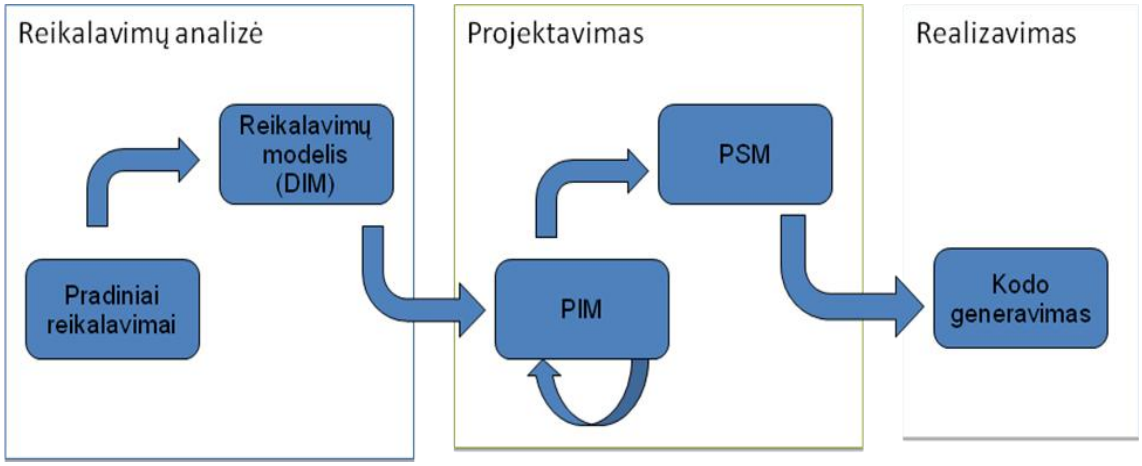
## *1.2 Modeliais paremtas kūrimas*

Informacijos sistemų kūrime svarbus faktorius yra technologija ir platforma, naudojama informacijos sistemos realizavimui. Buvo daug bandymų standartizuoti šią

platformą ir sukurti tokią sistemų kūrimo aplinką, kuri nepriklausytų nuo operacinės sistemos ir programavimo kalbos. Tačiau bandymai sukurti vieningą sistemų kūrimo standartą nepasiekė savo galutinio tikslo. Šiuo metu OMG plėtoja idėją įvesti bendrus standartus ne programavimo, bet aukštesniame, modelių kūrimo, lygmenyje. Ši kryptis pavadinta modeliais grindžiama architektūra MDA (angl. *Model Driven Architecture*) [11].

Modeliai leidžia lengviau suprasti kuriamą sistemą, atlikti geresnius sistemos įgyvendinimo sprendimus. Tokiais atvejais dažnai naudojamos diagramos paremtos grafine notacija UML. MDA pagrindas yra nuo technologijos nepriklausantis sistemos aprašas, kuris sudaromas naudojant unifikuotą modeliavimo kalbą UML (angl. *Unified Modelling Language*). Projektuotojų sudarytas kuriamos sistemos UML aprašas neapribojamas konkrečia programavimo aplinka ar kūrimo platforma. Šis aprašas yra aiškus ir suprantamas visiems projektuotojams ir gali būti įvertinamas, taisomas bei redaguojamas tada, kai pakeitimus padaryti yra paprasčiausia ir pigiausia – tai yra, prieš kuriant programos kodą.

Kaip matome MDA technologija koncentruojasi į nuo platformos nepriklausomų modelių PIM (angl. *Platform Independent Model*) transformavimą į konkrečių platformų modelius PSM (angl. *Platform Specific Model*) ir pastarųjų transformavimą į programos kodą, šių modelių pozicija IS kūrimo procese pateikiama 2 paveiksle.

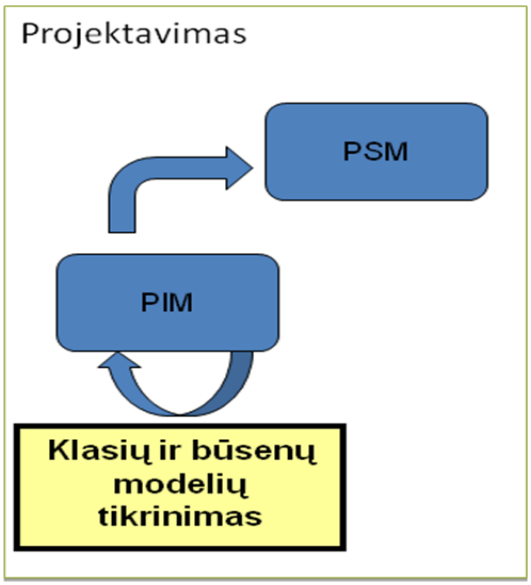



**2 pav. PIM ir PSM IS kūrimo procese**

Šiuo metu labiausiai MDA analizuojama sritis yra transformacija tarp PIM ir PSM modelių, bei UML profilių, reikalingų transformacijoms iš PIM į PSM ir įvairias realizavimo platformas, apibrėžimas.

Šiame darbe bus daugiau koncentruojamasi į projektavimo metu sudaromo PIM modelio sudarymo problemas, tokias kaip modelio korektiškumo užtikrinimas klasių

diagramų ir būsenų mašinų tarpusavyje, taip pat būsenų mašinos korektiškumo ir pilnumo užtikrinimas (3 pav.).




**3 pav. Problemos sritis (PIM) modelis**

Nuo platformos nepriklausančio modelio korektiškumas, išbaigtumas reikalingas tam, kad būtų galima generuoti platformai specifikuotą modelį. Platformai specifikuotas modelis naudojamas generuojat programos kodą.

## *1.3 CASE įrankių apžvalga*

Šioje dalyje apžvelgsime kai kurių CASE įrankių galimybes [19][12][20][22][21][25]. Šios analizės metu buvo stengiamas koncentruotis į savybes susijusias su nagrinėjama problema. Buvo išbandyti šie įrankiai :

- ✓ MagicDraw 14.0,
- ✓ Enterprise Architect 6.5,
- ✓ Visual Paradigm,
- ✓ Altova Umodel 2007,
- ✓ Rational Rose
- ✓ EclipseUml 2007.

Nagrinėjant įrankius, buvo atsižvelgiama tik į savybes, susijusias su nagrinėjama problema – panagrinėtos įrankių galimybės modelio bei atskirų diagramų suderinamumo

patikrinimui, kodo generavimui, įrankio funkcionalumui praplėsti, UML ir OCL palaikymas. Šių įrankių palyginimai pateikti 1 lentelėje. Pasirinktuose įrankiuose naudojamas UML 2.x standartas, palaikomos visų 13 diagramų.

*1 lentelė. Įrankių palyginimo lentelė*

| Įrankio pavadinimas | Savybės | | | | | |
|---|---|---|---|---|---|---|
| | UML 2.1 palaiky-mas | OMG MDA palaikymas | OCL palaiky-mas | Modelių tikrini-mas | XML, WSDL, DB generavimas | Galimybė naudoti įskiepius |
| MagicDraw 14 | + | + | + | +/- | + | + |
| EclipseUML 2007 | + | + | - | - | + | + |
| Enterprise Architect 6.5 | + | + | + | +/- | + | - |
| Altova Umodel 2007 | + | + | + | +/- | + | - |
| Rational Rose | + | - | - | - | + | + |
| Visual Paradingm | + | + | + | +/- | + | + |
| MS Visio | - | - | - | - | +/- | - |

Iš lentelės matome, kad praktiškai visuose nagrinėtuose įrankiuose galima praplėsti jų funkcionalumą (išskyrus UModel 2007, MS Visio ir Enterprise Architect 6.5). Rational Rose (RR), Enterprise Architect (EA) ir MagicDraw (MD) įrankiai yra panašiausi nagrinėjamomis savybėmis.

MagicDraw įrankis yra bene sparčiausiai tobulėjantis, kuriamas Lietuvoje UML įrankis. Baltijos programinė įranga, kurianti šį įrankį, bendradarbiauja su Informacijos sistemų katedra, kurioje vykdomi moksliniai tyrimai siejami su įrankio tobulinimu. Įrankis plačiai naudojamas tiek Lietuvoje, tiek pasaulyje. Tai pagrindinės priežastys, dėl ko buvo pasirinktas būtent šis įrankis. Įrankio kūrėjai teikia pagalbą įrankio vartotojams – per kelias dienas atsako į pateiktus klausimus lietuvių arba anglų kalba. Dar viena šio įrankio pasirinkimo priežastis – kitų projektų vykdymas, kuriant būtent MagicDraw funkcionalumo praplėtimą. Kuriamas projektas vėliau galėtų būti integruojamas su jau sukurtais ir dar kuriamais produktais, kurie praplečia MD įrankio funkcionalumą.

## *1.4 Esami būsenų mašinų tikrinimo metodai*

UML yra pusiau formali kalba, sintaksė ir statinė semantika (modelio elementai, jų tarpusavio ryšiai ir taisyklingumo taisyklės) yra aprašytos formaliai, bet jų dinaminė semantika specifikuota neformaliai. Problema yra ta, kad visa UML yra labai sudėtinga konstrukcija, kuri iš pirmo žvilgsnio atrodo net per daug sudėtinga aplinka, kurioje reikia atlikti formalų verifikavimą. Todėl tikslinga identifikuoti lengvai apdorojamus UML savybių poaibius, kurie leistų racionaliai ir nuosekliai formuoti modelių aprašymą.

**UML tiesioginio modelių vykdymo tikrintojai (UMC)**

UML būsenų diagramų verifikavimui naudojami tiesioginio vykdymo (angl. *on-the-fly*) modelių tikrintojai (UMC) [14]. Toks tiesioginis tikrinimas vykdomas intuityviai, kai modelių tikrintojas patikrina ir validuoja pagal laikiną būsenos formulę, paskui bandoma gražinti rezultatą, stebint vidinius būsenos nustatymus (reikšmes, atributus), ir rekursyviai patikrinamos kitos formulės kitai būsenai. Šiuo būdu (priklausome nuo patikrinimo formulės) tik fragmentas visos būsenos gali būti reikalingas, kurį reikėtų analizuoti, norint gauti reikiamą rezultatą. Logika pagrįsta UMC, $\mu ACTL^{+}$ - tai papildyta $\mu ACTL$ [17] logika, kuri turi pilną μ-skaičiavimą (angl. *μ-calculus*). Ši logika leidžia abu dalykus: specifikuoti įprastus nustatymus, kokius turi atitikti būsena, ir kurti paprastu predikatus su aukštesne logika arba laikinais kintamaisiais.

**UML tiesioginio modelių vykdymo tikrintojo (UMC) prielaidos**

- ✓ Būsenų perėjime, visos veiksmų sekos sudaro „veiksmo dalį“, jos turėtų būti vykdomos kaip nedaloma vienetinė veikla pvz. du lygiagretūs būsenų perėjimai, paleidžiami kartu būsenų mašinos konfigūracijoje, jie negali vienas su kitu sąveikauti, bet vykdomi nuosekliu būdu(tvarka nepaisoma).
- ✓ Duotas modelis sudarytas iš daugiau nei vienos būsenų mašinos, sistemos vystimasis priklauso nuo kiekvienos būsenos mašinos. Pvz. būsenų mašinos perėjimai laikomi vientisais ir nedalomais.
- ✓ Signalų sklidimas viduje būsenų mašinos ir tarp būsenų mašinų, laikomas momentaliu, ir be praradimų.
- ✓ Įvykių eilė asocijuojama su būsenų mašina, yra FIFO(angl. *First In First Out*) eilė.
- ✓ Perėjimų apjungimas visada gerai aprašomas ir statiškai nustatytas.

**UML tiesioginio modelių vykdymo tikrintojo (UMC) ribojimai**

- ✓ Įvykiai: Palaikomi tik asinchroniniai signalai. Kvietimo įvykių, laiko įvykių, pakitimų įvykių vėlavimai nėra palaikomi.
- ✓ Būsenos: Palaikomos būsenos: pradinė, galinė, kombinuotos ir lygiagrečios. Būsenų vidiniai perėjimai ir būsenos įėjimo/išėjimo/vykdymo perėjimai nepalaikomi.
- ✓ Perėjimai: Palaikomi perėjimai šie: vienos krypties, išsišakojantys ir apjungiami. Pradiniai perėjimai turi būti be įvykių; statiškai ir dinamiškai parinkti perėjimai nepalaikomi. Galinis perėjimas negali rodyti į daugiau nei viena galinę būseną.
- ✓ Kita: Sub-mašinos nepalaikomos. Veiksmais gali būti paprasti perdavimai ar signalai. Vienintelis reikšmių, signalų duomenų tipas sudarytas iš sveikųjų skaičių.

**Spin verifikavimo įrankis**

Tai gan populiarus atviro kodo įrankis, naudojamas daugelio žmonių visame pasaulyje, kuris skirtas kuriamų sistemų verifikacijai. Įrankis sukurtas Bell Labs. Įrankis kuriamas nuo 1991 ir vis dar tobulinamas. 2002 metais [15], balandžio mėnesį, buvo apdovanotas programinės įrangos apdovanojimu (angl. *System Software Award*) ACM grupės.

Spin įrankio sistemų verifikavimo galimybės:

- ✓ Spin pagrindinis tikslas verifikuoti sistemą. Įrankis palaiko aukšto lygio kalbas, specifikuojant sistemos aprašymą, tokios kaip PROMELA (angl. *PROcess MEta LAnguage*). Spin naudojamas patikrinti loginio dizaino klaidas esančias sukurtam sistemos modelyje, tokias kaip operacinės sistemos, duomenų perdavimo protokolų, sistemų keitimo, algoritmų sutapimą ir kita. Įrankis taip pat patikrina specifikacijos logiką. Jis aptinka mirties taškus, nespecifikuotas sąlygas, vėliavų klaidas ir t.t.
- ✓ Spin palaiko taip pat įterptinį C kalbos kodą, kaip dalį modelio specifikacijų. Tai leidžia tiesiogiai patikrinti programos specifikacijos interpretavimą.
- ✓ Spin veikia tiesioginio vykdymo (angl. *on-the-fly*), tai leidžia išvengti globalaus būsenų grafo konstravimo.

Norint patikrinti formalius modelius naudojama PROMELA įvedimo kalba.

**OCLE verifikavimo įrankis**

OCLE tai yra sutrumpinimas (angl. *Object Constraint Language Environment*). OCLE [16] buvo sukurtas atlikti modelių patikrinimui, UML statinė semantika išreiškiama naudojant objektų apribojimo kalba (OCL) ji turi daug semantinių, sintaksinių ir abstrakčių klaidų. Naudojant šį įrankį, lengvai galima aptikti klaidas kuriamame modelyje.

OCLE įrankio verifikavimo galimybės:

- ✓ Tikrinant UML modelių taisyklingumą (angl. *Well Formedness Rules*), gali aptikti daug projektuotojo padarytų klaidų;
- ✓ taip pat gali patikrinti ir metodologines taisykles, vaizdavimo taisykles išreiškiamas OCL kalboje modelio ir metamodelio lygyje;
- ✓ galima gauti visas UML modelio metrikas;
- ✓ tai pilnai nepriklausomas įrankis nuo CASE įrankių kuriais, kuriami grafiniai modeliai, tik juos reikia peteikti XMI formate.

Taigi šis įrankis taip pat kaip ir Spin reikalauja, kad diagramos būtų transformuojamos į tam tikrą aprašą (XMI. standartu). Šį aprašą generuoja daugelis UML CASE įrankių, kurie palaiko OMG standartus, todėl perėjimas nėra problema, tačiau yra problematiškas, prailgina projektavimo procesą ir projektuotojai priversti jo atsisakyti.

Taigi kuriant CASE įrankiais UML diagramas ir norint patikrinti jų korektiškumą tenka konvertuoti į tam tikras kalbas, kurias supranta verifikavimo įrankiai.

**Klasių diagramų suderinamumas su būsenų mašinomis**

Klasių ir būsenų mašinų suderinamumas nėra realizuotas nei viename nagrinėtame įrankyje, taigi toks patikrinimas galima sakyti yra unikalus.

## *1.5 Analizės išvados*

1. Klasių ir būsenų mašinų suderinamumo ir būsenų mašinos patikros ir pilnumo tikrinimas yra aktualus kuriant modeliais paremtos architektūros modelius.
2. Nagrinėtuose CASE įrankiuose yra galimybė atlikti MDA transformacijas, tačiau klasių ir būsenų mašinų suderinamumo, būsenų mašinos patikros ir pilnumo tikrinimo galimybės nei vienas įrankis neturi.
3. Norint tikrinimui naudoti nesuderintus su CASE įrankiais programinius paketus, (pvz., Spin) reikėtų atlikti modelių konvertavimą į specifines kalbas, tačiau

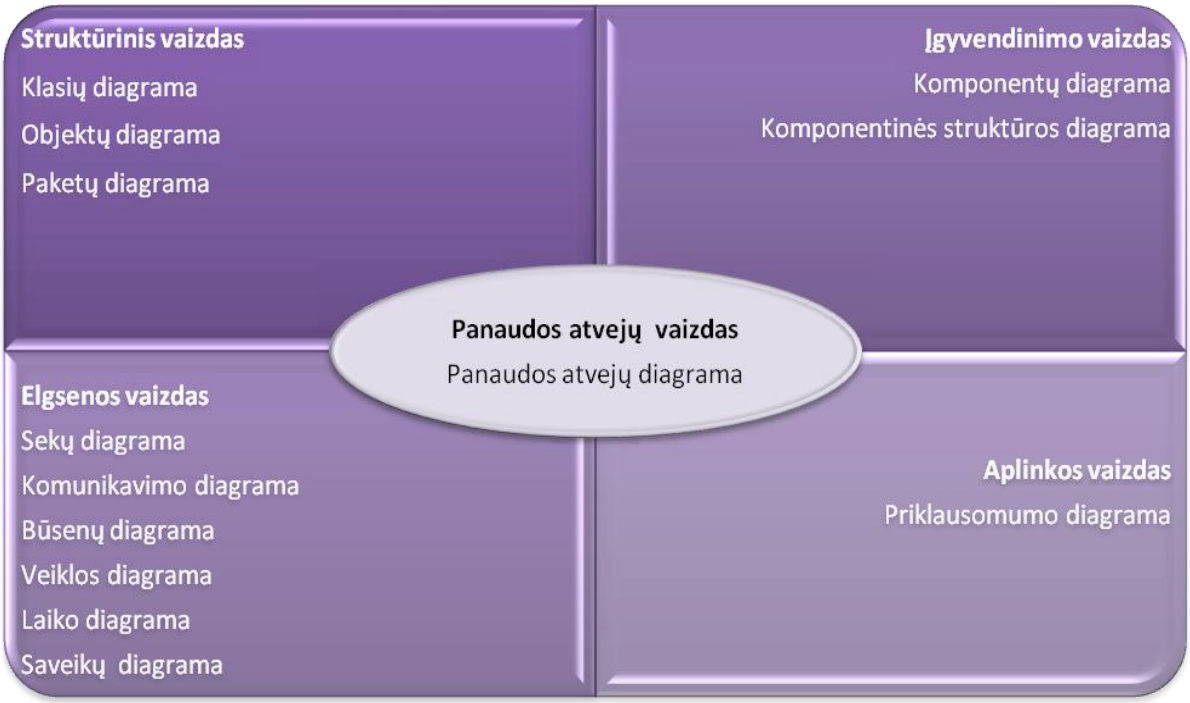
tokios galimybės nėra realizuotos, o rankiniu būdu toks tikrinimas praktiniuose projektuose truktų per ilgai ir būtų per brangus.

