KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

Laura Rimavičiūtė

INTERNETO PASLAUGŲ SISTEMŲ PROJEKTAVIMO METODIKA

Magistro darbas

Vadovė
doc. dr. L. Nemuraitė

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
INFORMACIJOS SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. R. Butleris

INTERNETO PASLAUGŲ SISTEMŲ PROJEKTAVIMO METODIKA

Informatikos mokslų magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė

Vadovė
doc. dr. L. Nemuraitė

Recenzentas
doc. S. Maciulevičius

Atliko
IFM 9/1 gr. stud.
L. Rimavičiūtė

KAUNAS, 2005

SUMMARY

Web services are widely used in banking, finance, insurance, tourism, e-business sectors and so on. Web services have emerged only recently, so the issues are typical for new technologies: ideas had arisen on technological level, and methods for modeling and development are coming only later. Existing development processes and notations such as Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA) frameworks, and Business Process Modeling (BPM) provide us with capabilities to identify and describe appropriate architectural abstractions, but it's not enough for realization of Service Oriented Architecture (SOA).

In this work the methodology is proposed for design of stand-alone services and combining them into larger system. Proposed methodology joins main principles of Object-Oriented Analysis and Design, Enterprise Architecture frameworks, and Business Process Modeling. This methodology allows to describe service-oriented models using UML diagrams and to combine them with each other. Stages of designing stand-alone services are: business modeling, requirements analysis, identification of interfaces, analysis of interface interaction, specification of components, service model (WSDL diagram) design. Designing services systems consists of following steps: use case modeling, services identification, choreography modeling, business modeling and composite service modeling.

The examples of proposed methodology for designing stand-alone services and combining them into larger systems are provided for explanatory purposes.

TURINYS

Lentelių sąrašas

Paveikslėlių sąrašas

ĮVADAS

Vis daugiau programinės įrangos kuriama interneto paslaugų pavidale. Tokios paslaugos plačiai naudojamos bankininkystėje, finansų, draudimo, turizmo industrijoje, elektroniniame versle ir kt. [18]. Prognozuojama, kad ateities programos jau neturės didelių sukompiliuotų paleidžiamųjų kodų (EXE, DLL ir kt.). Programos bus padalintos į mažesnes paslaugas, kurias paprasčiau sukurti, palaikyti ir valdyti.

Tinklo paslauga (angl. *Web Service*) – tai programinės įrangos dalis, prieinama per internetą ir naudojanti standartizuotą XML (angl. *Extensible Markup Language*) [11] pranešimų sistemą. Pagrindinis tinklo paslaugų privalumas – jos leidžia programoms dalintis duomenimis ir pasinaudoti kitų programų teikiamomis galimybėmis nepaisant to, kaip tos programos buvo sukurtos, kokioje operacinėje sistemoje ar platformoje jos dirba ir kokie įrenginiai reikalingi toms programoms pasiekti.

Tinklo paslaugos sparčiai populiarėja ir dažniau naujų sistemų realizacijai pasirenkama būtent šios technologijos. Jos atsirado palyginti neseniai, todėl šioje srityje susiduriama su tam tikromis problemomis, būdingomis naujoms technologijoms: idėjų plėtojimas vyksta technologiniame lygmenyje, o modeliavimo bei projektavimo metodai atsiranda vėliau. Esami projektavimo metodai (objektinė analizė ir projektavimas (angl. *Object-Oriented Analysis and Design – OOAD*) [24], organizacijų architektūros karkasai (angl. *Enterprise Architecture (EA) frameworks*) [24], veiklos procesų modeliavimas (angl. *Business Process Modeling – BPM*) [24]) teikia dideles galimybes identifikuojant ir apibrėžiant reikiamas architektūrines abstrakcijas, tačiau paslaugų architektūros realizacijai to nepakanka.

Šio darbo tikslas – sudaryti vieningą tinklo paslaugų projektavimo metodiką, skirtą pavienių paslaugų projektavimui ir jau esamų paslaugų panaudojimui kuriant informacinių sistemų projektus.

Tinklo paslauga yra programa, identifikuojama URI (angl. *Unified Resource Identifier*). Jos sąsajos ir ryšiai apibrėžiami, aprašomi ir randami panaudojant XML artefaktus. Ji tiesiogiai sąveikauja su kitomis programomis, naudodama XML pranešimus ir interneto protokolus [6].

Tinklo paslaugų technologija išsivystė iš penkių technologijų: tinklo (angl. *Network*), transportavimo (angl. *Transport*), pakavimo (angl. *Packaging*), aprašymo (angl. *Description*) ir atradimo (angl. *Discovery*). Todėl tinklo paslaugų realizacija remiasi dažniausiai naudojamais šių technologijų standartais: UDDI [1] [23], WSDL [7] [23] [12], SOAP [19] [23], HTTP [23], TCP/IP [23].

Tinklo paslaugos yra neatsiejamos nuo tinklo paslaugų architektūros, kuri pateikia konceptualųjį tinklo paslaugų modelį, padeda suprasti tinklo paslaugas, nustato ryšius tarp konceptualiojo modelio komponentų, tačiau nepateikia specifikacijų kaip tinklo paslaugą realizuoti ir neapriboja paslaugų tarpusavio sujungimo galimybių [6].

Tinklo paslaugų architektūrą apibendrina abstrakti paslaugų architektūra (angl. *Service-Oriented Architecture – SOA*) [3] [9] [10] [12] [16] – tai iš komponentų sudarytas sistemos modelis, kuris per aiškiai apibrėžtas sąsajas jungia skirtingo funkcionalumo programos elementus, vadinamus paslaugomis. Pagrindinės paslaugų architektūros esybės – paslauga, paslaugos vartotojas, paslaugos tiekėjas ir paslaugų katalogas. Paslaugų architektūros savybės: tiksliai apibrėžtos sąsajos, nepriklausomas paslaugų modelis, paslaugų skaidymas, paslaugų saugumas. Jos privalumai: paprastesnis integravimas ir valdymas, trumpesnis sukūrimo laikas, mažesnė kaina ir didesnės pakartotino naudojimo galimybės, lengvas keitimas ir tobulinimas.

Realizuojant tinklo paslaugų architektūrą naudojamas paslaugų analizės ir projektavimo metodas (angl. *Service-Oriented Analysis and Design – SOAD*) [10] [24], kuris jungia reikiamus egzistuojančių projektavimo metodų (objektinė analizė ir projektavimas, veiklos procesų modeliavimas, organizacijų taikomųjų programų integravimas, komponentinis projektavimas [10]) elementus.

Siūlomam tinklo paslaugų projektavimo metodui įgyvendinti pasirinktas MagicDraw paketas [13] [20], kadangi jis turi UML [5] išplėtimus WSDL diagramoms projektuoti ir galimybes generuoti WSDL dokumentus. Paslaugai realizuoti pasirinkta PHP [2] [4] technologija, kadangi buvo įdomu išbandyti, kaip realizuoti paslaugas, naudojant šią technologiją, kuri retai pasirenkama siekiant realizuoti tinklo paslaugas, tačiau plačiai naudojama kuriant įvairias sistemas.

Magistro darbą sudaro 4 dalys:

- Esamų projektavimo metodų analizė paslaugų projektavimo požiūriu,
- Tinklo paslaugų projektavimo metodas,
- Tinklo paslaugų projektavimo metodika CASE įrankiuose,
- Išvados.

Skyrius „Esamų projektavimo metodų analizė paslaugų projektavimo požiūriu“ susideda iš keturių dalių: tinklo paslaugų architektūros nagrinėjimo, esamų projektavimo metodų analizės, projektavimo ir realizacijos priemonių pasirinkimo bei analizės išvadų.

Pirmojoje analizės dalyje aprašoma tinklo paslaugos sąvoka, jos evoliucija, teikiama nauda. Apžvelgti technologijų, iš kurių išsivystė tinklo paslaugų technologija, lygmenys (tinklas, transportavimas, pakavimas, aprašymas, atradimas) ir pagrindiniai šių technologijų realizacijos standartai. Tinklo paslaugos yra neatsiejamos nuo tinklo paslaugų architektūros. Analizėje trumpai apžvelgiamos pagrindinės šios architektūros dalys: pranešimų, paslaugų, resursų bei strategijos modeliai. Tinklo paslaugų architektūrą apibendrina abstrakti paslaugų architektūra. Analizuojamos pagrindinės konceptualiojo paslaugų architektūros modelio esybės (paslauga, paslaugų teikėjas, paslaugų vartotojas, paslaugų lokatorius, paslaugų brokeris) ir pagrindinės paslaugų architektūros sąvokos (tiksliai apibrėžtos sąsajos, nepriklausomas paslaugų modelis, paslaugų skaidymas, paslaugų saugumas). Aptariami šios architektūros lygmenys ir paminimi pagrindiniai jos privalumai. Taip pat

šioje dalyje nagrinėjama tinklo paslaugų aprašymo kalba WSDL, jos dokumento struktūra ir pagrindiniai šio dokumento elementai.

Antrojoje analizės dalyje aptariami esami projektavimo metodai (objektinė analizė ir projektavimas, organizacijų architektūros karkasai, veiklos procesų modeliavimas, komponentinis projektavimas, sąsajų projektavimas, sluoksninis sistemų projektavimas) ir apibrėžiami reikalavimai tinklo paslaugoms projektuoti.

Analizės pabaigoje aptariami projektavimo ir realizacijos priemonių pasirinkimo kriterijai bei pateikiamos analizės išvados.

Skyriuje „Tinklo paslaugų projektavimo metodas“ aptariami paslaugų architektūrai realizuoti tinkami objektinės analizės ir projektavimo, organizacijų architektūros karkasų ir veiklos procesų modeliavimo elementai. Taip pat paaiškinama, kodėl būtų naudinga minėtų metodų elementus sujungti su tam tikrais naujais elementais, aptariamas paslaugų analizės ir projektavimo metodas, pagrindiniai jo etapai (identifikavimas, specifikavimas, realizavimas) ir artefaktai. Pateikiama sudaryta paslaugų projektavimo metodika, kuri sujungia objektinio projektavimo, veiklos modeliavimo, organizacijų ir paslaugų architektūros principus. Ji leidžia aprašyti paslaugoms būdingus modelius UML diagramomis ir suderinti jas tarpusavyje. Siūloma metodika susideda iš pavienių paslaugų ir paslaugų sistemų projektavimo. Pavienių paslaugų projektavimo etapai yra: veiklos modeliavimas, reikalavimų analizė, sąsajų identifikavimas, sąsajų sąveikų analizavimas, komponentų specifikavimas, paslaugos modelio (WSDL diagramos) sudarymas. Paslaugų sistemų projektavimas susideda iš šių etapų: panaudojimo atvejų modeliavimas, sąsajų identifikavimas, choreografijos modeliavimas, veiklos modeliavimas, kompozicinės paslaugos projektavimas.

„Tinklo paslaugų projektavimo metodika CASE įrankiuose“ dalyje pateikiami paslaugų projektavimo metodikos, suformuluotos skyriuje „Tinklo paslaugų projektavimo metodas“, realizavimo pavyzdžiai – pavienės kalendoriaus paslaugos projektas ir internetu transliuojamų laidų peržiūros sistemos projektas. Taip pat šioje dalyje aprašyti reikalavimai ir rekomendacijos CASE įrankiams, kurių pagalba būtų galima projektuoti paslaugas ir jų sistemas.

Šio darbo tema parašytas straipsnis „Interneto paslaugų sistemų projektavimo metodika“ (žr. 2 Priedas), kuris buvo pristatytas 10-oje tarpuniversitetinėje magistrantų ir doktorantų konferencijoje „INFORMACINĖS TECHNOLOGIJOS“, vykusioje KTU š.m. balandžio mėn. 29 d. Kaune.

1 ESAMŲ PROJEKTAVIMO METODŲ ANALIZĖ PASLAUGŲ PROJEKTAVIMO POŽIŪRIU

1.1 Tinklo paslaugų architektūra

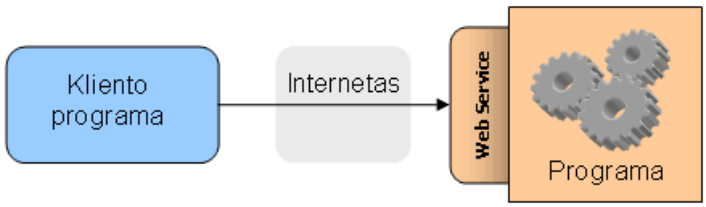
1.1.1 Tinklo paslaugų sąvokos

Tinklo paslauga – tai programinės įrangos dalis, prieinama per internetą ir naudojanti standartizuotą XML (angl. *Extensible Markup Language*) [11] pranešimų sistemą (1.1 paveikslas) [23].

Išsamų tinklo paslaugos apibrėžimą pateikia *WWW Consortium* (*W3C*):

Tinklo paslauga yra programa, identifikuojama URI (angl. *Unified Resource Identifier*). Jos sąsajos ir ryšiai apibrėžiami, aprašomi ir randami panaudojant XML (angl. *Extensible Markup Language*) artefaktus. Ji tiesiogiai sąveikauja su kitomis programomis, naudodama XML pranešimus ir interneto protokolus [6].

XML yra tinklo paslaugų pagrindas, kuris buvo sukurtas siekiant aprašyti duomenis. Pagrindinės XML funkcijos – tai bendravimo tarp programų užtikrinimas, duomenų integravimas ir programos bendravimas su išoriniais partneriais. Dėl XML standartizavimo, skirtingos programos gali daug lengviau tarpusavyje bendrauti.



1.1 pav. Tinklo paslauga leidžia pasiekti programos funkcionalumą per internetą

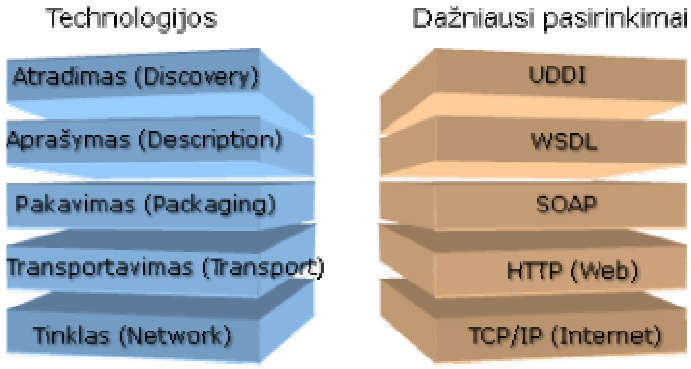
Tinklo paslaugos leidžia programoms dalintis duomenimis ir pasinaudoti kitų programų teikiamomis galimybėmis nepaisant nuo to, kaip tos programos buvo sukurtos, kokioje operacinėje sistemoje ar platformoje jos dirba ir kokie įrenginiai reikalingi toms programoms pasiekti. Kol XML tinklo paslaugos išlieka nepriklausomos viena nuo kitos, jos gali laisvai prisijungti prie bendradarbiaujančių grupių, atliekančių tam tikrus uždavinius.

Tinklo paslaugos neatsirado staiga. Tai nebuvo revoliucinė naujovė, o tiesiog puikus egzistuojančių tinklo protokolų evoliucionavimo pavyzdys. Nuo 1994 metų, keletas kompanijų kūrė paskirstytuosius objektus (angl. *Distributed Objects*) ir juos vadino skirtingais vardais. NeXT juos vadino nešiojamaisiais paskirstytaisiais objektais (angl. *Portable Distributed Objects*), Microsoft – komponentų modeliu (angl. *Compnent Object Model - COM*), IBM – sistemos objektų modeliu (angl. *System Object Model - SOM*), o Apple – OpenDoc. Šios kompanijos (išskyrus Microsoft) suformavo objektų valdymo grupę (angl. *Object Management Group* – OMG) ir priėmė standartą, pavadintą

bendra objektų kreipinių brokerių sistema (angl. *Common Object Request Broker Architecture – CORBA*). Šio standarto esmė yra kurti architektūrą, kuri leistų programoms „įsilieti į programų magistralę“ ir iškviesti tam tikrą paslaugą. Remiantis objektiniais moduliais ši architektūra buvo pavadinta specializuotais programinės įrangos komponentais. Šiandien bendradarbiaujantys objektai vadinami tinklo paslaugomis (angl. *Web services*).

