KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Viktoras Mickūnas

ELEKTRONINIŲ PASLAUGŲ MODELIŲ VERIFIKAVIMO GALIMYBIŲ TYRIMAS

Magistro darbas

Vadovas
doc. dr. L. Nemuraitė

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. R. Butleris

ELEKTRONINIŲ PASLAUGŲ MODELIŲ VERIFIKAVIMO GALIMYBIŲ TYRIMAS

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių k. katedros lekt.
dr. J. Mikelionienė

Vadovas
doc. dr. L. Nemuraitė

Recenzentas
doc. dr. A. Lenkevičius

Atliko
IFM 9/1 gr. stud.
V. Mickūnas

KAUNAS, 2005

SUMMARY

Web services provide means to computerize e-business processes. To be able to satisfy business requirements web services must be developed rapidly and in high quality. Code generation techniques are frequently employed. In such context, verification of web service models emerges as a relevant issue: model verification makes it easier to detect and remove errors of system specification at the early stages of software development cycle.

Available model verification tools require transformation of the objective model into complex formal notations, supported by the tool. It would be convenient to implement model verification directly in a CASE tool.

This thesis describes a method for checking web service models based on verification of UML state machines. Conceptual algorithms for checking state machine's safety criteria: reachability, completeness and consistency are provided. These algorithms, implemented in CASE tools could help to ensure the correctness and consistency of behavioral models. Also, a state machine template for composite web services is presented. Template complements state machine with transitions representing unsuccessful scenarios. Designer is freed from necessity to repeat reoccurring unsuccessful transitions for every event. This template is useful for automatic code generation.

TURINYS

Lentelių sąrašas

Paveikslėlių sąrašas

ĮVADAS

Programinės įrangos projektų efektyvumo akcentavimas lėmė gerai specifikuotų ir standartizuotų projektavimo metodų bei kalbų kūrimą. Tokios kalbos turėtų tenkinti keletą pagrindinių reikalavimų. Pirma, projektuotojams reikia bendro taikymo kalbos, kuri būtų lengvai suprantama, artima jų mąstysenai, kuri leistų manipuliuoti vizualiais modeliais, atspindinčiais sistemą įvairiais rakursais. Antra, projekto validavimas ir formalus verifikavimas reikalauja tikslios sintaksės ir semantikos, apibrėžtos matematiniais terminais.

Universali modeliavimo kalba UML yra sukurta, atsižvelgiant į šiuos reikalavimus. Tai yra bendros paskirties objektinės orientacijos kalba, skirta įvairiems sistemos aspektams specifikuoti, vizualizuoti ir dokumentuoti. Deja, modeliai, specifikuoti UML kalba, dažnai būna nepilni, nenuoseklūs ir dviprasmiški. Aptikti ir ištaisyti klaidas specifikacijoje, ypač vėlesnėse įrangos kūrimo ciklo fazėse, yra ne tik labai sunku, bet ir brangu. Be to, modelių neapibrėžtumas užkerta kelia automatiniam programinio kodo generavimui. Taigi automatinis UML modelių verifikavimas turi didelę reikšmę.

Egzistuoja nemažai įrankių, analizuojančių įvairių būsenų modelių atmainų korektiškumą. Deja, visi jie remiasi savais formalizmais ir turi savas, specializuotas modelių aprašymo kalbas. Sintaksinė ir semantinė „spraga“ tarp modelio, specifikuoto UML ir tokio įrankio yra per didelė, kad projektuotojas galėtų patogiai verifikuoti modelį.

Todėl būtų naudingas algoritmas, analizuojantis būsenų modelio vientisumą ir nuoseklumą ne iš formalios, neretai sunkiai išmokstamos ir skaitomos specifikavimo kalbos, o tiesiogiai iš sistemos elgsenos pateikties būsenų mašinų modelio pavidalu.

Būsenų mašinų naudojimas sistemų veikimui tikrinti ypatingai aktualus tampa mūsų dienomis, kai didelę programinės įrangos dalį sudaro įvairios specifikacijos, kurias galima generuoti automatiškai. Įvairių metodų (pvz., MDA) dėka didėja ir automatiškai generuojamo reikšmingo kodo dalis. Be to, vis aiškiau suvokiama, kad tikrinimą reikia atlikti ankstesniame kūrimo etape: kuo abstraktesnis kūrimo modelis, tuo daugiau projekto rezultatų galima juo paremti. Reikia paminėti, kad ,,abstraktesnis“ nereiškia „netikslus“: atvirkščiai, siekiama modelių tikslumo. Šiame darbe nagrinėjamos UML būsenų mašinos tam tikrai sričiai: elektroninių paslaugų sistemoms tikrinti. Tai nereiškia, kad būsenų mašinos yra specifiškai apribotos, tačiau išanalizuota eksperimentinių taikymų aibė yra orientuota paslaugų architektūra paremtoms sistemoms, neapimant, pavyzdžiui, realaus laiko ar įterptinių sistemų problematikos.

Šio darbo tikslas – ištirti e. paslaugų būsenų diagramų verifikavimo galimybes ir, suformulavus statinius ir dinaminius korektiškumo kriterijus, pateikti verifikavimo algoritmą. Taip pat ištirti, ar galima būsenų mašiną pateikus pagal tam tikras taisykles iš karto užtikrinti statinių ir/arba dinaminių kriterijų

išpildymą, išvengiant ilgai trunkančios ir dažnai „būsenų sprogimą“ sukeliančios perėjimų tikrinimo operacijos; išsiaiškinti ir eksperimentiškai arba teoriškai parodyti, kada tikrinimo galima išvengti arba kada jis yra būtinas.

Tiriant e. paslaugų modeliavimo specifiką, buvo remtasi literatūros šaltiniais [1], [2], kuriuose aprašomos e. paslaugų technologijos, architektūros principai ir savybės. Kiti nagrinėti šaltiniai susiję su UML kalbos analize [3] ir bendriniais būsenų mašinų bei UML būsenų diagramų verifikavimo metodais [4], [5], [8]. Verifikavimo metodo realizacijos galimybių UML CASE įrankiuose analizė paremta [9], [12] šaltiniais. Darbo tikslai ir uždavinių formuluotė susiję su Informacinių sistemų katedroje sukurtu paslaugoms orientuotų informacinių sistemų modeliavimo metodu, kuris išdėstytas [6], [7] šaltiniuose.

E. paslaugos yra modulinės save aprašančios taikomosios programos, kurios gali būti publikuojamos, randamos ir iškviečiamos iš bet kurio žiniatinklio arba vietinio tinklo taško. E. paslaugos turi sąsają, aprašančią tinklu prieinamų operacijų rinkinį, pateikiamą XML standartu paremtu WSDL dokumentu. Klientas su e. paslauga sąveikauja keisdamasis pranešimais, kurie transportuojami atvirais standartais aprašytais protokolais (dažniausiai SOAP).

Kompozicine e. paslauga vadinama paslauga, kuri savo funkcionalumą realizuoja naudodama kitų e. paslaugų teikiamus servisus. Tokia paslauga yra sistema, kurios elementai – autonominės e. paslaugos, turinčios skirtingas sąsajas ir elgsenos ypatumus, todėl būtina suderinti įvairiausių tipų sąveikas, saistančias sistemos elementus. Siekiant suderinti šias sąveikas, jos apibendrinamos sudarant būsenų mašinas, kurias galima generuoti iš daugelio sąveikas aprašančių UML sekų diagramų ([7]).

Kuriant e. paslaugas dažnai pageidaujama naudoti automatinį kodo generavimą iš sistemos elgsenos modelių, kurių klasei priklauso ir UML būsenų mašinos. Deja, kokybiškas kodo generavimas įmanomas tiktai turint griežtai, vienareikšmiškai ir korektiškai specifikuotus būsenų modelius, todėl būsenų modelių verifikavimas yra aktuali problema. Be to, tikrinant sistemą dar modelio lygmenyje, galima anksčiau aptikti ir ištaisyti klaidas sistemos specifikacijoje, o tai leidžia tiesiogiai sumažinti e. paslaugų realizavimo išlaidas.

Egzistuojantys modelio tikrinimo įrankiai reikalauja objektinio modelio transformacijos į sudėtingas įrankio formaliąsias notacijas. Tam reikalingas papildomas analizės etapas bei aukšta analitiko kvalifikacija. Todėl būtų patogu e. paslaugų elgsenos, specifikuotos, būsenų mašinų modeliu, tikrinimą realizuoti tiesiogiai CASE įrankyje. Toks tikrinimo metodas padėtų užtikrinti būsenų modelio suderinamumą su kitais sistemos modeliais. Be to jis užtikrina pakartotinį panaudojimą, kadangi iš sudaryto korektiškos būsenų mašinų modelio galima generuoti sistemos realizaciją įvairioms programavimo kalbomis.

Šiame darbe sudaryti algoritmai, leidžiantys patikrinti e. paslaugų būsenų mašinų modelių saugumo kriterijus. Nagrinėti saugumo kriterijai apima būsenų pasiekiamumą, grįžtamumą, pilnumą ir nuoseklumą. Būsenų pasiekiamumo kriterijus reikalauja, kad modelyje nebūtų izoliuotų būsenų, t. y. neturinčių įeinančių perėjimų. Grįžtamumo kriterijus reiškia, kad iš kiekvienos e. paslaugos būsenos turi egzistuoti kelias į laukimo būseną, kurioje sistema yra pasiruošusi aptarnauti kliento užklausas. Pilnumas kiekvienai leidžiamai įvykių sekai užtikrina vienareikšmiškai apibrėžtą veiksmų seką. Nuoseklumas garantuoja perėjimų vykdymo sąlygų neprieštaringumą ir elgsenos determinizmą.

Norint pagreitinti e. paslaugų kūrimo procesą, pasiūlytas e. paslaugų būsenų mašinos šablonas. Būsenų mašina apibendrina paslaugų sistemos elgseną, kuri aprašoma įvairių tipų sąveikų modeliais. Paprastai sąveikų modeliai apima tik pagrindinius scenarijus, kadangi pilna scenarijų aibė, apimanti visus nesėkmingus atvejus, yra labai didelė. Sukurtas šablonas padeda papildyti kompozicinės e. paslaugos būsenų mašiną nesėkmingus vykdymo scenarijus atitinkančių perėjimų aibe. Kad sistema būtų stabili, iš kiekvienos būsenos turi egzistuoti kelias, leidžiantis grįžti į laukimo būseną. Be to, kiekvieną kartą po kreipinio į kitą paslaugą turi būti apibrėžti du perėjimai: vienas, kai ta paslauga įvykdoma sėkmingai ir kitas – kai nesėkmingai. Apibendrinant, kiekvienai paslaugai turi egzistuoti grįžimas į laukimo būseną, o kiekvienam įvykiui – sėkmės ir nesėkmės alternatyvos. Pasiūlytą šabloną tikslinga naudoti automatiškai generuojant e. paslaugų sistemos programos kodą ir specifikacijas, kadangi tuomet pakanka aprašyti tik pagrindinį scenarijų, o nesėkmingiems atvejams sukuriami standartiniai apdorojimo veiksmai.

Pirmame skyriuje atlikta e. paslaugų modelių verifikavimo problemos analizė. Pateikta e. paslaugų apibrėžtis, aprašyta e. paslaugų architektūra, apibendrinta paslaugų sąsają specifikuojančių WSDL dokumentų struktūra, atliktas e. paslaugų ir komponentų palyginimas. Apibrėžtos kompozicinės e. paslaugos. Apžvelgti būsenų modelių, kuriais patogu vaizduoti e. paslaugų elgseną, korektiškumo kriterijų tikrinimo metodai.

Antrame skyriuje pateiktas siūlomas e. paslaugų verifikavimo metodas, paremtas e. paslaugų būsenų mašinų modelių tikrinimu. Sudarytas būsenų mašinų semantiką atspindintis metamodelis, papildytas vykdymo elementais. Apibrėžti bendri būsenų modelių korektiškumo kriterijai – saugumas ir gyvybingumas. Suformuluoti saugumo kriterijų grupei priklausantys būsenų pasiekiamumo, grįžtamumo, pilnumo ir nuoseklumo kriterijai. Pasiūlyti algoritmai šiems kriterijams tikrinti. Taip pat pateiktas kompozicinės e. paslaugos šablonas, padedantis papildyti paslaugos modelį perėjimais, atitinkančiais nesėkmingus vykdymo scenarijus.

Trečiame skyriuje pateiktas kompozicinės e. paslaugos pavyzdys. Pavyzdžio fragmentui pademonstruotas siūlomo pilnumo ir nuoseklumo tikrinimo algoritmo bei kompozicinės e. paslaugos šablono taikymas.

Ketvirtame skyriuje suformuluotos darbo išvados.

Darbo tema atspausdintas straipsnis ir perskaitytas pranešimas konferencijoje „Informacinės technologijos ’05“ [11].