4. Tikrinimo algoritmų įgyvendinimui būtų naudinga praplėsti esamų CASE įrankių funkcionalumą, o ne kurti naujus įrankius.

# 2 BŪSENŲ MAŠINŲ PILNUMO, NEPRIEŠTARINGUMO IR SUDERINAMUMO SU KLASIŲ MODELIU TIKRINIMO METODIKA

Šioje darbo dalyje apžvelgiami atlikti tyrimai ir apibendrinami analizės rezultatai, taip pat pateikiama sukurtoje programų sistemoje naudojama tikrinimo metodika, jos apribojimai ir galimybės.

UML yra pagrindinis objektinių sistemų projektavimo aprašas, kuris plačiai naudojamas sistemų architektų visame pasaulyje. Šiuo metu UML turi trylika modeliavimo diagramų (4 pav.), kurios sistemą vaizduoja įvairiais aspektais. UML CASE įrankio MagicDraw sistemų modeliavimo aspektai pateikiami 4 paveiksle [7]: (panaudos atvejų vaizdas (angl. *Use Case View*), struktūrinis sistemos vaizdas (angl. *Structural View*), realizacijos vaizdas (angl. *Implementation View*), elgsenos vaizdas (angl. *Behavioral View*), aplinkos vaizdas (angl. *Environment View*).




**4 pav. UML kalba aprašomi sistemos vaizdai**

Vis dėlto UML nėra formali kalba, ja aprašomų modelių korektiškumas nėra užtikrinamas. Šioje darbo dalyje nagrinėjamas su dinamika susijęs sistemos vaizdas, t.y. būsenų mašinos, jų susiejimas su struktūriniu sistemos vaizdu – klasių diagrama, jų bendri elementai ir kaip jie atsispindi skirtingose diagramose.

## 2.1 Klasių diagramos

Klasių diagrama tai statinis sistemos vaizdas. Joje pateikiama klasių ir jų sąsajų (angl. *Interface*) vidinė struktūra ir ryšiai tarp šių elementų. Klasės vaizduoja objektų tipus, kurie naudojami sistemoje. Klasių diagramoje nepateikiama jokia su laiku susijusi informacija, joje aprašoma tik klasifikacija. Šie tipai (objektai) sukuriami tik sistemos vykdymo metu ir jie atvaizduojami objektų ir sąveikos diagramomis.

Klasės tarpusavyje gali būti susietos tam tikrais ryšiais: asociacijomis (angl. *Association*) (sujungtos viena su kita), priklausomybėmis (angl. *Dependency*) (viena klasė priklauso nuo kitos klasės), specializacijomis (angl. *Specialized*) (viena klasė yra kitos poklasis) ir supakuojamos (angl. *Packaged*) (sugrupuojamos kartu kaip vienas vienetas).

Sistema dažniausiai turi keletą klasių diagramų – ne visos klasės pateikiamos vienoje klasių diagramoje. Klasė gali turėti daug prasmių lygių ir gali būti vaizduojama keliose klasių diagramose.

## 2.2 Būsenų mašinos

Klasės objektų elgsena gali būti aprašoma būsenų ir įvykių elementais, tokių elementų diagrama ir vadinama būsenų mašinų diagrama (angl. *State machine diagram*). Būsenų mašina – objekto būsenų seka, kurias objektas įgyja įvykus tam tikriems įvykimas viso jo gyvavimo metu.

Dažniausiai būsenų diagrama aprašoma klasės elgsena, bet taip pat būsenų diagrama gali būti aprašoma kitų modelio esybių elgsena, tokių kaip panaudos atvejų, sistemos veikėjų (aktorių), operacijų ar metodų.

Būsenų diagrama yra grafas, kuriuo pateikiama būsenų mašina. Būsenos ir pseudobūsenos grafe vaizduojami būsenų ir pseudobūsenų simboliais, perėjimai pateikiami rodyklių simboliais, kurie jungia būsenas. Reikia pastebėti, kad kiekviena būsenų mašina turi pradinę būseną, iš kurios pasiekiamos kitos būsenos.

Būsenų mašinomis UML kalboje, dažniausiai aprašomos sistemos kontrolerių (angl. *Managers*) būsenos.

## 2.3 Būsenų mašinų tikrinimo taisyklės

Būsenų mašina pateikia universalias notacijas, leidžiančias lanksčiai atspindėti elgsenos subtilybes. Tačiau išraiškos priemonių įvairovė neretai sąlygoja semantinių netikslumų atsiradimą, galinčių apsunkinti modeliuojamos dalykinės srities suvokimą.

Būsenų mašina turi užtikrinti saugumo ir gyvybingumo kriterijus. **Saugumo kriterijus** reiškia, kad iš kiekvienos būsenos galima pasiekti galinę būseną. **Gyvybingumo kriterijus** reiškia, kad bent viena galinė būsena yra tikslo būsena (o ne vien tik klaidos ar nepageidaujamos būsenos).

Būsenų mašina yra grafas, kurio viršūnės yra būsenos, galinčios sietis tarpusavyje hierarchiniais ryšiais ir kurios yra sujungtos orientuotomis briaunomis. Būsena vadinama *aktyvia*, jeigu sistema duotu laiko momentu yra toje būsenoje. Aktyvios būsenos priklauso esamai vykdymo *konfigūracijai* (konfigūraciją apibrėšime kiek vėliau). Būsena *s* gali turėti hierarchiškai pavaldžias būsenas, vadinamas *vaiko būsenomis*. Jeigu *vaiko* būsenos *s* būseną pažymėsime *s'*, tai *s* vadinsime būsenos *s' tėvo būsena*. Vizualiai vaiko būsenos pateikiamos vaizduojant vieną būseną kitos viduje. Būsenos *s palikuoniais* vadinamos visos būsenos *s vaiko* būsenos, tų *vaiko* būsenų *vaiko* būsenos ir t. t. [5].

Jeigu būsena *s* turi *vaiko* būsenų, ją vadinsime *sudėtine būsena*. Priešingu atveju tai *paprastoji būsena*. Būna dviejų rūšių sudėtinės būsenos: *AND būsenos ir OR būsenos*. AND būsenos skirtos vaizduoti lygiagretumui ir joms galioja tokia taisyklė: jeigu *s* yra AND būsena ir *s* yra aktyvi, tai aktyvios yra ir visos jos *vaiko* būsenos. OR būsena žymi išskirtinį pasirinkimą: jeigu *s* yra OR būsena ir *s* yra aktyvi, tai viena ir tik viena būsenos *s* vaikinė būsena yra taipogi aktyvi. Apibrėžtumo dėlei pareikalaukime, kad būsenų mašinos aukščiausio lygmens būsena butų OR būsena; šią būseną mes vadinsime *šaknine būsena*.

Orientuota briauna atitinka *perėjimą* iš vienos būsenos (*šaltinio*) į kitą būseną (*tikslą*). Perėjimas gali turėti sąlygos išraiškas ir veiksmo išraiškas. Sąlygos išraiškos yra loginės išraiškos, kurios gali operuoti kintamaisiais ir konceptų būsenomis. Veiksmo išraiška yra vieno arba daugiau elementarių veiksmų seka. Tam, kad būtų vykdomas perėjimas iš pradinės būsenos į tikslo būseną, turi būti aktyvi perėjimo šaltinio būsena, turi įvykti įvykis, nurodytas perėjimo žymėje ir turi būti tenkinama sąlygos išraiška. Perėjimo metu pirmiausia paliekama šaltinio būsena, taigi ji tampa neaktyvi, po to vykdomi veiksmai, pateikti perėjimo signatūroje, galiausiai suaktyvinama tikslo būsena.

Būsenų aibę vadinsime *konfigūracija*, jei ji tenkina šiuos reikalavimus:

1. Šakninė būsena visada priklauso konfigūracijai.
2. Jeigu būsena priklauso konfigūracijai, tai konfigūracijai priklauso ir jos *tėvo* būsena.
3. Jeigu OR būsena priklauso konfigūracijai, tai būtinai viena ir tik viena jos *vaiko* būsenų taip pat priklausys konfigūracijai.

4. Jeigu AND būsena priklauso konfigūracijai, tai ir visos jos *vaiko* būsenos priklauso konfigūracijai.

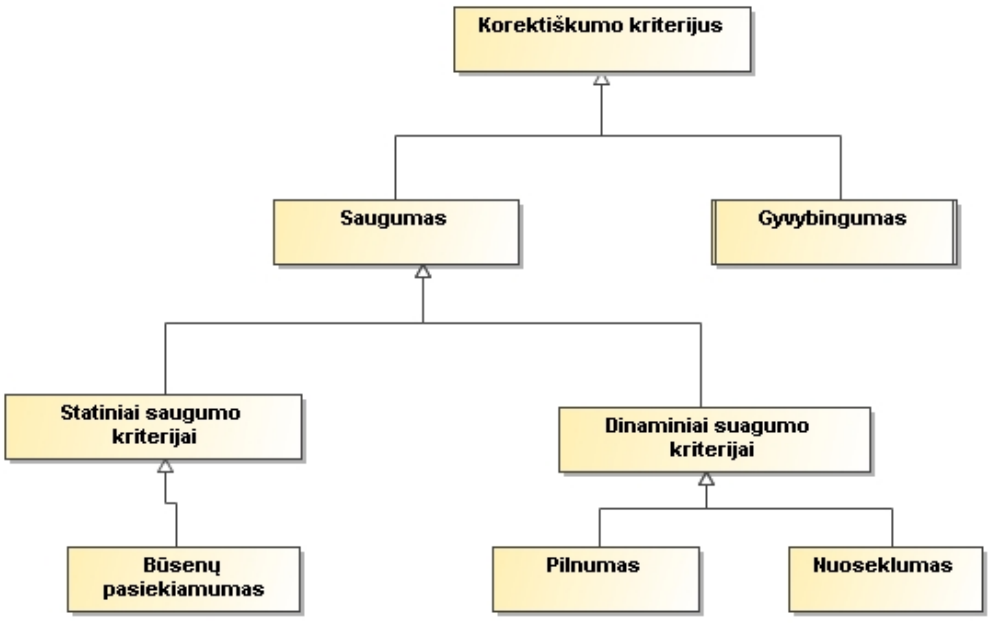
Sistemos konfigūracija gali kisti, vykdant perėjimus. Konfigūracijos pakitimai yra sąlygojami pakitimų, vykstančių sistemos aplinkoje. Aplinkos pokytis, kuris modeliuojant yra reikšmingas, vadinamas *įvykiu*.

Perėjimas, kuris konkrečiu laiko momentu gali būti vykdomas, vadinamas *galimu* (angl. *Enabled*). Perėjimas yra galimas tada ir tik tada, kai jo šaltinio būsena priklauso konfigūracijai, įvyksta įvykis, nurodytas perėjimo žymėje, ir sąlygos reiškinys yra teisingas. Kiekvienai briaunai *e* galima rasti mažiausią OR būseną, talpinančią tiek šaltinio, tiek ir tikslo būsenas. Tokia OR būsena vadinama briaunos *e kontekstu* (angl. *Scope*). Du galimi perėjimai yra *nesuderinami*, jeigu sutampa jų kontekstai, arba vienas jų yra kito palikuonis. Du nesuderinami įvykiai negali būti vykdomi tuo pačiu laiko momentu, nes tada nebūtų galima vienareikšmiškai apibrėžti, kokia bus konfigūracija, perėjimui įvykus.

## 2.4 *Būsenų mašinų korektiškumo kriterijai*

Bendriausi būsenų mašinų korektiškumo kriterijai yra saugumas (angl. *Safety*) ir gyvybingumas (angl. *Liveness*). Neformaliai šiuos kriterijus (5 pav.) galima apibrėžti taip:

- ✓ saugumas užtikrina, kad, esant bet kokiai įvykių sekai „neatsitiks nieko nepageidaujamo“.
- ✓ gyvybingumas garantuoja, kad „tai, ko laukiama, galiausiai įvyks“, t. y. sistema kada nors būtinai pasieks savo tikslinę būseną.




**5 pav. Būsenų mašinų korektiškumo kriterijų klasifikacija**

Būsenų modelio **statiniai korektiškumo kriterijai** – tai bendri reikalavimai, užtikrinantys sintaksinį elgsenos pateikimo teisingumą. Čia pateiksime du statinius korektiškumo kriterijus:

- ✓ **Būsenų pasiekiamumas** – šis kriterijus reikalauja, kad visos būsenų modelio būsenos būtų pasiekiamos, kitaip tariant, negali būti izoliuotų būsenų (neturinčių įeinančių perėjimų).

**Dinaminiai korektiškumo kriterijai** užtikrina sistemos elgsenos apibrėžtumą, besikeičiant aktyviai būsenai. Suformuluosime taip pat du dinaminio korektiškumo kriterijus:

- ✓ **Pilnumas**. Pilnumas užtikrina tai, kad kiekvienai leidžiamai įvykių sekai yra vienareikšmiškai apibrėžta veiksmų seka. Pilnumas reikalauja, kad *visų perėjimų, sužadinamų įvykus tam pačiam įvykiui e, vykdymo sąlygos sudarytų tautologiją* t. y., vienu momentu gali būti teisinga tik viena iš to įvykio perėjimų sąlygų. Formaliai šį reikalavimą galima užrašyti taip: $c_1 \vee c_2 ... \vee c_n = true$ (čia $\vee$ yra XOR operacija). T.y. tik vienos iš reiškinyje esančių $c_i$ reikšmė gali būti *true*; čia $c_i$ – perėjimo sąlygos reiškinys. Pilnumo kriterijų galima išreikšti ir tokiu reikalavimu:
  Jeigu tarp būsenų $s_i$ ir $s_j$ egzistuoja perėjimas $e_1$, susietas su įvykiu ir turintis nurodytą sąlygą $c_1$, tai tarp būsenų $s_i$ ir $s_j$ turi egzistuoti ir perėjimas $e_2$, susietas su tuo pačiu įvykiu I ir turintis sąlygą $c_2$, ekvivalenčią sąlygai $\overline{c_1}$.

  Tai išplaukia iš dviejų kitų reikalavimų:

  I. Būsenų modelyje turi būti aiškiai **apibrėžta sistemos elgsena** tiek, kai įvykio sąlyga yra įvykdoma ir tiek, kai neįvykdoma.
  II. Esant keliems perėjimams, susietiems su vienu įvykiu, **bent vieno perėjimo sąlyga įvykio metu turi būti įvykdoma**.

  Apibendrintai šį reikalavimą galima užrašyti taip:

  $\bigcup_{i=2}^{n} c_i = \overline{c_1}$ ši formulė išvedama iš $c_1 \vee c_2 ... \vee c_n = true$ ir iš matematinėje logikoje suformuluoto "trečio negalimo dėsnio" teigiančio kad, $A \vee \overline{A} = true$.

- ✓ **Nuoseklumas**. Nuoseklumas garantuoja reikalavimų neprieštaringumą ir elgsenos determinizmą. **Nuoseklumas reikalauja**, kad įvykus įvykiui, tuo pačiu metu būtų vykdomi tik tie perėjimai, kurie yra suderinami. Taip pat to paties įvykio sąlygojamų perėjimų, priklausančių tam pačiam kontekstui, vykdymo sąlygos turi būti poromis

nesikertančios: $c_i \wedge c_j = false, \forall j \neq i$ kai $i = 1,2,...,n; j = 1,2,...,n$. Šis apribojimas išplaukia iš reikalavimo, kad vienu metu galima vykdyti tik vieną perėjimą.

## 2.5 *Algoritmas būsenų mašinų ir klasių suderinamumui tikrinti*

Norint užtikrinti būsenų mašinos korektiškumą, reikia ją vaizduoti kanonine forma. Būsenų mašinos sudaromos kiekvienai esybei, perėjimuose nurodant valdiklių operacijas, ir kiekvienam valdikliui. Kiekvienai valdiklio operacijai, išskyrus *Query,* sukuriamas perėjimas su tuo pačiu vardu, kaip ir operacija. Gali būti, kad esybė turi tik vieną būseną, bet vis tiek ją reikia pavaizduoti. Gali būti, kad valdiklio būsenos tarpusavyje neturi ryšio.

Jei operacijos metodas aprašomas būsenų mašina, jos kontekstas yra tos operacijos klasė. Jei ši būsenų mašina nurodoma kaip efektas, efekto specifikacijoje nurodoma operacija, kurios metodą vaizduoja ta būsenų mašina.