1.1.2 Tinklo paslaugų technologijos

Tinklo paslaugų technologija išsivystė iš penkių technologijų: tinklo (angl. *Network*), transportavimo (angl. *Transport*), pakavimo (angl. *Packaging*), aprašymo (angl. *Description*) ir atradimo (angl. *Discovery*) [23]. Kiekviena iš šių technologijų priklauso nuo žemiau esančių technologijų, pavyzdžiui, be realizuoto tinklo negali būti transportavimo.



1.2 pav. Tinklo paslaugų technologijų lygmenys

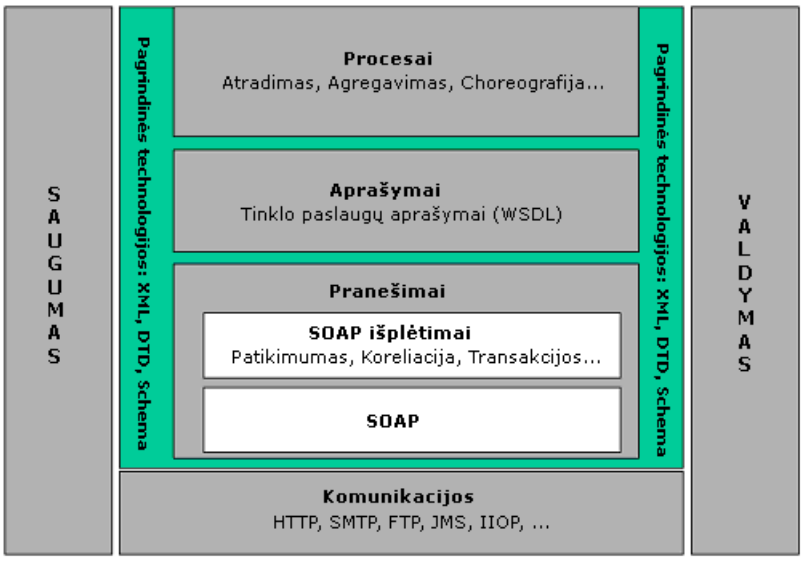
1.2 paveikslas iliustruoja tinklo paslaugų saugyklą (kairėje) ir dažniausius šių sluoksnių realizacijos pasirinkimus (dešinėje):

- **Atradimas (*Discovery*):** Norint kad programa naudotų tinklo paslaugą, pirmiausiai reikia tą paslaugą rasti. Šis sluoksnis atsakingas už paslaugų patalpinimą į bendrą registrą, lengvą paskelbimą ir radimą. Susietas serverių tinklas, kuris aprūpina globaliu tinklo paslaugų katalogu (angl. *The Universal Description, Discovery, and Integration (UDDI) project* [23] [1]) yra dabartinis standartas. UDDI specifikacijos gali padėti surasti reikiamą paslaugą iš milijonų paslaugų, jau publikuojamų internete.
- **Aprašymas (*Description*):** Prieš sukuriant tinklo paslaugų katalogą, reikia rasti būdą, kaip aprašyti tinklo paslaugą. Šis sluoksnis aprašo viešąją sąsają tam tikrai tinklo paslaugai. WSDL (angl. *Web Service Description Language*) [23] [7] [12] – XML pagrįsta kalba, kuri aprašo įvairias tinklo paslaugos atliekamas funkcijas. Kai tik tinklo paslaugos aprašas suformuojamas, programa jau turi viską, ko reikia tam, kad galėtų pasinaudoti paslauga.
- **Pakavimas (*Packaging*):** Tam, kad tinklo paslauga būtų prieinama bet kokiai programai, nekreipiant dėmesio į tai, kokia programavimo kalba ta programa parašyta, kokioje

operacinėje sistemoje ji dirba, tinklo paslaugos iškvietimai (angl. *requests*) ir jos atsakymai (angl. *responses*) turi būti užkoduoti standartiniu, nuo platformos nepriklausomu formatu. SAOP (*Simple Object Access Protocol*) [19] [23] – tai XML pagrįsta kalba, kuri leidžia supakuoti funkcijų iškvietimus ir dokumentus kaip tinklo paslaugos iškvietimus.

- **Transportavimas (*Transport*):** Kai tinklo paslaugos iškvietimas supakuotas (pavyzdžiui, kaip SOAP pranešimas), kažkokiu būdu jis turi būti perduotas tinklo paslaugai. Taip pat tinklo paslauga turi turėti galimybę kokiu nors būdu perduoti atsakymą. Tinklo paslaugų technologija yra gana lanksti ir gali naudoti beveik bet kokį protokolą (pavyzdžiui, Microsoft pristato gausybę paslaugų, teikiamų pasinaudojant skubiais pranešimais (angl. *Instant messaging*). Taip pat, netgi elektroninis paštas gali būti transportavimo mechanizmu). Dažniausiai naudojamas HTTP (*Hypertext Transport Protocol*) [23] protokolas (interneto naršyklėse iškviečiant ir gaunant tinklalapius).
- **Tinklas (*Network*):** Tam, kad būtų galima siųsti iškvietimus ir atsakymus, tarp kompiuterių turi būti tinklas. Dažniausiai šis tinklas – tai internetas, arba koks nors vidinis tinklas, paremtas TCP/IP (*Transmission Control Protocol/Internet Protocol*) [23] protokolu.

Taigi tinklo paslaugų architektūra apima daug skirtingų ir tarpusavyje susijusių technologijų. Yra įvairių būdų šioms technologijoms realizuoti ir atvaizduoti. Vienas paprastas tinklo paslaugų technologijų atvaizdavimo būdas pateiktas 1.2 paveiksle, kitas, detalesnis ir sudėtingesnis atvaizdavimo būdas, kurį pateikia *WWW Consortium* (*W3C*) [6], pateiktas 1.3 paveiksle.



1.3 pav. Tinklo paslaugų technologijų visuma

Nepaisant tinklo paslaugų sudėtingumo, priimta, kad tinklo paslauga – tai paslaugų architektūra papildomai turinti mažiausiai šiuos apribojimus:

- Sąsajos turi būti pagrįstos interneto protokolais: HTTP, FTP ir SMTP;

- Pranešimai turi būti užrašomi XML formatu (išskyrus binarinius duomenų priskyrimus).

Yra du pagrindiniai tinklo paslaugų tipai – SOAP tinklo paslaugos ir REST tinklo paslaugos [12].

SOAP tinklo paslaugos turi tokius apribojimus:

- Pranešimai turi būti perduodami SOAP pagalba (išskyrus binarinius duomenų priskyrimus);
- Paslaugos aprašymas turi būti pateiktas WSDL kalba.

SOAP tinklo paslaugos – tai dažniausiai pasitaikančios tinklo paslaugos. SOAP atlieka apvalkalo funkcijas ir saugo pranešimų turinį. SOAP privalumas yra toks, kad jis palaiko platų pasikeitimo pranešimais diapazoną: nuo tradicinio užklausa-atsakymas pranešimo, iki transliavimo bei sudėtingų pranešimų tarpusavio sąryšių.

REST tinklo paslauga – tai paslaugų architektūra paremta resurso sąvoka. Resursas – tai bet kokia esybė, turinti URI (angl. *Universal Resource Identifier*). Resursas gali turėti keletą atvaizdavimų, arba neturėti nei vieno. REST tinklo paslaugos turi tokius apribojimus:

1. Sąsajos yra apribotos HTTP protokolui. Yra apibrėžtos tokios semantikos:
   - HTTP GET naudojama norint gauti resurso atvaizdavimą. Vartotojas tai naudoja siekdamas išgauti resurso atvaizdavimą iš URI. Paslaugos teikiamos per šią sąsają negali gauti jokių nurodymų ar apribojimų iš vartotojo.
   - HTTP DELETE – skirta resurso atvaizdavimui pašalinimui.
   - HTTP POST – naudojama resurso atvaizdavimui atnaujinti ar naujam atvaizdavimui pridėti.
   - HTTP PUT – skirta naujam atvaizdavimui pridėti.
2. Daugelis pranešimų išreiškiami XML formatu, juos pritaikant prie schemos, užrašytos XML Schema [12] [22] ar RELAX NG [8] [12] būdu.
3. Paprasti pranešimai gali būti koduojami URL kodavimu.
4. Paslauga ir paslaugos tiekėjas turi būti resursais, o paslaugos vartotojas gali būti resursas.

REST tinklo paslaugos reikalauja nedidelio infrastruktūrinio palaikymo, išskyrus standartines HTTP ir XML technologijas, kurias palaiko dauguma programavimo kalbų ir platformų. REST tinklo paslaugos yra paprastos ir efektyvios, kadangi HTTP protokolas yra plačiausiai prieinama ir daugeliui programų pakankamai gera sąsaja. Tačiau daugeliu atvejų HTTP protokolo paprastumo nepakanka, kad tinklo paslaugai realizuoti būtų pasirinkta REST technologija. Taip yra todėl, kad REST reikalauja sudėtingo papildomo transportavimo sluoksnio įdiegimo.

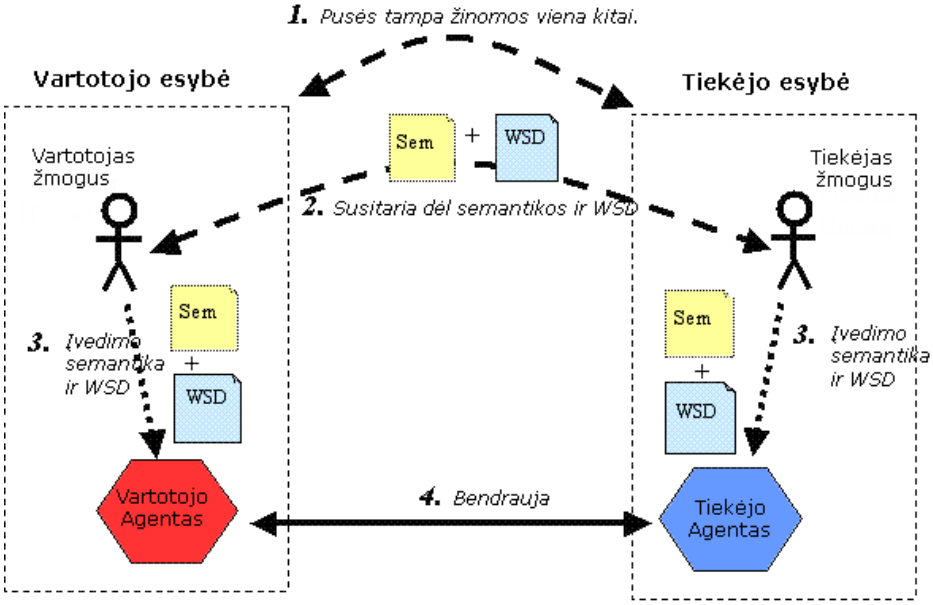
1.1.3 Tinklo paslaugų architektūra

Tinklo paslaugos yra neatsiejamos nuo tinklo paslaugų architektūros, kuri pateikia konceptualųjį tinklo paslaugų modelį, padeda suprasti tinklo paslaugas, nustato ryšius tarp konceptualiojo modelio komponentų, tačiau nepateikia specifikacijų, kaip tinklo paslaugą realizuoti ir neapriboja paslaugų tarpusavio sujungimo galimybių [6].

Tinklo paslauga – tai resursas, apibūdinamas abstrakčia teikiamo funkcionalumo aibe. Paslaugą realizuoja agentas – konkreti programinės ar aparatūrinės įrangos dalis, siunčianti ir gaunanti pranešimus. Įvykus pokyčiams agente (pavyzdžiui, agentą suprogramavus kita kalba) – tinklo paslauga išlieka nepakitusi.

Tinklo paslaugos pranešimų pasikeitimo mechanizmas dokumentuojamas tinklo paslaugos aprašymu (angl. *Web Service Description – WSD*). WSD – įrenginio apdorota tinklo paslaugos specifikacija, išreikšta WSDL kalba. Ji nustato pranešimų formatus, duomenų tipus, transportavimo protokolus, transportavimo išdėstymo serijomis formatus, kurie turi būti naudojami bendravimui tarp paslaugos vartotojo ir paslaugos tiekėjo realizuoti. Specifikacija taip pat nurodo vieną, ar kelis tinklo adresus, kur paslaugos tiekėjas gali būti iškviestas.

Be WSDL aprašo, tinklo paslauga turi apibrėžtą semantiką – tai bendrai aprašytas laukiamas paslaugos elgesys atsakant į vartotojo siųstus pranešimus. Kitais žodžiais tariant, semantika – tai kontraktas tarp paslaugos vartotojo ir paslaugos tiekėjo dėl bendradarbiavimo tikslo ir rezultatų. Šis kontraktas nebūtinai turi būti rašytinėje, ar aiškaus susitarimo formoje. Jis gali būti aiškus ir tikslus, arba numanomas, apdorojamas įrenginyje ar skirtas žmogui ir kt.

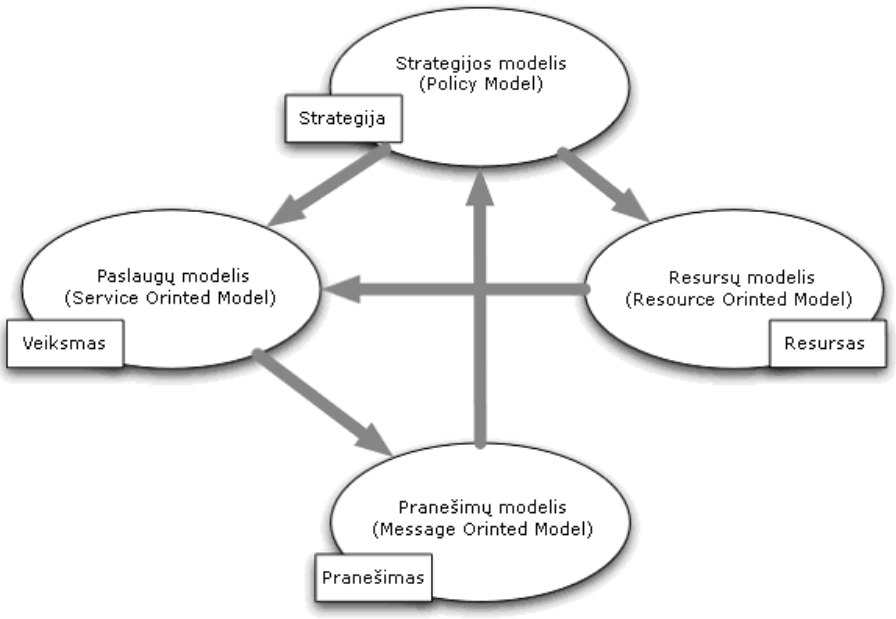


1.4 pav. Bendrasis tinklo paslaugos naudojimo procesas

Yra daug būdų, kaip vartotojas gali pasinaudoti paslauga. 1.4 paveiksle pavaizduoti pagrindiniai tinklo paslaugos panaudojimo žingsniai:

1. Vartotojo ir paslaugos tiekėjo esybės tampa žinomos viena kitai;
2. Vartotojo ir tiekėjo esybės kažkokiu būdu susitaria dėl ryšį tarp kliento ir tiekėjo valdančių paslaugos aprašymo ir semantikos;
3. Vartotojo ir tiekėjo esybės realizuoja paslaugos aprašymą ir semantiką;
4. Vartotojo ir tiekėjo esybės keičiasi pranešimais atliekančiais tam tikrus uždavinius.

Tinklo paslaugų architektūra turi keturis modelius, kurie pavaizduoti 1.5 paveiksle.