ELEKTRONINIŲ PASLAUGŲ MODELIŲ VERIFIKAVIMO PROBLEMOS ANALIZĖ

1.1. Pagrindinės elektroninių paslaugų architektūros sąvokos

1.1.1. Elektroninės paslaugos apibrėžtis

E. paslaugos (angl. *web services*) yra modulinės save aprašančios taikomosios programos, kurios gali būti publikuojamos, randamos ir iškviečiamos iš bet kurio žiniatinklio (angl. *Web*) arba vietinio tinklo taško. E. paslaugos tiekėjas ir vartotojas gali nesirūpinti operacine sistema, kalbos aplinka ar komponentų modeliu, naudotu kuriant e. paslaugą, kadangi pastarosios yra grindžiamos visuotinais ir atvirais interneto standartais - XML, HTTP ir SMTP.

E. paslauga – tai programinės įrangos sistema, suprojektuota, siekiant užtikrinti heterogeninių kompiuterių sistemų sąveiką tinkle. Ši sistema turi sąsają, specifikuotą standartizuota WSDL (angl. *Web Service Definition Language*) kalba. Kitos sistemos sąveikauja su e. paslauga, atsižvelgdamos į paslaugos nustatytas taisykles, SOAP (angl. *Simple Object Access Protocol*) protokolo pranešimais. E. paslaugos sąsaja, aprašanti tinklu prieinamų operacijų rinkinį, realizuojama pasitelkiant standartizuotą XML pranešimų mechanizmą. E. paslaugos yra fundamentalūs komponentai paskirstytųjų skaičiavimų skverbimosi į internetą kontekste. Atvirieji standartai ir sistemų sąveikavimo akcentavimas sąlygojo palankią terpę e. paslaugoms, tampančioms patrauklia taikomųjų programų integracijos platforma.

Egzistuoja daug e. paslaugų apibrėžimų, tačiau šias savybes mini dauguma jų:

- E. paslaugų funkcionalumas, prieinamas vartotojams, naudojantis standartiniu žiniatinklio protokolu. Dažniausiai naudojamas SOAP protokolas.
- E. paslaugos pateikia pakankamai išsamų ir tikslų savo sąsajos aprašymą. Aprašas paprastai pateikiamas WSDL dokumentu.
- E. paslaugos registruojamos ir katalogizuojamos, kad potencialūs vartotojai galėtų lengvai jas surasti. Tai užtikrina UDDI (angl. *Universal Discovery Description and Integration*) mechanizmas.

Vienas iš pagrindinių e. paslaugų architektūros privalumų – skirtingomis programavimo kalbomis parašytų ir skirtingose platformose veikiančių programų komunikavimo galimybė, naudojantis nesudėtingais, standartizuotais, XML grįstais protokolais.

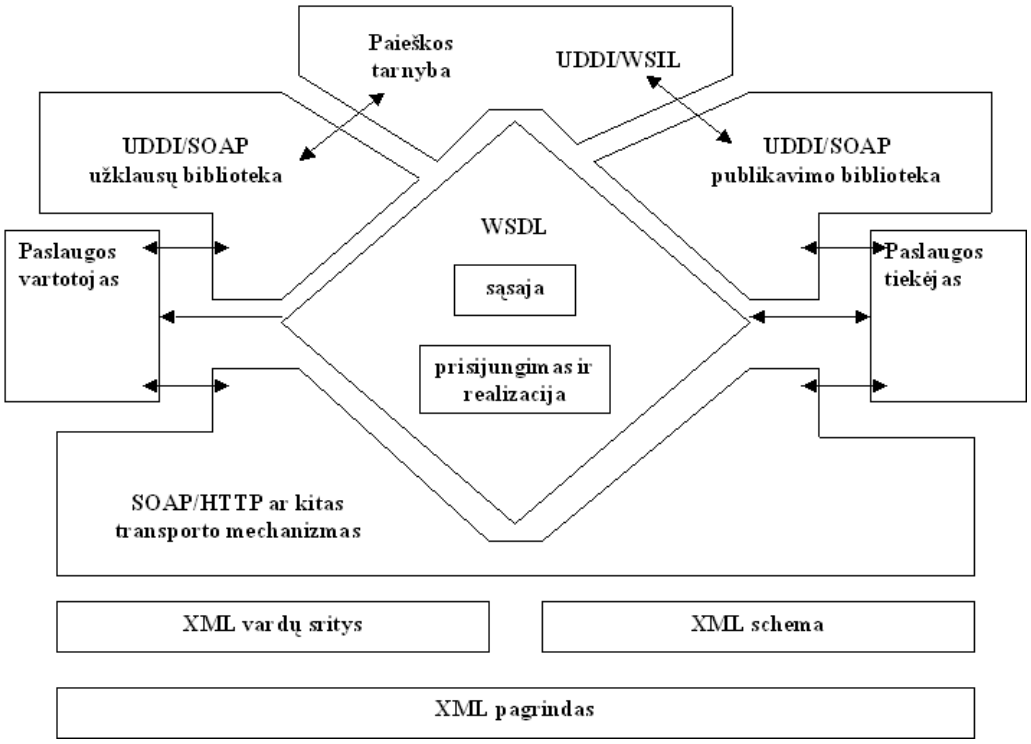
Kitas ženklus privalumas – finansinis aspektas. Kadangi dauguma įmonių savo infrastruktūroje jau naudoja standartinius HTTP ir TCP/IP protokolus ir turi apmokytą personalą, e. paslaugų naudojimas reikalauja mažesnių papildomų investicijų, nei kitos technologijos.

1.1.2. E. paslaugų architektūra

E. paslaugų architektūra sudaryta iš trijų posistemių: *paieškos*, *aprašymo* ir *vykdymo*. Paiešką užtikrina UDDI registras. Aprašymas pateikiamas WSDL kalba, o vykdymo mechanizmas remiasi SOAP protokolu.

Galima išskirti šias esybes:

- *Paslaugos tiekėjas*, siūlantis e. paslaugą internete, ekstranete arba intranete.
- *Paslaugos vartotojas*, ieškantis ir besinaudojantis e. paslauga, teikiama paslaugos tiekėjo. Informaciją apie taip, kaip prisijungti prie paslaugos tiekėjo ir sužadinti reikiamą paslaugą gaunama, kreipiantis į paieškos tarnybą.
- *Paieškos tarnyba* (angl. *discovery agency*), nukreipianti paslaugos vartotoją prie paslaugos tiekėjo. Ši tarnyba turi saugyklą, kurią, ieškodamas paslaugos, paslaugos vartotojas gali naršyti, gauti paslaugos sąsajos specifikaciją bei prisijungimo informaciją. Tiekėjas naudojasi saugykla publikuodamas paslaugas, kurias nori pasiūlyti.



1.1 pav. E. paslaugų architektūra

1.1 pav. pavaizduota, kaip sistemos esybės siejasi viena su kita. Apačioje matome XML pagrindą; visos kitos struktūrinės dalys remiasi juo.

Paslaugos vartotojas ir *paslaugos tiekėjas* sąveikauja trijuose lygmenyse:

- *WSDL lygmuo*. Charakterizuoja e. paslaugos sąsają. WSDL dokumentas aprašo sintaksinį susitarimą (angl. c*ontract)*, kurio paslaugos vartotojas turi laikytis. Toks susitarimas susideda iš dviejų dalių:
    - Abstraktaus *sąsajos aprašo*, specifikuojančio palaikomas operacijas, operacijų parametrus bei jų tipus;
    - *Prisijungimo ir realizacijos aprašo*, susiejančio abstraktų sąsajos aprašą su konkrečiu protokolu ir tinklo adresu.

  Tam pačiam sąsajos aprašui gali egzistuoti daug prisijungimo taškų ir realizacijų. Tai reiškia, kad e. paslauga gali būti prieinama, naudojantis skirtingais protokolais ir pasiekiama keliais skirtingais tinklo adresais.
- *SOAP lygmuo*. Atlieka ryšių protokolo funkciją tarp vartotojo, tiekėjo ir paieškos tarnybos. SOAP yra XML pranešimų ir nuotolinio procedūrų iškvietimo (angl. *Remote Procedure Call*) formatas.
- *UDDI* ir *WSIL* (angl. *Web Services Inspection Language*) *lygmuo*. UDDI ir WSIL yra dvi atskiros publikavimo / paieškos mechanizmo realizacijos.

  UDDI registras pateikia bendrą saugyklą, kurioje paslaugos tiekėjai publikuoja informaciją apie paslaugas, o paslaugos vartotojai tos informacijos ieško. UDDI apibrėžia registro struktūrą kartu su užklausų ir publikavimo biblioteka

  WSIL yra paprastesnė UDDI alternatyva.

1.1.3. WSDL dokumentai

Vartotojų programos, automatiškai besijungiančios prie e. paslaugų, turi gauti formalų paslaugos aprašymą, teikiantį informaciją apie paslaugos išdėstymą ir paslaugos sąsajas. Turėdamas šią informaciją, vartotojas žino, kaip sąveikauti su paslauga. WSDL yra formali kalba, būtent ir skirta specifikuoti šiai informacijai. WSDL dokumentas iš esmės yra paslaugos sąsajos kontraktas, parašytas XML kalba, kurio sąlygas vartotojas turi patenkinti. Vartotojas, ieškodamas paslaugos, geriausiai atitinkančios jo poreikius, gali peržvelgti keletą tokių dokumentų.

WSDL dokumento struktūra apibendrinta 1.1 - 1.3 lentelėse. Kaip ir minėta, e. paslaugos aprašymas susideda iš abstrakčios sąsajos aprašo ir iš realizacijų aprašų (1.1 lentelė).

1.1 lentelė. WSDL dokumento struktūra

Dokumento dalis	**Aprašymas**
abstrakti sąsaja	Sąsajos aprašas apibrėžia paslaugos palaikomas operacijas, operacijų parametrus ir abstrakčius duomenų tipus. Šis aprašas yra visiškai nepriklausomas nuo konkretaus tinklo adreso, komunikacinio protokolo ar konkrečių duomenų struktūrų.
konkreti realizacija	Realizacijos aprašas susieja abstrakčios sąsajos aprašą su konkrečiu tinklo adresu, protokolu ir konkrečiomis duomenų struktūromis. E. paslaugos vartotojas gali prisijungti prie tokios realizacijos ir sužadinti paslaugą.

1.2 lentelė. WSDL dokumento sąsajos aprašo elementai

Sąsajos aprašo elementas	**Aprašymas**
portType	Porto tipo elementas `portType` yra vardą turintis operacijų rinkinys.
operation	Operacijos elementas `operation` abstrakčiai aprašo paslaugos iškvietimą. Šiame elemente gali būti aprašytas įeigos pranešimo elementas `input` ir išeigos pranešimo elementas `output`. Taip pat galima nurodyti keletą klaidos pranešimo elementų `fault`.
message	Pranešimo elementas `message` yra abstraktus siunčiamų duomenų aprašymas. Pranešimas susideda iš loginių vienetų vadinamų dalimis.
part	Kiekvienas dalies elementas `part` yra susiejamas su duomenų tipu.
types	Typų elementas `types` yra duomenų tipų apibrėžčių talpykla. WSDL specifikacijos 1.1 versija rekomenduoja duomenų tipų aprašymui naudoti XML schemas.

Centrinis elementas, įkūnijantis sąsajos aprašymą (1.2 lentelė) yra `portType` elementas. Šiame elemente pateikiami visų e. paslaugos palaikomų operacijų aprašai.

Operacijos apraše išvardinami visi pranešimai, kuriais potencialiai gali keistis e. paslaugos tiekėjas ir vartotojas. Jeigu papildomai aprašytume klaidų (`fault`) pranešimus, jie taip pat būtų pateikiami kaip elemento `operation` nariai.

Pranešimas susideda iš dalių, o dalys yra susietos su tipais. Pranešimas atspindi operacijos duomenų tipą, o dalys gali tą tipą struktūrizuoti.

Perskaitęs WSDL sąsajos aprašą, klientas įgauna aiškų supratimą apie paslaugos sąsajos išplanavimą, tačiau jam vis dar trūksta konkrečios informacijos apie tinklo prieigą ir protokolą.

Centrinis realizacijos aprašo elementas (1.3 lentelė) yra sąryšio elementas `binding`. Šis elementas atvaizduoja porto tipą, t.y. paslaugos sąsajos aprašą, į egzistuojančią paslaugos realizaciją. Verta pastebėti, kad sąryšio elementas netalpina jokios su konkrečia programavimo kalba arba su konkrečia paslaugos realizacija susijusios informacijos. Tokia informacija yra už WSDL akiračio ribų.

Elementas `port` aprašo paslaugos tinklo adresą. Paslauga per portą keičiasi apibrėžto formato pranešimais. E. paslaugos vartotojas gali jungtis ir sąveikauti su paslauga tiktai tuo atveju, jeigu jis laikosi konkretaus pranešimų formato, kurio tikisi e. paslauga. Ta pati paslauga gali būti prieinama skirtingais duomenų formatais ir keliais skirtingais portais. Vartotojas, norintis sąveikauti su sistema gali pasirinkti vieną iš šių portų.