Būsenų mašinos ypač svarbios valdikliams, kai jų operacijos tarpusavyje susijusios ir sudaro būsenų mašiną. Tokį valdiklį galima pavaizduoti sudėtine operacija (pvz., veiklos transakcija).

Atominių operacijų metodus taip pat galima pavaizduoti būsenų mašinomis, bet jų elementai bus jau ne operacijos, bet paprasti veiksmai. Veiksmų tipus apibrėžia UML veiksmų semantika.

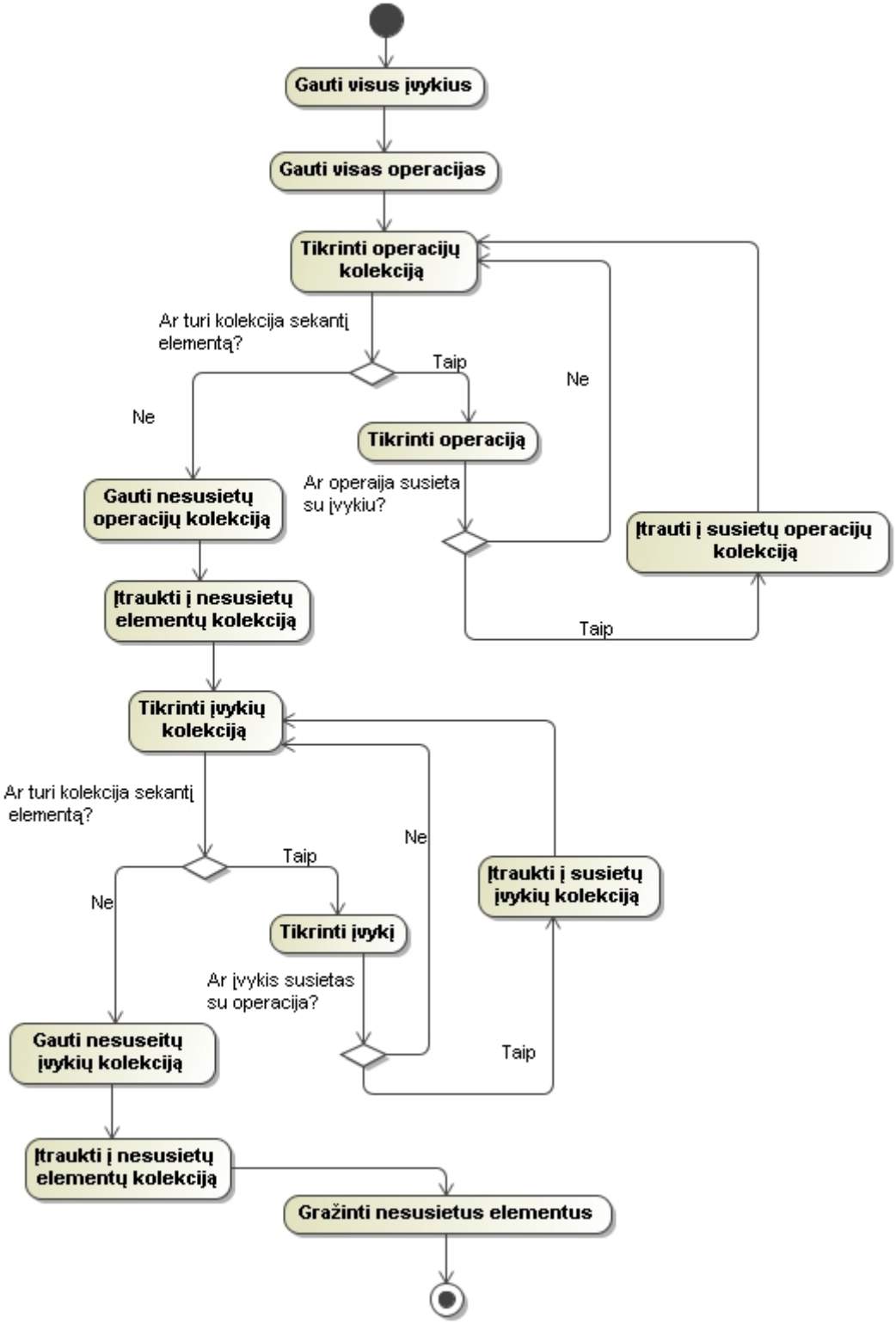
Perėjimų sąlygas reikia vaizduoti pasirinkimo (*Choice*) būsenomis. Kiekvienas *Choice* turi turėti ne daugiau kaip vieną perėjimą su tipine sąlyga [*else*]. Geriausia, kad visi *Choice* turėtų po du išeinančius perėjimus, tada galima garantuoti, kad yra kelias tam atvejui, kai sąlyga tenkinama ir kai netenkinama.

Jei A valdiklio operacijos parametras yra B esybė, iš A valdiklio į B esybę klasių diagramoje turi būti ryšys. Jei A valdiklio operacija yra nurodyta B valdiklio būsenų mašinos perėjime kaip sužadinantis įvykis, klasių diagramoje iš A valdiklio turi būti ryšys į B. Jei A valdiklio operacija yra nurodyta B valdiklio būsenų mašinos perėjime kaip efektas, klasių diagramoje iš B valdiklio turi būti ryšys į A.

Tikrinimas atliekamas pakete. Jo viduje gali būti kiti paketai. Kiekvienos būsenų mašinos įvykiai (operacijų kvietimai, signalai ir t.t.) ir kviečiamos operacijos turi būti aprašyti jos pakete arba aukštesniame pakete, kuris priklauso tikrinamam paketui. Negali būti už tikrinamo paketo ribų arba to paties lygio pakete.

Jei perėjimą sužadina operacijos kvietimas, pasirenkama ta operacija.

Jei perėjimo metu reikia iškviesti dar vieną operaciją arba jei perėjimą sužadina laiko, pasikeitimo ar signalo įvykis, efekte nurodoma kviečiamos arba sužadinamos operacijos būsenų mašina ir operacijos vardas (kuris turi sutapti su būsenų mašinos vardu). 6 paveiksle pateikiamas grafinis algoritmo vaizdas.




**6 pav. Būsenų mašinų ir klasių suderinamumo tikrinimas**

Klasių ir būsenų mašinų suderinamumo patikrinimo algoritmas:

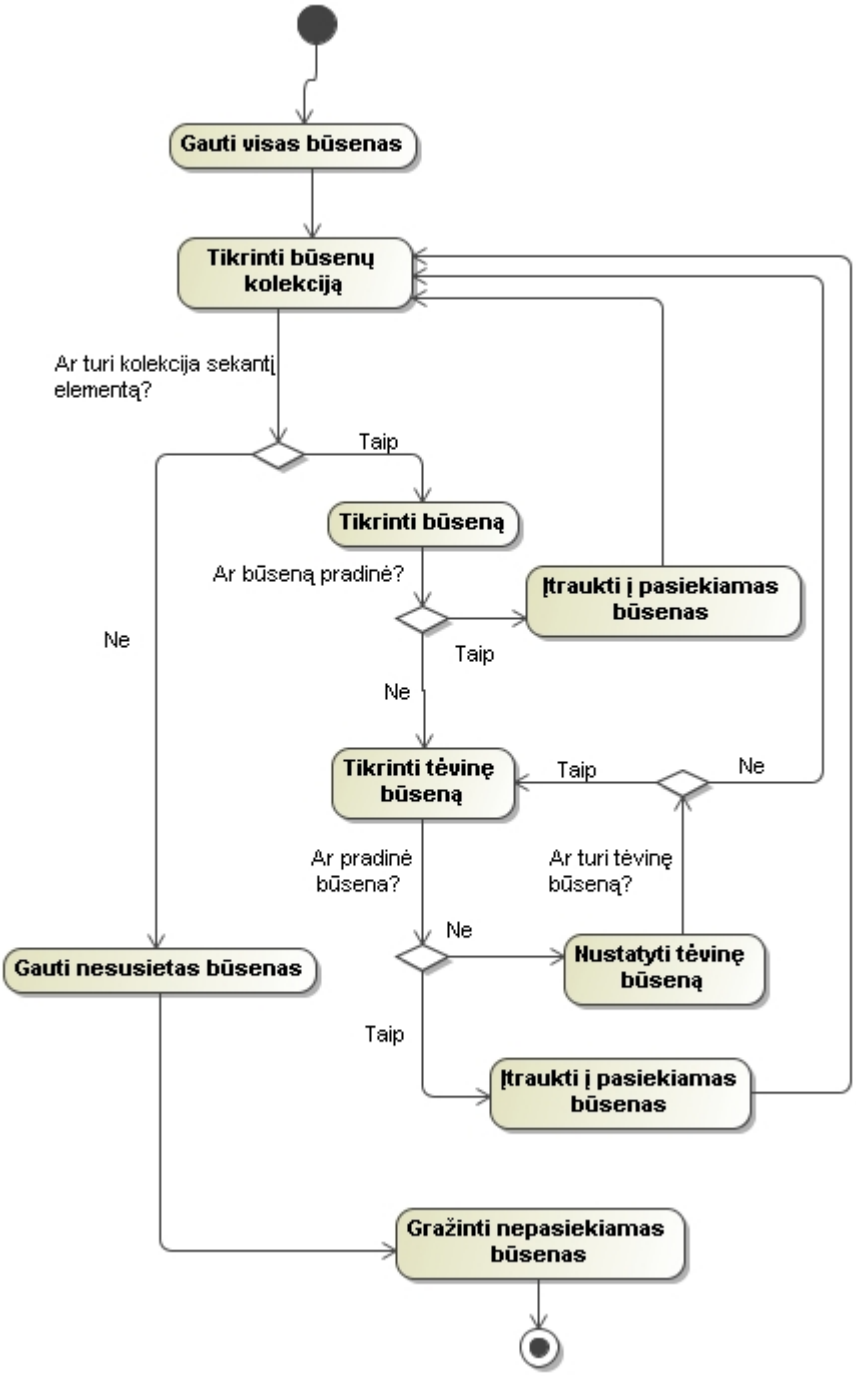
```
metodas patikrintiSuderinamumą(Paketas paketas)
{
        visiĮvykiai:= paketas.gautiĮvykius();
        visisosOperacijos :=paketas.gautiOperacijas();
        susietiElementai := ∅;
        foreach(Įvykis  i ∈ visiĮvykiai)
        {
                if (i.GautiTipa = CallEvent)
                        if (visisosOperacijos .Turi(i.susietaOperacija))
                                susietiElementai.add(i);
        }
        nesusietiElementai:= visiĮvykiai – susietiElementai;
        susietiElementai := ∅;
        foreach(Operacija o ∈ visisosOperacijos)
        {
                if (visiĮvykiai.arĮvykisSusietaSuOpracija(o.Pvadinimas, CallEvent))
                susietiElementai.add(i);
        }
        nesusietiElementai:= nesusietiElementai + visisosOperacijos- susietiElementai;
        return nesusietiElementai;
}
```

## *2.6 Algoritmas būsenų mašinų pasiekiamumui tikrinti*

Šio algoritmo tikslas - surasti ar visos būsenos yra pasiekiamos. Reikia patikrinti ar visos būsenos pasiekiamos iš pradinės būsenos, gražinti tas būsenas kurios nepasiekiamos (7 pav.).

```
metodas gautiNepasiekiamasBūsenas(Kolekcija visų BūsenųKolekcija);
{
        visosBūsenos := visų BūsenųKolekcija;
        pasiekiamosBūsenos := ∅;
        foreach(Būsena b ∈ visosBūsenos)
                {
                        if (!pasiekiamosBūsenos.Turi(b) AND b.gautiTipa()≠PradinėBūsena)
                        {
                                if( patikrintiArPasiekimaPradinėBūsena(b))
                                        pasiekiamosBūsenos.add(b);
                        }
                        else if (b.gautiTipa()=PradinėBūsena)
                                pasiekiamosBūsenos.add(b);
                }
        nepasiekiamosBūsenos:= visosBūsenos – pasiekiamosBūsenos;
        return  nepasiekiamosBūsenos;
}
metodas patikrintiArPasiekimaPradinėBūsena(Būsena b)
{
        if (b.turiTėvinęBūseną())
        {
                if (b.turiVienąTėvinęBūseną)
                        if (b.Tėvinė.gautiTipą= PradinėBūsena)
                                return true;
                        else
```

```
                    patikrintiArPasiekimaPradinėBūsena(b.Tėvinė);
            else
                    foreach(Būsena b ∈b.gautiTėvines())
                    {
                            patikrintiArPasiekimaPradinėBūsena(b);
                    }
        }
        return false;
}
```




**7 pav. Būsenų pasiekiamumo tikrinimas**

## 2.7 *Algoritmas būsenų mašinų pilnumui ir nuoseklumui tikrinti*

Šio algoritmo realizacijai reikia įvesti tam tikrus apribojimus.

Du vienam kontekstui priklausančius perėjimus vadinsime *panašiais*, jeigu juos sužadina tas pats įvykis. Algoritmu tikrinsime panašių perėjimų sąlygų nesikertamumą. Taip pat tikrinsime, ar įvykus įvykiui, visada bus teisinga viena kuri iš tų sąlygų.

Šio algoritmo tikslas gražinti nekorektiškai specifikuotų perėjimų aibę.

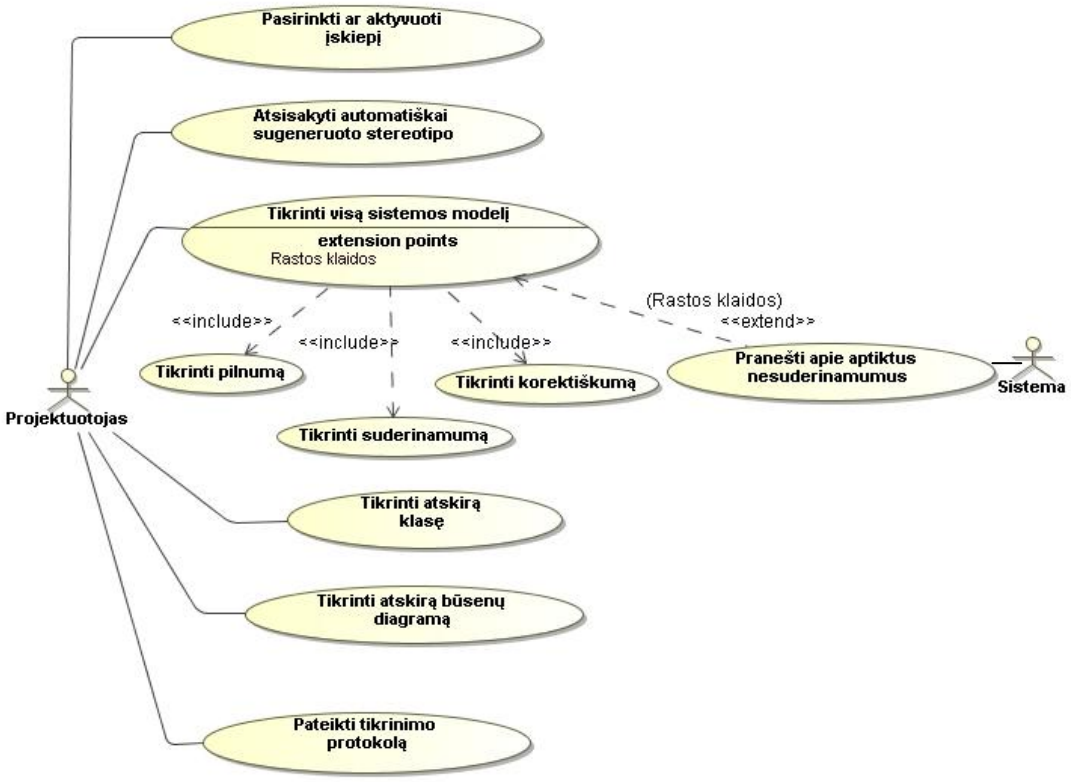
```
metodas dinamikosTikrinimas(BūsenųAibė)
{
        Korektiški := ∅;
        Stekas := ∅;
        b := PradinėBusena(BūsenųAibė);
        Stekas.push(b);
        aplankytosBūsenos := aplankytosBūsenos ∪ {b} ;

        repeat
            b := Stekas.pop
            if (gautiVaikus(b) ≠ ∅)
                    if (AND_būsena(b))
                        foreach( v ∈ gautiVaikus(b))
                        {
                                Korektiški := Korektiški ∪ (visiVaikai(v) - dinamikosTikrinimas
                    (visiVaikai(v)) );
                        }
                    else
                        Korektiški := Korektiški ∪ (visiVaikai(s)- dinamikosTikrinimas (visiVaikai (s))
                    );

            pan := gautiPanašius(b);
            if (pan = ∅)
                 Korektiški := Korektiški ∪ gautiIšėjimus(b);
            else
                 if(nesikertaPoromis(gautiSąlygas(pan)) AND bentViena(gautiSąlygas(pan)) )
                        Korektiški := Korektiški ∪ gautiIšėjimus(b);

            foreach( k ∈  gautiKaimynui(b))
            {
              if (k ∉ aplankytosBūsenos)
                  {
                        Stekas.push(k);
                       Aplankytos := Aplankytos ∪ {k};
                  }
            }

        until Stekas = ∅
        Nekorektiški := BūsenųAibė – Korektiški ;
        return Nekorektiški;
}
```

# 3 MAGICDRAW ĮRANKIO IŠPLĖTIMO PROJEKTAS

Šiame skyriuje pateikta informacija apie kuriamos sistemos numatytus panaudojimo atvejus, jo architektūrinius sprendimus, keliamus funkcinius ir nefunkcinius reikalavimus, naudojamas duomenų struktūras, realizacines galimybes ir apribojimus. Kadangi sistema realizuojama MagicDraw aplinkoje taip pat bus pateikiami aptikti MagicDraw *OpenAPI* [8] galimybių aprašymai. Kūrimo aplinkos ir įskiepio suderinamumo su MagicDraw CASE įrankiu ypatumai. Sukurto įrankio kodinis pavadinimas CSMChecker (Class StateMachine Checker).