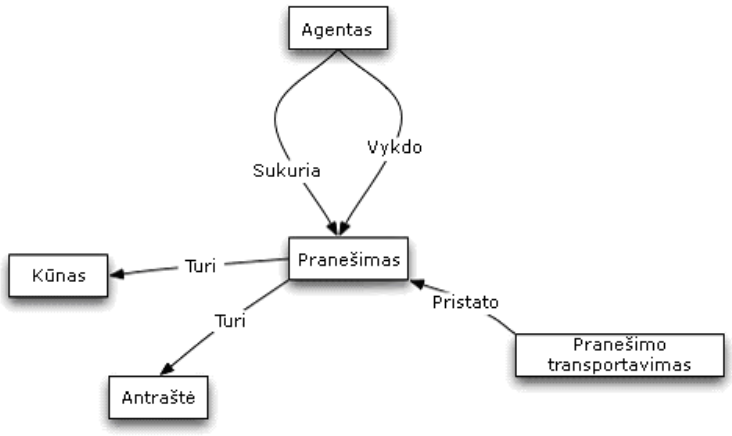


1.5 pav. Paslaugų architektūros metamodelis

Pagrindinės architektūros metamodelio dalys yra:

- panešimų modelis,
- paslaugų modelis,
- resursų modelis,
- strategijos (angl. *Policy*) modelis.

Pranešimų modelyje daugiausiai dėmesio skiriama pranešimams, pranešimo struktūrai, transportavimui ir kt. Šiame modelyje nepateikiama jokia informacija apie pranešimų priežastis ar jų svarbumą. Pranešimų modelio sąvokų žemėlapis (angl. *Concept map*) pateiktas 1.6 paveiksle.

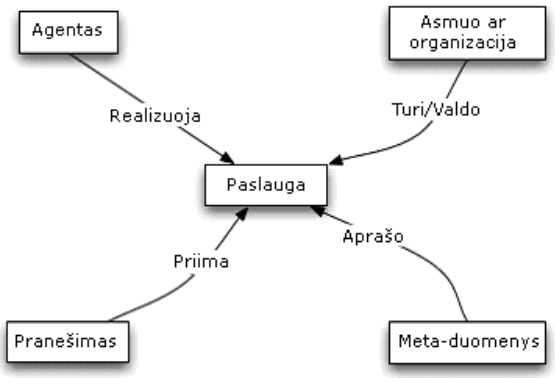


1.6 pav. Pranešimų modelio sąvokų žemėlapis

Pagrindinės pranešimų modelio sąvokos:

- agentas (arba vartotojas) kuris siunčia ir gauna pranešimus,
- pranešimo struktūra susidedanti iš pranešimo antraštės ir pranešimo kūno,
- pranešimams perduoti naudojami specialūs mechanizmai.

Paslaugų modelyje koncentruojamasi ties paslaugos, veiksmo ir kitais aspektais. Paslaugų modelio sąvokų žemėlapis pateiktas 1.7 paveiksle.

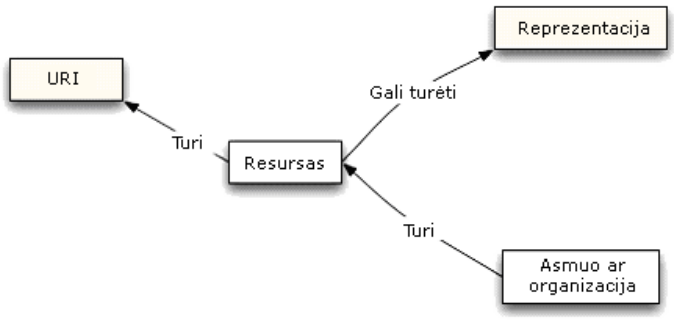


1.7 pav. Supaprastintas paslaugų modelio sąvokų žemėlapis

Tai yra pats sudėtingiausias iš visų keturių architektūros modelių. Pagrindinės šio modelio idėjos – paslaugą realizuoja agentas, ją naudoja kitas agentas. Paslaugos pasiekiamos keičiantis pranešimais tarp paslaugos agento ir vartojo agento.

Labai svarbus paslaugų aspektas – tai jų sąryšis su realiu pasauliu, kadangi paslaugos dažniausiai kuriamos tam, kad teiktų tam tikrą funkcionalumą realiam pasauliui (asmeniui, ar organizacijai). Be to paslaugų modelis naudoja metaduomenis. Jie naudojami norint dokumentuoti daugelį paslaugų aspektų: sąsajos elementus ir transportavimo sąryšius su paslaugos semantika ir paslaugos elgsenos apribojimus.

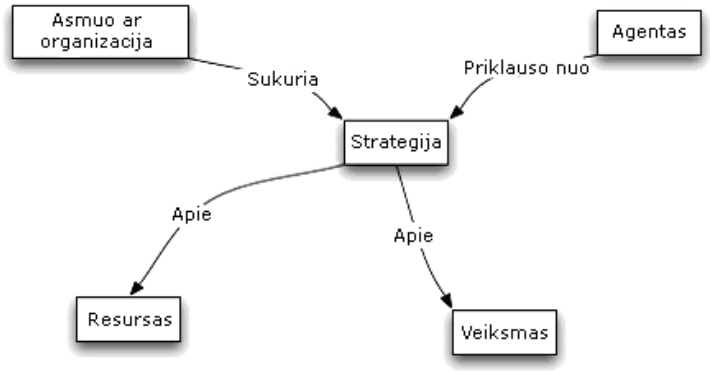
Resursų modelyje koncentruojamasi ties egzistuojančiais ir turinčiais savininkus resursais. Šio modelio sąvokų žemėlapis pateiktas 1.8 paveiksle.



1.8 pav. Resursų modelio sąvokų žemėlapis

Resursų modelis įkūnija paslaugų architektūros resurso sąvoką. Šiame modelyje sąvoka išplėsta, kad būtų galima atvaizduoti ryšius tarp resursų ir jų savininkų.

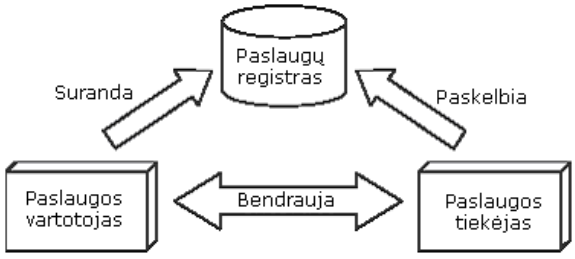
Strategijos (angl. Policy) modelyje didžiausias dėmesys skiriamas paslaugų ir agentų elgsenos apribojimams. Strategijos modelio sąvokų žemėlapis pateiktas 1.9 paveiksle.



1.9 pav. Strategijos modelio sąvokų žemėlapis

Strategijos taikomos agentams, kurie gali bandyti šiuos resursus pasiekti. Strategijos sudaro ar patvirtina už šiuos resursus atsakingi žmonės. Strategijos gali atitikti saugumo, paslaugos kokybės ir valdymo interesus.

Tinklo paslaugų architektūrą apibendrina abstrakti paslaugų architektūra (angl. *Service-Oriented Architecture – SOA*) [3] [9] [10] [12] [16] – tai iš komponentų sudarytas sistemos modelis, kuris per aiškiai apibrėžtas sąsajas jungia skirtingo funkcionalumo programos elementus vadinamus paslaugomis. Paslaugų architektūra pavaizduota 1.10 paveiksle.



1.10 pav. Paslaugų architektūra

Pagrindinės paslaugų architektūros modelio esybės:

- Paslaugų teikėjas (angl. *Service provider*) – tai programinė esybė, realizuojanti paslaugų specifikacijas.
- Paslaugų vartotojas (angl. S*ervice consumer, user, requestor*) – tai programinė esybė, iškviečianti paslaugų teikėją. Nuo tradicinio kliento skiriasi tuo, kad ši esybė gali būti paslauga ar programa.
- Paslauga (angl. *Service*) – tai paslaugų tiekėjo atliekamas darbinis modulis apibrėžiamas viena ar daugiau sąsajų, skirtas pageidaujamiems paslaugų vartotojo galutiniams rezultatams pasiekti.
- Paslaugų katalogas (angl. *Service locator, Service broker*) – specialus paslaugų teikėjas, kuris veikia kaip registras ir leidžia peržiūrėti paslaugų teikėjų sąsajas ir jų buvimo vietas.

Tiek paslaugų tiekėjas, tiek paslaugų vartotojas – tai programinės įrangos agentai, dirbantys jų savininkų vardu.

Paslaugos rezultatas paprastai keičia paslaugos vartotojo būseną, tačiau gali keisti ir paslaugos tiekėjo arba ir paslaugos tiekėjo ir paslaugos vartotojo būsenas.

Paslaugų architektūra nėra naujas dalykas, tai alternatyva labiau įprastiems, stipriai susietiems objektiniams modeliams. Paslaugų architektūra pagrįstos sistemos gali būti sudarytos iš pakartotinai naudojamų paslaugų, sukurtų remiantis objektiniais projektavimo principais, tačiau ji pati nėra objektinė. Skirtumas tarp šių dviejų architektūrų – tai sąsaja, jungianti skirtingo funkcionalumo programos elementus. Sąsaja neturi priklausyti nuo naudojamos techninės įrangos, operacinės sistemos ir programavimo kalbos, kuria paslauga yra realizuota. Toks neutralumas leidžia paslaugas realizuoti įvairiose sistemose, nesvarbu ar paslaugų sąsajos panašios, ar visiškai skirtingos ir yra vadinamas laisvu paslaugų susiejimu (angl. *Loose coupling*). Sistemos, kurių komponentai jungiami laisvai susiejant, pasižymi lankstumu, joms lengviau pritaikomi įvairūs struktūriniai bei vidiniai realizacijos pakeitimai. Tokias sistemas galima nesunkiai pritaikyti prie nuolat besikeičiančių sąlygų, pavyzdžiui, pasikeitusi įmonės strategija, poreikiai ir kiti faktoriai darantys įtaką įmonės veiklai. Paslaugų architektūroje laisvas susiejimas tarp bendradarbiaujančių programinės įrangos agentų realizuojamas dviem architektūriniais apribojimais:

- Mažas paprastų ir visur esančių sąsajų rinkinys, skirtas visiems aktyviems programinės įrangos agentams. Sąsajose užkoduota tik bendra semantika. Sąsajos turi būti universalios ir prieinamos visiems paslaugų tiekėjams ir vartotojams.
- Aprašų pranešimai ribojami sąsajų pateikiamomis išplečiamomis schemomis. Pranešimai nurodo labai mažą (ar netgi visai nenurodo) sistemos elgesio dalį. Schemos riboja pranešimų struktūrą ir žodyną. Išplečiamų schemų dėka naujos paslaugų versijos gali būti realizuotos nenutraukiant paslaugos darbo.

Tvirtas paslaugų susiejimas (angl. *Tight-coupling*) reiškia, kad sąsajos tarp skirtingų sistemos komponentų stipriai susiję tiek funkcionalumu, tiek forma. Todėl, kai prireikia bet kokių pakeitimų, ryšiai tarp komponentų tampa netvirti.

Apžvelgus laisvą ir tvirtą paslaugų susiejimą, galima daryti išvadą, kad paslaugų architektūroje sąsajos yra labai svarbios. Sąsaja apibrėžia viešai prieinamų operacijų signatūras – kontraktą tarp kliento ir paslaugų teikėjo. Bet kuri sąsajos realizacija turi užtikrinti visus jos metodus. Jei sąsaja neveikia arba klaidingai veikia, tai ir visa sistema negali funkcionuoti. Sąsajos aprašo sistemos elgseną ir labai sudėtinga šį aprašymą realizuoti be klaidų, kadangi paslaugoms įgyvendinti naudojamos skirtingos platformos ir kalbos.

Kuriant architektūros modelį nepakanka tik paslaugos aprašymo. Turi būti apibrėžta, kaip sistemoje bus valdomi darbų srautai (angl. *Workflow*) tarp įvairių paslaugų. Taip pat paslaugų architektūra turi sieti veiklos procesus su juos realizuojančiais techniniais procesais. Taigi, norint, kad sistema dirbtų efektyviai, turi būti apibrėžtos tam tikros paslaugų tarpusavio bendradarbiavimo taisyklės, užtikrinančios, jog reikiami procesai veiks taip, kaip buvo tikėtasi.

Paslaugų architektūra pateikia sistemų projektavimo sprendimus, kurie prireikus gali būti išplėsti ar pakeisti. Tokie sprendimai susideda iš pakartotinai naudojamų paslaugų, su tiksliai apibrėžtomis, paskelbtomis ir standartus atitinkančiomis sąsajomis. Paslaugų architektūra pateikia vis dar naudojamų, pasenusių (angl. *legacy*) sistemų sujungimo mechanizmą, nesvarbu kokia programavimo kalba ta sistema parašyta, ar kokioje operacinėje sistemoje ji dirba.

Yra trys pagrindiniai paslaugų architektūros abstrakcijos lygiai:

- Operacijos: transakcijos išreiškiančios bendrus loginius darbo vienetus (angl. *Single Logical Unit of Work – LUW*). Operacijos vykdymas paprastai iššaukia nuolatinius duomenų įrašų nuskaitymus, įrašymus ar redagavimus. Paslaugų architektūroje operacijos tiesiogiai prilyginamos objektiniams metodams.
- Paslaugos: išreiškia logines operacijų grupes.
- Veiklos procesai: ilgas tam tikrų veiksmų vykdymas siekiant įgyvendinti vienokius ar kitokius veiklos tikslus. Veiklos procesas paprastai apima kreipimąsi į daugelį paslaugų.

Veiklos procesas susideda iš operacijų sekos. Operacijų vykdymo eiliškumas priklauso nuo veiklos taisyklių. Operacijų parinkimas, eiliškumo sudarymas bei tų operacijų vykdymas vadinamas paslaugos ar proceso choreografija (angl. *choreography*).

Paslaugų architektūroje naudojamos sąvokos:

- tiksliai apibrėžtos sąsajos,
- nepriklausomas paslaugų modelis,
- paslaugų skaidymas,
- paslaugų saugumas.

Tiksliai apibrėžtos sąsajos reiškia, kad paslaugų sąryšiai turi būti apibrėžti. Vartotojas, norėdamas bendrauti su paslauga, turi gauti tam tikrą informaciją. Paslaugų aprašymas apibrėžia operacijas, kurios:

- leidžia bendrauti su paslauga,
- apibrėžia pranešimus, iškviečiančius operacijas,
- nustato pranešimų sudarymo instrukcijas,
- teikia informaciją, kur siųsti pranešimus, siekiant gauti pranešimų sudarymo instrukcijas.

Nepriklausomas paslaugų modelis reiškia, kad paslaugos turi būti nepriklausomos ir savarankiškos užklausų (angl. *request*) prasme. Užklausos privalo turėti pakankamai informacijos operacijai vykdyti ir neturi reikalauti informacijos ar būsenos, kurias perdavė prieš tai buvusios užklausos. Paslaugos taip pat neturėtų priklausyti nuo kitų paslaugų konteksto ar būsenos. Jei priklausomybė yra reikalinga, tuomet ji turi būti apibrėžta veiklos procesų, funkcijų ir duomenų modeliuose.

Paslaugų skaidymas – tai labai svarbi paslaugų architektūros sąvoka. Išoriniams taikymams rekomenduojama naudoti stambiai (angl. *coarse-grained*) suskaidytas sąsajas, tuo tarpu smulkiai suskaidytos (angl. *fine-grained*) sąsajos gali būti taikomos paslaugoje, realizuojant smulkias veiklos operacijas. Smulkiai suskaidytos sąsajos suteikia daugiau lankstumo vartotojo sistemai, tačiau bendravimo su paslauga modelis tampa sudėtingesnis.