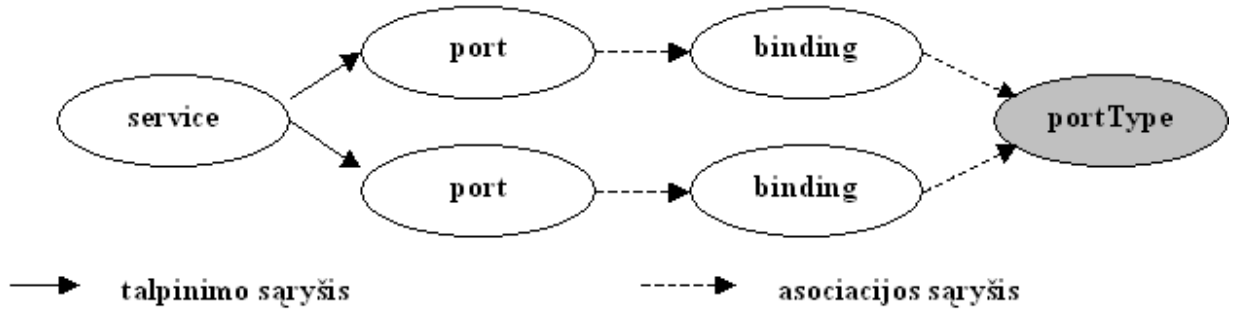
1.3 lentelė. WSDL dokumento realizacijos aprašo elementai

Realizacijos aprašo elementas	**Aprašymas**
binding	Sąryšio elementas `binding` pateikia konkrečias tam tikro porto tipo (`portType`) protokolo bei duomenų formato specifikacijas.
port	Porto elementas `port` aprašo paslaugos tinklo adresą. Kartu su sąryšio elementu (`binding`), šis elementas nusako vieną konkretų paslaugos prieigos tašką.
service	Elementas `service` aprašo portų rinkinį. Šie portai gali turėti tokį patį porto tipą (`portType`), tačiau skirtingus sąryšius (`bindings`) ar tinklo adresus. Taip pat, kiekvienas portas gali būti konkreti paslaugos prieigos taško realizacija atitinkanti kurį nors porto tipą.

Verta pastebėti elemento `binding` atskyrimą nuo elemento `port`. Elementas `binding` yra pakartotinio panaudojimo informacinė esybė. Ta pati paslaugos realizacija gali būti prieinama skirtingais tinklo adresais.

Galiausiai elementas `service` išvardina visus portus, t.y. tinklo adresus, kuriais yra prieinama e. paslauga.

1.2 pav. parodomas sąryšis tarp porto tipo, pateikiančio abstraktų paslaugos sąsajos aprašą, nuspalvintą pilkai, ir paslaugos realizacijos aprašo struktūrinių dalių.



1.2 pav. Sąryšis tarp sąsajos aprašo ir realizacijos aprašo elementų

1.2. E. paslaugų ir komponentų palyginimas

Nors tarp komponentų ir e. paslaugų yra nemažai panašumų, tačiau yra ir skirtumų. Pagrindinis skirtumas yra tas, kad komponentai yra kuriami, mąstant apie perspektyvą naudoti juos, derinant su kitais komponentais, o e. paslaugos yra visiškai nepriklausomos. E. paslaugos kūrimo metu visai nebūtina žinoti, kokia kita paslauga gali su ja kooperuoti. Tai savo ruožtu lemia mažesnį susietumą.

Kitas skirtumas glūdi transportavimo mechanizme. Komponentai dažniausiai naudoja tarpines integruojančias programines technologijas, tokias kaip CORBA, e. paslaugos ryšiui naudoja žiniatinklį.

1.4 lentelė. Komponentų ir e. paslaugų palyginimas

Savybė	**Komponentai**	**e. paslaugos**
Gali būti savarankiškai realizuojami, testuojami bei modifikuojami	taip	taip
Sąsajos specifikacija	taip	taip
Tik išreikštinės kontekstinės priklausomybės	taip	taip
Transportavimas	tarpinės programinės technologijos	žiniatinklis

Standartai	komerciniai	atviri
Susietumas	glaudus	lengvas
Realizacija	atskira	savarankiška
Sąsajos	formalios	sutartinės

1.3. Kompozicinės e. paslaugos

Kompozicine e. paslauga vadinama paslauga, kuri savo funkcionalumą realizuoja naudodama kitų e. paslaugų teikiamus servisus. Pažymėtina, kad kompozicinės e. paslaugos klientas sąveikauja tik su ja viena ir gali nieko nežinoti apie šios paslaugos ryšius su kitais paslaugų tiekėjais.

Kadangi kompozicinė e. paslauga yra sistema, kurios elementai – autonominės e. paslaugos, turinčios skirtingas sąsajas ir elgsenos ypatumus, korektiškas šios sistemos darbas įmanomas tiktai tada, kada yra suderinamos įvairiausių tipų sąveikos, saistančios sistemos elementus. Šios sąveikos apima: sąveikas su vartotoju, aprašomas kompozicinės e. paslaugos palaikomomis sąsajomis; sistemos sąsajų sąveikas, kurių visuma sudaro orkestruotę; globalaus proceso modelį, kuris susideda iš sąveikų fragmentų, vadinamų choregrafijomis. Siekiant suderinti šias sąveikas, jos apibendrinamos sudarant būsenų mašinas, kurias galima generuoti iš daugelio sąveikas aprašančių UML sekų diagramų ([7]).

1.4. Korektiškumo kriterijų tikrinimo metodai

Korektiškumo kriterijų tikrinimui dažniausiai pasitelkiami įrankiai, kadangi „rankinis“ modelio tikrinimas užima daug laiko ir reikalauja didelių projektuotojo pastangų. Galima išskirti šiuos automatizuotus metodus ir priemones:

- Sistemos būsenų erdvės *pasiekiamumo analizė* gali aptikti dviprasmiškas situacijas, sąlygojamas nepasiekiamų būsenų arba nepageidautinos įvykių sekos. Dar vienas tikrinimo objektas – perėjimai, kurie niekuomet nėra aktyvuojami. Globalios būsenų erdvės tyrimas reikalauja pasiekiamumo grafo (angl. *reachability graph*) generavimo.
- *Teoremų įrodinėjimo sistemos* grindžiamos aksiomatine matematika. Tiek sistemos specifikacija, tiek ir sistemos realizacija aprašoma logine formule. Tada tikrinama, ar iš sistemos specifikacijos logiškai seka sistemos realizacija, t. y. siekiama įrodyti teoremą. Jei teoremą pavyksta įrodyti, galima teigti, kad realizacija tenkina specifikaciją. Šis metodas yra pusiau automatinis ir reikalauja kvalifikuoto specialisto įsikišimo. Dažniausiai naudojamas akademinėms užduotims verifikuoti (pvz., tikrinti aritmetiniams algoritmams ir pan.). Pagrindiniai metodo trūkumai: patogių automatizuotų įrankių stoka, sunkiai pritaikomas programinių sistemų elgsenos verifikavimui, negarantuojamas įrodymo radimas, sudėtingas

įrodymo procesas, nelokalizuojamas realizacijos fragmentas, prieštaraujantis specifikacijai (angl. *couterexample*).

- *Modelių tikrinimas* (angl. *model checking*) – tai metodas, skirtas verifikuoti lygiagrečiosioms sistemoms, turinčioms baigtinį būsenų skaičių [10]. Tokių sistemų pavyzdžiais galima laikyti sekvencinės logines schemas ar ryšio protokolus. Lyginant su tradiciniais metodais, grindžiamais simuliacija, testavimu ir dedukciniu išvedimu, modelių tikrinimo metodas turi svarių privalumų. Pirmiausia, šis metodas pilnai automatizuotas ir paprastai yra gana greitas. Be to, jeigu modelyje yra klaida, tai ji bus ne tik rasta, bet ir lokalizuota.

  Modelių tikrinimo priemonės tikrina, ar sistemos specifikacija tenkina savybes, išreikštas laikine logika (angl. *temporal logics*). Pačią modelių tikrinimo problemą formaliai galima suformuluoti taip: turint Kripkės struktūrą

  $$M = (S, R, L)$$

  (čia $S = \{s_1, s_2, ..., s_i, ...s_n\}$ – būsenų aibė,

  $R = \{(s_i, s_j)\}$ – perėjimų tarp būsenų santykis,

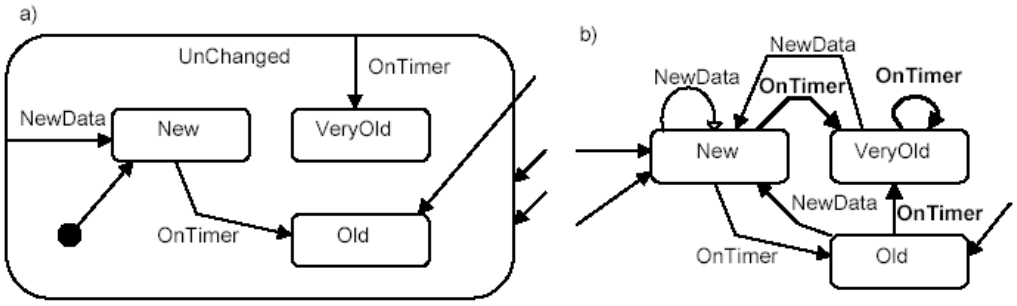
  $L : S \to 2^{AP}$ - žymių funkcija. Šios funkcijos reikšmė $L(s)$ yra visų atominių teiginių poaibis, kurio elementai būsenoje *s* turi reikšmę *true*),

  kuri aprašo baigtinių būsenų skaičiaus lygiagrečiąją sistemą, ir laikinės logikos formulę *f*, išreiškiančią kažkokią savybę, reikia rasti visas būsenas iš aibės *S*, tenkinančias savybę *f*.

  Pagrindiniai modelių tikrinimo metodo sunkumai išryškėja susidūrus su būsenų erdvės sprogimo problema. Ši problema pasireiškia sistemose, turinčiose daug tarpusavyje sąveikaujančių komponentų arba sistemose, naudojančiose duomenų struktūras, galinčias įgyti didelį reikšmių skaičių. Tokiais atvejais globalių būsenų skaičius gali išaugti nevaldomai.

  Modernios modelių tikrinimo priemonės būsenų erdvės aprėpiamumo problemą bando spręsti, taikydamos sudėtingas metodikas, atvaizduojant ir analizuojant globaliąją būsenų erdvę.

- *Grafų transformavimo metodai.* UML būsenų mašina iš esmės galima laikyti grafu. Taigi, grafų transformavimo taisykles galima taikyti būsenų modeliams modifikuoti arba transformuoti. Tikrinant korektiškumo kriterijus, grafų transformavimo metodus galima taikyti dvejopai:
  - Modelį galima transformuoti į formą, tinkamesnę tikrinimui. Pvz., prieš tikrinant OCL išraiškas, galima supaprastinti, „ištiesinti“ hierarchinį modelį;



1.3 pav. Hierarchinio modelio „ištiesinimas“

- Sistematiškai pašalinus iš modelio pilnas ir nuoseklias dalis, galiausiai modelyje liks tik tie specifikacijos fragmentai, kurie turi projektinių trūkumų.

Apskritai, grafų transformavimo taisyklės apibrėžiamos, pateikiant kairiąją taisyklės pusę (sąlygas) ir dešiniąją taisyklės pusę (rezultatą). Sistema bando ieškoti modelyje fragmentų, tenkinančių kairiąją taisyklės pusę. Jeigu rasti pavyksta, kairiąją taisyklės pusę atitinkanti modelio dalis transformuojama taip, kaip reikalauja dešinioji taisyklės pusė.

1.5 lentelė apibendrina trijų automatizuoto modelių tikrinimo įrankių palyginimo rezultatus.

Spin yra efektyvus programinės įrangos verifikavimo įrankis, orientuotas į lygiagrečiųjų ir paskirstytųjų sistemų modelių tikrinimą. Spin naudotas operacinių sistemų, duomenų perdavimo protokolų, lygiagrečiųjų algoritmų loginio projektavimo klaidų diagnostikai. Sistemos specifikacija šiam įrankiui pateikiama PROMELA kalba. Įrankis tikrina specifikacijos loginį nuoseklumą, aptinka aklavietes (angl. *deadlock*). Tikrinamas korektiškumo savybes galima specifikuoti naudojant sistemos ar procesų invariantus, LTL logiką, Biuchi automatus.

Įrankis Optimyz Web Service Tester teikia priemones e. paslaugų funkciniam ir orkestruotės testavimui. Programos duomenys – e. paslaugų sąsajos aprašas, pateikiamas WSDL dokumentu bei veiklos proceso aprašas, pateikiamas BPEL kalba. Programa leidžia interaktyviai simuliuoti veiklos proceso vykdymą, peržiūrėti programos vykdymo trasas.

LTSA-BPEL4WS įrankis turi integruotą aplinką, padedančią projektuoti, realizuoti ir tikrinti e. paslaugų kompozicijas. Įrankis leidžia palyginti sistemos specifikacijos, pateikiamos sekų diagramomis ir sistemos realizacijos BPEL kalba trasas. Tam, kad būtų galima atlikti palyginimą, tiek specifikacija, tiek ir BPEL realizacija transformuojama į baigtines būsenų mašinas. Vartotojas peržiūrėjęs trasas gali įvertinti realizacijos nukrypimus nuo specifikacijos arba pačios specifikacijos loginio nuoseklumo trūkumus.