## *3.1 Sistemos panaudojimo atvejai*

Iš užsakovo pateiktų reikalavimų buvo sudaryti sistemos panaudojimo atvejai, jų diagrama pateikiama 8 paveiksle. Žemiau pateiktas panaudojimo atvejų sąrašas.




**8 pav. Panaudojimo atvejų diagrama**

**Panaudojimo atvejų specifikacijos:**

1. PANAUDOJIMO ATVEJIS: Pasirinkti ar aktyvuoti įskiepį

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Modeliavimo pradžioje projektuotojas pasirenka ar aktyvuoti įskiepio papildomą funkcionalumą.

**Prieš sąlyga:** Paleidžiamas Magic Draw CASE įrankis.

**Sužadinimo sąlyga:** Pranešimo lentelėje pasirenkame „Yes“

**Po-sąlyga:** Aktyvuotas patikros komponentas, kuris praplečia MagicDraw funkcionalumą .

2. PANAUDOJIMO ATVEJIS: Atsisakyti automatiškai sugeneruoto stereotipo

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Vartotojas kuria būsenų diagramą susietą su klasių diagrama, naudodamas papildomus meniu punktus

**Prieš sąlyga:** Turi būti aktyvuotas CSM įskiepis.

**Sužadinimo sąlyga:** Spaudžiame „Undo“ MagicDraw įrankių juostoje.

**Po-sąlyga:** Panaikintas „csm“ stereotipas.

3. PANAUDOJIMO ATVEJIS: Tikrinti pilnumą

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Vartotojas kuria sistemos modelį.

**Prieš sąlyga:** Turi būti aktyvuotas CSM įskiepis

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas ant modelio paketo “Check Completenes“

**Po-sąlyga:** Pilnumas patikrintas, pateiktas tikrinimo protokolas.

4. PANAUDOJIMO ATVEJIS: Tikrinti suderinamumą.

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Vartotojas kuria sistemos modelį.

**Prieš sąlyga:** Turi būti aktyvuotas CSM įskiepis

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas ant modelio paketo“Check Compabtibility“

**Po-sąlyga:** Suderinamumas patikrintas, pateiktas tikrinimo protokolas

5. PANAUDOJIMO ATVEJIS: Tikrinti korektiškumą

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Vartotojas kuria sistemos modelį.

**Prieš sąlyga:** Turi būti aktyvuotas CSM įskiepis

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas ant modelio paketo“Check Correctness“

**Po-sąlyga:** Korektiškumas patikrintas, pateiktas tikrinimo protokolas

6. PANAUDOJIMO ATVEJIS: Pateikti tikrinimo protokolą

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Tikrinamas korektiškumas kuriamų diagramų, apie aptinkamus nesuderinamumus informuojamas vartotojas ir pateikiamas nesuderinamumo aprašas. Viskas kaupiama klaidų protokole.

**Prieš sąlyga:** Turi būti aktyvuotas susietų diagramų kūrimas

**Sužadinimo sąlyga:** Pasirenkamas tikrinimas.

**Po-sąlyga:** Gautas klaidų pranešimų ir klaidų aprašymų sąrašas.

7. PANAUDOJIMO ATVEJIS: Tikrinti atskirą klasę

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Tikrina, ar klasės metodai panaudoti būsenų mašinoje.

**Prieš sąlyga:** Turime jau sukurtą pilną būsenų mašinos diagramą.

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas „CSM Check Class “

**Po-sąlyga:** Atlikta patikrinimo operacija, visos aptiktos klaidos užregistruotos ir pateiktos ekrane.

8. PANAUDOJIMO ATVEJIS: Tikrinti atskirą būsenų diagramą

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Atlieka būsenų mašinos ir jos subbūsenų korektiškumo ir pilnumo patikrinimą.

**Prieš sąlyga:** Turime jau sukurta pilną būsenų mašinos diagramą.

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas „CSM Check StateMachine“

**Po-sąlyga:** Atlikta patikrinimo operacija, visos aptiktos klaidos užregistruotos ir pateiktos ekrane.

9. PANAUDOJIMO ATVEJIS: Pateikti tikrinimo protokolą

**Vartotojas/Aktorius:** Projektuotojas

**Aprašas:** Pateikiamas protokolas apie kūrimo procese aptiktas klaidas, apie aptiktus nesuderinamumus būsenų ir klasių modeliuose.

**Prieš sąlyga:** Turi būti aktyvuotas CSM įskiepis.

**Sužadinimo sąlyga:** Pasirenkamas meniu punktas „Show Check Protocol“

**Po-sąlyga:** Ekrane gaunamas visų derinimo metu užfiksuotų klaidų protokolas.

## *3.2 Funkciniai reikalavimai sistemai*

Šioje darbo dalyje pateikiami sistemos funkciniai reikalavimai, kurie buvo taikomi sistemos kūrimo metu, jų aprašymas pateikiamas formose. Funkciniai sistemos reikalavimai nurodo, ką sistema turi sugebėti daryti.

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 1 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 1/1 |
| Aprašymas: | Sistema turi informuoti apie atliktus neteisingus veiksmus. | | | | |
| Pagrindimas: | Vartotojas turi būti informuojamas apie neteisingai atliekamus veiksmus, bei turi gauti tų klaidų aprašymą. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Kūrimo metu pateikiamos klaidos apie neteisingus veiksmus | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | 1 | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 1. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 2 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 1/1 |
| Aprašymas: | Įrankis turi turėti papildomus meniu punktus kurie leis dirbti susietame režime. | | | | |
| Pagrindimas: | Galimybė pasirinkti norimą atlikti operaciją | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Ar visos reikalingos funkcijos įjungiamos įjungus patikros įrankį. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 3 |
| Priklausomybės | 2 | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 8. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 3 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 1/1 |
| Aprašymas: | Įrankis turi leisti kurti naujas diagramas susietame režime. | | | | |
| Pagrindimas: | Vartotojas pasirinkęs susietą kūrimą gauna daugiau funkcionalumo | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Sukūrus naują modelį, aktyvuojamas iškart patikros komponentas. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 5. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 4 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 2/2 |
| Aprašymas: | Įrankis turi neturėti papildomų meniu punktų ir stereotipo | | | | |
| Pagrindimas: | Paketas neturi turėti nereikalingų veiksmų | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Ar visos reikalingos funkcijos išjungiamos, išjungus patikros įrankį. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 3 |
| Priklausomybės | 2 | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 8. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 5 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 3/6 |
| Aprašymas: | Registruoti klaidas ir pateikti klaidų aprašymą | | | | |
| Pagrindimas: | Tai padės vartotojui išvengti tam tikrų kūrimo klaidų. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Sistema turi aptikti įvestas klaidas | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 10. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 6 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 4/3 |
| Aprašymas: | Įrankis turi patikrinti, ar klasės turi visas operacijas iš būsenų mašinos vaizdo. | | | | |
| Pagrindimas: | Susietame režime visos klasės operacijos turi būti atvaizduotos būsenų mašinoje (išskyrus užklausų operacijas, tačiau tą nusprendžia vartotojas, o ne sistema) | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Atlikus pakeitimą klasių ar būsenų mašinos diagramoje (įvedus\ištrynus) naują būseną ar operaciją), sistema turi pranešti apie operacijos atvaizdavimą būsenų diagramoje. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 7. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 7 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 5/5 |
| Aprašymas: | Įrankis turi atlikti būsenų mašinos patikrinimą, ar nėra kilpų, bereikalingų perėjimų ir kitų su būsenų mašinomis nesuderinamų loginių klaidų. | | | | |
| Pagrindimas: | Būtina įsitikinti kad sudaryta korektiška ir logiką atitinkanti būsenų mašina. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Pateikus būsenų mašiną su klaidomis, sistema pateikia informaciją apie klaidas. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 8 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 6/4 |
| Aprašymas: | Sistema turi atlikti išsamų susietam režime kuriamų būsenų mašinų ir klasių modelių patikrinimą, ar visi būsenų mašinos įvykiai atvaizduojami klasių diagramoje. | | | | |
| Pagrindimas: | Abu modeliai buvo kuriami susietame režime, taigi turi vienas kitą atitikti. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Atlikus pakeitimą klasių ar būsenų mašinos diagramoje (įvedus\ištrynus) naują būseną ar operaciją), sistema turi pranešti apie būsenų mašinos įvykio atvaizdavimą klasių diagramoje. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 3. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 9 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 7/9 |
| Aprašymas: | Turi būti registruojamos visos aptiktos klaidos, kaupiamas jų sąrašas. | | | | |
| Pagrindimas: | Tai leis geriau suprasti daromas klaidas, ir jų išvengti ateityje. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Sistema turi aptikti visas padarytas klaidas | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 10. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 10 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 8/7 |
| Aprašymas: | Sistema turi atlikti būsenų mašinų ir klasės patikrinimą, ar visos klasės operacijos atvaizduotos būsenų mašinų diagramose (išskyrus užklausų operacijas, tačiau tą nusprendžia projektuotojas) | | | | |
| Pagrindimas: | Abu modeliai buvo kuriami susietame režime, taigi turi vienas kitą atitikti. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Atlikus pakeitimą klasėje ar būsenų mašinos diagramose (įvedus\ištrynus naują būseną ar operaciją), sistema turi pranešti apie klasės operacijos atvaizdavimą būsenų mašinos diagramoje. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 3. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 11 | Reikalavimo tipas: | 9a | Įvykis/panaudojimo atvejis #: | 9/8 |
| Aprašymas: | Sistema turi atlikti išsamų susietam režime kuriamų būsenų mašinų ir klasių modelių patikrinimą, ar visi vienos būsenų mašinos įvykiai atvaizduojami klasių diagramoje. | | | | |
| Pagrindimas: | Abu modeliai buvo kuriami susietame režime, taigi turi vienas kitą atitikti. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Atlikus pakeitimą klasių ar būsenų mašinos diagramoje (įvedus\ištrynus) naują būseną ar operaciją), sistema turi pranešti apie tikrinamos būsenų mašinos įvykio atvaizdavimą klasių diagramoje. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 3. | | | | |

## *3.3 Nefunkciniai reikalavimai*

Šioje dalyje pateikiami sistemos nefunkciniai reikalavimai jie aprašomi taip pat šablono pagalba.

**Reikalavimai sistemos išvaizdai**

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 12 | Reikalavimo tipas: | 10 | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Vartotojo sąsaja paprasta ir aiški. Lengvai suprantamas meniu, intuityvi veiksmų seka. | | | | |
| Pagrindimas: | Būsimasis vartotojas turėtų greitai perprasti veikimo principą. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Vartotojas orientuojasi meniu ir pateikiamuose pranešimų languose | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | Nėra | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

## Reikalavimai panaudojamumui

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 13 | Reikalavimo tipas: | 11a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Įrankis turi būt nesudėtinga naudoti vartotojui, turinčiam ir jau naudojančiam MagicDraw paketą. | | | | |
| Pagrindimas: | Dauguma vartotojų bus potencialūs MagicDraw įrankio vartotojai. | | | | |
| Šaltinis: | Vadovybė | | | | |
| Tikimo kriterijus: | Turi būti perprantamas per vieną darbo valandą. | | | | |
| Užsakovo tenkinimas: | 5 | | | Užsakovo netenkinimas: | 5 |
| Priklausomybės | Nėra | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 14 | Reikalavimo tipas: | 11a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Įrankis turi padėti vartotojui nedaryti klaidų, arba išsamiai apie jas informuoti jį. | | | | |
| Pagrindimas: | Kuriant didelius modelius neretai pasitaiko klaidų. | | | | |
| Šaltinis: | Vadovybė | | | | |
| Tikimo kriterijus: | Mažas padaromų klaidų kiekis. | | | | |
| Užsakovo tenkinimas: | 2 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | Nėra | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

## Reikalavimai vykdymo charakteristikoms

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 15 | Reikalavimo tipas: | 12a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Įrankis turėtų netrikdyti MagicDraw paketo veikimo, bei bendrauti su juo, veikti sparčiai. | | | | |
| Pagrindimas: | Darbo metu turi būti atliekama patikra, ir automatiškai informuojamas vartotojas apie jo atliktas klaidas. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Kūrimo procese nepastebimi pašaliniai trikdžiai, ar sistemos darbo sutrikimai. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 3 |
| Priklausomybės | Nėra | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

## Reikalavimai veikimo sąlygoms

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 16 | Reikalavimo tipas: | 13a | Įvykis/panaudojimo atvejis #: | |
| Aprašymas: | Įrankis turi atlikti greitą patikrą, | | | | |
| Pagrindimas: | Vartotojas neturi jausti diskomforto, laukti. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Pateiksime didelę diagramą ir stebėsim vykdymo laiką. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 3 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

## Reikalavimai sistemos priežiūrai

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 17 | Reikalavimo tipas: | 14a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Turi būti vartotojo dokumentacija. | | | | |
| Pagrindimas: | Vartotojui turi būti instrukcija kurioje būtų parodoma kaip naudotis įrankiu. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Duodame nežinančiam žmogui instrukciją ir stebime kaip greitai įsisavinamas įrankis. | | | | |
| Užsakovo tenkinimas: | 4 | | | Užsakovo netenkinimas: | 4 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Kovo 9. | | | | |

## Reikalavimai sistemos saugumui

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 18 | Reikalavimo tipas: | 15a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Duomenys patikros metu neturi būti keičiami, ar kitaip transformuojami, turi išlikti modelio originalas | | | | |
| Pagrindimas: | Naudojant susietą režimą modelis turi būti peržiūrimas patikros komponento, bet nekoreguojamas. | | | | |
| Šaltinis: | Projektuotojas | | | | |
| Tikimo kriterijus: | Tikrinimo metu nekeičiami duomenys tik aptinkamos klaidos. | | | | |
| Užsakovo tenkinimas: | 3 | | | Užsakovo netenkinimas: | 3 |
| Priklausomybės | Nėra | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 kovo 9 | | | | |

## Kultūriniai-politiniai reikalavimai

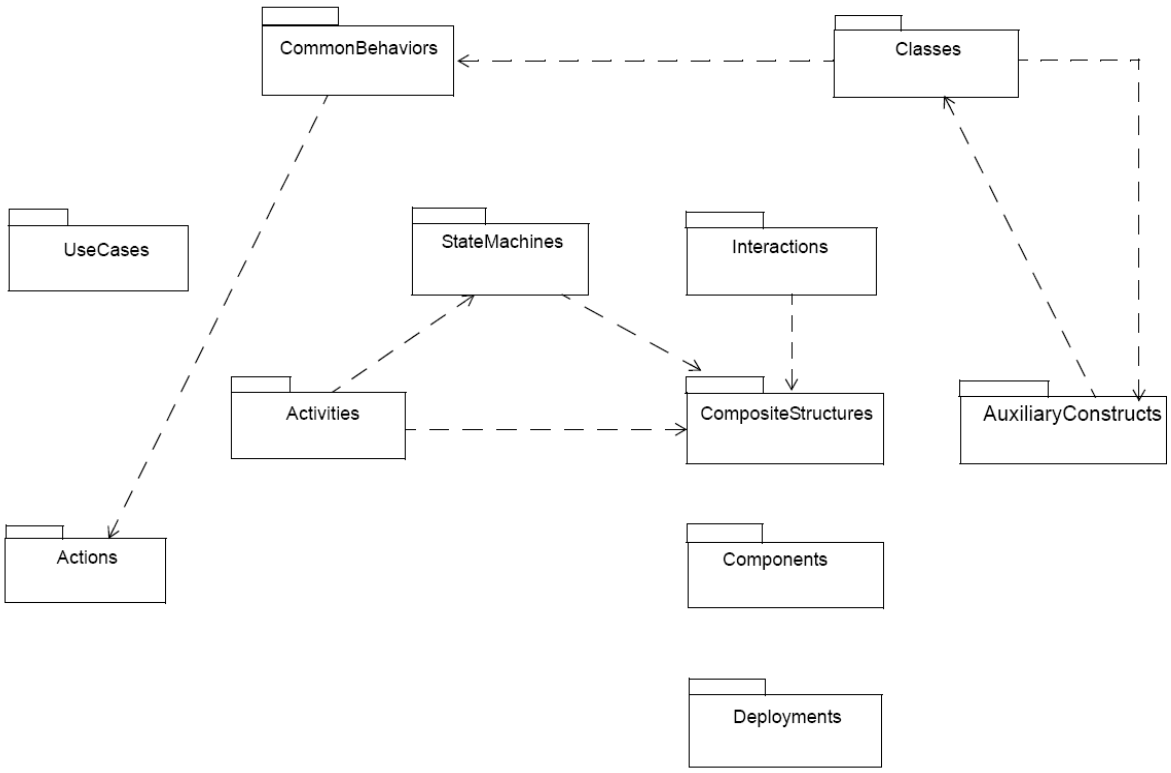
| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 19 | Reikalavimo tipas: | 16a | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Programos meniu punktuose ar ikonose negali būti dviprasmiškų ar įžeidžiančių žodžių, paveikslėlių | | | | |
| Pagrindimas: | Bus naudojama kitose šalyse. | | | | |
| Šaltinis: | Vadovybė | | | | |
| Tikimo kriterijus: | Punktų pavadinimai ir ikonų paveiksliukai korektiški | | | | |
| Užsakovo tenkinimas: | 2 | | | Užsakovo netenkinimas: | 2 |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 Sausio 20. | | | | |

## Teisiniai reikalavimai

| | | | | | |
|---|---|---|---|---|---|
| Reikalavimas #: | 20 | Reikalavimo tipas: | | Įvykis/panaudojimo atvejis #: | 1-10 |
| Aprašymas: | Modelius kuriant turi būti laikomasi UML kalbos taisyklių. | | | | |
| Pagrindimas: | Modeliuose bus naudojama UML kalba. | | | | |
| Šaltinis: | Vadovybė | | | | |
| Tikimo kriterijus: | Naudojama UML specifikaciją atitinkanti kalba. | | | | |
| Užsakovo tenkinimas: | | | | Užsakovo netenkinimas: | |
| Priklausomybės | - | | | Konfliktai: | Nėra |
| Papildoma medžiaga: | - | | | | |
| Istorija: | Užregistruotas 2006 kovo 9 | | | | |