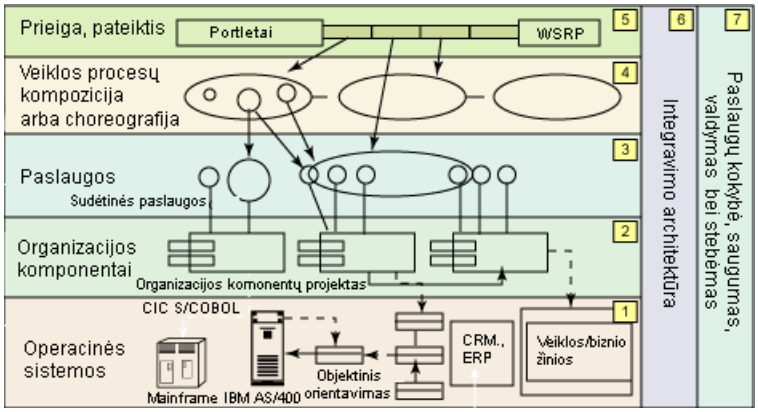
Stambiai suskaidytos sąsajos užtikrina, kad paslaugos vartotojai šią paslaugą naudos nuosekliai (pavyzdžiui, nebandys atlikti tam tikrų operacijų, kurias galima vykdyti tik atlikus tam tikrus veiksmus). Paslaugų architektūra nereikalauja naudoti tik stambiai suskaidytų sąsajų, tačiau rekomenduoja naudoti tokį suskaidymą, kadangi tai yra geriausias išorinio paslaugų sujungimo sprendimas. Kuriant stambiai suskaidytas sąsajas, kurios vykdo veiklos procesus, susidedančius iš smulkiai suskaidytų operacijų, naudojama paslaugų choreografija.

Paslaugų saugumas reiškia, jog reikia nepamiršti apie kuriamų paslaugų saugumą, kadangi šios paslaugos gali būti naudojamos sistemose, kurios galbūt valdo svarbių veiklos procesų vykdymą. Interneto protokolai (naudojami pranešimams tarp paslaugos tiekėjo ir paslaugos vartotojo perduoti) nėra patikimi, todėl reikia užtikrinti, kad pranešimas bus perduotas ir apdorotas. Jei pranešimo perduoti neįmanoma, paslaugos vartotojas kažkokiu būdu turi būti informuotas, kad jo užklausa neįvykdyta.

Tinklo paslauga susideda iš komponentų, kurie veikia kartu, kad atliktų tam tikras tinklo paslaugos funkcijas. Komponentai yra smulkesni nei tinklo paslaugos. Komponentiniame projektavime komponentai kuriami taip, kad jie atitiktų veiklos esybes ir apjungtų tam tikrą, laukiamą esybių elgseną. Paslaugų projektavime, kuriant paslaugas nesiremiama veiklos esybėmis. Čia kiekviena paslauga – tai savarankiška dalis, kuri vykdo tam tikras operacijas esybių rinkinyje.

Įmonėje paslaugų architektūrą palaiko organizacijos paslaugų magistralė (angl. *Enterprise Service Bus – ESB*). Abstrakčiai paslaugų architektūrą būtų galima pavaizduoti, kaip veiklos procesus vykdančią, iš dalies suskaidytą lygmenimis ir iš paslaugų sudarytą architektūrą. Tokį požiūrį iliustruoja 1.11 paveikslas.

Ryšius tarp tinklo paslaugų ir komponentų galima paaiškinti taip, kad organizaciniai komponentai (stambūs verslo lygio komponentai) realizuoja paslaugas ir yra atsakingi už paslaugų funkcionalumą bei kokybę. Programose veiklos procesų srautus valdo paslaugų, realizuojamų šiais komponentais, choreografija. Integruojamoji architektūra palaiko šių paslaugų, komponentų ir procesų srautų maršrutizavimą, pasiekimą, siuntimą ir transliavimą naudojant organizacijos paslaugų magistralę. Sukurtos paslaugos turi būti patikrintos ar jos atitinka kokybės ir nefunkcinius reikalavimus.



1.11 pav. Paslaugų architektūros lygmenys

Kiekvienam iš šių lygmenų turi būti parinkti architektūriniai sprendimai.

Trumpi šių lygmenų aprašymai:

1. *Vykdomasis sisteminis lygmuo.* Jis susideda iš egzistuojančių programų, vadinamų pasenusiomis sistemomis (angl. *Legacy systems*). Paslaugų architektūra, susidedanti iš įvairių sluoksnių, gali reikiamai įtakoti egzistuojančias sistemas ir jas sujungti naudojant paslaugų sujungimo metodus.
2. *Organizacijos komponentų lygmuo.* Šie komponentai atsakingi už funkcionalumo realizaciją ir publikuojamų paslaugų kokybę (angl. *Quality of Service – QoS*). Šiame lygmenyje paprastai naudojamos talpyklų technologijos (pavyzdžiui, taikomųjų programų serveriai), kurias naudojant realizuojami komponentai, valdomas darbo krūvis ir pan.
3. *Paslaugų lygmuo.* Šis lygmuo suteikia galimybę pasirinkti paslaugas. Paslaugos gali būti surastos, susietos tarpusavyje arba, panaudojant paslaugų choreografiją, įtrauktos į didesnes sudėtines paslaugas.
4. *Veiklos procesų kompozicijos arba choreografijos lygmuo.* Kompozicijos ir choreografijos, naudojamos trečiajame lygmenyje, yra sudaromos šiame architektūros

sluoksnyje. Paslaugos įtraukiamos į choreografiją ir veikia kaip viena sistema. Tokios programos atitinka tam tikrus panaudojimo atvejus ir veiklos procesus.

5. *Prieigos arba pateikties lygmuo.* Nors šis lygmuo dažniausiai neįtraukiamas į paslaugų architektūros aptarimą, tačiau jis tampa vis aktualesnis. Modelyje šis lygmuo pateiktas, kadangi jam yra nemažai standartų (pavyzdžiui, *Web Services for Remote Portlets Version 2.0* ir kt.), siūlančių sprendimus taikomųjų programų sąsajų ir pateikties lygmenyje, kai naudojamos paslaugos.
6. *Integravimo lygmuo.* Šis lygmuo leidžia sujungti paslaugas naudojant, pavyzdžiui, maršrutizavimą, apsikeitimą protokolais ir kitas galimybes.
7. *Paslaugų kokybės lygmuo.* Čia teikiamos galimybės, kurių pagalba galima stebėti ir valdyti paslaugos kokybės rodiklius, pavyzdžiui, saugumą, našumą, pasiekiamumą.

Pagrindiniai paslaugų architektūros privalumai:

- Paprastesnis integravimas ir valdymas. Paslaugų architektūroje tinklo paslaugos ir kliento susiejimas pagrįstas paslaugos specifikacija, o ne jos realizacija. Todėl pasikeitus paslaugos realizacijai, išvengiama pokyčių kliento dalyje.
- Trumpesnis sukūrimo laikas. Galimybė kurti naujas paslaugas tiesiog apjungiant esamas paslaugas – tai didelis pranašumas, nes sutrumpinamas programinės įrangos kūrimo ciklas, kadangi dauguma reikalavimų jau būna realizuoti, nereikia projektuoti, kurti ir testuoti jau esamų paslaugų.
- Mažesnė kaina ir didesnės pakartotinio naudojimo galimybės. Kadangi paslaugos laisvai susiejamos, todėl, atsižvelgiant į veiklos poreikius, gali būti prijungiamos įvairios paslaugos.
- Lengvas keitimas ir tobulinimas. Veiklos procesai, kurie susideda iš įvairių paslaugų rinkinio, gali būti lengvai kuriami, tobulinami ir keičiami, kad atitiktų pasikeitusius poreikius.

1.1.4 Tinklo paslaugų apibrėžimo kalba WSDL

Nuo tada, kai komunikavimo protokolai ir pranešimų formatai yra standartizuoti pasaulinio žiniatinklio (angl. *web*) bendruomenėje, galimybė programų bendravimą aprašyti tam tikru struktūriniu būdu tampa vis svarbesnė. Šį poreikį realizuoja WSDL [7] [12] [23], apibrėždama XML gramatiką, kuri tinklo paslaugas aprašo kaip galutinių tinklo taškų, galinčių keistis pranešimais, rinkinį.

WWW Consortium (*W3C*) pateikia tokį WSDL apibrėžimą:

WSDL – tai XML formatas, kurio paskirtis – tinklo paslaugas aprašyti kaip galutinių taškų, apdorojančių dokumentais ar procedūromis grįstą informaciją, rinkinį. Operacijos ir pranešimai yra aprašomi abstrakčiai ir susiejami su konkrečiu tinklo protokolu ir pranešimų formatu tam, kad būtų

nusakytas galutinis taškas. Susiję konkretūs galutiniai taškai jungiami į abstrakčius galutinius taškus (paslaugas) [7]. WSDL galima išplėsti – galutinius taškus ir jų pranešimus galima aprašyti neatsižvelgiant į tai, koks to pranešimo formatas arba koks tinklo protokolas naudojamas bendravimui.

WSDL dokumentas apibrėžia paslaugas kaip tinklo galutinių taškų, ar jungčių rinkinį. WSDL kalboje abstraktus galutinių taškų ir pranešimų aprašymas yra atskirtas nuo konkretaus jų išdėstymo, realizavimo tinkle ar nuo duomenų formatų sąryšių. Tai leidžia atkartoti abstrakčius aprašus (angl. *definitions*): duomenų abstrakčius aprašus – pranešimus ir jungčių tipus – abstrakčius operacijų rinkinius. Konkretaus protokolo ir duomenų formato specifikacijos tam tikram jungties tipui – tai atkartojamas sąryšis (angl. *binding*). Jungtis apibrėžiama tinklo adresą susiejant su atkartojamu sąryšiu. Jungčių rinkinys apibrėžia paslaugą.

WSDL dokumeto struktūra

WSDL dokumentas – tai apibrėžimų rinkinys. Dokumento pagrindas yra apibrėžimo (angl. *definition*) elementas, kurio viduje yra kiti aprašymai. WSDL dokumento gramatika:

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>

    <import namespace="uri" location="uri"/>*

    <wsdl:documentation .... /> ?

    <wsdl:types> ?
        <wsdl:documentation .... />?
        <xsd:schema .... />*
        <-- extensibility element --> *
    </wsdl:types>

    <wsdl:message name="nmtoken"> *
        <wsdl:documentation .... />?
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </wsdl:message>

    <wsdl:portType name="nmtoken">*
        <wsdl:documentation .... />?
        <wsdl:operation name="nmtoken">*
           <wsdl:documentation .... /> ?
           <wsdl:input name="nmtoken"? message="qname">?
               <wsdl:documentation .... /> ?
           </wsdl:input>
           <wsdl:output name="nmtoken"? message="qname">?
               <wsdl:documentation .... /> ?
           </wsdl:output>
           <wsdl:fault name="nmtoken" message="qname"> *
               <wsdl:documentation .... /> ?
           </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="nmtoken" type="qname">*
        <wsdl:documentation .... />?
        <-- extensibility element --> *
        <wsdl:operation name="nmtoken">*
           <wsdl:documentation .... /> ?
           <-- extensibility element --> *
           <wsdl:input> ?
               <wsdl:documentation .... /> ?
               <-- extensibility element -->
           </wsdl:input>
           <wsdl:output> ?
               <wsdl:documentation .... /> ?
               <-- extensibility element --> *
           </wsdl:output>
           <wsdl:fault name="nmtoken"> *
```

```
                <wsdl:documentation .... /> ?
                <-- extensibility element --> *
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="nmtoken"> *
        <wsdl:documentation .... />?
        <wsdl:port name="nmtoken" binding="qname"> *
            <wsdl:documentation .... /> ?
            <-- extensibility element -->
        </wsdl:port>
        <-- extensibility element -->
    </wsdl:service>

    <-- extensibility element --> *

</wsdl:definitions>
```

Pastaba: simboliai „?“ ir „*“ šalia atributų arba elementų rodo kiek tokių elementų arba atributų gali būti: „?” – 0 arba 1, „*“ – 0 ar daugiau.

Taigi, aprašant tinklo paslaugą WSDL dokumente naudojami tokie elementai:

- **Tipai** (angl. *Types*). Jie teikia duomenų tipų aprašus, naudojamus pasikeitimui pranešimais aprašyti.
- **Pranešimai** (angl. *Message*) – abstraktus perduodamų duomenų aprašas. Pranešimas susideda iš loginių dalių, iš kurių kiekviena susieta su tam tikra sistema;
- **Jungties tipas** (angl. *Port Type*) – abstrakti operacijų aibė. Kiekviena operacija kreipiasi į įvedimo ir į išvedimo pranešimą.
- **Sąryšis** (angl. *Binding*) – konkretaus protokolo ir duomenų formato specifikacijos tam tikro porto tipo aprašomiems pranešimams ir operacijoms.
- **Jungtis** (angl. *Port*) – vienas galutinis taškas apibrėžtas kaip sąryšio ir tinklo adreso kombinacija.
- **Paslauga** (angl. *Service*) – susijusių galutinių taškų rinkinys.

Šiuos WSDL dokumento elementus aptarsime plačiau.

Tipai

Šis elementas prideda duomenų tipų aprašus, reikalingus pasikeisti pranešimais. Tam, kad būtų užtikrinta bendradarbiavimo kokybė ir nepriklausomybė nuo platformos, WSDL naudoja XSD ir ją traktuoja kaip esminę tipų sistemą.

```
<definitions .... >
    <types>
        <xsd:schema .... />*
    </types>
</definitions>
```

Jei paslauga turės daug sąryšių tam pačiam pranešimui arba, jeigu bus tik vienas sąryšis, tačiau šio sąryšio tipas neturės plačiai naudojamos tipų sistemos, tuomet koduojant abstrakčius tipus XSD rekomenduojama:

- naudoti elementų (ne atributų) formą,

- neįtraukti elementų ar atributų kuriuos sudėtinga užkoduoti,
- masyvų tipai turėtų išplėsti SOAP v1.1 kodavimo schemoje apibrėžtą masyvo tipą,
- lauko ar parametro, kuris gali įgyti bet kokio tipo reikšmę, aprašymui naudoti xsd:anyType tipą.

Kadangi neprotinga tikėtis, jog abstrakčių tipų aprašymui gali būti naudojama tik viena gramatinė sistema, WSDL leidžia pridėti tipų sistemas, naudojant išplėtimo elementus. Išplėtimo elementas gali būti įterptas po tipų elemento tam, kad identifikuotų naudojamą tipų aprašų sistemą ir kad suteiktų XML konteinerį tipų aprašui. Išplėtimo elemento vaidmuo gali būti prilygintas XML Schema kalbos elementui:

```
<definitions .... >
    <types>
        <-- type-system extensibility element --> *
    </types>
</definitions>
```

Pranešimai

Panešimai susideda iš vienos ar keleto loginių dalių (angl. *parts*). Kiekviena dalis yra susieta su tam tikru tipu iš tipų sistemos, naudojant pranešimų tipo atributą.

Pranešimo sintaksė parodyta žemiau. Pranešimo tipo atributai (jie gali būti kitokie atsižvelgiant į naudojamos sistemos tipą) pažymėti paryškintu šriftu:

```
<definitions .... >
    <message name="nmtoken"> *
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </message>
</definitions>
```

Pranešimo vardo atributas (angl. *Message name attribute*) pranešimui suteikia unikalų vardą, kurio negali įgyti nei vienas kitas to paties WSDL dokumento pranešimas. Dalies vardas (angl. *Part name attribute*) daliai suteikia unikalų vardą, kurio negali įgyti nei viena kita to paties pranešimo dalis.

Jungčių tipai

Jungties tipas (angl. *Port type*) – tai turintis vardą, abstrakčių operacijų ir abstrakčių pranešimų rinkinys.