1.5 lentelė. Būsenų modelių tikrinimo metodų ir įrankių palyginimas

Formalumo lygis	**Įrankis**	**Naudotojas**	**Privalumai**	**Trūkumai MDA požiūriu**
Baigtinės būsenų mašinos	Spin	Tyrėjai moksliniuose darbuose, reikalavimų inžinieriai, projektuotojai, testuotojai	Pripažintas klasikinis verifikavimo metodas	Reikalauja specialaus paruošimo, perėjimo nuo vieno įrankio prie kito, ilgai trunka
Kompiuterizuotas srautų modelis, leidžia peržiūrėti programos vykdymo trasas	Optimyz Web Service Tester	Programuotojas	Automatizuoja veiklos proceso valdymo programos (BPEL) tikrinimą	Specializuotas naudojimas BPEL programavimo kalbai; mažiau sąnaudų būtų generuoti BPEL turint korektišką proceso modelį
Kompiuterizuotas, transformuoja BPEL į baigtines būsenų mašinas ir leidžia palyginti su UML sekų diagramomis, taip pat transformuotomis į baigtines būsenų mašinas	LTSA-BPEL4WS	Programuotojas - analitikas	Automatizuoja programuotojo ir analitiko rezultatų suderinamumo tikrinimą	Specializuotas BPEL programavimo kalbai; iškyla problemos dėl semantinio vientisumo (programuotojas ir analitikas gali naudoti skirtingus žymėjimus)

Atsižvelgiant į išanalizuotus poreikius, šiame darbe norima sudaryti formalizuotą tikrinimo metodą, e. paslaugoms vaizduoti naudojant UML būsenų mašinas. UML pasirinkta todėl kad ji yra programų sistemų projektavimo standartas, tinkamas verslui ir informacinėms sistemoms modeliuoti, projektuoti, dokumentuoti. UML plačiai naudojama projektuotojų, programuotojų ir verslo analitikų, todėl toks įrankis patenkintų plataus vartotojų rato poreikius.

Būsenų mašina turi būti tikrinama todėl, kad šis modelis yra objektinių sistemų elgsenos pagrindas. E. paslaugų modeliavimo metodai dar nėra galutinai nusistovėję, e. paslaugų kompozicijai naudojami procesų modeliai (UML veiklos diagramos arba analogiški), tačiau procesų modeliai vaizduoja valdymo srautus. Tuo tarpu patį elgsenos mechanizmą – įvykius ir jų iššauktus sistemos būsenų pokyčius – vaizduoja būsenų mašinos. Literatūroje pripažįstama, kad tinkamiausias modelis elektroninių paslaugų elgsenai vaizduoti yra būsenų mašina. Informacijos sistemų katedroje daromas mokslinis darbas, kuriame specialios formos būsenų mašina naudojama sistemos sąveikoms suderinti [6]. Tačiau pati būsenų mašina gali būti nekorektiška. Taigi šiame darbe siekiama sukurti algoritmus, kurie leistų patikrinti būsenų mašinos korektiškumą ir užpildytų spragą projektavimo procese.

Sudarytą korektišką būsenų mašiną galima naudoti daugeliui realizacijų (BPEL, Java, Net ir t. t.). Tuo siūlomo metodo realizacija būtų pranašesnė už tikrinimo įrankius, skirtus tam tikrai programavimo kalbai (Web Service Tester, LTSA-BPEL4WS).

Šiame darbe siūlomas tikrinimo metodas būtų integruotas su kitais projekto modeliais ir turėtų didelį pakartotino panaudojimo laipsnį, kadangi pagal MDA iš konceptualaus modelio generuojama daug projekto artefaktų.

Metodas turėtų būti suformuluotas taip, kad jį būtų galima naudoti UML CASE įrankių išplėtimui. Galima realizacija būtų naudojant JMI (Java Metamodel Interface) ir Magic Draw API.

1.5. Analizės išvados

- ✓ E. paslaugos teikia priemones e. verslo procesams kompiuterizuoti. Jos turi patenkinti verslo poreikius, todėl turi būti kuriamos greitai ir kokybiškai, dažniausiai pasitelkiant automatinį kodo generavimą. Tokiame kontekste tampa aktuali e. paslaugų modelių verifikavimo problema. Sistemos tikrinimas modelio lygmenyje leidžia anksčiau aptikti ir ištaisyti klaidas sistemos specifikacijoje.
- ✓ Egzistuojantys modelio tikrinimo įrankiai reikalauja objektinio modelio transformacijos į sudėtingas įrankio formaliąsias notacijas. Būtų patogu būsenų tikrinimą realizuoti tiesiogiai

CASE įrankyje. Toks tikrinimo metodas padėtų užtikrinti būsenų modelio suderinamumą su kitais sistemos modeliais.

- ✓ E. paslaugų elgsenos verifikavimui pasirinktas UML būsenų mašinų modelis, kadangi UML yra programinių sistemų projektavimo standartas, o būsenų mašinų modelis yra objektinių sistemų elgsenos pagrindas.
- ✓ Šiame darbe siekiama sudaryti algoritmus, leidžiančius patikrinti e. paslaugų būsenų mašinų modelių saugumo kriterijus. Saugumo kriterijai apima būsenų pasiekiamumą, grįžtamumą, pilnumą ir nuoseklumą.
- ✓ Šiame darbe išdėstytas e. paslaugų tikrinimo metodas užtikrintų pakartotinį panaudojimą, kadangi iš sudaryto korektiškos būsenų mašinų modelio būtų galima generuoti sistemos realizaciją įvairioms programavimo kalbomis.

ELEKTRONINIŲ PASLAUGŲ VERIFIKAVIMO METODAS

1.6. E. paslaugų būsenų mašinų semantika

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

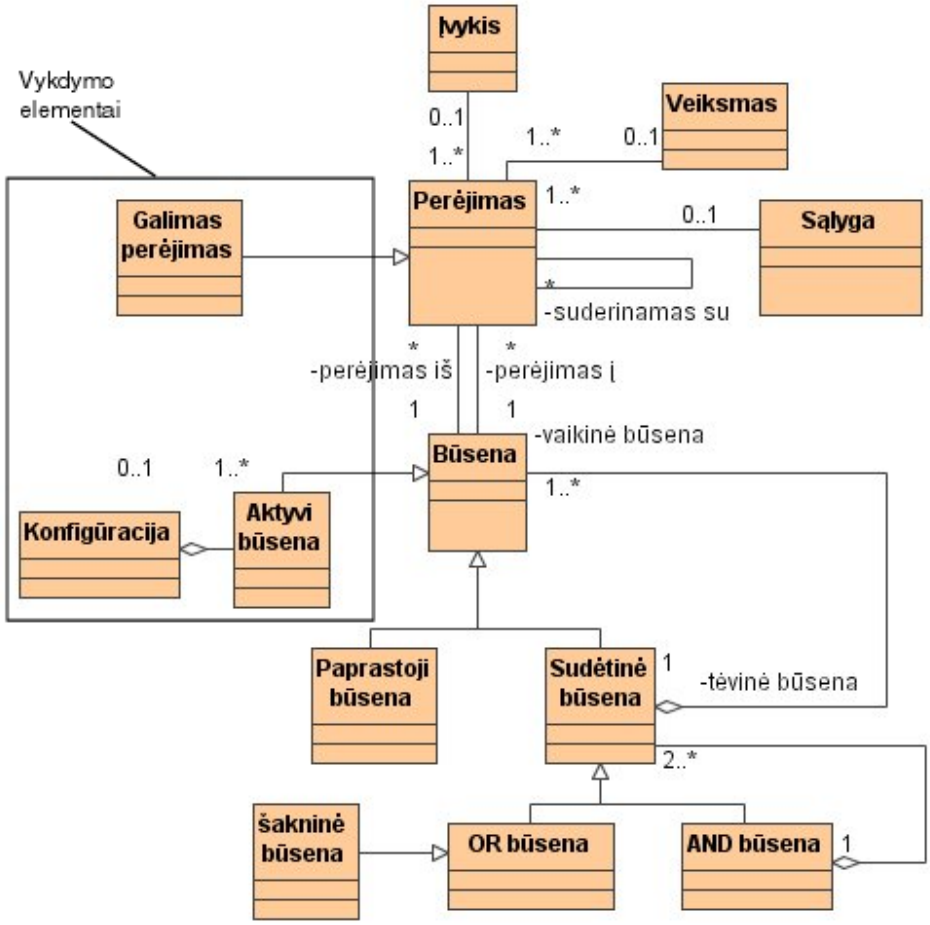
Apibrėšime būsenų mašinų versiją, orientuotą elektroninių paslaugų būsenų vaizdavimui, nagrinėjamam [8]. Šiame darbe naudojama būsenų mašina išlaiko pagrindines [8] savybes, tačiau jos metamodelis sugriežtintas ir papildytas būsenų perėjimų vykdymo informacija, kuri leidžia realizuoti kompozicinių e. paslaugų tikrinimo algoritmą, pateiktą tolesniame skyriuje. Nors visi apibrėžimai ir algoritmai tinka hierarchinėms būsenų mašinoms, e. paslaugų būsenų mašinos tikrinimą supaprastina jos modulinė struktūra: įdėtinių (angl. *Include*) būsenų mašinų korektiškumas tikrinamas atskirai, o kompozicinės e. paslaugos (jas aptarsime šiek tiek vėliau) būsenų mašinoje jos nagrinėjamos kaip paprastos būsenos, todėl visuomet tikrinama tik plokščia būsenų mašina.

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 *vaikinėmis būsenomis*. Jeigu vaikinę būsenos *s* būseną pažymėsime *s'*, tai *s* vadinsime būsenos *s' tėvine būsena*. Vizualiai vaikinės būsenos pateikiamos vaizduojant vieną būseną kitos viduje. Būsenos *s palikuoniais* vadinamos visos būsenos *s* vaikinės būsenos, tų vaikinių būsenų vaikinės būsenos ir t. t. [2].

Jeigu būsena *s* turi vaikinių būsenų, ją vadinsime *sudėtine būsena*. Priešingu atveju tai *paprastoji būsena*. Būna dviejų rūšių sudėtinės būsenos: *AND būsenos ir OR būsenos*. AND būsenos skirtos vaizduoti paralelizmui ir joms galioja tokia taisyklė: jeigu *s* yra AND būsena ir *s* yra aktyvi, tai aktyvios yra ir visos jos vaikinės 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 šaltinio 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.



2.1 pav. Sugriežtintas būsenų mašinų metamodelis

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 tėvinė būsena.

3. Jeigu OR būsena priklauso konfigūracijai, tai būtinai viena ir tik viena jos vaikinių būsenų taip pat priklausys konfigūracijai.

Jeigu AND būsena priklauso konfigūracijai, tai ir visos jos vaikinės būsenos priklauso konfigūracijai.

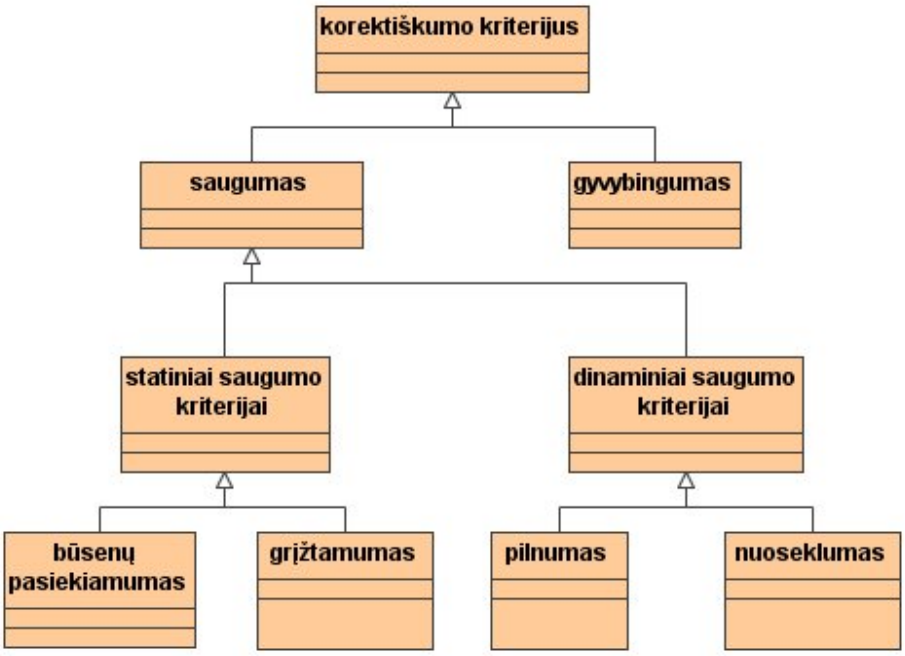
Sistemos konfigūracija gali kisti, vykdant perėjimus. Konfigūracijos pakitimai yra sąlygojami pakitimų, vykstančių sistemos aplinkoje. Aplinkos pokytis, kuris modeliuojant yra reikšmingas, vadinamas *įvykiu*.

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, įvykdžius perėjimą.

1.7. Būsenų mašinų korektiškumo kriterijai

Bendriausi būsenų mašinų korektiškumo kriterijai yra saugumas (angl. *safety*) ir gyvybingumas (angl. *liveness*). Neformaliai šiuos kriterijus 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.



2.2 pav. 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).