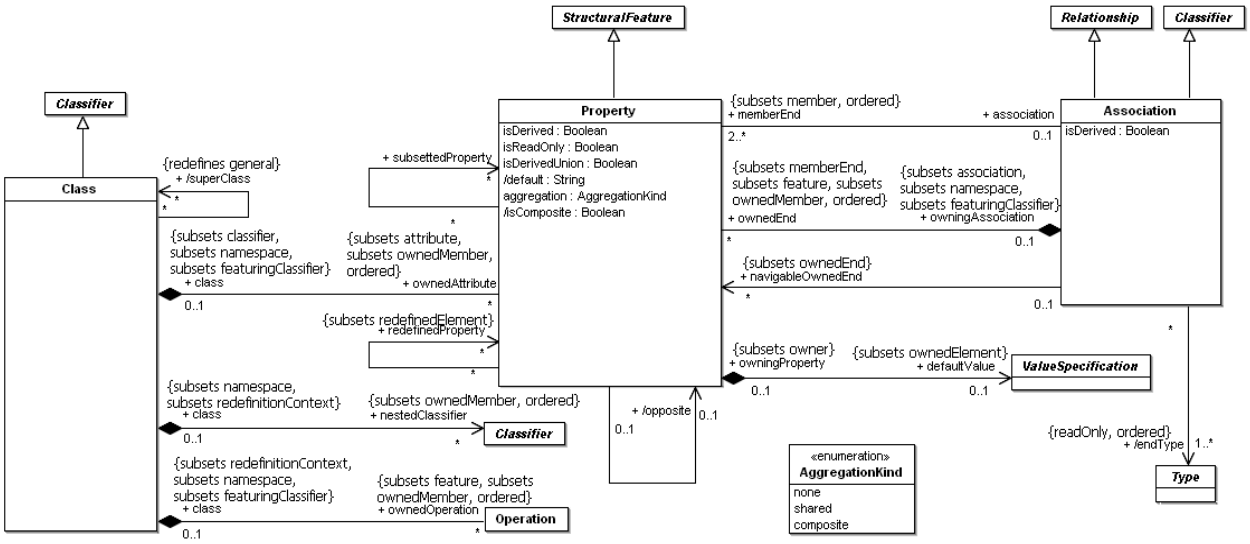
## *3.4 Duomenų struktūra*

Kadangi kuriama sistema yra kitos sistemos dalis, teks prisitaikyti prie pagrindinės sistemos, šiuo atveju MagicDarw duomenų struktūrų. Norint atrasti reikiamus duomenis ir ryšius tarp jų, pirmiausiai riekėtų paanalizuoti UML modelio struktūrą. UML specifikacija yra lengvai prieinama per internetą [9] šaltinis. Specifikacijoje išsamiai aprašomi UML modelio lementai, sąsajos ir hierarchijos tarp jų. UML paketas, suskaidytas į tam tikrus paketus, kurie naudojami struktūriniame ir elgsenos modeliuose (9 pav.) [9].



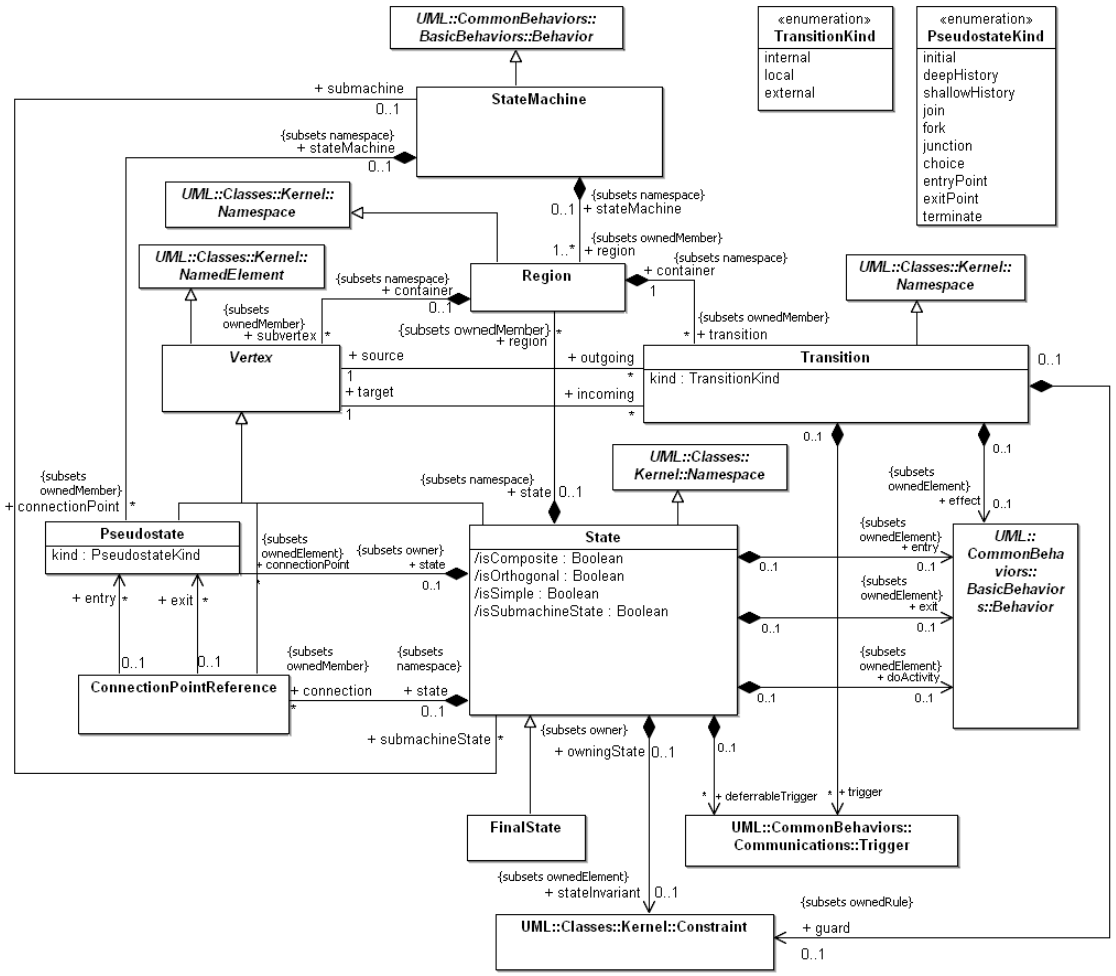

**9 pav. Aukščiausio lygio paketų struktūra UML 2.1.1 Superstruktūroje**

Dabar panagrinėkime, kaip yra saugomos klasės ir būsenų mašinos detaliau. UML modelyje klasės saugomos tam tikroje struktūroje kuri pateikiama (10 pav.) [9].



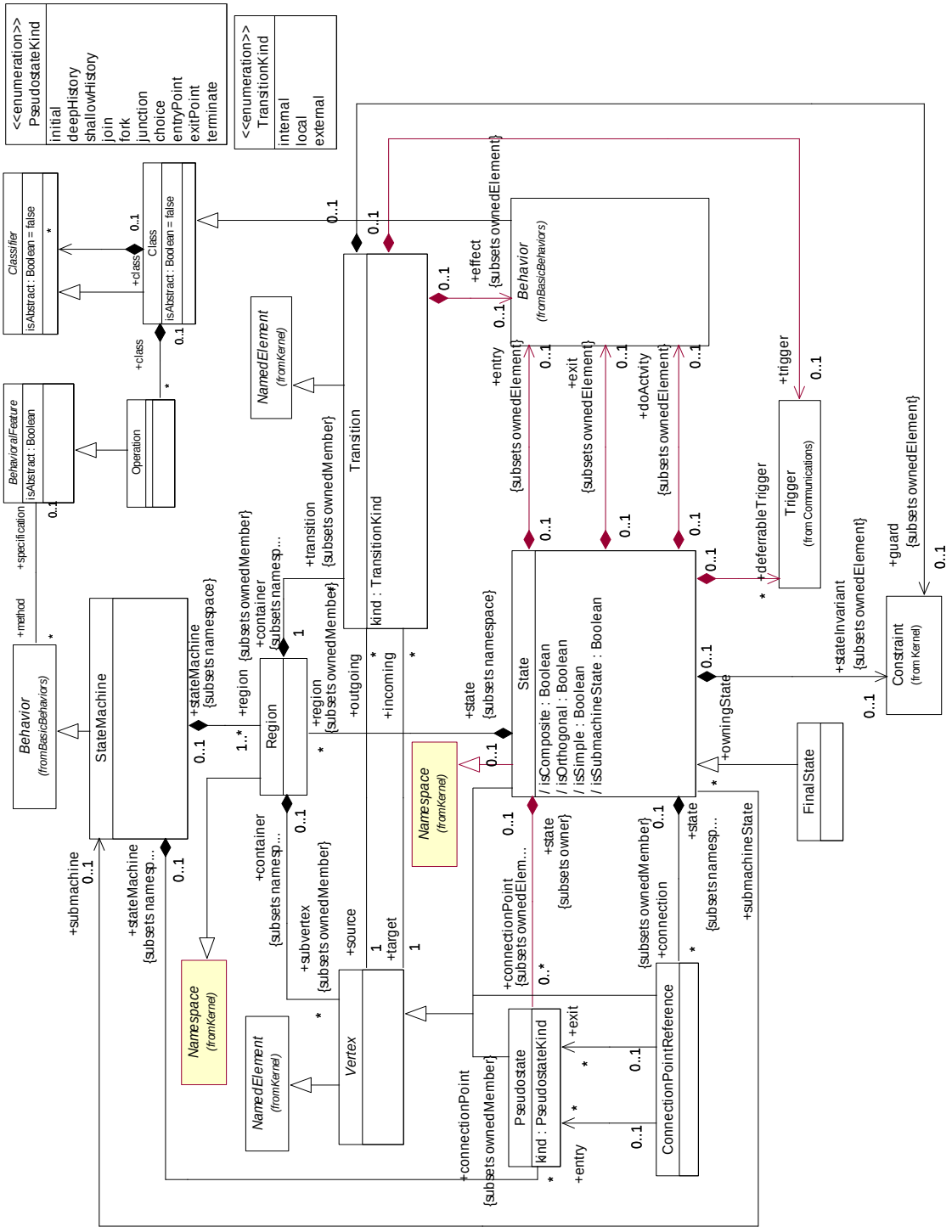

**10 pav. Klasės meta modelis**

Uml modelyje būsenų mašinos (*angl. State Machine* ) saugomos tam tikroje struktūroje, kuri pateikiama 11 paveiksle. Paveikslas paimtas iš [9] šaltinio.




**11 pav. Būsenų mašinos (angl. State Machine) meta modelis**

Dabar lieka tik surasti kaip šie lementai susiejami UML modelyje t.y., kaip klasių operacijos susiejamos su būsenų mašinų įvykiais. Šių elementų ryšiai pateikiami (12 pav.).




**12 pav. Būsenų meta modelis kuriame įtraukti ryšiai su klasėmis**

## *3.5 Sistemos architektūra*

Kuriamos sistemos architektūros tikslai:

- ✓ Nepriklausomumas nuo platformos – sistema turi veikti keliose plačiai naudojamose platformose.
- ✓ Sistema turi būti realizuojam JAVA kalba, nes tik tokia kalba parašytus įskiepius palaiko MagicDraw aplinka.
- ✓ Daugiakalbiškumas -sistemoje galima naudoti skirtingų kalbų simbolius
- ✓ duomenų įvedimui/išvedimui (tą užtikrina MagicDraw įrankis).
- ✓ Sistemos integravimas į MagicDraw. Sistema turi būti MagicDraw CASE įrankio dalis.

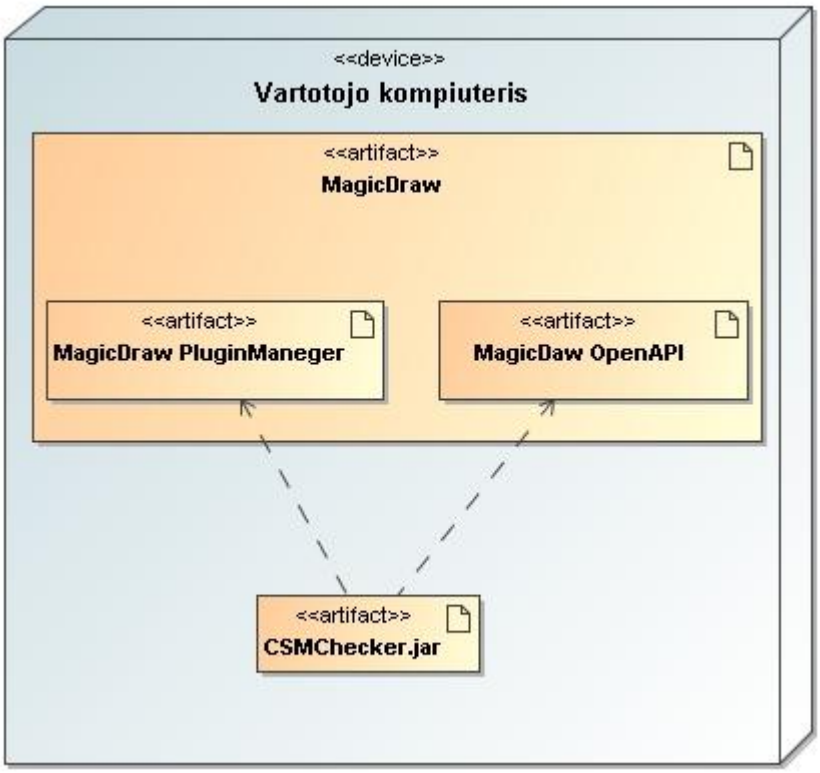
Kuriamai sistemai taikomi architektūros apribojimai:

- ✓ Veiklos logika turi būti realizuota, remiantis užsakovo pateiktų duomenų struktūros šablonu.

Kuriama sistema turi būti integruota i MagicDraw UML CASE įrankį.

13 paveiksle pavaizduotas sistemos išdėstymo vaizdas, atitinkantis sistemos architektūros reikalavimus.

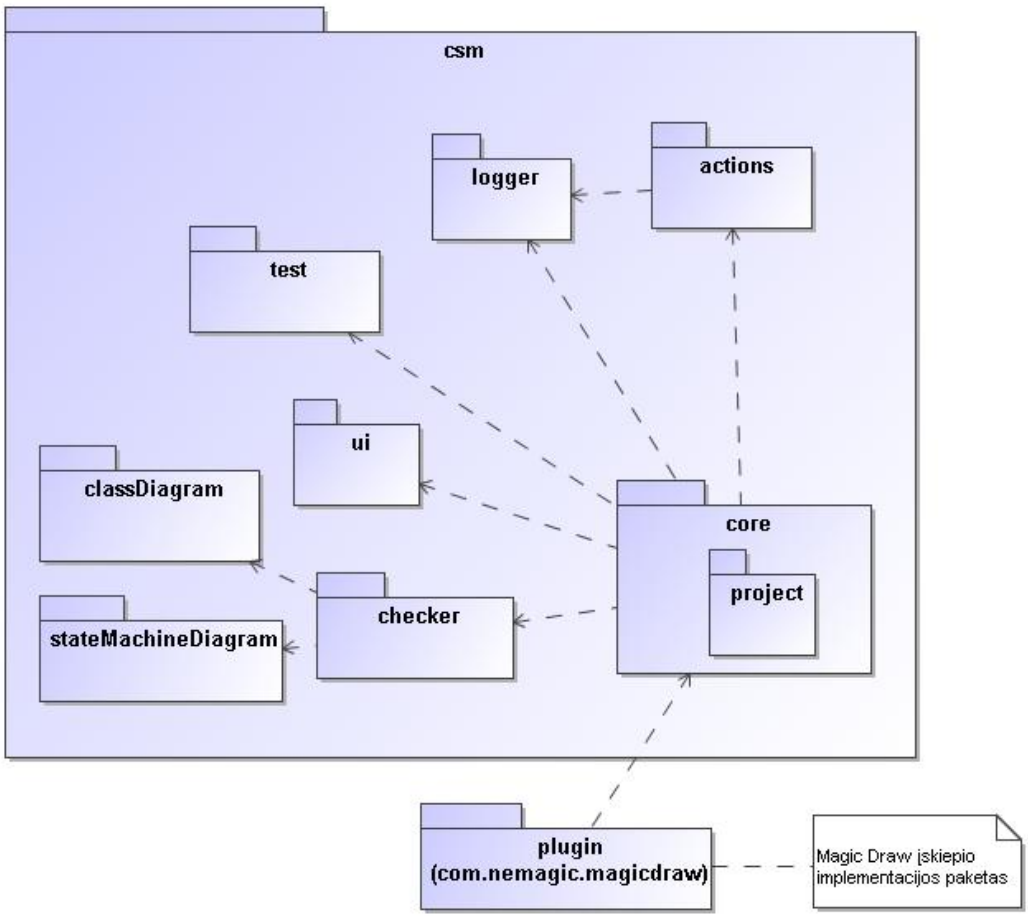
Kuriamą sistemą sudaro tokie komponentai:




**13 pav. Sistemos komponentai**

## *3.6 Paketų struktūra*

Programinė įranga sekų ir būsenų suderinamumui tikrinti kuriama kaip CASE įrankio (MagicDraw UML įrankio) įskiepis (angl. *pugin*). Kuriamos sistemos realizacijai reikalinga paketų struktūra pavaizduota 14 paveiksle. Kuriant paketų struktūrą buvo stnegiamsi išlaikyti MagicDraw paketų struktūros logiką ir pavadinimus.