```
<wsdl:definitions .... >
    <wsdl:portType name="nmtoken">
        <wsdl:operation name="nmtoken" .... /> *
    </wsdl:portType>
</wsdl:definitions>
```

Jungties tipo vardo atributas (angl. *Port type name attribute*) Jungties tipui suteikia unikalų vardą, kurio negali įgyti nei vienas kitas to paties WSDL dokumento jungties tipas.

WSDL turi keturis perdavimo primityvus, kuriuos galutinis taškas gali palaikyti:

- **Vienakryptis** (angl. *One-way*). Galutinis taškas gauna pranešimus.

- **Užklausa-atsakymas** (angl. *Request-response*). Galutinis taškas gauna pranešimą ir siunčia bendravimo (koreliavimo) pranešimą.
- **Prašymas-atsakymas** (angl. *Solicit-response*). Galutinis taškas siunčia pranešimą ir gauna bendravimo (koreliavimo) pranešimą.
- **Pranešimas** (angl. *Notification*). Galutinis taškas siunčia pranešimą.

WSDL šiuos primityvus traktuoja kaip operacijas. Užklausa-atsakymas ar prašymas-atsakymas primityvai gali būti abstrakčiai modeliuojami naudojant du vienakrypčius panešimus. Šiuos pranešimus naudinga modeliuoti kaip primityvius operacijų tipus, kadangi:

- jie yra labai bendri,
- šios sekos gali būti koreliuojamos neįvedant sudėtingos srautų informacijos,
- kai kurie galutiniai taškai gali tik gauti pranešimus, jei šie pranešimai yra sinchroninės užklausos rezultatai,
- iš šių primityvų paprastas srautas gali būti perduotas algoritminiu būdu, kai tik srauto aprašo prireikia galutiniam taškui.

Sąryšiai

Sąryšis nustato pranešimo formatą ir tam tikro jungties tipo aprašomų pranešimų bei operacijų protokolo detales. Duotas jungties tipas gali turėti bet kokį sąryšių skaičių. Sąryšio gramatika atrodytu taip:

```
<wsdl:definitions .... >
    <wsdl:binding name="nmtoken" type="qname"> *
        <-- extensibility element (1) --> *
        <wsdl:operation name="nmtoken"> *
           <-- extensibility element (2) --> *
           <wsdl:input name="nmtoken"? > ?
               <-- extensibility element (3) -->
           </wsdl:input>
           <wsdl:output name="nmtoken"? > ?
               <-- extensibility element (4) --> *
           </wsdl:output>
           <wsdl:fault name="nmtoken"> *
               <-- extensibility element (5) --> *
           </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

Sąryšio vardo atributas (angl. *binding name attribute*) sąryšiui suteikia unikalų vardą, kurio negali įgyti nei vienas kitas to paties WSDL dokumento sąryšis.

Sąryšio išplėtimo elementai naudojami tam, kad būtų galima specifikuoti konkrečią gramatiką įvedimui (3), išvedimui (4) ir klaidos pranešimui (5). Informacija sąryšio operacijai (2) ir pačiam sąryšiui (1) taip pat gali būti specifikuojama.

Operacijos elementas sąryšyje specifikuoja sąryšio informaciją, skirtą operacijai, pavadintai tuo pačiu vardu ir esančiai sąryšio jungties tipe. Kadangi operacijų vardai nereikalauja unikalumo, tai vardo atributo sąryšio operacijos elemente nepakanka, kad reikalinga operacija būtų identifikuota. Šiuo

atveju, reikiama operacija yra identifikuojama pagal atributų, atitinkančių wsdl:input ir wsdl:output elementus, vardus.

Sąryšis turi specifikuoti tik vieną protokolą, ir neturi specifikuoti adreso informacijos.

Jungtys

Jungtis aprašo atskirą sąveikos tašką ir nurodo jam vieną adresą.

```
<wsdl:definitions .... >
    <wsdl:service .... > *
        <wsdl:port name="nmtoken" binding="qname"> *
           <-- extensibility element (1) -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Jungties vardo atributas (angl. *port name attribute*) jungčiai suteikia unikalų vardą, kurio negali įgyti nei viena kita to paties WSDL dokumento jungtis. Sąryšio išplėtimo elementai (1) naudojami informacijos apie jungties adresą specifikacijai. Jungtis turi specifikuoti tik vieną adresą. Ji neturi aprašyti jokios sąryšio informacijos, tik informaciją apie adresą.

Paslaugos

Paslauga jungia susijusių jungčių aibę.