- *Grįžtamumas*. E. paslaugų atveju šis kriterijus reiškia, kad iš visų būsenų turi egzistuoti kelias į laukimo būseną.

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., bet būtų teisinga kuri nors viena iš vykdymo sąlygų (2.3 pav). Formaliai šį reikalavimą galima užrašyti taip:

$$c_1 \vee c_2 \ldots \vee c_n = true \qquad \textbf{(2.1)}$$

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 I 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:

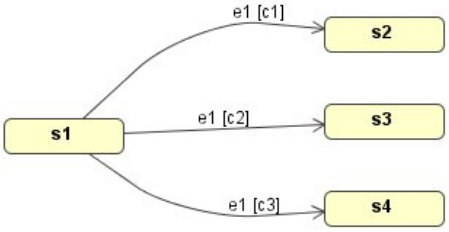
1. Būsenų modelyje turi būti aiškiai apibrėžta sistemos elgsena, tiek kai įvykio sąlyga yra patenkinama, tiek ir kai nepatenkinama.
2. Esant keliems perėjimams, susietiems su vienu įvykiu, bent vieno perėjimo sąlyga įvykio metu turi būti įvykdoma.

Apibendrintai šį reikalavimą galima užrašyti taip:

$$\bigcup_{i=2}^{n} c_i = \overline{c_1} \qquad \textbf{(2.2)}$$

Ši formulė išvedama iš **(2.1)** ir iš matematinėje logikoje suformuluoto „trečio negalimo“ dėsnio, teigiančio, kad:

$$A \vee \overline{A} = true \qquad \textbf{(2.3)}$$



2.3 pav. Būsenų mašinos pavyzdys

- *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 nepersikertančios* (2.3 pav):

$$c_i \wedge c_j = false, \forall j \neq i, \quad i = 1,2,...,n \quad j = 1,2,...,n \qquad \textbf{(2.4)}$$

Ši apribojimas išplaukia iš reikalavimo, kad vienu metu galima vykdyti tik vieną perėjimą.

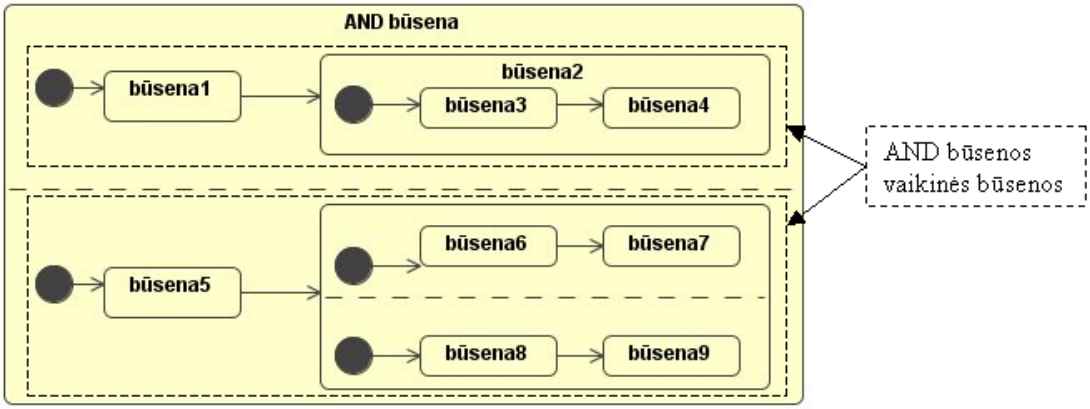
1.8. Siūlomi algoritmai saugumo kriterijams tikrinti

Šiame skyrelyje pateikiami konceptualūs algoritmai, skirti šiems saugumo kriterijams tikrinti: būsenų pasiekiamumui, grįžtamumui, pilnumui ir nuoseklumui.

Visų pateiktų algoritmų įėjimo duomenys – būsenų mašinos būsenų aibė *S*. Algortimuose naudojamų aibių ir funkcijų aprašymai apibendrinami šalia pateiktomis lentelėmis.Algoritmai analizuoja visas įmanomas perėjimų vykdymo trasas. Trasos pradedamos formuoti nuo pradinės būsenų mašinos būsenos. Būsenos apeinamos paieškos į plotį metodu. Pažymėtina, kad įvertinamos ir sudėtinės AND ir OR būsenos.

Reikalavimai ir pastabos tikrinamai būsenų mašinai:

- tiek būsenų mašinai, tiek ir kiekvienai jos OR būsenai turi būti nurodyta pradinė būsena;
- pradinė būsena turi būti nurodyta ir visoms modelio AND būsenų vaikinėms būsenoms (2.4 pav.). Laikoma, kad *n* lygiagrečiųjų procesų vaizduojanti AND būsena turi lygiai *n* vaikinių OR būsenų (vizualiai šios OR būsenos kontūras paprastai nevaizduojamas).



2.4 pav. AND būsenos vaikinės būsenos

1.8.1. Algoritmas būsenų pasiekiamumui tikrinti

Šis algoritmas tikrina, ar visos būsenų mašinos būsenos yra pasiekiamos, t.y. ar iki kiekvienos būsenos egzistuoja bent vienas kelias iš pradinės būsenos, ir grąžina nepasiekiamų būsenų aibę.

Būsenų pasiekiamumo kriterijų tikrinsime iteracijomis formuodami pasiekiamų būsenų aibę. Algoritmo darbo pradžioje pasiekiamų būsenų aibė yra tuščia. Pirmiausia, pasiekiamų būsenų aibei priskiriamos pradinės būsenos kaimynės (būseną s_j vadinsime būsenos s_i *kaimynine būsena*, jeigu egzistuoja perėjimas iš būsenos s_i į būseną s_j.). Tada pasiekiamų būsenų aibei priskiriamos tų kaimyninių būsenų kaimyninės būsenos ir t. t. Algoritmui baigus darbą, nepasiekiamų būsenų aibė gaunama iš visų būsenų aibės atėmus pasiekiamas būsenas.

Jeigu algoritmo grąžinta nepasiekiamų būsenų aibė nėra tuščia, vadinasi, būsenų mašina turi izoliuotų būsenų ir būsenų pasiekiamumo kriterijus nėra išpildomas.

```
Pasiekiamos := ∅
Stekas := ∅
s := PradinėBusena(S)
Stekas.push(s)
Pasiekiamos := Pasiekiamos ∪ {s}
repeat
    s := Stekas.pop
    if(Vaikai(s) ≠ ∅)
      if(AND_būsena(s))
          for v ∈ Vaikai(s)
              Pasiekiamos := Pasiekiamos ∪ (Palikuonys(v) - Pasiekiamumas(Palikuonys(v)))
          end for
      else
          Pasiekiamos := Pasiekiamos ∪ (Palikuonys(s) - Pasiekiamumas(Palikuonys(s)))
      end if
    end if

    for k ∈  Kaimynai(s)
      if k ∉ Pasiekiamos
          Stekas.push(k)
          Pasiekiamos := Pasiekiamos ∪ {k}
      end if
    end for
until Stekas = ∅
Nepasiekiamos := S - Pasiekiamos
return Nepasiekiamos
```

2.5 pav. Pasiekiamumas(S) – algoritmas, tikrinantis būsenų pasiekiamumą

2.2 lentelė. Algoritmo Pasiekiamumas(S) naudojamos aibės ir funkcijos

Aibė arba funkcija	**Aprašymas**
Pasiekiamos	Būsenų mašinos pasiekiamų būsenų aibė
PradinėBūsena(S)	Funkcija, grąžinanti būsenų mašinos pradinę būseną
Stekas	Duomenų struktūra, atitinkanti LIFO (angl. Last In First Out) principą. Stekas turi operacijas *push* ir *pop*. Operacija *push* įrašo elementą sekos pabaigoje. Operacija *pop* – grąžina paskutinį sekos elementą ir jį pašalina iš steko
Vaikai(s)	Funkcija, grąžinanti būsenos *s* vaikines būsenas, arba ∅, jei *s* tokių neturi
AND_būsena(s)	Loginė funkcija, grąžinanti reikšmę *true,* jeigu būsena *s* yra AND būsena
Palikuonys(s)	Funkcija, gražinanti visų būsenos *s* palikuonių, t. y. vaikinių būsenų ir, savo ruožtu, visų pastarųjų vaikinių būsenų aibę
Kaimynai(s)	Funkcija, grąžinanti būsenos *s* kaimyninių būsenų aibę
Nepasiekiamos	Būsenų mašinos nepasiekiamų būsenų aibė

1.8.2. Algoritmas grįžtamumui tikrinti

Pavadinkime būseną, kuriai egzistuoja kelias (t. y. perėjimų seka) iki e. paslaugos laukimo būsenos, kurioje paslauga yra pasiruošusi aptarnauti kliento užklausas, *grįžtančiąja būsena*.

Taip pat, tarkime turime būsenas s_i *ir* s_j. Būseną s_i vadinsime būsenos s_j *pirmtake*, jeigu egzistuoja bent vienas perėjimas iš būsenos s_i į būseną s_j.

Algoritmas grąžina negrįžtančiųjų būsenų aibę. Grįžtamumo kriterijų tikrinsime iteracijomis formuodami grįžtančiųjų būsenų aibę. Pirmiausia, grįžtančiųjų būsenų aibėje yra tik vienas elementas – laukimo būsena. Tada į aibę įtraukiamos visos laukimo būsenos pirmtakės (kadangi iš pirmtakių jau pagal apibrėžimą egzistuoja perėjimas, o tuo pačiu ir kelias į laukimo būseną), pastarųjų pirmtakės ir t. t.

Algoritmui baigus darbą, negrįžtančiųjų būsenų aibė randama iš visų būsenų aibės atėmus grįžtančiąsias būsenas.

Jeigu algoritmo grąžinta negrįžtančiųjų būsenų aibė turi bent vieną elementą, vadinasi būsenų mašinoje egzistuoja tokių būsenų, iš kurių neįmanoma pasiekti e. paslaugos laukimo būsenos, taigi,

galima situacija, kai dėl modelio netikslumo paslauga nustos aptarnauti naujų klientų užklausas. Grįžtamumo kriterijus netenkinamas.

```
Grįžtančiosios := ∅
Stekas := ∅
s := LaukimoBūsena
Stekas.push(s)
Grįžtančiosios := Grįžtančiosios ∪ {s}
repeat
    s := Stekas.pop

    for p ∈ Pirmtakai(s)
       if p ∉ Grįžtančiosios
            Stekas.push(p)
            Grįžtančiosios := Grįžtančiosios ∪ {p}
       end if
    end for

until Stekas = ∅
Negrįžtančiosios := S – Grįžtančiosios
return Grįžtančiosios
```

2.6 pav. **Grįžtamumas(S) – algoritmas, tikrinantis grįžtamumą ir grąžinantis grįžtančiųjų būsenų aibę.**

2.3 lentelė. Algoritmo Grįžtamumas(S) naudojamos aibės ir funkcijos

Aibė arba funkcija	**Aprašymas**
Grįžtančiosios	Būsenų mašinos grįžtančiųjų būsenų aibė
LaukimoBūsena(S)	Funkcija, gražinanti būsenų mašinos S laukimo būseną, t. y. būseną, į kurią e. paslauga visada turi grįžti, baigusi bet kurios iš savo teikiamų paslaugų vykdymą
Pirmtakai(s)	Funkcija, grąžinanti būsenos *s* būsenų pirmtakių aibę
Negrįžtančiosios	Būsenų mašinos negrįžtančiųjų būsenų aibė

1.8.3. Pilnumo ir nuoseklumo tikrinimo algoritmas

Prieš pateikdami algoritmą, įvesime dar keletą apibrėžimų.

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

Perėjimą laikysime *nekorektišku*, jeigu jie netenkina pilnumo ir nuoseklumo kriterijų reikalavimų (2.2 skyrelis).

Algoritmas grąžina nekorektiškai specifikuotų perėjimų aibę.

```
Korektiški := ∅
Stekas := ∅
s := PradineBusena(S)
Stekas.push(s)
Aplankytos := Aplankytos ∪ {s}

repeat
    s := Stekas.pop
    if(Vaikai(s) ≠ ∅)
       if(AND_busena(s))
            for v ∈ Vaikai(s)
                 Korektiški := Korektiški ∪ (Palikuonys(v) - Dinamika(Palikuonys(v)) )
            end for
       else
            Korektiški := Korektiški ∪ (Palikuonys(s) - Dinamika(Palikuonys(s)) )
       end if
    end if

    pan := Panašūs(s)
    if(pan) = ∅
            Korektiški := Korektiški ∪ Išėjimai(s)
    else
            if( PoromisNesikerta(Sąlygos(pan)) AND BentViena(Sąlygos(pan)) )
                    Korektiški := Korektiški ∪ Išėjimai(s)
            end if
    end if

    for k ∈  Kaimynai(s)
       if k ∉ Aplankytos
            Stekas.push(k)
            Aplankytos := Aplankytos ∪ {k}
       end if
    end for

until Stekas = ∅
Nekorektiški := S – Korektiški
return Nekorektiški
```

2.7 pav. Dinamika(S) – algoritmas, tikrinantis dinaminius pilnumo ir nuoseklumo kriterijus.