**14 pav. CSMChecker sistemos paketų struktūra**

**Paketų aprašymas**

Com.nomagic.magicdraw.plugins – MagicDraw modeliavimo įrankio paketas, realizuojantis išorinę sąsają, leidžiančią įskiepiams naudotis įrankio funkcionalumu.

core – tai pagrindinis CSM įskiepio paketas, kuriame saugomos klasės, vykdančios pagrindinė įskiepio logiką ir valdymą.

project – saugomos klasės, kurios dirba su MagicDarw aplikacijoje sukurtu projektu atlieka jo elementų konfigūravimus ir pagalbinį projekto valdymą.

action – pakete laikomos visos klasės susijusios su CSM įskiepio įvykiais.

logger – pakete laikomos su sekimo įrašų išsaugojimu susijusios klasės.

test – paketas buvo sukurtas norint atlikti funkcijų testavimą su JUnit karkasu, jame patalpintos klasės, kurios imituoja UML duomenų struktūras.

ui – CSM įskiepio vartotojo sąsajos klasių paketas.

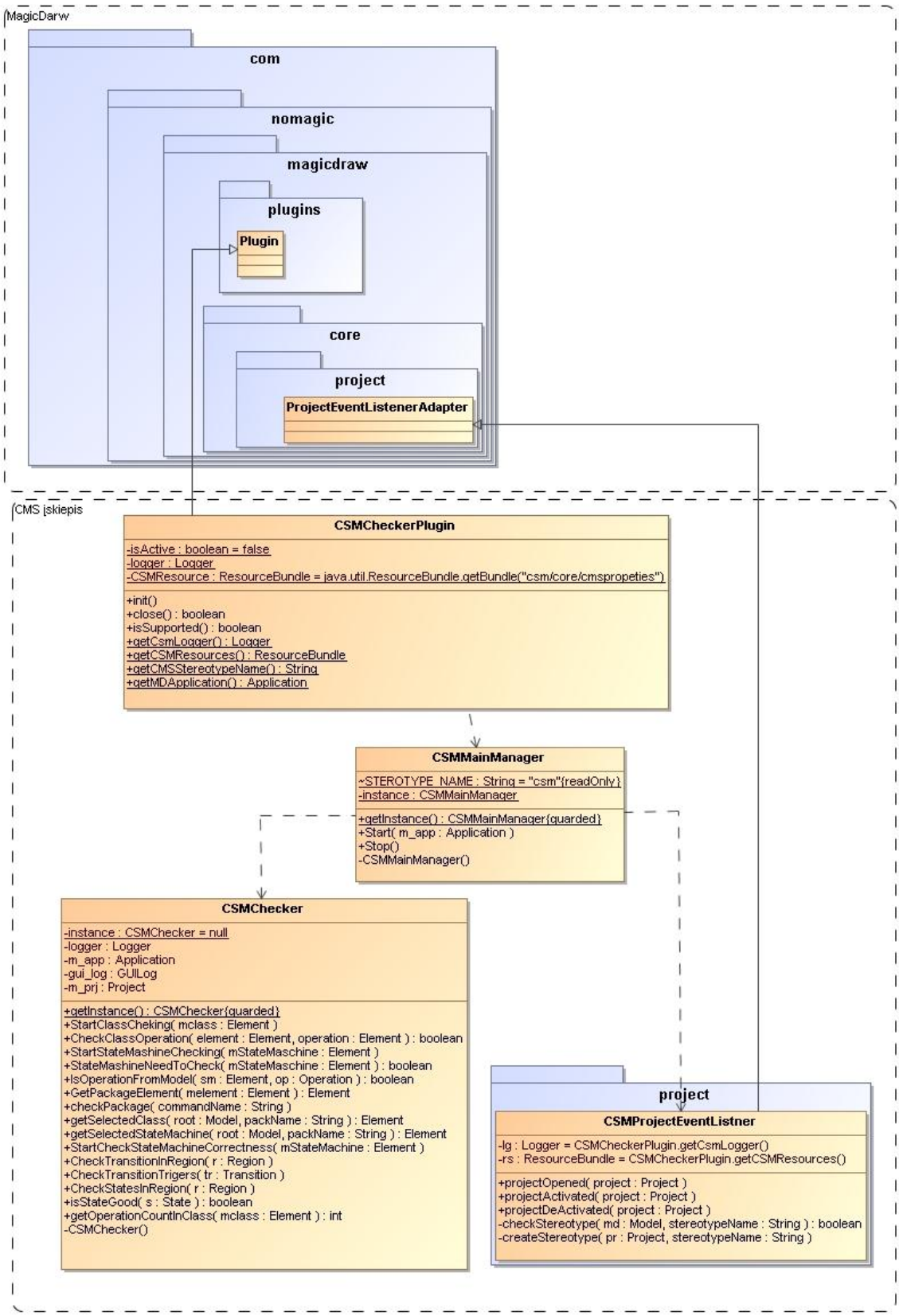
checker – šiame pakete saugomos visos klasės, kurios dirba su paketu, klasių diagrama, būsenų mašinos diagrama.

classDiagram – pakete saugomos klasės, kurios dirbas su klasių diagramomis.

stateMachineDiagram – pakete saugomos klasės, kurios dirba su būsenų mašinų diagramomis.

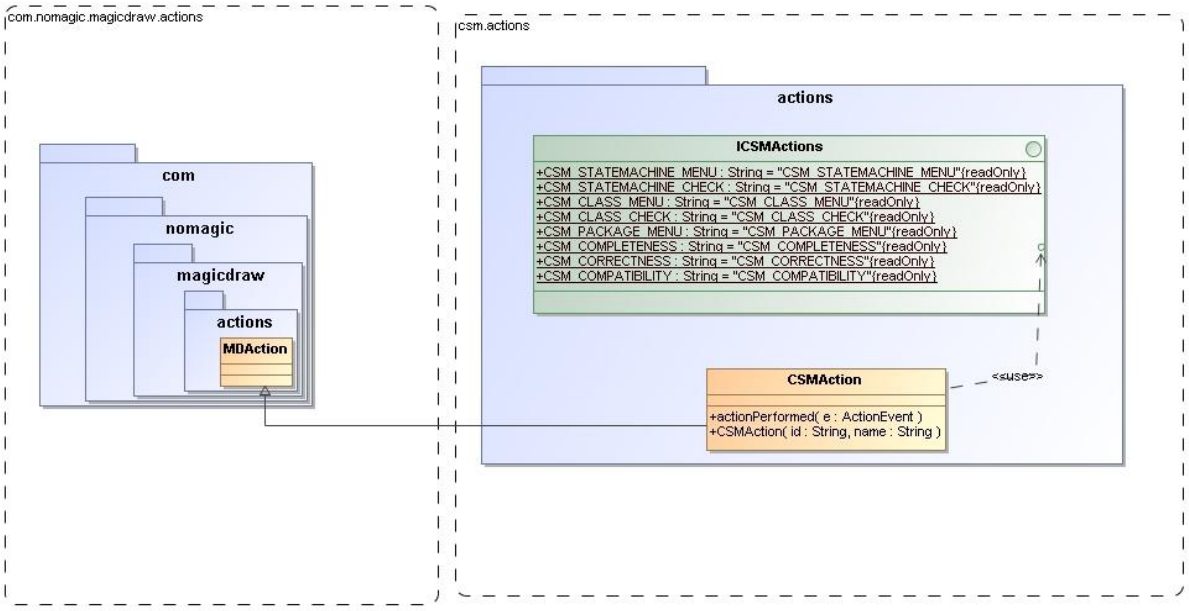
## *3.7 Esminiai projektavimo sprendimai*

Sistema buvo skaidoma į tam tikrus loginius vienetus, stengiantis išlaikyti MagicDarw koncepciją ir logiką. Sukurtos sistemos paketų struktūrą apžvelgėme ankstesniame skyriuje, dabar panagrinėkime, kokie sprendimai atlikti klasių lygyje.



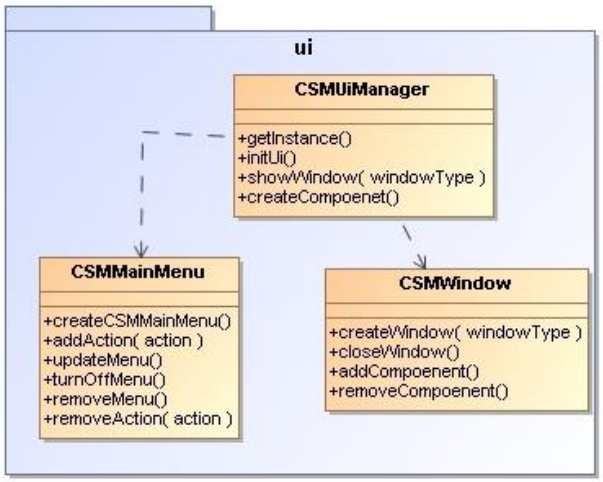

**15 pav. Sistemos paketas (csm.core)**

Šioje schemoje (15 pav.) patiekiami pagrindiniai sistemos valdikliai, kurie atlieka įskiepio darbo valdymą. CMSCeckerPlugin atlieka įskiepio sukūrimo darbus, toliau viskas valdo CSMMainManager, kuris sukuria modelyje reikalingus meniu, atidarinėja reikalingus langus, CSMChecker tai tikrinimo elementas jis atleika visus reikalingus įskiepio tikrinimus.



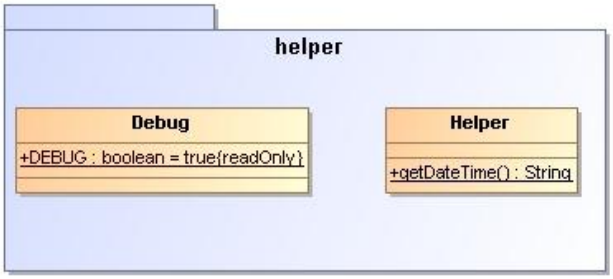

**16 pav. Sistemos paketas (csm.actions)**

Pakete csm.action aprašomi visi sistemos galimi įvykiai tam panaudojamas ICSMAction. Įvykio lementas yra paveldimas iš MagicDarw aplinkos, kadangi įvykių aprašymas turi atitikti MagicDraw aplinkos įvykių aprašymą (16 pav.).



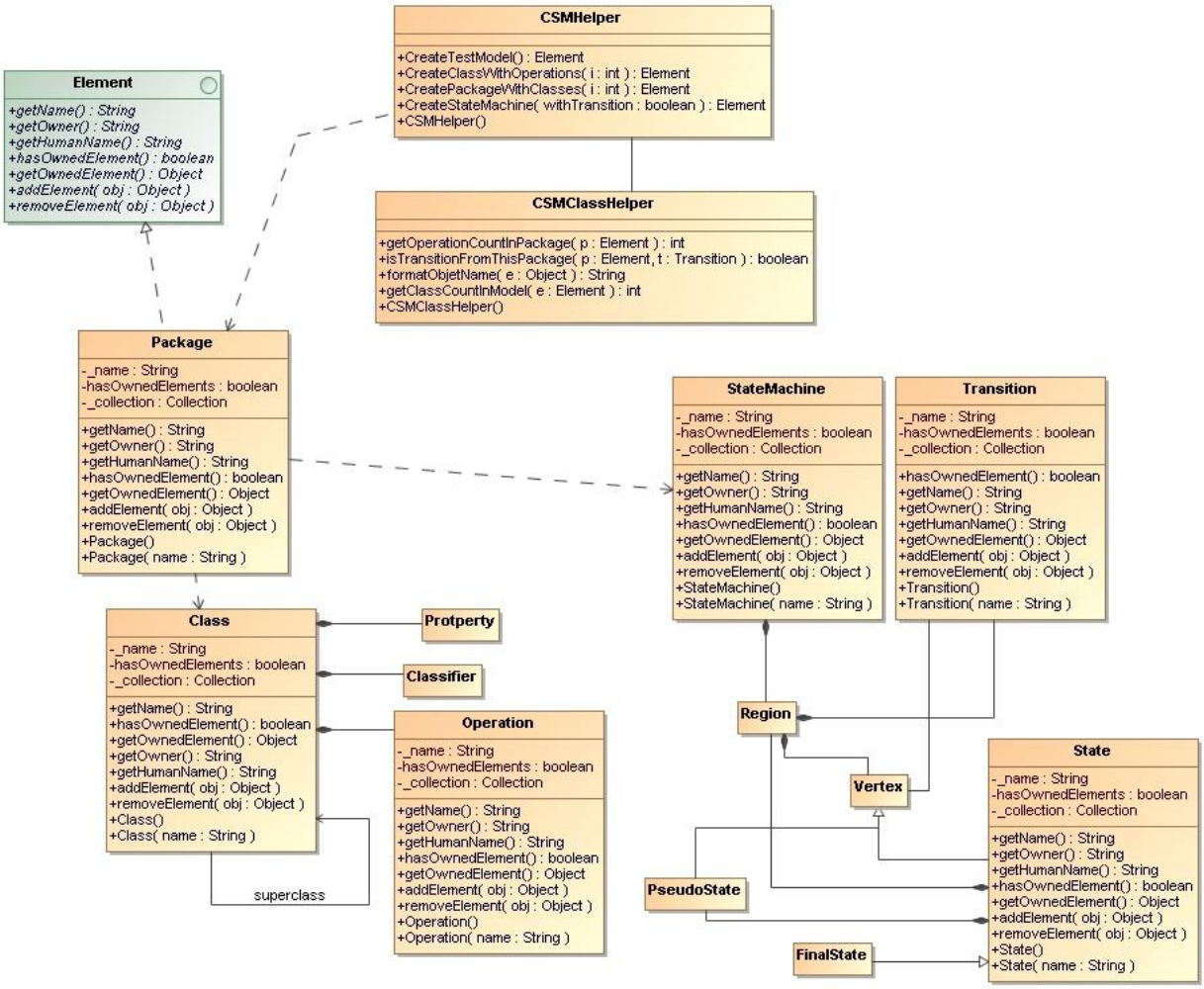

**17 pav. Sistemso paketas (cms.ui)**

Pakte csm.ui aprašomi elementai reikalingi darbui su grafine sąsaja, CSMUIManager atleika grafinės sąsajos valdymo darbus (17 pav.).




**18 pav. Sistemos paketas (csm.helper)**

Paketas csm.helper buvo sukurtas tam kad būtų galima įrankį paleisti derinimo ir taip pat testavimo rėžime (18 pav.).




**19 pav. Sistemos paketas (cms.test)**

Paketas csm.test buvo sukurtas tam, kad būtų galima atlikti sistemos testavimus naudojant JUnit karkasą. Kaip matome, klasių diagramoje teko susikurti savo duomenų struktūras ir testo metu jas padavinėti tikrinimo funkcijoms (19 pav.).

## *3.8 Sukurtos programų sistemos charakteristikos*

Sukurta sistema buvo remiantis OOP paradigma t.y., sistemos komponentai buvo kuriami kaip atskiri objektai su savo logika, atributais ir metodais. Tokius elementu nesunku panaudoti tolimesniuose darbuose, išplečiant jų funkcionalumą. Sistemos įverčiai pateikiami 2 lentelėje.

*2 lentelė. Sistemos parametrų įverčiai*

| **Elementas** | **Parametrai** |
|---|---|
| **Paketų kiekis** | 7 |
| **Klasių kiekis** | 29 |
| **Metodų kiekis** | 100 |
| **Perkrautų metodų** | 15 |
| **Maksimalus parametrų metode kiekis** | 5 |
| **Sąsajų** | 10 |
| **Paveldėjimų** | 5 |
| **Kodo eilučių** | 4000 |

Nemažai laiko buvo skiriama MagicDraw openAPI analizei, nes projektas yra įterptinė sistema, kuri priklauso nuo MagicDraw veikimo principų. Toki projektai visuomet būna sudėtingesni nes reikia taikyti struktūras prie pagrindinės sistemos.

# 4 REALIZACIJA IR TESTAVIMAS

## Realizacija

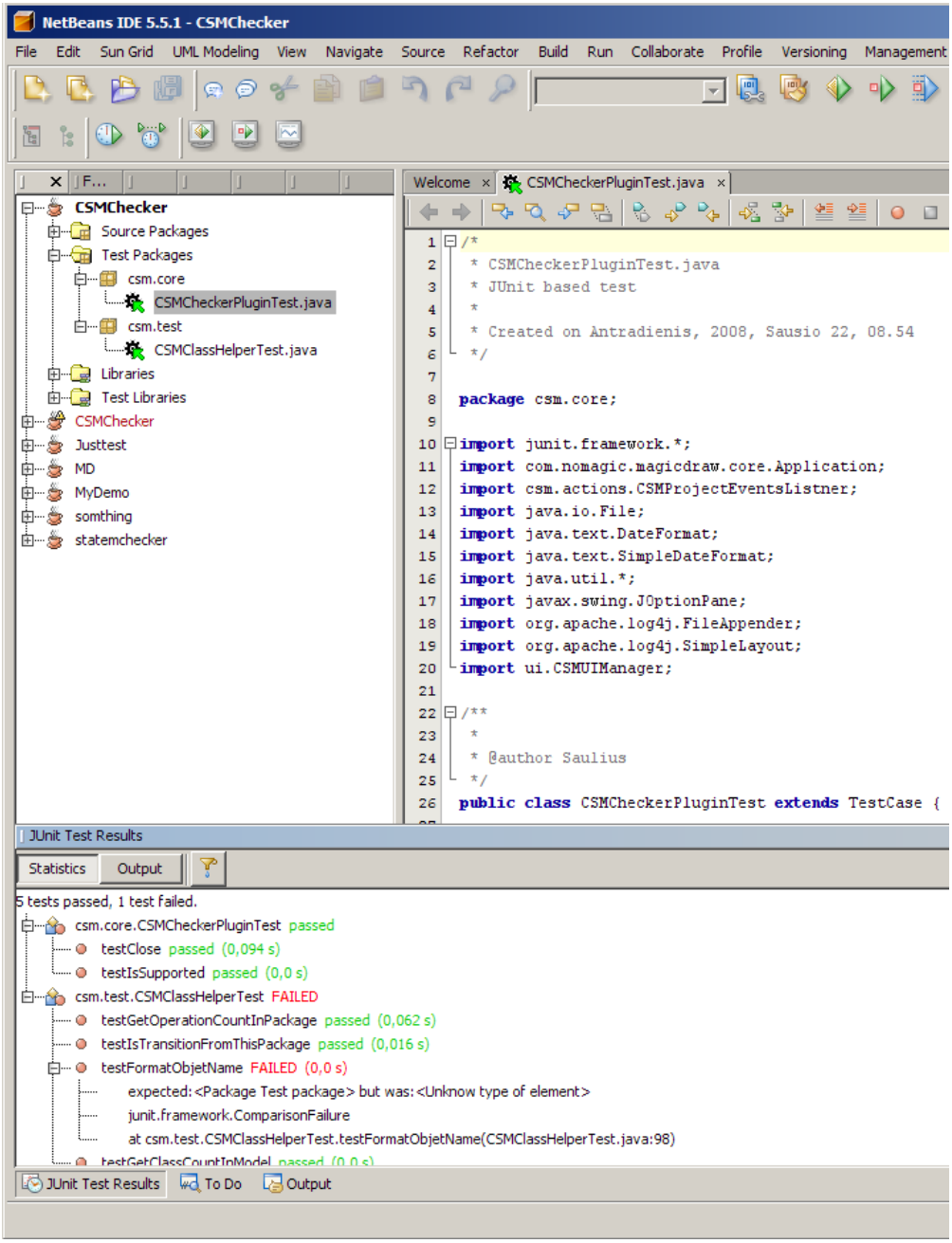
Realizacijos metu buvo susiduriama su sunkumais dėl tam tikrų MagicDraw OpenAPI dokumentacijos nesklandumų, taip pat dėl ribojamo priėjimo prie sistemos komponentų. Teko nemažai atlikti eksperimentų norint sustatyti tam tikrus sistemos veiksmus ir elgseną. Kadangi MagicDraw produktas yra mokamas, tai natūralu, kad yra stengiamasi apsaugoti sistema nuo plagijavimo, daugelis elementų yra obfuskuoti t.y. jų vardai pakeisti į nereikšmingus simbolius, kurie neleidžia sistemą lengvai analizuoti. Nors ir buvo bandoma sistemą analizuoti, bet visada tekdavo ieškoti kitų sprendimo variantų.