```
<wsdl:definitions .... >
    <wsdl:service name="nmtoken"> *
        <wsdl:port .... />*
    </wsdl:service>
</wsdl:definitions>
```

Paslaugos vardo atributas (angl. *service name attribute*) paslaugai suteikia unikalų vardą, kurio negali įgyti nei viena kita to paties WSDL dokumento paslauga.

Paslaugos jungtys yra susietos tokiu būdu:

- nei viena jungtis nebendrauja viena su kita;
- jei paslauga turi keletą jungčių, kurios visos naudoja tą patį jungties tipą, tačiau skirtingus sąryšius ar adresus, tai tos jungtys yra alternatyvios. Kiekviena jungtis pateikia semantiškai ekvivalenčią elgseną. Tai leidžia WSDL dokumento vartotojui pasirinkti tam tikrą jungtį;
- išbandant paslaugos jungtis, galima nustatyti jungčių tipus. Tai leidžia WSDL dokumento vartotojui nuspręsti ar jis nori toliau bendrauti su tam tikra paslauga. Tai naudinga, jei yra realizuoti ryšiai tarp jungties tipo operacijų ir tų operacijų, kurios turėtų atlikti tam tikrus uždavinius.

WSDL išplėtimai

Kaip jau buvo minėta ankščiau, WSDL galima išplėsti – leisti aprašyti galutinius taškus ir jų pranešimus neatsižvelgiant į tai, koks tų pranešimų formatas ar kokie tinklo protokolai naudojami

bendravimui. Trumpai aptarsime, kaip WSDL naudojamas sąjungoje su SOAP 1.1, HTTP GET/POST ir MIME.

WSDL kalboje yra specialūs sąryšiai skirti SOAP 1.1 galutiniams taškams, kurie palaiko tokią protokolo informacijos specifikaciją:

- požymį, kad sąryšis yra apribotas SOAP protokolui;
- būdą, kaip specifikuoti adresą galutinam SOAP taškui;
- URI skirtą SOAPAction HTTP antraštei, kuri savo ruožtu skirta SOAP HTTP sąryšiui;
- apibrėžimų sąrašą antraštėms, kurios yra perduodamos kaip SOAP apvalkalo dalis.

Šis sąryšio gramatikos aprašymas nėra išsami specifikacija, kadangi SOAP sąryšių aibė vis didėja.

WSDL kalba palaiko sąryšį skirtą HTTP GET ir POST žodžiams. Naudojant šį sąryšį galima aprašyti interneto naršyklės ir tinklalapio bendravimą. Gali būti specifikuojama tokia protokolo informacija:

- požymis, kad sąryšis naudoja HTTP GET ir POST,
- jungties adresas,
- atitinkamas adresas kiekvienai operacijai.

WSDL taip pat turi galimybę susieti abstrakčius tipus ir konkrečius pranešimus, naudojant tam tikrą MIME formatą. Sąryšiai yra nustatyti tokiems MIME tipams:

- Multipart/related,
- Text/xml,
- application/x-www-form-urlencoded (formatas skirtas formos patvirtinimui HTML),
- kiti (specifikuojant MIME tipo eilute).

Apibrėžtų MIME tipų aibė yra didelė ir besivystanti, taigi WSDL neturi tikslo išsamiai aprašyti XML gramatiką kiekvienam MIME tipui. Esant poreikiui, gali būti apibrėžta papildoma gramatika papildomo MIME tipo aprašui.

1.2 Esami projektavimo metodai

1.2.1 Objektinė analizė ir projektavimas

Objektinės analizės ir projektavimo metodas (angl. *Object-Oriented Analysis and Design – OOAD*) [10] [24] – tai probleminė sritis ir loginis sprendimas objektų (daiktų, konceptų ar esybių) požiūriu. Objektiškumo tikslas projektavimo lygyje – įgalinti greitą ir efektyvų lanksčios ir išplečiamos sistemos projektavimą, kūrimą, vykdymą. Objektai – tai programinės įrangos konstrukcijos, kurios elgiasi lyg šiais objektais modeliuojamos realaus pasaulio esybės. Objektai identifikuojami ir aprašomi nagrinėjant probleminę sritį, o objektiniame projektavime jie pavirsta į

loginius programinės įrangos objektus, kurie bus realizuoti objektine programavimo kalba. Objektai charakterizuojami operacijų rinkiniu ir būsena, kuri įsimena šių operacijų poveikį.

Pagrindiniai objektinės paradigmos principai yra:

- inkapsuliacija (angl. *Encapsulation*),
- informacijos paslėpimas (angl. *Information hiding*),
- klasės ir jų egzemplioriai (angl. *Classes and instances*),
- sujungimai ir paveldėjimas (angl. *Associations and inheritance*),
- keitimasis pranešimais (angl. *Messaging*),
- polimorfizmas (angl. *Polymorphism*).

Objektinė paradigma palaiko pilną sistemos analizės, projektavimo ir kūrimo ciklą:

- Objektinė analizė ir projektavimas bando surasti optimalų objektų rinkinį ir natūraliausią klasių hierarchiją, kurios pagalba būtų galima šiuos objektus realizuoti.
- Objektinis kūrimas remiasi palaipsniu kūrimu, kai vienam laiko intervale sukuriamas vienas scenarijus ar panaudojimo atvejis.
- Objektinio vykdymo aplinka (angl. *runtime environment*) (pavyzdžiui, *Java™ Virtual Machine*) vykdo kodą, teikia sistemos paslaugas (pavyzdžiui, šiukšlių surinkimas (nebereikalingų resursų pašalinimas)), kartu su karkasais (pavyzdžiui, *J2EE*) suteikia mechanizmą, leidžiantį bendrauti skirtinguose serveriuose patalpintiems objektams.

Objektinėje analizėje ir projektavime, norint supaprastinti sudėtingų veiklos scenarijų analizavimą, tam tikri objekto aspektai gali būti inkapsuliuoti. Tam, kad sumažinti modelio sudėtingumą, nagrinėjamos tik svarbios ir būtiniausios objektų charakteristikos, o ne tokios svarbios gali būti neįtraukiamos į objektų aprašus.

1.2.2 Komponentinis projektavimas

Komponentinis projektavimas (angl. *Component-based design*) [10] – tai ne nauja technologija. Ji išsivystė iš objektinės paradigmos. Dar ankstyvoje objektinės analizės ir projektavimo gyvavimo stadijoje, kai kurie stambūs objektai buvo skirti pakartotiniam panaudojimui, tačiau tokie objektai turėjo per žemą suskaidymo lygį ir tuo metu dar nebuvo jokių pakartotinio naudojimo efektyvumo standartų. Stambūs komponentai tapo tarsi programinės įrangos kūrimo ir integravimo tikslu. Tokie stambūs komponentai teikia tam tikrą funkcionalumą, kurį realizuoja smulkesnių komponentų rinkinys.

Kai tik organizacija pasiekia aukštesnį architektūrinį lygį, pagrįstą atskirais funkciniais komponentais, veiklą palaikančios programos gali būti apibendrintos į dar stambesnius komponentus. Komponentai gali būti traktuojami kaip tam tikri mechanizmai, kurių pagalba skirtingos paslaugos „supakuojamos“ ir valdomos. Komponentai gali suderinti įvairias technologijas ir jas naudoti kartu,

pavyzdžiui, stambūs organizaciniai komponentai, realizuojantys veiklos lygio panaudojimo atvejus. Jie gali būti sukurti naudojant naujesnes objektinės programinės įrangos kūrimo technologijas kartu su pasenusiomis technologijomis.

1.2.3 Veiklos procesų modeliavimas

Veiklos procesų modeliavimas (angl. *Business Process Modeling – BPM*) [24] turi daug ir įvairių stilių, notacijų ir rinkinių. Pavyzdžiui, UML naudojimas programos srityje gali būti išplėstas panaudojimui veiklos procesų modeliavimo lygyje. Dar viena dažnai naudojama technologija – tai įvykiais pagrįstų procesų grandinių (angl. *event-driven process chains*), vaizduojančių konceptualius procesų srautus, aprašymas. Ši technologija naudoja ne UML, o kitą notaciją.

Nauja tendencija – apibrėžti standartinio vykdomų srautų modelių vaizdavimo būdą, pavyzdžiui, veiklos procesų vykdymo kalba, skirta paslaugoms (angl. *Business Process Execution Language for Web Services – BPEL)).* Ši kalba išplečia procesų modelių taikymą nuo analizės iki realizacijos. Tokie vykdomieji modeliai iškelia daug naujų klausimų:

- Kurie aspektai turėtų būti aprašomi BPEL, o kurie WSDL? Kur yra riba tarp procesų modelių ir tradicinių programavimo modelių?
- Kaip kai kurie aspektai (pavyzdžiui, nefunkciniai reikalavimai, paslaugos kokybės charakteristikos) gali būti įtraukti į modelius?
- Kokią veiklos logikos dalį vykdys BPEL variklius išplečianti programavimo kalba, o kokią dalį – tradicinė programavimo kalba (Java, php ir kt.)?
- Kokie vykdomų procesų modeliai yra kokybiški ir kaip geriausia tą kokybę pasiekti?
- Kas turi atlikti PBEL srautų modeliavimą – veiklos ekspertas (analitikas) ar sistemos architektas?

Visi veiklos procesų modeliavimo būdai gali būti panaudoti kaip paslaugų analizės ir projektavimo pradžia, tačiau jie turi būti patobulinti, kad būtų galima nustatyti galimas paslaugas (paslaugas „kandidates“) ir joms priskirti operacijas iš procesų modelių. Be to, procesų modeliavimas, paslaugų analizės ir projektavimo procese, turi būti sinchronizuotas su projektavimo lygyje vykdomu panaudojimo atvejų modeliavimu. Taip pat turi būti surasti atsakymai į panašius kaip BPEL keliamus klausimus.

1.2.4 Organizacijų taikomųjų programų integravimas

Verslo programų ir IT infrastruktūros perkėlimas į paslaugų architektūrą gali būti labai sudėtingas žingsnis, paliečiantis daugelį veiklos ir organizacijos dalių. Todėl, siekiant palaikyti nuoseklumą tarp atskirų sprendimų, turi būti taikomi organizacijų architektūros karkasai ir architektūriniai nurodymai (pavyzdžiui, *The Open Group Architecture Framework (TOGAF)* [24]).

Dauguma egzistuojančių organizacijų architektūros karkasų ir metodų turi tam tikrų apribojimų vienoje ar kitoje srityje. Pavyzdžiui, jokie veiklos lygio procesai ar paslaugų vaizdai negali būti gauti, kadangi svarbiausias dalykas – kaip žemiausio lygio blokai, atvaizduojantys techninius įrenginius, apjungiami makrolygyje. Paslaugų architektūros kontekste toks mąstymo būdas netinka. Čia dėmesys sutelkiamas į paslaugas atitinkančius loginius blokus ir koncentruojamasi ties sąsajų apibrėžimu ir paslaugų lygio susitarimais (angl. *Service Level Agreements – SLAs*) tarp paslaugų. Be to, dauguma verslo lygio (angl. *enterprise-level*) architektūrinių nurodymų bei karkasų yra gana bendri ir nepaliečia projektavimo srities. Tokia aukšto lygio architektūra nepadeda sistemos projektuotojams ir programuotojams priimti su sistemos realizacija susijusių sprendimų.

Paslaugų analizė ir projektavimas turėtų padėti paslaugų architektūros projektuotojui susidaryti bendrą veiklos lygio paslaugos įspūdį. Šiuolaikiniai organizacijų architektūros karkasai to padaryti negali.

1.2.5 Paslaugų projektavimas

Literatūros šaltiniuose pateikiamas paslaugos apibrėžimas:

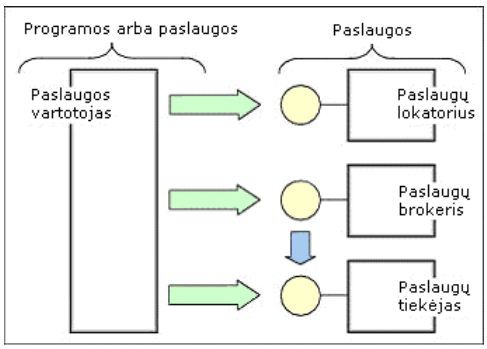
„Paslauga - tai vykdoma kodo dalis, kuri tam tikras, susijusias paslaugas inkapsuliuoja juodos dėžės principu. Tos paslaugos gali būti pasiektos per paskelbtą sąsają. Turi būti galimybė komponentą prijungti prie kitų komponentų“ [10].

Paslauga realizuojama kaip stambi, atrandama programinės įrangos esybė, kuri bendrauja su kitomis programomis ir paslaugomis, naudodama laisvai susietą, pranešimais pagrįstą bendravimo modelį.

Pagrindinės paslaugų projektavimo (angl. *Service-Oriented Design*) sąvokos:

- Paslauga – loginė esybė, turinti vieną ar kelias paskelbtas sąsajas;
- Paslaugos tiekėjas – programinės įrangos esybė realizuojanti paslaugos specifikaciją;
- Paslaugos vartotojas – programinės įrangos esybė, kuri kviečia paslaugos tiekėją.
- Paslaugos lokatorius – specialus paslaugų teikėjas, kuris veikia kaip registras ir leidžia peržiūrėti paslaugų teikėjų sąsajas ir jų buvimo vietas.
- Paslaugų tarpininkas (brokeris) – specialus paslaugų teikėjas, kuris gali siųsti paslaugų užklausas vienam ar daugiau paslaugų teikėjų.

Sąsajos tarp šių sąvokų parodytos 1.12 paveikslėlyje.



1.12 pav. Paslaugų projektavimo sąvokos

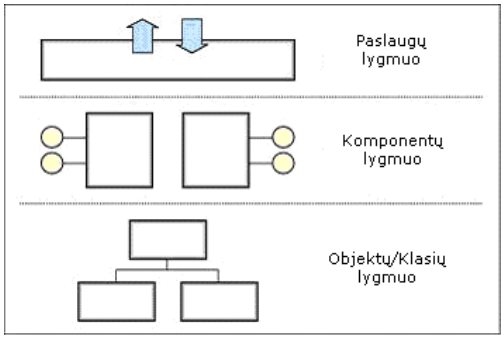
1.2.6 Sąsajų projektavimas

Sąsaja – tai labai svarbi sąvoka tiek komponentų, tiek paslaugų projektavime. Pagrindiniai sąsajų projektavimo (angl. *Interface-based design*) [10] terminai:

- Sąsaja – apibrėžia viešai pasiekiamų metodų rinkinį. Sąsaja nusako kaip paslaugos tiekėjas ir paslaugos vartotojas galėtų bendrauti.
- Paskelbta sąsaja – tai unikaliai identifikuojama sąsaja, pasiekiama per specialų registrą.
- Vieša sąsaja – tai sąsaja, kurią vartotojai gali naudoti, tačiau ji nėra paskelbta.
- Dviguba sąsaja – sąsajos kuriamos poromis ir priklausomos viena nuo kitos.

1.2.7 Sluoksninis sistemų projektavimas

Kaip minėta ankščiau, objektinės technologijos ir kalbos tinka komponentų realizacijai. Nors komponentai yra geriausias būdas paslaugų realizacijai, tačiau gera komponentinė programa nebūtinai bus gera paslaugų programa. Skirtumas tarp komponentinio ir paslaugų modelio – paslaugų modelis turi dar vieną architektūrinį lygį. 1.13 paveiksle pavaizduoti sistemos architektūriniai lygmenys [10].



1.13 pav. Sistemos realizacijos lygmenys: paslaugų, komponentų, objektų

1.2.8 Reikalavimai tinklo paslaugų projektavimui

Išnagrinėjus pagrindinius projektavimo metodus, galima sudaryti šių metodų analizę apibendrinančią lentelę (1.1 lentelė), kuri padės suformuluoti reikalavimus paslaugų projektavimo ir analizės metodui (1.2 lentelė).

1.1 lentelė Projektavimo metodų analizė

Metodas	Savybės	Kodėl netinka paslaugų architektūrai?
Objektinė analizė ir projektavimas	• Įgalina greitą ir efektyvų lanksčios bei išplečiamos sistemos projektavimą, kūrimą, vykdymą; • Nagrinėja žemiausio lygio abstrakcijas (klases, objektus); • Paremtas panaudojimo atvejų sudarymu; • Pagrindiniai principai: o Inkapsuliacija, o informacijos paslėpimas, o klasės ir jų egzemplioriai, o sujungimai ir paveldėjimas, o keitimasis pranešimais, o polimorfizmas.	• Programos projektas susideda iš daugelio panaudojimo atvejų modelių, todėl bendras veiklos procesų modelis tampa painus ir neaiškus; • Panaudojimo atvejų modeliai ne visada būna sinchronizuoti su veiklos procesų modeliais; iš to paties panaudojimo atvejų rinkinio galima suformuoti daug procesų modelių, tačiau objektinis projektavimas to nenagrinėja; • Išskaidymas (klasėmis) pernelyg smulkus paslaugų modeliavimui; • Stiprūs sąryšiai (paveldėjimas) tvirtai susieja modelio dalis tarpusavyje.