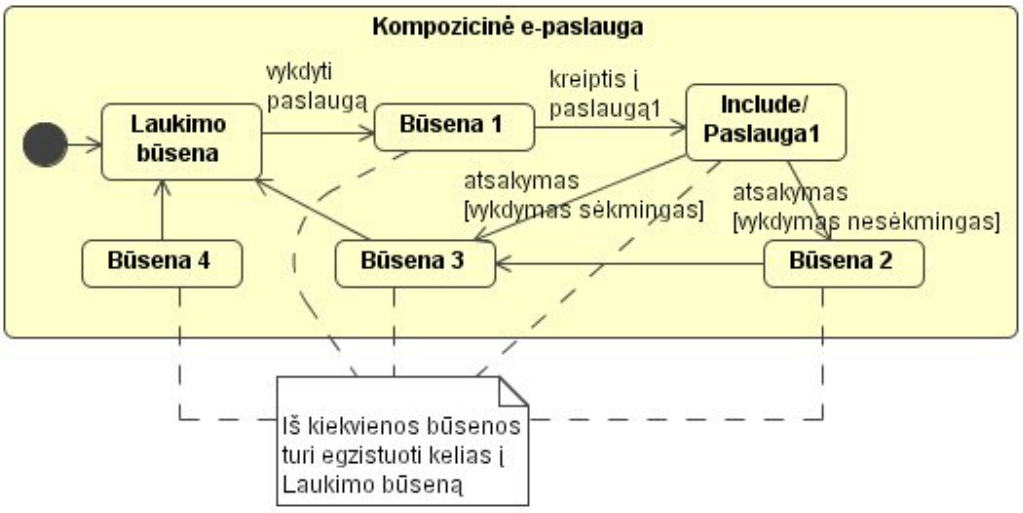
2.4 lentelė. Algoritmo Dinamika(S) naudojamos aibės ir funkcijos

Aibė arba funkcija	**Aprašymas**
Korektiški	Korektiškų perėjimų aibė
LaukimoBūsena(S)	Funkcija, gražinanti būsenų mašinos S laukimo būseną, t. y. būseną, į kurią e. paslauga visada turi grįžti, baigusi bet kurios iš savo teikiamų

	paslaugų vykdymą
Aplankytos	Algoritmo jau aplankytų būsenų aibė
Panašūs(s)	funkcija, grąžinanti visų iš būsenos *s* išeinančių panašių perėjimų aibę
Išėjimai(s)	Funkcija, grąžinanti visų iš būsenos *s* išeinančių perėjimų aibę
Sąlygos(T)	Funkcija, grąžinantį aibę visų sąlygų, susietų su perėjimais iš aibės *T*
PoromisNesikerta(C)	Funkcija, kurios reikšmė yra *true*, jeigu sąlygų aibės *C* elementai yra poromis nepersikertantys
BentViena(C)	Funkcija, kurios reikšmė yra *true*, jeigu sąlygų aibės *C* elementai tenkina sąlygą $\bigcup_{i=2}^{n} c_i = \overline{c_1}$
Nekorektiški	Nekorektiškų perėjimų aibė

1.9. Kompozicinės e. paslaugos būsenų mašinos šablonas

Būsenų mašina apibendrina paslaugų sistemos elgseną, kuri aprašoma įvairių tipų sąveikų modeliais. Paprastai sąveikų modeliai apima tik pagrindinius scenarijus, kadangi pilna scenarijų aibė, apimanti visus nesėkmingus atvejus, yra labai didelė. Todėl siūlome šabloną, kuris padeda papildyti būsenų mašiną nesėkmingų atvejų aibe.



2.8 pav. Kompozicinės e. paslaugos šablonas

Kompozicinė e. paslauga gali kreiptis į kitas e. paslaugas, pvz. „Paslauga1“ (2.8 pav.).

Kad sistema būtų stabili, iš kiekvienos būsenos turi egzistuoti kelias, leidžiantis grįžti į laukimo būseną. Be to, kiekvienam įvykiui „atsakymas“, kuris įvyksta, gavus atsakymą iš naudojamos paslaugos

(pvz., „Paslauga1“), turi būti apibrėžti du perėjimai: vienas, kai paslauga įvykdoma sėkmingai ir vienas – kai nesėkmingai.

Apibendrinant, kiekvienai paslaugai turi egzistuoti grįžimas į laukimo būseną, o kiekvienam įvykiui – sėkmės ir nesėkmės alternatyvos.

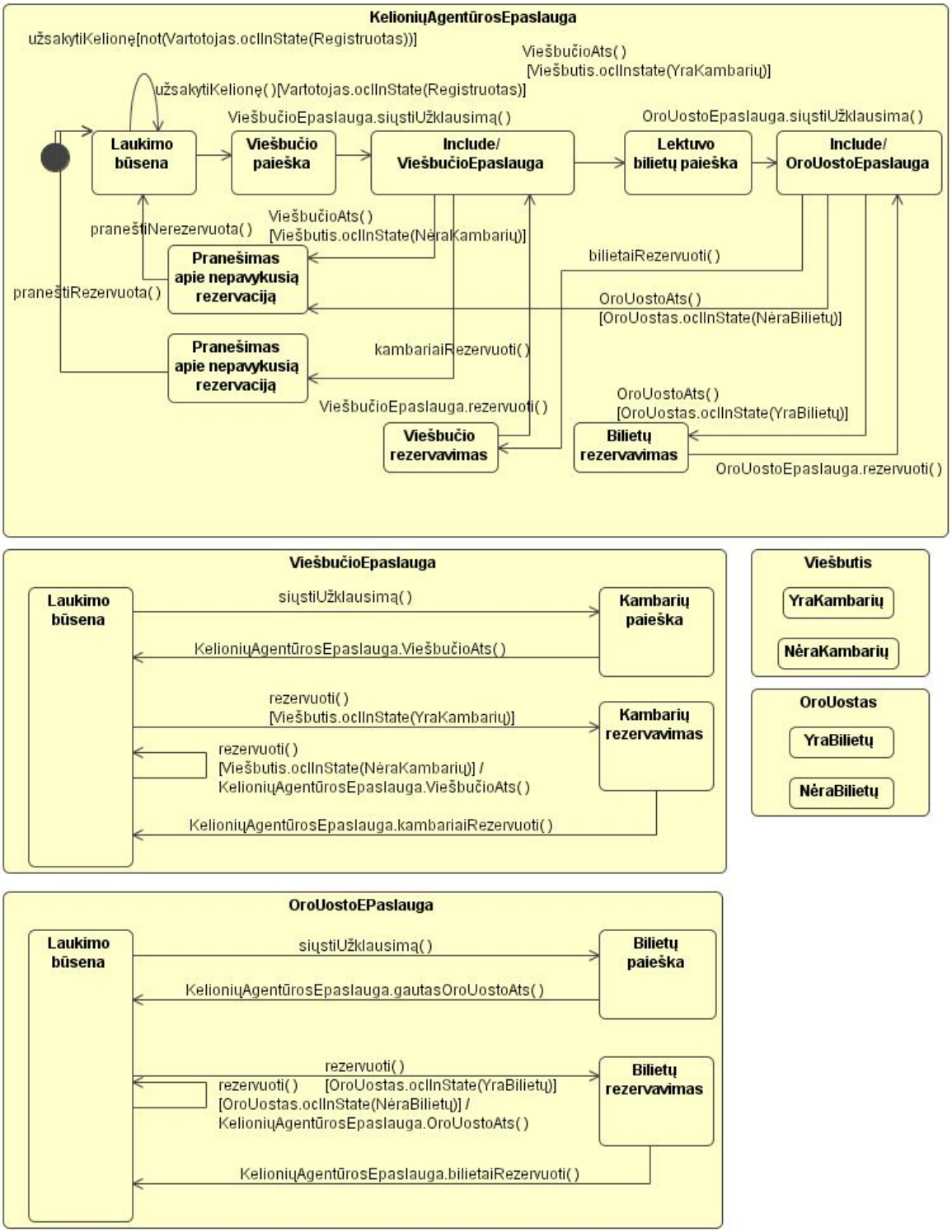
EKSPERIMENTINIO ALGORITMŲ TAIKYMO PAVYZDYS

1.10. Kompozicinės e. paslaugos pavyzdys

Panagrinėkime kelionių agentūros e. paslaugos pavyzdį. Trumpas jos darbo scenarijaus aprašymas:

- Vartotojas pateikia e. paslaugai planuojamos kelionės parametrus: kelionės tikslą ir keliaujančių asmenų skaičių;
- Kelionių agentūros e. paslauga kreipiasi į viešbučio e. paslaugą, pateikdama jai užklausą, ar nurodytą dieną viešbutyje bus laisvų kambarių nurodytam asmenų skaičiui;
- Viešbučio e. paslaugai pateikus atsakymą, kad kambarių bus, kelionių agentūros paslauga kreipiasi į oro uosto e. paslaugą, norėdama patikrinti ar bus bilietų skrydžiui;
- Jei yra ir vietų viešbutyje, ir bilietų skrydžiui, rezervuojami apartamentai ir bilietai, o vartotojui pateikiamas patvirtinimas;
- Jei nepavyksta rezervuoti kambarių arba bilietų, vartotojui siunčiamas pranešimas apie nepavykusią rezervaciją.

Kelionių agentūros paslaugų būsenų mašinos modelis pateiktas 2.9 pav.



2.9 pav. Kelionių agentūros e. paslaugos būsenų mašinos pavyzdys

Buvo pabandyta patikrinti šį pavyzdį algoritmu *Dinamika(S).*

Algoritmo vykdymo metu perrenkamos visos įmanomos būsenų mašinos perėjimų vykdymo trasos. Pateiksime kelias trasas, gautas pritaikius šį algoritmą kelionių agentūros paslaugų būsenų mašinos paslaugų pavyzdžiui:

1. `užsakytiKelionę()[Vartotojas.oclInState(Registruotas)] ->`
   `ViešbučioEpaslauga.siųstiUžklausimą() ->`
   `ViešbučioAts()[Viešbutis.oclInState(NėraKambarių)] ->`
   `praneštiNerezervuota();`
2. `užsakytiKelionę()[Vartotojas.oclInState(Registruotas)] ->`
   `ViešbučioEpaslauga.siųstiUžklausimą() ->`
   `ViešbučioAts()[Viešbutis.oclInState(YraKambarių)] ->`
   `OroUostoEpaslauga.siųstiUžklausimą()->`
   `OroUostoAts()[Viešbutis.oclInState(NėraBilietų)] -> praneštiNerezervuota().`

1.11. Kompozicinės e. paslaugos būsenų mašinos šablono taikymo pavyzdys

Patikrinkime, ar kelionių agentūros e. paslaugos pavyzdžio fragmentas tenkina suformuluoto kompozicinių e. paslaugų šablono rekomendacijas.

Tarkime, nagrinėjamu momentu sistema yra būsenoje „**`Include/ViešbučioEpaslauga`**“.

- ✓ Pirmiausia patikriname, ar iš šios būsenos egzistuoja kelias į laukimo būseną
  Iš tiesų, toks kelias egzistuoja:

  `Include/ViešbučioEpaslauga` `-> ViešbučioAts()[Viešbutis.oclInState(NėraKambarių)] ->`
  `Pranešimas apie nepavykusią rezervaciją` `-> praneštiNerezervuota() ->`
  `Laukimo būsena;`

  čia pastorintu šriftu žymimos sistemos būsenos, o paprastu – perėjimai tarp jų.

- ✓ Matome, kad būsenai „**`Include/ViešbučioEpaslauga`**“ yra apibrėžtas įvykis „`ViešbučioAts()`“.

  Šis įvykis gali aktyvuoti du perėjimus. Vienas perėjimas turi vykdymo sąlygą `[Viešbutis.oclInState(YraKambarių)]`, kitas - `[Viešbutis.oclInState(NeraKambarių)]`

  Kaip ir rekomenduojama, įvykis „`ViešbučioAts()`“ turi du perėjimus: vieną, kai vykdymo sąlyga patenkinama, ir kitą – kai nepatenkinama:
  `[Viešbutis.oclInState(NėraKambarių)] = NOT( [Viešbutis.oclInState(YraKambarių)] )`

IŠVADOS

- ✓ Modelių verifikavimo problema labai aktuali kuriant elektroninių paslaugų sistemas, kadangi paslaugų technologijomis realizuojami e. verslo procesai kuriami pagal poreikį ir kinta labai greitai, todėl tokioms sistemoms programos kodas turi būti generuojamas automatiškai ir reikia užtikrinti jo korektiškumą
- ✓ Verifikavimą tikslinga atlikti modelio lygyje, kadangi iš vieno modelio generuojama daug artefaktų
- ✓ Literatūros šaltinių analizė parodė, kad naujos kartos CASE įrankiuose, skirtuose automatiniam kodo generavimui modelių pagrindu, nėra tinkamų modelių verifikavimo priemonių. Esami verifikavimo įrankiai nesuintegruoti su projektavimo įrankiais, vieni iš jų skirti formaliam tikrinimui ir reikalauja transformuoti projekto modelius į formalias specifikacijas, kiti susieti su konkrečiomis programavimo aplinkomis
- ✓ Kadangi e. paslaugų sistemos elgseną geriausiai aprašo būsenų mašina, korektiškumo kriterijus suformuluotas remiantis bendriniu būsenų mašinų saugumo kriterijum, kurį galima išskaidyti į pilnumo ir nuoseklumo kriterijus.
- ✓ Sudarytas e. paslaugų tikrinimo metodas, paremtas UML būsenų mašinos tikrinimu. Tam sudarytas sugriežtintas būsenų mašinos metamodelis, į kurį įtraukti elementai būsenų mašinos vykdymui modeliuoti.
- ✓ Metamodelio pagrindu sudaryti verifikavimo algoritmai, tikrinantys būsenų mašiną pagal būsenų pasiekiamumo, grįžtamumo, pilnumo ir nuoseklumo kriterijus.
- ✓ Pasiūlytas šablonas, padedantis papildyti kompozicinės e. paslaugos būsenų mašiną nesėkmingų atvejų perėjimų aibe. Tokiu būdu sudaryta būsenų mašina išsamiai aprašo elgseną ir ją galima naudoti kodui generuoti. Projektuotojas išvaduojamas nuo rutininio darbo, kadangi jam nereikėtų aprašinėti pasikartojančių nesėkmingų perėjimų kiekvienam įvykiui, sąlygai ar veiksmui.
- ✓ Būsenų tikrinimo algoritmai, įdiegti CASE įrankiuose, padėtų projektuotojams užtikrinti kuriamų sistemų elgsenos modelių suderinamumą, pagreitintų projektavimą.