Taip pat MagicDraw OpenAPI nėra pateikiama, kaip kuriamus įrankius paleisti derinimo rėžime, todėl pradžioje buvo dirbama be derinimo rėžimo, toliau analizuojant sistemą atradau savo padarytas klaidas nustatinėjant kūrimo aplinkos parametrus.

Realizavimo metu buvo naudojami nauji įrankiai NetBeans 5.5.1, paskutiniam sistemos modifikavimui teko panaudoti naujausią NetBeans 6.0. Ši programavimo aplinka lengvai leidžia derinti sistemą, ją analizuoti, atlikti atvirkštinę inžineriją. Taip pat yra suderinta su MagicDraw, taigi MagicDraw projekto ir sistemos realizacijos metu galime atlikti pastovų modelio ir realizuojamo kodo sinchronizaciją.

## Testavimas

Testavimas buvo atliekamas tokias būdai: kūrimo metu naudojant JUnit karkasą ir csm.test pakete sukurtas duomenų struktūras. Tokio testavimo metu buvo stengiamasi eliminuoti sistemos funkcijose paliktas klaidas. Toks testavimas labai pravertė sistemos kūrimo pradžioje.



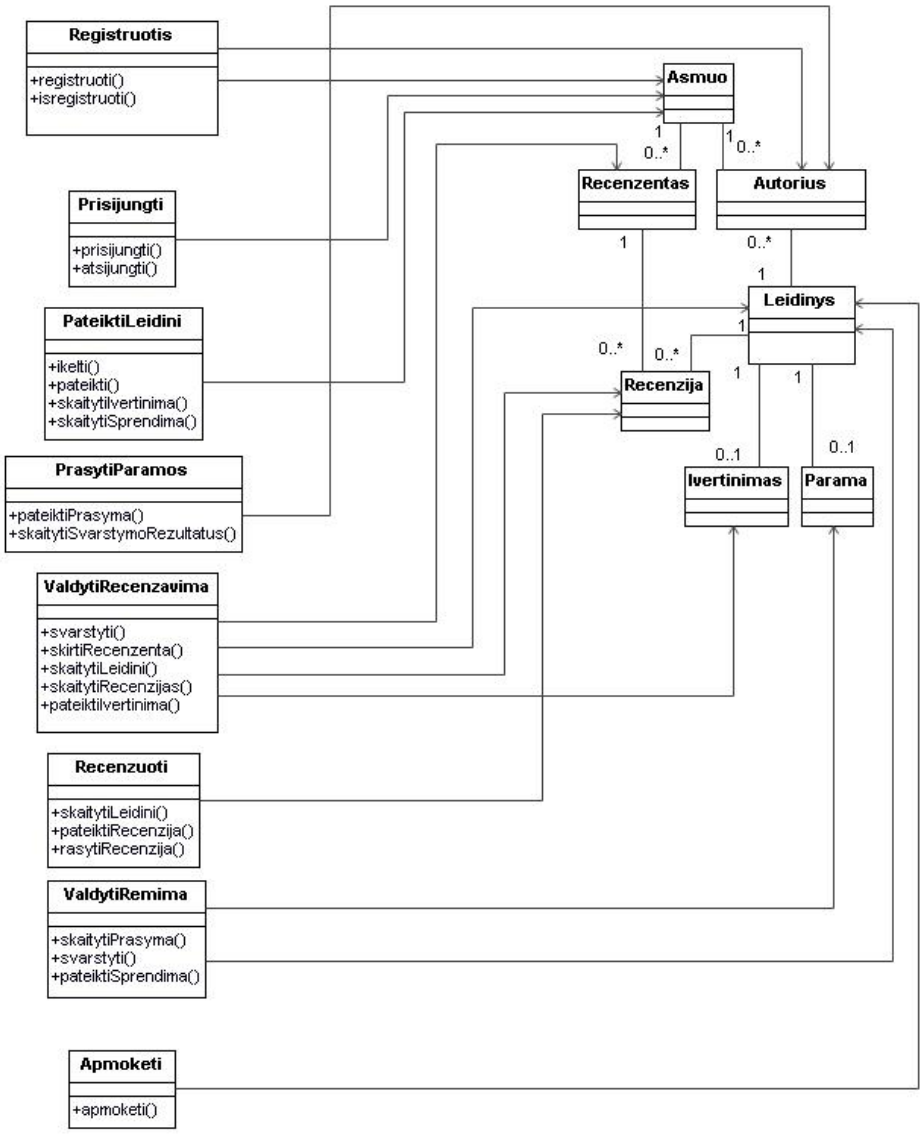

**20 pav. JUnit testavimas**

Toks funkcijų testavimas leido lengvai eksperimentuoti su galimomis duomenų struktūromis bei užtikrinti sistemos funkcijų veikimo korektiškumą ir teisingumą.

# 5 EKSPERIMENTINIO TAIKYMO PAVYZDYS

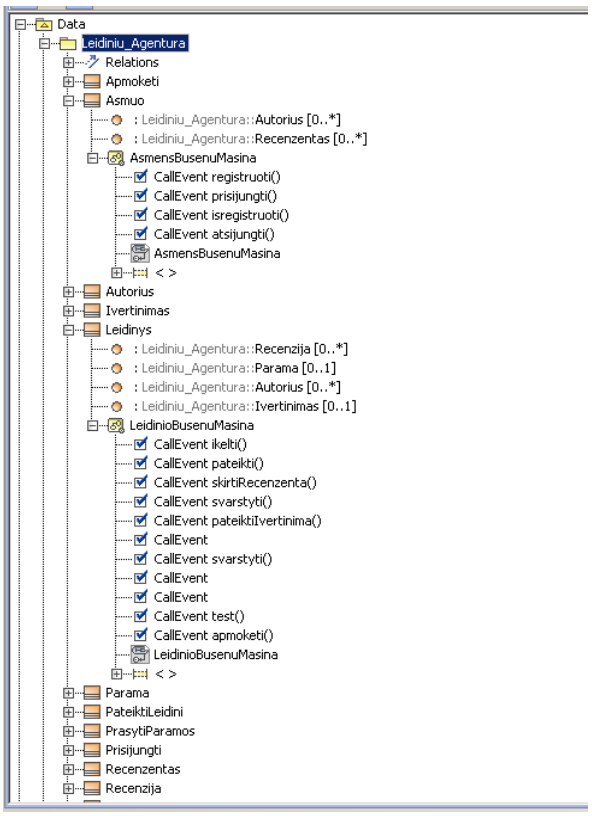
## Testavimo modelio pavyzdys

Šio taikymo metu buvo sukurtas nuo platformos nepriklausomas (PIM) modelis, su kuriuo buvo atlikti testavimai ir bandymai. Pateikiamas modelio klasių diagramos vaizdas (21 pav.)

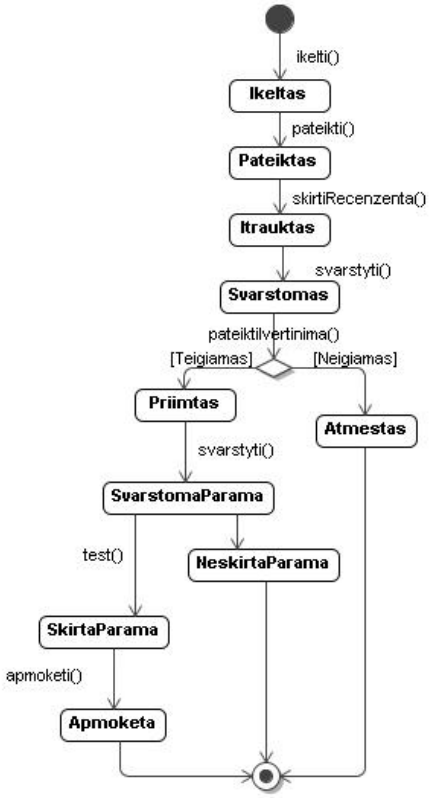



**21 pav. Nuo platformos nepriklausantis PIM modelis**

Šio modelio valdikliams buvo sukurtos būsenų mašinos (22 pav., 23 pav.). Būsenų mašinose buvo atvaizduojama valdiklių dinamika. Šis modelis nėra labai sudėtingas, jame tik 15 klasių ir 7 būsenų mašinos.

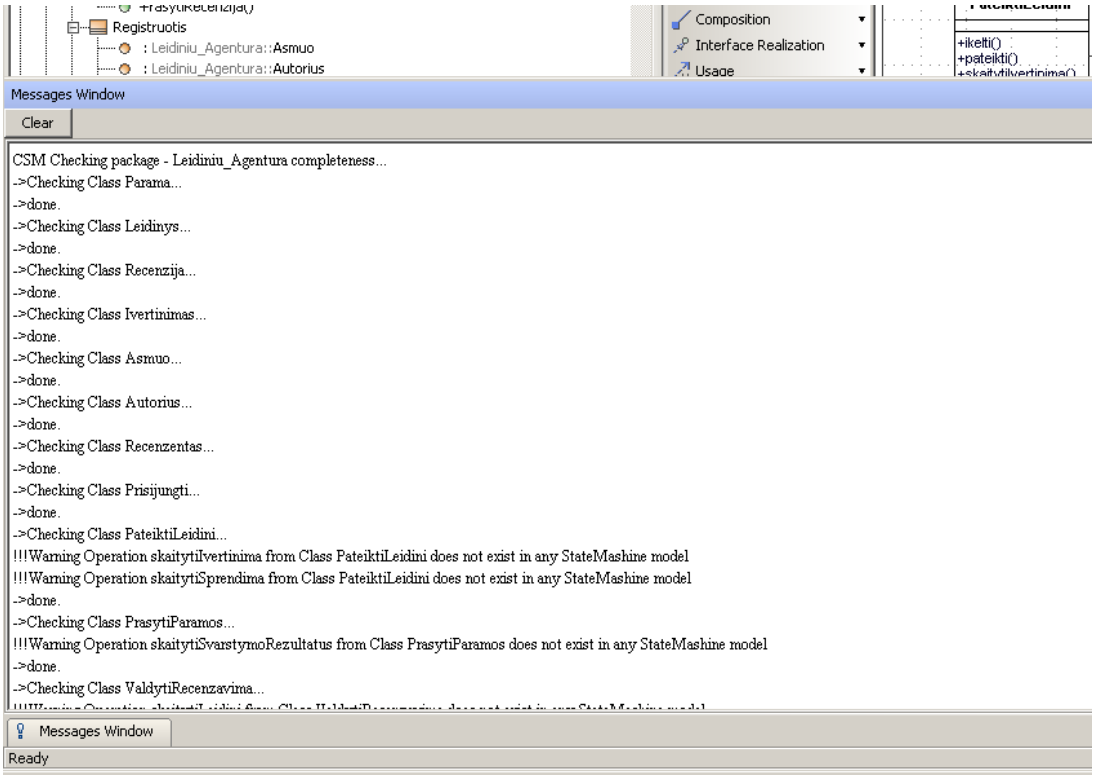



**22 pav. Modelio fragmentas**




**23 pav. Leidinio būsenų mašina**

Tokio modelio patikrai užtuktume nuo 2~4 min priklausomai nuo kvalifikacijos ir susipažinimo su modeliu. Sukurtas patikros įrankis su šia problema susidoroja greitai tai trunka apie ~1 sekundę.




**24 pav. Modelio patikrinimo ataskaita**

Gavę ataskaita kurioje pateikiamas neatitikimo pranešimas, jei ši neatitikimas toleruotinas t.y. operacija nebūtina būsenų mašinos modelyje ją galima ignoruoti ir netaisyti.

# 6 ĮVERTINIMAS

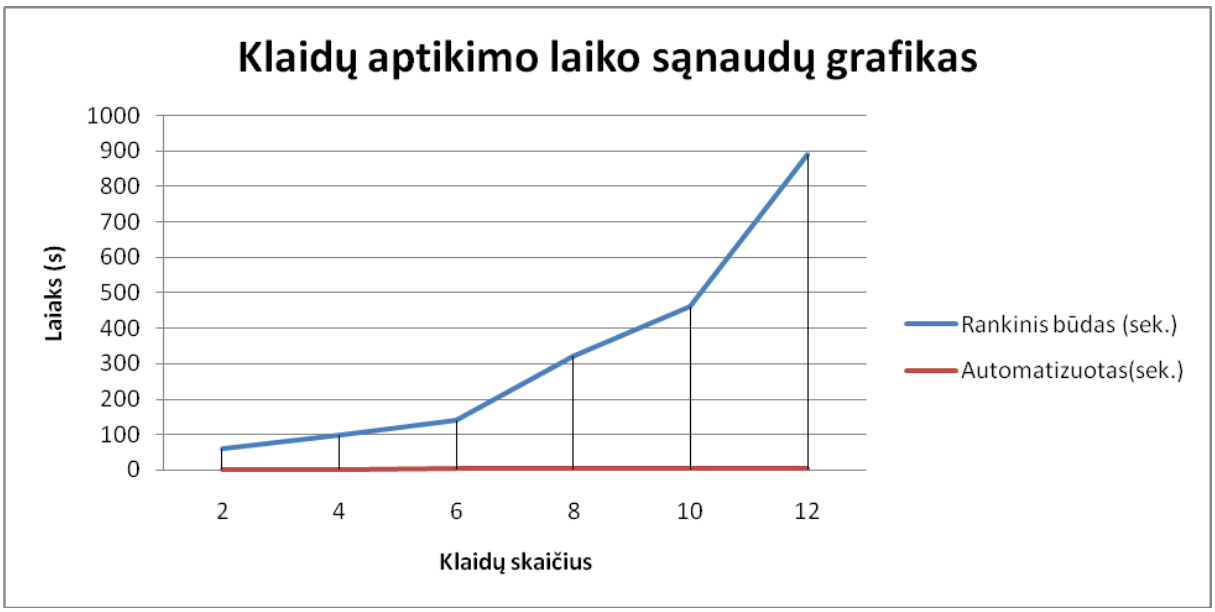
## 6.1 *Sukurto įskiepio efektyvumo tyrimas*

Įrankio efektyvumui tirti buvo sukurtas PIM modelio prototipas, kurį buvo stengiamasi vis papildyti ir stebėti įrankio darbo rezultatus. Įvertinimui atlikti buvo papildytas įskiepio funkcionalumas, skaičiuojantis sistemos klaidų aptikimo laikus. Taip pat buvo stengiamasi atlikti patikrą rankiniu būdu kuomet modelio kūrėjas įvelia klaidas ir kitas žmogus (aišku - susipažinęs su modeliu) bando atrasti tas klaidas ir jas ištaisyti. Šie tyrimo rezultatai buvo gaunami palaipsniui didinant modelio apimtis. Automatinio patikrinimo laikas buvo skaičiuojamas paspaudus tikrinimo mygtuką iki tikrinimo pabaigos ir rezultatų pateikimo pabaigos. Tokio įvertinimo rezultatai parodo ryškų automatizuoto tikrinimo pranašumą. Kadangi sistemos sudėtingumas auga, jos elementų skaičius didėja, natūralu, kad ir klaidų paieškos laikas ilgėja tai darant rankiniu būdu.

*3 lentelė. Duomenys apie klaidų aptikimo spartą (rankinis ir automatizuotas budai).*

| Diagramos elementų skaičius | Klaidu kiekis | Rankinis būdas (sek.) | Automatizuotas(sek.) |
|---|---|---|---|
| Klasės 4 būsenų mašinos 4 | 2 | 60 | 1 |
| Klasės 8 būsenų mašinos 8 | 4 | 98 | 1 |
| Klasės 12 būsenų mašinos 12 | 6 | 140 | 2 |
| Klasės 16 būsenų mašinos 16 | 8 | 320 | 2 |
| Klasės 20 būsenų mašinos 20 | 10 | 460 | 2 |
| Klasės 24 būsenų mašinos 24 | 12 | 890 | 2,5 |

Šiais gautais duomenimis buvo sudarytas klaidų aptikimo laiko sąnaudų grafikas, kuriame matome, kad sistema yra pranašesnė prieš žmogų. Toks klaidų aptikimas leidžia greičiau taisyti paliktas klaidas.