Organizacijos architektūra	• Leidžia integruoti atskiras sistemos dalis ir sprendimus.	• Leidžia aprašyti informacinės sistemos architektūrą ir atskirų dalių ryšius, tačiau nenagrinėja veiklos procesų ir modeliavimo.
Veiklos procesų modeliai	• Teikia pilną funkcinių vienetų vaizdą.	• Nenagrinėja architektūros ir realizacijos; • Veiklos procesų modeliai ne visada sinchronizuoti su panaudojimo atvejų modeliais; • Veiklos procesų modeliai dažniausiai neintegruoti su duomenų modeliais.

1.2 lentelė Paslaugų analizei ir projektavimui keliami reikalavimai

Reikalavimų tipas	Reikalavimų aprašymas
Bendri reikalavimai	• Procesas ir notacija turi būti formaliai (ar bent jau pusiau formaliai) apibrėžti, lygiai kaip ir bet kurioje kitoje projektavimo metodologijoje. • Paslaugų analizė ir projektavimas neturėtų prasidėti nuo „nulio“ – visų pirma turi būti parinkti ir apjungti objektiniai, komponentiniai, veiklos procesų modeliavimo ir organizacijų architektūros elementai, o tik tada, jei reikia galima pridėti naujus elementus. • Šis metodas turi palengvinti projektavimą „nuo pradžios iki pabaigos“, taip pat, turi turėti prieinamus projektavimo įrankius. • Metodas turi būti apibrėžiamas sintakse, semantika ir elgsena. Tai reikalinga „ad hoc“ kompozicijai, semantiniam tarpininkavimui ir paslaugos suradimui vykdymo metu.
Objektiniai reikalavimai	• Vidiniam paslaugų realizacijos projektavimui turi būti naudojamos objektinės esybės - klasės ir objektai. • Paslaugų analizės ir projektavimo metodai neturi priklausyti nuo panaudojimo atvejų. Panaudojimo atvejai „nusileidžia“ į žemesnį projektavimo lygmenį.
Komponentiniai reikalavimai	• Susijusios sistemos dalys turi būti jungiamos į stambesnius komponentus.
Organizacijų architektūros reikalavimai	• Atskiri komponentai integruojami į bendrą sistemą - paslaugą naudojant organizacijų architektūros integravimo metodus.
Veiklos procesų modeliavimo reikalavimai	• Paslaugų tarpusavio bendradarbiavimui (choreografijai) modeliuoti turi būti naudojami įvykiais pagrįsti procesų modeliai.

1.3 Projektavimo bei realizacijos priemonių pasirinkimas

Pasirenkant projektavimo įrankį, pagrindiniai kriterijai buvo galimybė kurti WSDL diagramas, atlikti WSDL kodo tiesioginę ir atvirkštinę inžineriją bei RUP (angl. *Rational Unified Process*) [21] palaikymas. Yra keletas įrankių, leidžiančių generuoti WSDL kodą (pavyzdžiui, BindStudio), tačiau įrankių, turinčių galimybę kurti WSDL diagramas bei atlikti WSDL kodo tiesioginę ir atvirkštinę inžineriją, yra nedaug [20].

Darbe pateikiamoms diagramoms braižyti pasirinktas paketas MagicDraw [13]. Šį paketą galima naudoti projektuojant tinklo paslaugas taikant „UML – WSDL – Realizacija“ metodą. Paketas skirtas verslo analitikams, programinės įrangos analitikams, programuotojams, dokumentuotojams. MagicDraw palaiko kodo tiesioginę ir atvirkštinę inžineriją J2EE, C#, C++, CORBA IDL, .NET, XML Schema, WSDL programavimo kalboms.

MagicDraw palaiko UML 2.0 notaciją. Su MagicDraw galima sukurti UML modelį, iš jo sugeneruoti išeities kodą, jį papildyti, atlikti atvirkštinę kodo inžineriją. Jokia informacija neprarandama, atliekant kodo inžineriją (tiesioginę ar atvirkštinę).

Su MagicDraw, galima suprojektuoti duomenų bazės schemą, naudojant klasių diagramą ir iš jos sugeneruoti DDL kodą. Taip pat galima atlikti duomenų bazės atvirkštinę kodo inžineriją.

MagicDraw veikia daugelyje operacinių sistemų, pavyzdžiui, Windows98/ME/NT/2000/XP, Solaris, OS/2, Linux, HP-UX, AIX, MacOS(X) ir visur kitur, kur palaikoma Java 1.4.

Eksperimentinei sistemai realizuoti pasirinkta PHP [2] [4] programavimo kalba. Pagrindiniai pasirinkimo kriterijai:

- Kaina – skirtingai nei daugelis technologijų, PHP yra nemokama;
- MySQL – PHP puikiai suderintas su MySQL [15]. Galima lengvai prisijungti prie duomenų bazės ir redaguoti jos duomenis;
- Platformų palaikymas - PHP veikia daugelyje operacinių sistemų (pavyzdžiui, Linux, Windows) ir serverių;
- Tinklo paslaugų realizacijai dažniausiai naudojamos .NET arba Java technologijos, todėl buvo įdomu išbandyti kaip realizuoti paslaugas PHP technologija, kuri retai naudojama paslaugų realizacijai, tačiau plačiai naudojama kuriant įvairias sistemas;
- Naujai sistemai realizuoti pasirinkta PHP programavimo kalba, dėl jos galimybių realizuoti tinklo paslaugas, dėl paprastų programavimo konstrukcijų.

Pasirinkta duomenų bazės valdymo sistema – MySQL. Pasirinkimą nulėmė tai, kad nevertėjo naudoti didelę DBVS, be to MySQL gerai suderinama su PHP.

Ypač svarbus pasirinkimo kriterijus buvo programinės įrangos kaina – tiek PHP, tiek MySQL yra nemokamos sistemos.

1.4 Analizės išvados

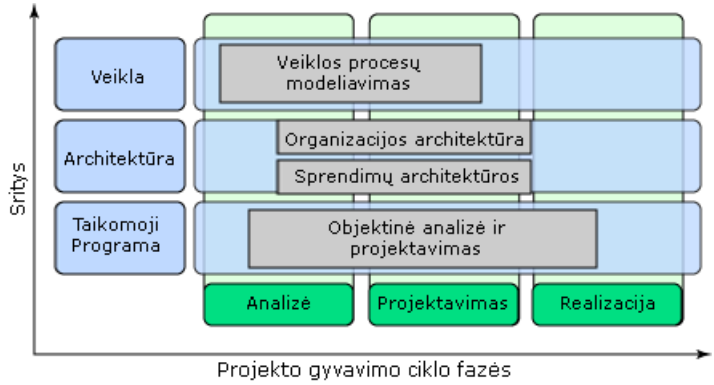
1. Analizės dalyje išanalizuotos pagrindinės su darbo tikslais susijusios sąvokos: tinklo paslauga ir tinklo paslaugų architektūra. Pateiktas tinklo paslaugos apibrėžimas, jos evoliucija ir teikiama nauda. Apžvelgti technologijų, iš kurių išsivystė tinklo paslaugų technologija, lygmenys (tinklas, transportavimas, pakavimas, aprašymas, atradimas).
2. Išnagrinėta tinklo paslaugų architektūra, kurios pagrindinės dalys yra pranešimų, paslaugų, resursų ir strategijos modeliai. Tinklo paslaugų architektūrą apibendrina paslaugų architektūra. Šios architektūros pagrindinės konceptualiojo modelio esybės yra paslauga, paslaugų teikėjas, paslaugų vartotojas ir paslaugų katalogas.
3. Pagrindinės paslaugų architektūros savybės: tiksliai apibrėžtos sąsajos, nepriklausomas paslaugų modelis, paslaugų skaidymas, paslaugų saugumas. Jos privalumai: paprastesnis integravimas ir valdymas, trumpesnis sukūrimo laikas, mažesnė kaina ir didesnės pakartotinio naudojimo galimybės, lengvas keitimas ir tobulinimas.
4. Išnagrinėti tinklo paslaugų realizacijoje naudojami standartai: UDDI, WSDL, SOAP, HTTP, TCP/IP. Didelę reikšmę turi tinklo paslaugų aprašymo kalba WSDL, kuri aprašo paslaugas ir jas susieja su konkrečiais tiekimo taškais. Pagrindiniai WSDL dokumento elementai yra tipas, pranešimas, jungtis, sąryšis, paslauga.
5. Išanalizuoti esami projektavimo metodai: objektinė analizė ir projektavimas, organizacijų architektūros karkasai, veiklos procesų modeliavimas, komponentinis projektavimas, sąsajų projektavimas bei sluoksninis sistemų projektavimas. Kiekvienas iš šių metodų apima dalį reikalavimų keliamų paslaugoms projektuoti. Todėl, norint sukurti paslaugų projektavimo metodą, reikia sujungti šių metodų savybes: objektinių metodų - išskaidymą klasėmis ir objektais, veiklos procesų modelių – įvykiais pagrįstus procesų modelius bei organizacijų architektūros karkasų teikiamas atskirų sistemos dalių ir sprendimų integravimo galimybes.
6. Metodui įgyvendinti pasirinktas MagicDraw paketas, kadangi jis turi UML išplėtimus WSDL diagramoms projektuoti ir galimybes generuoti WSDL dokumentus. Paslaugai realizuoti pasirinkta PHP technologija, kadangi buvo įdomu išbandyti kaip realizuoti paslaugas naudojant šią technologiją, kuri retai pasirenkama tinklo paslaugoms realizuoti, tačiau plačiai naudojama kuriant įvairias sistemas.

2 TINKLO PASLAUGŲ PROJEKTAVIMO METODAS

2.1 Projektavimo etapai ir artefaktai

Egzistuojantys projektavimo procesai ir notacijos (objektinė analizė ir projektavimas, organizacijų architektūros karkasai bei veiklos procesų modeliavimas) „padengia“ tik dalį paslaugų architektūros reikalavimų. Kadangi, paslaugų architektūros požiūris stiprina tvirtai nusistovėjusius pagrindinius programinės įrangos principus (pavyzdžiui, informacijos paslėpimas, moduliškumas bei sąvokų atskyrimas), jis taip pat prideda papildomas sąvokas (pasaugų choreografija, paslaugų katalogai, tarpinės paslaugų magistralės šablonas), reikalaujančias papildomo dėmesio projektuojant.

2.1 paveiksle pavaizduotos objektinės analizės ir projektavimo, organizacijų architektūros karkasų bei veiklos procesų modeliavimo metodų taikymo sritys. Horizontaliojoje ašyje vaizduojamos projekto gyvavimo ciklo fazės, vertikaliojoje – skirtingi sričių abstrakcijos lygiai, kur paprastai ir vykdomas modeliavimas.

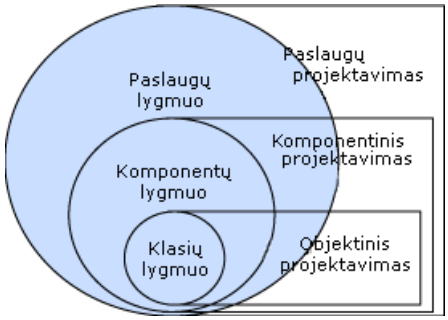


2.1 pav. Objektinės analizės ir projektavimo, organizacijų architektūros bei veiklos procesų modeliavimo išsidėstymas

Objektinės analizės ir projektavimo metodologija padeda identifikuoti paslaugų architektūrą. Ši metodologija nagrinėja žemiausio lygio abstrakcijas, pavyzdžiui, klases ar atskirus objektų egzempliorius. Toks išskaidymas (klasėmis) pernelyg smulkus paslaugų modeliavimui. Stiprūs sąryšiai (paveldėjimas) tvirtai susieja modelio dalis tarpusavyje. Tuo tarpu projektuojant paslaugas ir naudojant laisvą susiejimą, siekiama lankstumo ir judrumo.

Daugelį metų architektūros lygmeniui nagrinėti buvo įprasta naudoti objektinę analizę ir projektavimą bei UML. Taikant šią projektavimo metodiką, kiekvienai probleminei sričiai kuriami savarankiški panaudojimo atvejų (angl. *Use Case*) modeliai, todėl programos projektas, visas verslo/veiklos modelis tampa painus ir neaiškus. Be to, dėl įvairių priežasčių panaudojimo atvejų modeliai ne visada būna sinchronizuoti su veiklos procesų modeliais. Paslaugų analizės ir projektavimo metode panaudojimo atvejai vis dar bus svarbūs, tačiau šis metodas turėtų būti pagrįstas procesais, o ne panaudojimo atvejais.

Objektinį projektavimą sunku tiesiogiai pritaikyti paslaugų architektūrai kurti. Tačiau jis naudingas paslaugų struktūrai (klasėms ir komponentams) modeliuoti. Objektinio projektavimo vietą paslaugų projektavimo procese iliustruoja 2.2 paveikslas.



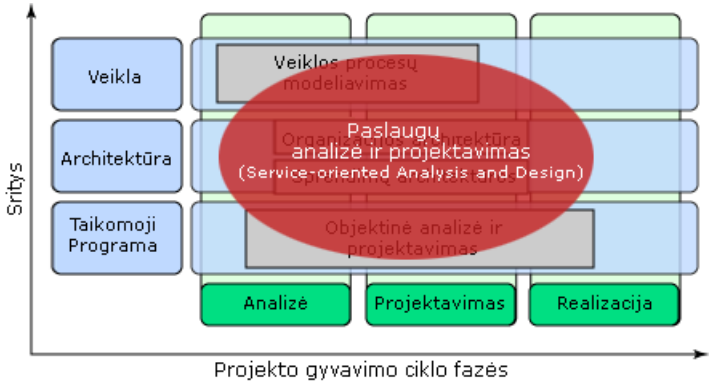
2.2 pav. Projektavimo lygmenys

Organizacijų architektūros integravimo metodai (pavyzdžiui, *Treasury Enterprise Architecture Framework – TEAF*, *Feature-Oriented Domain Analysis – FODA* [24]) naudojami siekiant integruoti skirtingai realizuotas sistemos dalis, tačiau jie nepaliečia projektavimo srities.

Nors veiklos procesų modeliavimo metodai (angl. *Business Process Management Initiative – BPMI*) [24] teikia pilną funkcinių vienetų vaizdą, tačiau jie nenagrinėja architektūrinės ir realizacijos srities.

Nei vienas, iš egzistuojančių modelių nenurodo kaip pritaikyti esamas programas paslaugų architektūrai. Egzistuojančios sistemos paprastai turi didelį kiekį kritiškai svarbių duomenų ir saugo veiklos logiką, todėl jos negali būti taip paprastai pakeistos. Taigi, analizė „iš viršaus į apačią“ taip pat turi būti valdoma taip, kad ją naudojant būtų galima pakeisti esamas strategijas.

Sudėtinis paslaugų architektūros projektavimas sudarytas iš reikiamų objektinės analizės ir projektavimo, organizacijų architektūros bei veiklos procesų modeliavimo elementų, juos jungiant su naujais elementais. 2.3 paveikslas vaizduoja paslaugų analizės ir projektavimo (angl. *Service-Oriented Analysis and Design – SOAD*) [10] [24] elementus bei technologijas.



2.3 pav. Paslaugų analizė ir projektavimas bei jo sudedamosios dalys

Projektuojant paslaugas, gali būti nustatyti pagrindiniai paslaugų kokybės rodikliai:

- Gerai sukurtos paslaugos veiklai duoda lankstumą ir judrumą. Kadangi, naudojamas laisvas susiejimas, inkapsuliacija ir informacijos paslėpimas, todėl paslaugas lengviau perkonfigūruoti ir pakartotinai panaudoti.
- Gerai suprojektuotos paslaugos yra naudingos ir pritaikomos ne tik verslo sistemoms. Sąryšių tarp paslaugų skaičius kiek įmanoma sumažinamas ir tokiu būdu gaunamas sistemos aiškumas.
- Paslaugos abstrakcijos yra susijusios viena su kita, išbaigtos ir nuoseklios. Pavyzdžiui, kuriant paslaugas ir jų operacijų aprašus, reikia remtis „sukurk, skaityk, atnaujink, sunaikink, ieškok“ (angl. Create, Read, Update, Delete, and Search – CRUDS) metafora.
- Dažnai daromos prielaidos, kad paslaugos yra nepriklausomos.
- Paslaugų pavadinimai turi būti suprantami srities ekspertams ir neturi reikalauti tam tikrų gilesnių techninių žinių.
- Paslaugų architektūroje visos paslaugos turi atitikti tą pačią projektavimo filosofiją (išreikštą šablonų pagalba) ir tuos pačius sąveikos šablonus.

Nagrinėjant paslaugų architektūrą, priešingai nei objektiniame projektavime ir analizėje, reikia priimti dvi projektavimo perspektyvas – paslaugos vartotojo ir paslaugos tiekėjo (paslaugos agentas šiuo atveju nenagrinėjamas) (2.4 paveikslas).

Paslaugų architektūrai skirta projektavimo strategija prasideda ne „iš viršaus į apačią“ nagrinėjimo požiūriu, kaip dažnai pasitaiko su paslaugomis susijusiuose projektavimo požiūriuose ar metoduose.

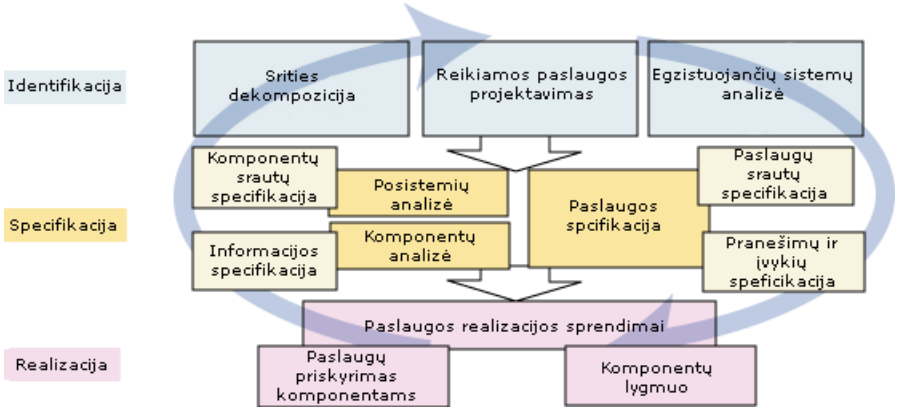
Egzistuoja visa eilė svarbių sprendimų ir veiksmų, įtakojančių ne tik integracijos, bet tuo pačiu verslo ir taikomųjų programų architektūras.

Rolė	Rolės veiksmai				
Vartotojo aspektas	Paslaugos identifikacija	Paslaugos kategorizacija	Paslaugos panaudojimo sprendimai	Choreografija arba kompozicija	Paslaugos kokybė