LITERATŪRA

1. W3 Consortium. *Web Services Architecture* [interaktyvus]. 2004 [žiūrėta 2004 04 15]. Prieiga per internetą: <http://dev.w3.org/cvsweb/~checkout~/2002/ws/arch/wsa/wd-wsa-arch-review2.html>.
2. Zimmermann O., Tomlinson M. R., Peuser S. *Perspectives on Web Services. Applying SOAP, WSDL, and UDDI to Real-World Projects*. Springer Verlag, 2003.
3. Object Management Group. *OMG Unified Modelling Language Specification. Version 1.5* [interaktyvus]. 2005 [žiūrėta 2005 04 15]. Prieiga per internetą: <http://www.omg.org/docs/formal/05-04-01.pdf>.
4. Pap Zs., Majzik I., Pataricza A., Szegi A. *Completeness and Consistency Analysis of UML Statechart Specifications* [interaktyvus]. 2001 [žiūrėta 2004 05 20]. Prieiga per internetą: <http://www.inf.mit.bme.hu/FTSRG/Publications/DDECS2001a.pdf>.
5. Pap Zs., Majzik I., Pataricza A. *Checking General Safety Criteria on UML Statecharts* [interaktyvus]. 2001 [žiūrėta 2004 04 15]. Prieiga per internetą: <http://home.mit.bme.hu/~majzik/publicat/safecomp01.pdf>.
6. Čeponienė L., Nemuraitė L. *Design Independent Modeling of Information Systems*. In: Barzdins, J., Caplinskas, A. (red.) Databases and Information Systems. Selected Papers from the Sixth International Baltic Conference DB&IS'2004, IOS Pres., 2005, p. 224-237.
7. Nemuraitė, L., Čeponienė L. *UML būsenų diagramų sudarymas iš sekos diagramų*. Informacinės technologijos 2003. Kaunas: Technologija, 2003, Sekcija XIV, p. 62- 67.
8. Eshuis R., Jansen D. N., Wieringa R. *Requirements-level semantics and model checking of object-oriented statecharts* [interaktyvus]. 2002 [žiūrėta 2005 01 15]. Prieiga per internetą: <http://is.tm.tue.nl/staff/heshuis/req.pdf>.
9. Kovse J., Härder T. *Generic XMI-Based UML Model Transformations* [interaktyvus]. 2002 [žiūrėta 2003 12 10]. Prieiga per internetą: <http://wwwdvs.informatik.uni-kl.de/pubs/papers/KH02.OOIS.pdf>.
10. Clark E. M, Jr., Grumberg O., Peled D. A. *Model checking.* Cambrige, Massachusetts: The MIT press, 1999.
11. Mickūnas V., Nemuraitė L. *Elektroninių paslaugų modelių verifikavimas, taikant UML būsenų mašinas*. Informacinės technologijos 2005. Kaunas: Technologija, 2005, II sekcija, p. 107- 111.

12. Foster H., Uchitel S., Magee J., Kramer J. *LTSA-BPEL4WS: Tool Support for Model-based verification of Web Service Compositions* [interaktyvus]. 2003 [žiūrėta 2004 02 23]. Prieiga per internetą: <http://www.doc.ic.ac.uk/~hfl/phd/downloads/LTSABPEL4WS_Tool_Foster_H.pdf>.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

BPEL (angl. *Business Process Execution Language*)	Standartizuota e. paslaugų orkestruotės aprašymo kalba
CASE (angl. *Computer Aided Software Engineering*)	Kompiuterizuotas programinės įrangos projektavimas
MDA (angl. *Model Driven Architecture*)	Modeliais grįsta architekūra
SOAP (angl. *Simple Object Access Protocol*)	Protokolas, transportuojantis pranešimus e. paslaugoms
UDDI (angl. *Universal Discovery Description and Integration*)	E. paslaugų registravimo ir katalogizavimo mechanizmas
UML (angl. *Unified Modeling language*)	Universali modeliavimo kalba
WSDL (angl. *Web Services Description Language*)	E. paslaugų sąsajos aprašymo kalba
WSIL (angl. *Web Services Inspection Language*)	Paprastesnė UDDI mechanizmo alternatyva

1 PRIEDAS. Straipsnis, skaitytas konferencijoje „Informacinės technologijos 2005“

Elektroninių paslaugų modelių verifikavimas, taikant Uml būsenų MAŠINas

Viktoras Mickūnas, vadovė Lina Nemuraitė

Kauno technologijos universitetas, Informacijos sistemų katedra

Straipsnyje nagrinėjamas UML būsenų mašinų panaudojimas e. paslaugų modelių verifikavimui. Apžvelgiama būsenų mašinų semantika. Apibrėžiami būsenų mašinos pilnumo ir nuoseklumo kriterijai. Sudaromas sugriežtintas būsenų mašinų metamodelis, į kurį įtraukti papildomi elementai būsenų mašinos vykdymui modeliuoti. Metamodelio pagrindu pasiūlytas algoritmas būsenų modelių korektiškumui tikrinti. Pateikiamas kompozicinės e. paslaugos šablonas, padedantis užtikrinti modelio korektiškumą.

1. Įvadas

Kompozicine e. paslauga vadinama paslauga, kuri savo funkcionalumą realizuoja naudodama kitų e. paslaugų teikiamus servisus. Pažymėtina, kad kompozicinės e. paslaugos klientas sąveikauja tik su ja viena ir gali nieko nežinoti apie šios paslaugos ryšius su kitais paslaugų tiekėjais.

Kadangi kompozicinė e. paslauga yra sistema, kurios elementai – autonominės e. paslaugos, turinčios skirtingas sąsajas ir elgsenos ypatumus, korektiškas šios sistemos darbas įmanomas tiktai tada, kada yra suderinamos įvairiausių tipų sąveikos, saistančios sistemos elementus. Šios sąveikos apima: sąveikas su vartotoju, aprašomas kompozicinės e. paslaugos palaikomomis sąsajomis; sistemos interfeisų sąveikas, kurių visuma sudaro orkestruotę; globalaus proceso modelį, kuris susideda iš sąveikų fragmentų, vadinamų choregrafijomis. Siekiant suderinti šias sąveikas, jos apibendrinamos sudarant būsenų mašinas, kurias galima generuoti iš daugelio sąveikas aprašančių UML sekų diagramų [1].

Būsenų mašinomis galima patogiai ir vaizdžiai specifikuoti sudėtingą, įvykiais grįstą sistemos elgseną, todėl jas patogu vartoti, modeliuojant e. paslaugų dinamiką. Deja, laisvai sudaryti ar iš sekų diagramų sugeneruoti būsenų mašinų modeliai taip pat gali būti nepilni, nenuoseklūs ir dviprasmiški. Aptikti ir ištaisyti specifikacijos klaidas vėlesnėse programinės įrangos kūrimo ciklo fazėse yra ne tik sunku, bet ir brangu. Be to, modelių neapibrėžtumas užkerta kelia automatiniam programinio kodo generavimui.

Taigi, automatinis būsenų mašinų verifikavimas yra svarbi problema, tuo labiau kad kompozicinės e. paslaugos dažnai turi būti formuojamos labai greitai, pagal iškilusį poreikį, ir jų kūrimą siekiama automatizuoti. Esami modelių tikrinimo įrankiai (pavyzdžiui, Spin) reikalauja modelio transformavimo į jų specifines formaliąsias notacijas, kas reikalauja nemažai projektuotojo laiko ir pastangų. Siekiant kūrimą pagreitinti, būsenų mašinų tikrinimą tikslinga atlikti e.paslaugų projektavimo ir kūrimo CASE įrankiuose. Tam tikslui šiame darbe siūlomas būsenų tikrinimo algoritmas, kurį būtų galima realizuoti UML CASE įrankiuose.

2. Kompozicinių paslaugų būsenų mašinų semantika

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

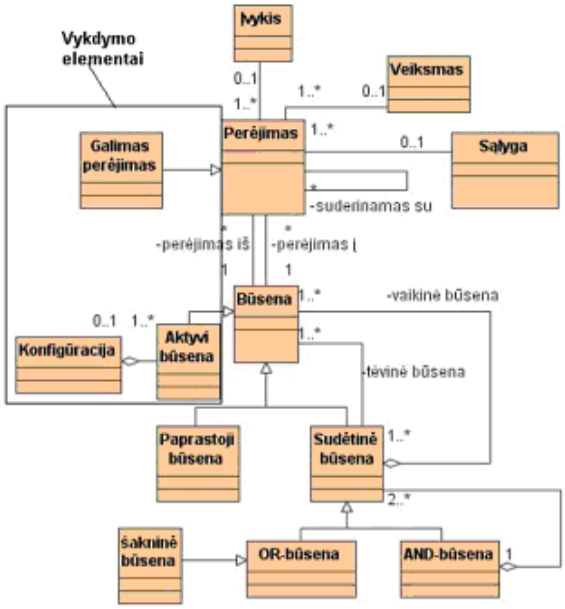
Apibrėšime būsenų mašinų versiją, orientuotą elektroninių paslaugų būsenų vaizdavimui, nagrinėjamam [1]. Šiame darbe naudojama būsenų mašina išlaiko pagrindines [1] savybes, tačiau jos metamodelis sugriežtintas ir papildytas būsenų perėjimų vykdymo informacija, kuri leidžia realizuoti kompozicinių e. paslaugų tikrinimo algoritmą, pateiktą tolesniame skyriuje. Nors visi apibrėžimai ir algoritmai tinka hierarchinėms būsenų mašinoms, e. paslaugų būsenų mašinos tikrinimą supaprastina jos modulinė struktūra: įdėtų (*Include*) būsenų mašinų korektiškumas tikrinamas atskirai, o kompozicinės paslaugos būsenų mašinoje jos nagrinėjamos kaip paprastos būsenos, todėl visuomet tikrinama tik plokščia būsenų mašina.

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 sub-būsenas, vadinamas *vaikinėmis būsenomis*. Jeigu vaikinę būsenos *s* būseną pažymėsime *s'*, tai *s* vadinsime būsenos *s' tėvine būsena*. Vizualiai vaikinės būsenos pateikiamos vaizduojant vieną būseną kitos viduje. Būsenos *s palikuoniais* vadinamos visos būsenos *s* vaikinės būsenos, tų vaikinių būsenų vaikinės būsenos ir t.t. [2].

Jeigu būsena *s* turi vaikinių būsenų, ją vadinsime *sudėtine būsena*. Priešingu atveju tai *paprastoji būsena*. Būna dviejų rūšių sudėtinės būsenos: *AND-būsenos ir OR-būsenos*. AND-būsenos skirtos vaizduoti paralelizmui ir joms galioja tokia taisyklė: jeigu *s* yra AND-būsena ir *s* yra aktyvi, tai aktyvios yra ir visos jos vaikinės 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 šaltinio 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.



1.pav. Būsenų mašinos metamodelis.

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

1. Šakninė būsena priklauso visada priklauso konfigūracijai.

2. Jeigu OR-būsena priklauso konfigūracijai, tai būtinai viena ir tik viena jos vaikinių būsenų taip pat priklausys konfigūracijai.

3. Jeigu būsena priklauso konfigūracijai, tai konfigūracijai priklauso ir jos tėvinė būsena.

4. Jeigu AND-būsena priklauso konfigūracijai, tai ir visos jos vaikinės 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* (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 mažiausia OR-būseną, talpinančią tiek šaltinio, tiek ir tikslo būsenas. Tokia OR-būsena vadinama briaunos e *kontekstu* (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, įvykdžius perėjimą.

3. Būsenų mašinų korektiškumo kriterijai

Bendriausi būsenų mašinų korektiškumo kriterijai yra saugumas (angl. *safety*) ir gyvybingumas (angl. *liveness*). Detaliau jie išreiškiami *pilnumo, nuoseklumo, būsenų pasiekiamumo* bei *grįžtamumo* kriterijais. Pilnumas užtikrina tai, kad kiekvienai leidžiamai įvykių sekai yra vienareikšmiškai apibrėžta veiksmų seka. Specifikacijos nuoseklumas garantuoja reikalavimų neprieštaringumą ir elgsenos determinizmą.