**25 pav. Klaidų aptikimo laiko sąnaudų grafikas**

Iš grafiko (25 pav.) nesunkiai galime pastebėti, kad į modelį įvedus papildomai komponentų ir padidinus klaidų skaičių, jų aptikimo laiko sąnaudos auga sparčiai, jas bandant aptikti rankiniu būdu. Automatiškas sistemos tikrinimas su sukurtu CSMChecker įrankiu užtrunka vos keletą sekundžių, per kurias išanalizuojamas paketas ir jame esančios būsenų ir klasių diagramos.

### Įskiepio tyrimo išvados

1. Specifinio proceso automatizavimas leidžia sumažinti laiko sąnauda ir atlikti jį greičiau.
2. Tai turėtų leisti greičiau kurti PIM modelius, kurių klasės ir būsenų mašinos bus suderintos tarpusavyje, o apie aptiktus nesuderinamumus bus informuojamas sistemos projektuotojas.
3. Tai leis sutaupyti sistemos kūrimo metu reikalingų resursų ir laiko sąnaudas.

## 6.2 Apibendrinimas

Šis įrankis leidžia greičiau aptikti sistemoje paliktas klaidas, kurios gali įtakoti sistemos modelio teisingumą ir korektiškumą. Toks klaidų tikrinimas aišku yra specifinis, bet tai leidžia užtikrinti modelio korektiškumą ir išbaigtumą.

## 6.3 Sistemos ateities darbai

Naujų sistemos algoritmų kūrimą gali įgyvendinti programuotojas. Reikėtų patobulinti tikrinimo algoritmą kad būtų galima pritaikyti hierarchinių būsenų mašinų patikrai ir validavimui.

Įskiepis buvo pradėtas kurti, kai Nomagic kompanija išleido MagicDraw 12 verisją, kuri palaikė UML 2.0 meta modelio notaciją, šiuo metu rinkoje jau pateikta 15.1 versija, kurioje yra atlikta nemažai pakeitimų ir realizuota jau UML 2.1.2 meta modelio notacija. Taip pat MagicDraw openAPI buvo modifikuota ir keletas anksčiau naudotų funkcijų buvo pakeisti kitomis. Reikėtų atlikti MagicDraw openAPI atliktų pakeitimų analizę ir suderinti įskiepį su nauja MagicDraw versija.

# IŠVADOS

1. Modeliais grindžiamo kūrimo procesui reikalingi suderinti, taisyklingi IS modeliai, tačiau esamuose CASE įrankiuose tikrinimo priemonės yra ribotos
2. IS elgsenos suderinamumui užtikrinti tikslinga tikrinti būsenų mašinų teisingumą ir susieti būsenų mašinas su klasėmis
3. Kadangi formalų tikrinimą atliekančių įrankių taikymas yra brangus, buvo sudaryti tikrinimo algoritmai, kuriuos galima tiesiogiai realizuoti CASE įrankiuose.
4. Šių algoritmų realizavimui pasirinktas UML CASE įrankis MagicDraw
5. Sukurtas produktas leidžia projektuotojui greičiau aptikti paliktas klaidas ar nesuderinamumus:
   - ✓ analizuoti abipusį atitikimą tarp klasių ir būsenų mašinų modelių;
   - ✓ analizuoti būsenų mašinų modelių korektiškumą.
6. Eksperimentinis įrankio įvertinimas parodė, kad įrankis randa numatytų tipų klaidas ir sutaupo projektuotojo laiką ir pagerina modelio kokybę.
7. Sukurtą įskiepį galima plėtoti ir naudoti tolimesniuose eksperimentuose:
   - ✓ Įvertinti perėjimų sąlygas;
   - ✓ Išplėsti tikrinimo algoritmus hierarchinėms būsenų mašinoms;
   - ✓ Nagrinėti kitų tipų įvykius: signalų gavimus ir siuntimus;
   - ✓ Generuoti klasių operacijas iš būsenų diagram įvykių.

# LITERATŪRA


[1]. L.Čeponienė, L.Nemuraitė. UML klasių, būsenų ir sekos diagramų suderinimas. Informacinės technologijos ‘2003. ISBN 9955-09-335-8, Kaunas, Technologija, 2003, Sekcija XIV p.62-67.

[2]. Čeponienė L., Nemuraitė L. Patobulintas UML klasių, būsenų ir sekos diagramų suderinimo algoritmas// Informacinės Technologijos’2004.ISBN 9955-09-588-1, Kaunas, Technologija, 2004, Sekcija XI, p. 37- 47.

[3]. L.Čeponienė. Informacijos sistemų reikalavimų analizės ir transformavimo į projektą metodas. Daktaro disertacija.

[4]. Čeponienė L., Nemuraitė L., Design Independent Modeling of Information Systems Using UML and OCL. 2004

[5]. R.Eshuis, D.Jansen and R.Wieringa. Requirements-level semantics and model checking of object-oriented statecharts. *Springer Verlag*, 2002

[6]. MagicDraw product feature overview. [Žiūrėta 2008 01 02], prieiga internete <http://www.magicdraw.com/main.php?ts=navig&NMSESSID=706771c1badd1c157bb2eaff1b07a8be&cmd_show=1&menu=feature_list&NMSESSID=706771c1badd1c157bb2eaff1b07a8be>.

[7]. MagicDraw UserManual version 14.0. NoMagic, Inc, 2007, 768p.

[8]. MagicDraw OpenAPI User‘s Guide. NoMagic, Inc, 2007. [Žiūrėta 2007 10 15], prieiga internete <http://www.magicdraw.com/files/manuals/14/MagicDraw%20OpenAPI%20UserGuide.pdf?NMSESSID=469bba9a3c2fadebc22b3d0490befba6>.

[9]. Unified Modeling Language. Superstructure Specification. Version 2.1.1 OMG document formal/07-02-05, 2007. [Žiūrėta 2007 10 11], prieiga internete <http://www.omg.org/cgi-bin/doc?formal/07-02-05>.

[10]. T.Pender. UML Bible. ISBN:0764526049. John Wiley & Sons, 2003, 940p.

[11]. OMG 2003. MDA Guide Version 1.0.1. [Žiūrėta 2007 10 15], prieiga internete <http://www.omg.org/docs/omg/03-06-01.pdf>.

[12]. What is MagicDraw? No Magic Inc. [Žiūrėta 2007 04 20], prieiga internete <http://www.magicdraw.com/main.php?ts=navig&NMSESSID=4db1e1632f98b380cbbd74cce71ff725&cmd_show=1&menu=what_is>

[13]. M. von der Beeck. Formalization of UML Statecharts, in Proc. UML 2001 Conf., LNCS 2185 Springer-Verlag, pp.406-421, 2001.

[14]. G. Reggio, M.Cerioli, E.Astesiano**.** An Algebraic Semantics of UML Supporting it's Multiview Approach. – Italy: DISI-Universita di Genova, 2001

[15]. ON-THE-FLY, LTL MODEL CHECKING with SPIN [interaktyvus]. 2002 [žiūrėta 2007 10 20]. Prieiga per internetą <http://spinroot.com/spin/whatispin.html>

[16]. Object Constraint Language Environment OCLE [interaktyvus]. [žiūrėta 2007 11 26]. Prieiga per internetą <http://lci.cs.ubbcluj.ro/ocle/>

[17]. S. Gnesi, F. Mazganti $\mu ACTL^{+}$ : A temporal logic for UML statecharts diagrams. Istituto di Scienza e Tecnologie dell'Informazione – ISTI – C.N.R., Pisa

[18]. Unified Modeling Language: OCL Version 2.0, 2006 OMG document formal/06-05-01 [Žiūrėta 2007 03 20] <http://www.omg.org/cgi-bin/doc?formal/2006-05-01>

[19]. Enterprise Architect – UML design tools. [Žiūrėta 2007 11 15], prieiga internete <http://www.sparxsystems.com.au/products/ea.html>.

[20]. UModel 2007 UML 2 Software Development Tool [Žiūrėta 2007 11 15], prieiga internete <http://www.altova.com/UModel2007_013007.html>

[21]. EclipseUML Free Edition 2007 [Žiūrėta 2007 11 15] prieiga internete <http://www.eclipsedownload.com/>

[22]. Visual Paradigm for UML. [Žiūrėta 2007 11 30], prieiga internete <http://www.visual-paradigm.com/news/vpsuite30sp2/vpuml60sp2.jsp>.

[23]. Case Tools [interaktyvus] [Žiūrėta 2007 11 20]. Prieiga per internetą: <http://educ.queensu.ca/~compsci/units/casetools.html>

[24]. Java 2 Platform, Enterprise Edition (J2EE) [interaktyvus] [Žiūrėta 2007 11 26]. Prieiga per internetą: <http://java.sun.com/j2ee/index.jsp>

[25]. MagicDraw. [interaktyvus] [Žiūrėta 2007 11 18]. Prieiga per internetą: <http://www.magicdraw.com/>

# SANTRUMPŲ IR TERMINŲ ŽODYNAS

**API** - Aplikacijų programavimo sąsaja, kurią suteikia kompiuterinė sistema tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis (Application Programming Interface).

**CASE** įrankis – įrankis, skirtas programinės įrangos kūrimo automatizavimui (angl. *Computer-Aided Software Engineering*).

**DIM** – nuo projektavimo metodo nepriklausantis reikalavimų modelis (angl. *Design Independent Model*).

**MDD** – modeliais grindžiamas kūrimas (angl. *Model Driven Development*).

**IS** – informacijos sistema (angl. *Information System*).

**JMI** – java kalbos meta duomenų sąsaja (angl. *Java Metadata Interface*).

**MDA** – modeliu paremta architektūra (angl. *Model Driven Architecture*).

**OCL** – objekto apribojimų kalba (angl. *Object Constraint Language*).

**OGM** – objektų valdymo grupė (*Object Management Group http://www.omg.org*).

**OMG** – Objektų valdymo grupė (angl. *Object Management group*).

**OO** – objektiškai orientuotas (angl. *Object Orientated*).

**CIM** – nuo skaičiavimų nepriklausomas modelis, kitaip vadinamas veiklos modeliu(angl. *Computation Independent Model*).

**DIM** –nuo projektavimo metodo nepriklausantis reikalavimų modelis (angl. *Design Independent Model*).

**PIM** – nuo platformos nepriklausomas modelis (angl. *Platform Indemependent Model*).

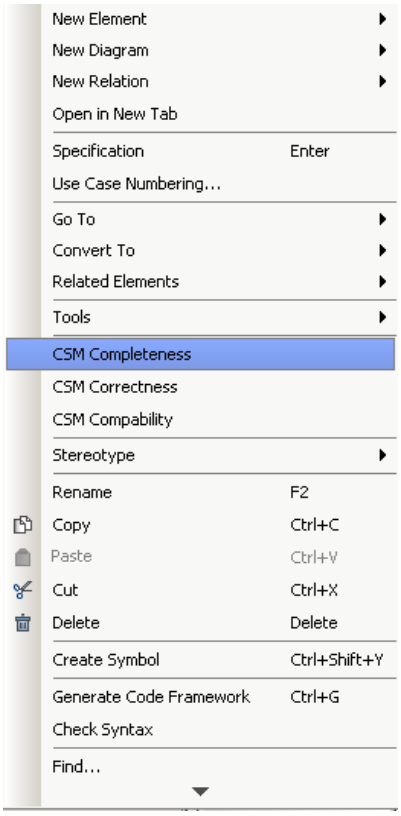
**PSM** – platformai specifinis modelis (angl. *Platform Specific Model*).

**UCM** – panaudos atvejų planas (angl. *Use Case Map*).

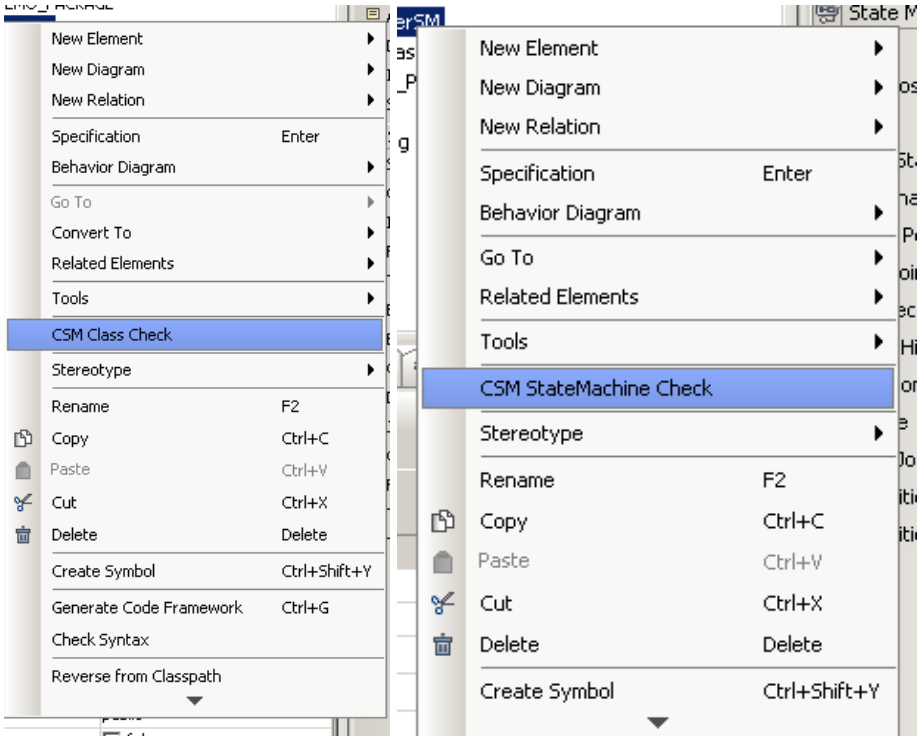
**UML** – unifikuota modeliavimo kalba (angl. *Unified Modeling Language*).

**XMI** – formatas, skirtas keitimuisi UML modeliais tarp CASE įrankių, pagrįstų XML (angl. *XML Metadata Interchange*).

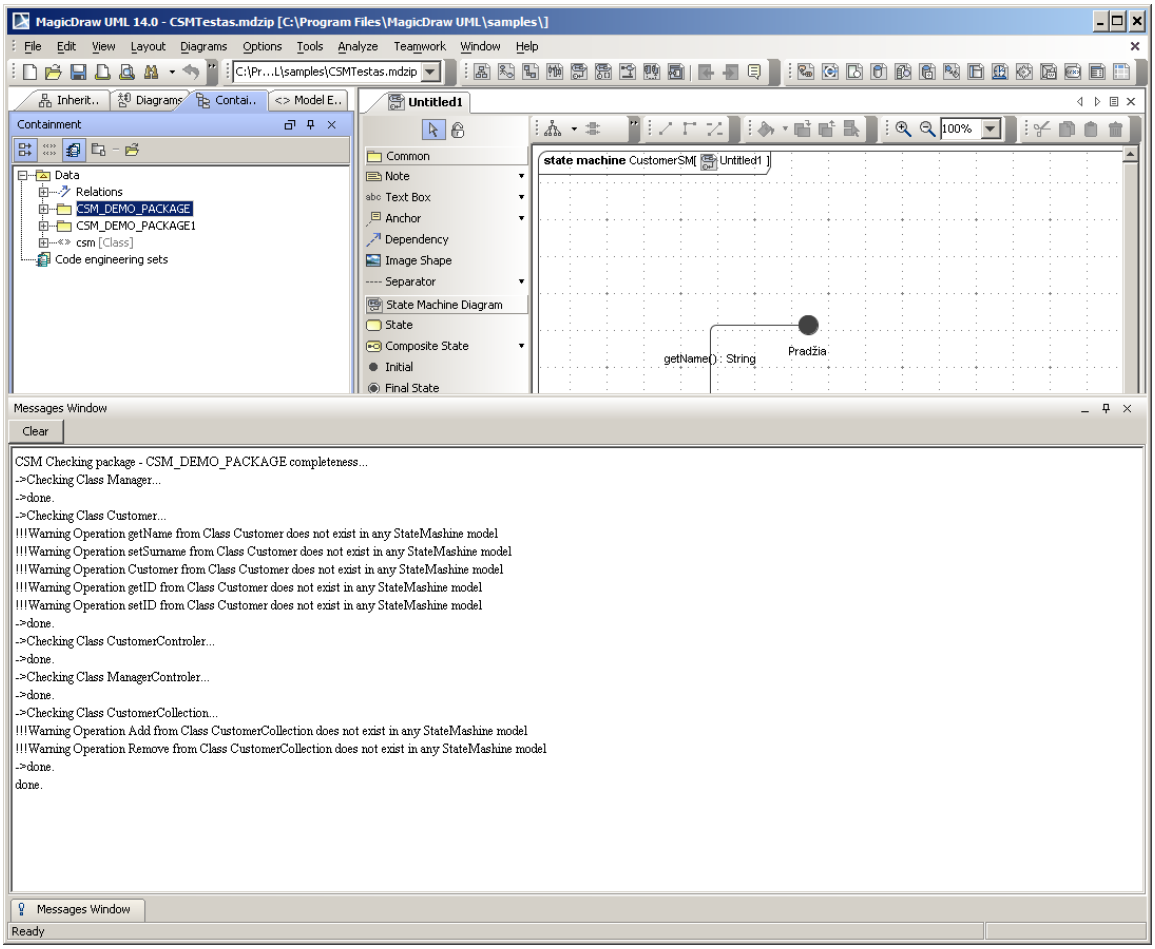
**XML** – bendros paskirties duomenų struktūrų bei jų turinio aprašomoji kalba (angl. *Extensible Markup Language*).




**26 pav. MagicDraw paketo meniu išplėtimas**




**27 pav. MagicDraw klasės, ir būsenų mašinos meniu išplėtimas**




**28 pav. MagicDraw aptiktų klaidų pranešimų langas**