Pilnumas apima keletą reikalavimų: kiekviena būsena turi turėti bent vieną įeinantį perėjimą [3]; kiekvienai būsenai turi galioti tokia sąlyga: visų perėjimų, sužadinamų įvykus tam pačiam įvykiui, vykdymo sąlygos turi sudaryti tautologiją, t.y., bet kuriuo laiko momentu turi būti teisinga kuri nors viena vykdymo sąlygų. Nuoseklumas reikalauja, kad įvykus įvykiui, tuo pačiu metu būtų vykdomi tik tie perėjimai, kurie yra suderinami. Be to, jeigu egzistuoja keletas perėjimų iš tos pačios būsenos, ir tie perėjimai yra sužadinami vieno ir to paties įvykio, viena ir tik viena vykdymo sąlyga gali būti teisinga.

Pilnumas ir nuoseklumas dažnai vadinami dinaminiais sistemos korektiškumo kriterijais. Statiniams kriterijams priskiriamas *būsenų pasiekiamumas*. Šis kriterijus reikalauja, kad būsenų mašinoje nebūtų izoliuotų būsenų, t.y. būsenų, neturinčių įeinančių perėjimų. Kitas statinis kriterijus, būdingas e. paslaugoms – *grįžtamumas*, reikalaujantis, kad iš visų būsenų egzistuotų kelias į laukimo būseną [4], tolygus bendresniam saugumo reikalavimui, kad iš bet kurios būsenos būtų galima pereiti į galinę būseną.

4. Siūlomas algoritmas dinaminiams korektiškumo kriterijams tikrinti

Prieš pateikdami mūsų siūlomą algoritmą, įvesime dar vieną apibrėžimą. Du vienam kontekstui priklausančius perėjimus vadinsime *panašiais*, jeigu juos sužadina tas pats įvykis. Algoritmu tikrinsime tų pačių įvykių iššaukiamų panašių perėjimų sąlygų nepersikertamumą. Taip pat tikrinsime, ar, įvykus įvykiui, visada bus teisinga viena kuri iš tų sąlygų.

Algoritmas grąžina visų nekorektiškai specifikuotų perėjimų aibę. Jame naudojamų aibių ir funkcijų aprašymą pateikiame 2 paveiksle.

Korektiški – korektiškų perėjimų aibė;

Nekorektiški – nekorektiškų perėjimų aibė. Ši aibė yra algoritmo darbo rezultatas;

Stekas – duomenų struktūra, atitinkanti LIFO (Last-In-First-Out) principą. Stekas turi operacijas *push* ir *pop*. Operacija push įrašo elementą sekos pabaigoje. Operacija pop – grąžina paskutinį sekos elementą ir jį pašalina iš steko;

PradineBusena(S) – funkcija, grąžinanti būsenų mašinos, aprašomos būsenų aibe *S*, pradinę būseną;

Aplankytos – algoritmo aplankytų būsenų aibė

Vaikai(s) – funkcija, grąžinanti būsenos *s* vaikines būsenas, arba tuščią aibę, jei *s* tokių neturi;

AND_busena(s) – loginė funkcija, grąžinanti reikšmę *true*, jeigu būsena *s* yra AND busena;

Palikuonys(s) – funkcija, gražinanti visų būsenos *s* palikuonių, t.y. vaikinių būsenų ir savo ruožtu visų pastarųjų vaikinių būsenų aibę;

Kaimynai(s) – funkcija, grąžinanti būsenos *s* kaimyninių būsenų, t.y., tokių į kurias egzistuoja perėjimas iš būsenos *s*, aibę

Panašūs(s) – funkcija, grąžinanti visų iš būsenos *s* išeinančių panašių perėjimų aibę;

Išėjimai(s) – funkcija, grąžinanti visų iš būsenos *s* išeinančių perėjimų aibę;

Sąlygos(T) – funkcija, grąžinantį aibę visų sąlygų, kurias turi perėjimai iš aibės *T*;

PoromisNesikerta(C) – funkcija, kurios reikšmė yra *true*, jeigu sąlygų aibės *C* elementai yra poromis nepersikertantys;

BentViena(C) – funkcija, kurios reikšmė yra *true*, jeigu sąlygų aibės *C* elementai tenkina sąlygą

$$\bigcup_{i=2}^{n} c_i = \overline{c_1}$$

2 pav. Algoritme naudojamų funkcijų ir aibių aprašymas.

Dinamika(S) – algoritmas, tikrinantis dinaminius kriterijus ir grąžinantis nekorektiškų perėjimų aibę.

S – visų būsenų aibė.

```
Korektiški := ∅
Stekas := ∅
s := PradineBusena(S)
Stekas.push(s)
Aplankytos := Aplankytos ∪ {s}

repeat
    s := Stekas.pop
    if(Vaikai(s) ≠ ∅)
      if(AND_busena(s))
          for v ∈ Vaikai(s)
              Korektiški := Korektiški ∪ (Palikuonys(v) -
Dinamika(Palikuonys(v)) )
          end for
      else
          Korektiški := Korektiški ∪ (Palikuonys(s) -
Dinamika(Palikuonys(s)) )
      end if
   end if
   pan := Panašūs(s)
   if(pan) = ∅
          Korektiški := Korektiški ∪ Išėjimai(s)
   else
          if( PoromisNesikerta(Sąlygos(pan)) AND
BentViena(Sąlygos(pan)) )
              Korektiški := Korektiški ∪ Išėjimai(s)
          end if
   end if
   for k ∈ Kaimynai(s)
     if k ∉ Aplankytos
          Stekas.push(k)
          Aplankytos := Aplankytos ∪ {k}
     end if
   end for
until Stekas = ∅
Nekorektiški = S – Korektiški
return Nekorektiški
```

3 pav. Algoritmas „Dinamika(S)"

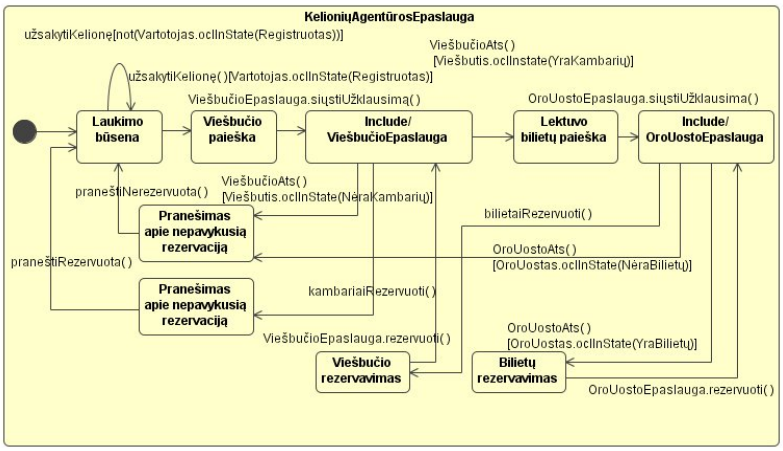
5. Kompozicinės e. paslaugos pavyzdys

Panagrinėkime kelionių agentūros e. paslaugos pavyzdį. Trumpas jos darbo scenarijaus aprašymas:

Vartotojas pateikia e. paslaugai planuojamos kelionės parametrus: kelionės tikslą ir keliaujančių asmenų skaičių. Kelionių agentūros e. paslauga kreipiasi į viešbučio e. paslaugą, pateikdama jai užklausą, ar nurodytą dieną viešbutyje bus laisvų kambarių nurodytam asmenų skaičiui. Viešbučio e. paslaugai pateikus atsakymą, kad kambarių bus, kelionių agentūros paslauga kreipiasi į oro uosto e. paslaugą, norėdama patikrinti ar bus bilietų skrydžiui. Jei yra ir vietų viešbutyje, ir bilietų skrydžiui, rezervuojami apartamentai ir bilietai, o vartotojui pateikiamas patvirtinimas. Jei nepavyksta rezervuoti kambarių arba bilietų, vartotojui siunčiamas pranešimas apie nepavykusią rezervaciją. Kelionių agentūros paslaugų būsenų mašinos fragmentas pateiktas 3 paveiksle (be jame pateiktų būsenų, į Kelionių agentūros paslaugų būsenų mašiną įeina ViešbučioEPaslaugos, OroUostoEPaslaugos, Viešbučio bei OroUosto būsenų mašinos).

Algoritmo vykdymo metu perrenkamos visos įmanomos būsenų mašinos perėjimų vykdymo trasos. Pateiksime kelias trasas, gautas pritaikius šį algoritmą kelionių agentūros paslaugų būsenų mašinos paslaugų pavyzdžiui (4 pav.):

```
1. užsakytiKelionę()[Vartotojas.oclInState(Registruotas)] ->
    ViešbučioEpaslauga.siųstiUžklausimą() ->
      ViešbučioAts()[Viešbutis.oclInState(NėraKambarių)] -> praneštiNerezervuota();
2. užsakytiKelionę()[Vartotojas.oclInState(Registruotas)] ->
      ViešbučioEpaslauga.siųstiUžklausimą() ->
 ViešbučioAts()[Viešbutis.oclInState(YraKambarių)] ->
 OroUostoEpaslauga.siųstiUžklausimą()->
 OroUostoAts()[Viešbutis.oclInState(NėraBilietų)] -> praneštiNerezervuota().
          ...
```

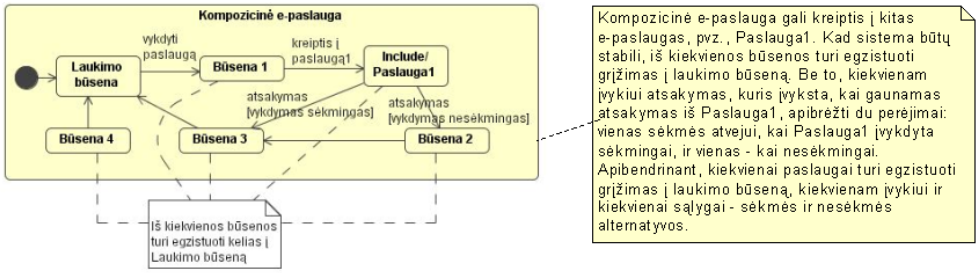


4 pav. Kelionių agentūros e. paslaugos būsenų modelio fragmentas

Šio modelio nekorektiškų perėjimų aibė yra tuščia, vadinasi, visi perėjimai tenkina užduotus korektiškumo kriterijus.

6. Elektroninės paslaugos būsenų mašinos šablonas

Šiame skyrelyje pateikiamas kompozicinės e. paslaugos būsenų mašinos šablonas (5 pav.), kuris padeda užtikrinti pilnumo ir nuoseklumo kriterijų išpildymą, todėl toks šablonas gali būti naudojamas kodui generuoti.



5 pav. Kompozicinės e. paslaugos šablonas

7. Išvados

Straipsnyje pateiktas siūlomas kompozicinių e. paslaugų elgsenos tikrinimo metodas, paremtas būsenų mašinos tikrinimu. Tam tikslui sudarytas sugriežtintas kompozicinių e. paslaugų būsenų mašinos metamodelis, į kurį įtraukti papildomi elementai būsenų mašinos vykdymui modeliuoti.

Metamodelio pagrindu sudarytas tikrinimo algoritmas, kuris tikrina būsenų mašiną pagal pilnumo ir nuoseklumo kriterijus, kurie išreiškia bendrinį būsenų mašinų saugumo kriterijų.

Būsenų mašina apibendrina paslaugų sistemos elgseną, kuri aprašoma įvairių tipų sąveikų modeliais. Paprastai sąveikų modeliai apima tik pagrindinius scenarijus, kadangi pilna scenarijų aibė, apimanti visus nesėkmingus atvejus, yra labai didelė. Todėl pasiūlytas šablonas, kuris padeda papildyti būsenų mašiną nesėkmingų atvejų aibe. Tokiu būdu sudaryta būsenų mašina išsamiai aprašo elgseną ir ją galima naudoti kodui generuoti.

Literatūros sąrašas

[1] **L.Čeponienė, L.Nemuraitė.** Design Independent Modeling of Information Systems. *In: Barzdins, J., Caplinskas, A. (red.) Databases and Information Systems. Selected Papers from the Sixth International Baltic Conference DB&IS'2004, IOS Pres*., 2005, p. 224-237.

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

[3] **Zs. Pap, I. Majzik1, A. Pataricza and A. Szegi.** Completeness and Consistency Analysis of UML Statechart Specifications.

[4] **Zs. Pap, I. Majzik1, A. Pataricza and A. Szegi.** Checking General Safety Criteria on UML Statecharts.

Verification of Web Services Models, Using UML State Machines

The article discusses application of UML state machines for Web service model verification. Semantics of state machines and criteria for their completeness and consistency are defined. The metamodel for state machines is supplemented with additional elements for execution modeling. The algorithm for verification of state machines is proposed suitable for implementation in UML CASE tools. At the last, the pattern for compositional Web services is presented, helping to ensure the correctness of the model.