Tiekėjo aspektas	Komponentų identifikacija	Komponentų specifikacija	Paslaugos realizacija	Paslaugos valdymas	Panaudojimo standartai
	Paslaugų išdėstymas komponentuose	SOA lygių pritaikymas	Techninių prototipų realizacija	Produkto pasirinkimas	Architektūriniai sprendimai (srautai, būsenos ir t.t.)

2.4 pav. Paslaugų modeliavimo veiksmai

2.4 paveikslas vaizduoja kiekvieno veikėjo (paslaugos vartotojo ir paslaugos tiekėjo) veiksmus. Svarbu pastebėti, kad paslaugos tiekėjas gali atlikti ir paslaugos vartotojo veiksmus (pavyzdžiui, paslaugos tiekėją gali dominti paslaugos identifikavimas, skirstymas pagal kategorijas ir kt.). Šiame

paveiksle parodyti veiksmai gali būti pavaizduoti kaip srautas, paslaugų projektavimo ir architektūros modelyje (2.5 paveikslas).



2.5 pav. Paslaugų projektavimo ir architektūros modelis.

2.5 paveiksle parodytas modelis susideda iš trijų pagrindinių žingsnių: identifikacijos, specifikacijos ir paslaugų, komponentų bei srautų (paprastai choreografijos) realizacijos. Kiekvienam iš pagrindinių paslaugų projektavimo ir architektūros žingsnių galima išskirti du elementų tipus:

- Tradiciniai paslaugų analizės ir projektavimo elementai, pagrįsti objektinės analizės ir projektavimo, veiklos procesų modelių bei organizacijų architektūros aspektais:
  - Paslaugų identifikavimo ir apibrėžimo etapas:
    - srities dekompozicija,
    - tiesioginė ir netiesioginė veiklos analizė.
  - Paslaugų specifikavimo etapas:
    - posistemių analizė,
    - paslaugos išskaidymas,
    - susitarimai dėl pavadinimų.
  - Paslaugų realizacijos etapas:
    - komponentų specifikavimas,
    - paslaugų lokalizavimas,
    - paslaugų realizavimas.
- Tikrieji paslaugų analizės ir projektavimo elementai:
  - Paslaugų identifikavimo ir apibrėžimo etapas:
    - „*Meet-in-the-middle*“ procesas.
  - Paslaugų specifikavimo etapas:
    - paslaugų agregavimas ir priskyrimas kategorijoms,
    - politikos ir tam tikri aspektai.
  - Paslaugų realizacijos etapas:
    - semantinis tarpininkavimas,

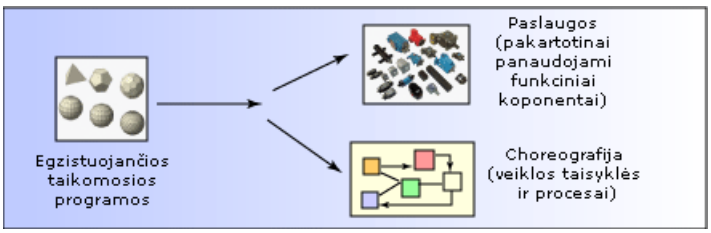
- paslaugų realizacija ir tarpininkavimas perduodant žinias.

Plačiau apžvelgsime paslaugų analizės ir projektavimo etapus neišskiriant tradicinių bei tikrųjų paslaugų analizės ir projektavimo elementų.

Paslaugų identifikavimas ir apibrėžimas

Veiklos modeliavimo „Iš viršaus į apačią“ metodai (pavyzdžiui, komponentų veiklos modeliavimas (angl. *Component Business Modeling – CBM*) [24]) duoda pradžią paslaugų architektūros projektavimui. Tačiau, kaip jau buvo pastebėta, ši architektūra retai kada prasideda „nuo nulio“. Paslaugų architektūros sprendimai dažniausiai paliečia egzistuojančias pasenusias sistemas ir jų grupes, šias sistemas skaidant į paslaugas, operacijas, veiklos procesus ir veiklos taisykles.

Identifikuojant ir apibrėžiant paslaugas, egzistuojančios taikomosios programos skaidomos į konkrečių paslaugų rinkinius, kurie atlieka tam tikras, susijusias operacijas. Toks principas vadinamas „Iš viršaus į apačią“ skaidymo principu. „Iš viršaus į apačią“ vaizde numatomi panaudojimo atvejai tarnauja kaip veiklos paslaugų specifikacijos. Šis procesas dažnai vadinamas *srities dekompozicija*. Ji susideda iš veiklos srities skaidymo į funkcines sritis, posistemius ir aukšto lygio panaudojimo atvejus. Šie panaudojimo atvejai yra tarsi „kandidatai“ tapti paslaugomis. Srities dekompozicijos metu, skaidomos programos veiklos procesai ir veiklos taisyklės perkeliamos iš taikomosios programos į atskirą veiklos procesų modelį, kurį valdo veiklos choreografijos modelis. Sistemų dekompozicija pavaizduota 2.6 paveikslėlyje.



2.6 pav. Paslaugų dekompozicija

Kitas paslaugų identifikacijos ir apibrėžimo būdas – *tiesioginė ir netiesioginė veiklos analizė*. Veiklos procesų modeliai ir tiesioginė reikalavimų analizė (vartotojų apklausos, komponentų veiklos modeliai) yra akivaizdus ir tinkamas paslaugų „kandidačių“ identifikacijos būdas. Jis gali būti papildytas netiesioginiais metodais. Siekiant identifikuoti paslaugas „kandidates“, turi būti apklausiami įvairūs analizuojamos veiklos specialistai, nagrinėjami egzistuojantys panaudojimo atvejai (ne iš paslaugų architektūros projektų). Kuriamos sistemos aprašymų terminologija – tai taip pat geras informacijos šaltinis padedantis rasti operacijas „kandidates“.

Kai nagrinėjamos egzistuojančios sistemos, “iš apačios į viršų” procesas nepakankamai abstrakčiai apibrėžia veiklos paslaugas, o “iš viršaus į apačią” procesas gali būti nepakankamų nefunkcinių reikalavimų ir architektūrinių faktorių neatitikimo priežastimi. Taip pat, naudojant šį procesą, gali atsirasti neatitikimų tarp paslaugų ir komponentų. Todėl, identifikuojant ir apibrėžiant

paslaugas, dažnai taikomas *„Meet-in-the-middle“ procesas*, kai požiūriai “iš viršaus į apačią” ir “iš apačios į viršų” naudojami kartu.

Paslaugų specifikavimas

Identifikavus paslaugas atliekama paslaugų specifikacija. Šiame etape pirmiausiai *analizuojamos posistemės*. Šis veiksmas apibrėžia tarpusavio priklausomybes ir srautus tarp posistemių, identifikuotų sistemos suskaidymo metu. Posistemių analizė susideda iš objektų modelių, kurie vaizduoja posistemių veiklą ir jų projektus. Šių posistemių pagrindu bus realizuotos paslaugos. Posistemės projektas bus traktuojamas kaip stambaus komponento, realizuojančio tam tikrą veiklą, realizacijos modelis.

Pagrindinė paslaugų modeliavimo problema – parinkti tinkamą paslaugos išskaidymo lygį. Paslaugų modelis turi būti kiek galima stambiau suskaidytas, neprarandant paslaugų svarbos, nuoseklumo ir pilnumo. Paslauga gali būti realizuota vienu dideliu komponentu ar sudaryta iš smulkesnių paslaugų rinkinio.

Paslaugos turi skirtingus tikslus ir yra skirtingai naudojamos. Atominės paslaugos gali sudaryti aukštesnio lygio paslaugas. *Paslaugų agregavimą ir priskyrimą kategorijoms* palengvina įvairūs vykdomieji modeliai, pavyzdžiui, BPEL.

Paslaugos apibrėžiamos sintakse, semantika ir įvairiais kokybės rodikliais, kurie turi būti sumodeliuoti. Formalūs sąsajos kontraktai turi apimti daugiau nei apima WSDL.

Projektuojant paslaugų sistemą, turi būti apibrėžta visos programinės organizacijos pavadinimų schema (XML vardų erdvės, programinių paketų, interneto domenų ir t.t.)

Paslaugų realizacija

Paslaugų realizacijos etape pirmiausiai specifikuojami komponentai. Komponentui specifikuojama:

- duomenys,
- taisyklės,
- paslaugos,
- konfigūruojamas profilis,
- variacijos.

Taip pat, šiame žingsnyje apibrėžiamas pasikeitimo pranešimais mechanizmas, specifikuojami įvykiai.

Specifikavus komponentus, kitas žingsnis yra *paslaugų lokalizavimas*. Jis susideda iš paslaugų priskyrimo ankščiau identifikuotoms posistemėms ir komponentų, realizuojančių paslaugas, priskyrimo reikiamiems paslaugų architektūros sluoksniams.

Svarbus paslaugų analizės ir projektavimo etapas yra *semantinis tarpininkavimas*, kurio metu apibrėžiama bendravimo su paslauga sintaksė – formalus sąsajos kontraktas. Tai pagrindinis veiklos tarpusavio (angl. *Bussiness-to-Bussiness – B2B*) [24] bei dinaminio bendravimo scenarijus. Tokie scenarijai yra paslaugų architektūros lankstumo ir prisitaikymo prie kintančių reikalavimų vizijos pagrindas.

Paskutinis paslaugų projektavimo žingsnis – *paslaugos realizacija*. Šis žingsnis reikalauja, kad paslaugą realizuojanti programinė įranga būtų parinkta ar sukurta. Kitas pasirinkimas – naudojant paslaugas atgaivinti pasenusią sistemą. Šiame žingsnyje nusprendžiama kuri pasenusios sistemos dalis bus panaudota duotai paslaugai realizuoti ir kurios paslaugos bus sukurtos „nuo nulio“. Kiti sprendimai susiję su paslaugų realizacija ir nesusiję su jos funkcionalumu – tai paslaugų saugumas, valdymas ir stebėjimas.

Paslaugų realizacija ir tarpininkavimas perduodant žinias – tai žinių valdymo ir gyvavimo ciklo problema: kaip paslaugos galėtų būti paruoštos pakartotinai panaudoti, jei jos buvo konceptualizuotos? Paslaugos turi būti identifikuojamos ir kuriamos taip, kad jas būtų galima pakartotinai naudoti. Jei paslaugos negalima pakartotinai naudoti, ji neturėtų būti kuriama kaip savarankiška paslauga.

Paslaugų architektūros projektavimo tikslas – siekti, kad paslaugą naudotų daugiau nei vienas vartotojas. Tam gali pasitarnauti paslaugų registras (pavyzdžiui, UDDI), kurio pagalba vartotojai gali surasti reikalingas paslaugas.

2.2 Pavienių paslaugų projektavimas

Pavienių paslaugų projektavimo tikslas yra sudaryti paslaugos ir jos sąsajos modelį bei WSDL dokumentą, aprašantį tinklo paslaugą. Toks dokumentas pilnai specifikuoja tinklo paslaugą ir yra nepriklausomas nuo platformos. Suprojektavus WSDL diagramą, su kai kuriais CASE įrankiais galima sugeneruoti WSDL dokumentą. Naudojant specialiai kodo generavimui iš WSDL dokumento skirtus įrankius, galima sugeneruoti tinklo paslaugos realizacijos ir kliento klases. WSDL yra XML dokumentas, jo kūrimui naudojama XML kalba.

2.2.1 WSDL projektavimas naudojant UML

Sėkmingas WSDL projektavimas naudojant UML reikalauja išplėstos UML notacijos, kurią realizuoja WSDL profilis. Šis profilis susideda iš:

- stereotipų, kurie parodyti 2.7 paveiksle atvaizduotame WSDL metamodelyje,
- apribojimų,
- požymių elementų.

Paslaugų projektavimas gali būti išskirtas į dvi stambias WSDL dalis: nepriklausantį nuo platformos (angl. *Platform Independent Model – PIM*) ir priklausantį nuo platformos (angl. *Platform Specific Model – PSM*) modelį [17].

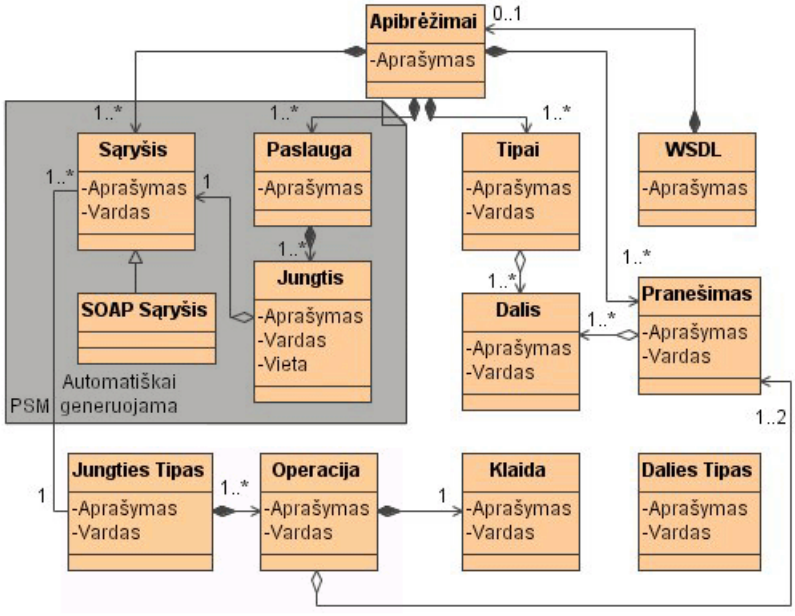
Nepriklausantis nuo platformos modelis atvaizduoja WSDL abstrakcijas:

- aprašą,
- paslaugą,
- jungties tipą (-us),
- pranešimus,
- pranešimų dalis,
- dalių tipus.

Priklausomas nuo platformos modelis atvaizduoja tik dalį WSDL sąryšių. Šiame modelyje atvaizduojami elementai yra:

- paslauga,
- jungtis,
- sąryšis (-iai).

Šiems modeliams priklausantys elementai parodyti 2.7 paveiksle atvaizduotame WSDL metamodelyje (PSM modelio elementai parodyti kitos spalvos fone nei PIM modelio elementai).



2.7 pav. WSDL metamodelis

2.2.2 Pavienių paslaugų projektavimo proceso variantai

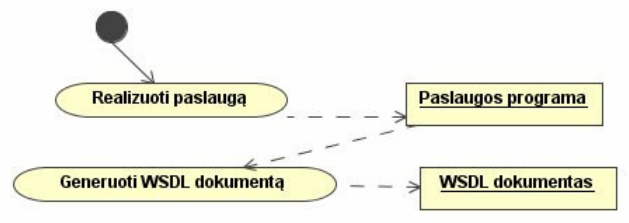
Šiuo metu egzistuoja trys tinklo paslaugų projektavimo metodai:

- „Realizacija – WSDL“;
- „WSDL – Realizacija“;

- „UML – WSDL – Realizacija“;

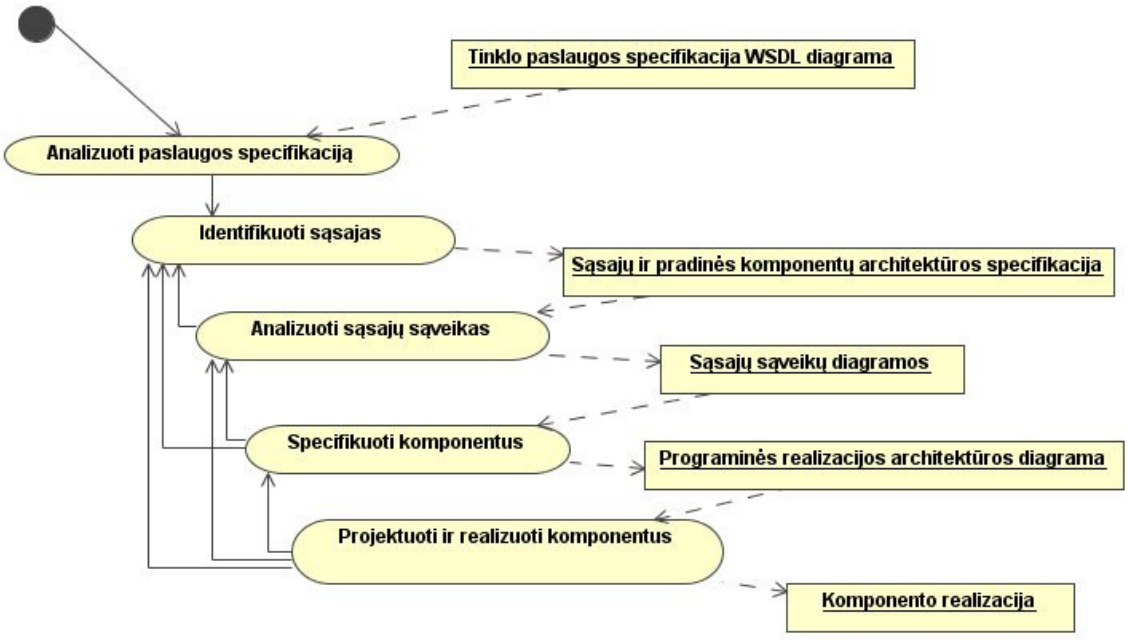
Daugelis tinklo paslaugų kūrimo platformų palaiko tiek „Realizacija – WSDL“, tiek „WSDL – Realizacija“ metodus. Metodui „UML – WSDL – Realizacija“ dar reikalingas UML CASE įrankis.

Naudojant „Realizacija – WSDL“ metodą (2.8 paveikslas), turint realizuotą tinklo paslaugą, sugeneruojama tinklo paslaugos specifikacija – WSDL dokumentas. Šis metodas yra labai paprastas, kadangi nereikia rūpintis WSDL dokumento projektavimu – tai atliekama automatiškai.



2.8 pav. Projektavimo metodo „Realizacija – WSDL“ procesas

Tam tikrais atvejais, turint WSDL dokumentą, gali prireikti sukurti sistemos modelį. Tai atliekama naudojant „WSDL – Realizacija“ metodą, kuris paslaugos projektavimą pradeda nuo WSDL dokumento, kurį panaudoja kodo generavimui, arba paslaugos modelio kūrimui. Šio metodo pagalba išvengiama specifinių programavimo kalbai duomenų tipų naudojimo, kurių transliavimas į SOAP elementus gali būti neteisingas dėl to, kad WSDL nepalaiko tam tikro duomenų tipo. Naudojant „WSDL – Realizacija“, pirmiausiai sukuriamas WSDL dokumentas ir jame aprašomi visi reikalingi duomenų tipai. Projektavimo metodo „WSDL – Realizacija“ procesas atvaizduotas 2.9 paveiksle:

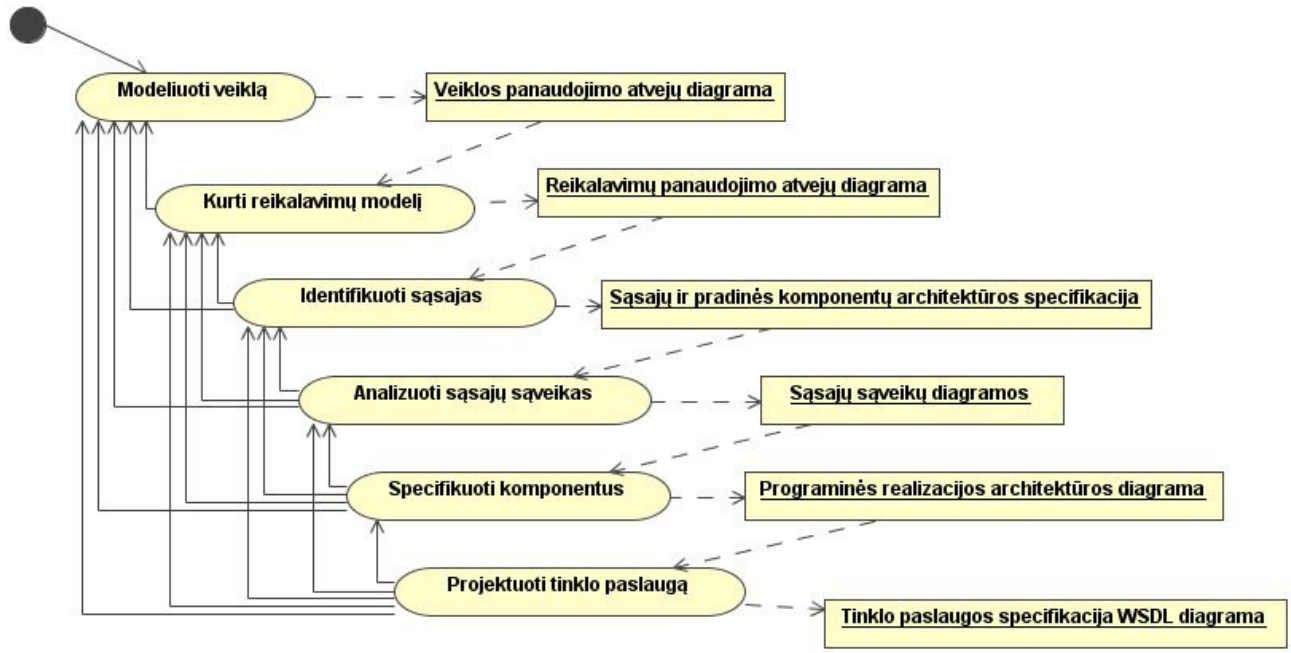


2.9 pav. Projektavimo metodo „WSDL – Realizacija“ procesas

WSDL projektavimas be jokių įrankių pagalbos gali būti sudėtingas, ypač kai tenka projektuoti sudėtingas tinklo paslaugas. Be to, WSDL kaip specifikacija turėtų būti vaizdi ir aiškiai suprantama. Tai pasiekiama naudojant „UML – WSDL – Realizacija“ metodą. Jis remiasi tinklo paslaugos

specifikacijos projektavimu, naudojant UML. Pasinaudojus UML CASE įrankiu ir atlikus tiesioginę kodo inžineriją, galime gauti WSDL dokumentą, kurį galima panaudoti kodo generavimui.

Projektavimo metodo „UML –WSDL – Realizacija“ procesas pavaizduotas 2.10 paveiksle:



2.10 pav. Projektavimo metodo „UML –WSDL – Realizacija“ procesas

„UML – WSDL – Realizacija“ – tai, iteracinis projektavimo metodas. Taikant šį metodą, pirmiausiai modeliuojama paslaugos veikla ir sudaromi veiklos panaudojimo atvejų modeliai.

Kitas žingsnis - sukurti reikalavimų modelį, t.y. sudaryti veiklos konceptų modelį ir identifikuoti panaudojimo atvejus.

Išanalizavus veiklą ir panaudojimo atvejus, identifikuojamos sąsajos. Šiame etape, pirmiausiai reikia sudaryti veiklos tipų modelį ir išskirti esminius veiklos tipus. Nagrinėjant panaudojimo atvejus, šiems esminiams veiklos tipams priskiriamos sąsajos, atsakingos už šių tipų informacijos apdorojimą (atsakomybė diagramoje vaizduojama kompozicijos ryšiu). Sudarius veiklos tipų modelį ir išskyrus esminius veiklos tipus, sudaroma pradinė komponentų architektūros specifikacija.

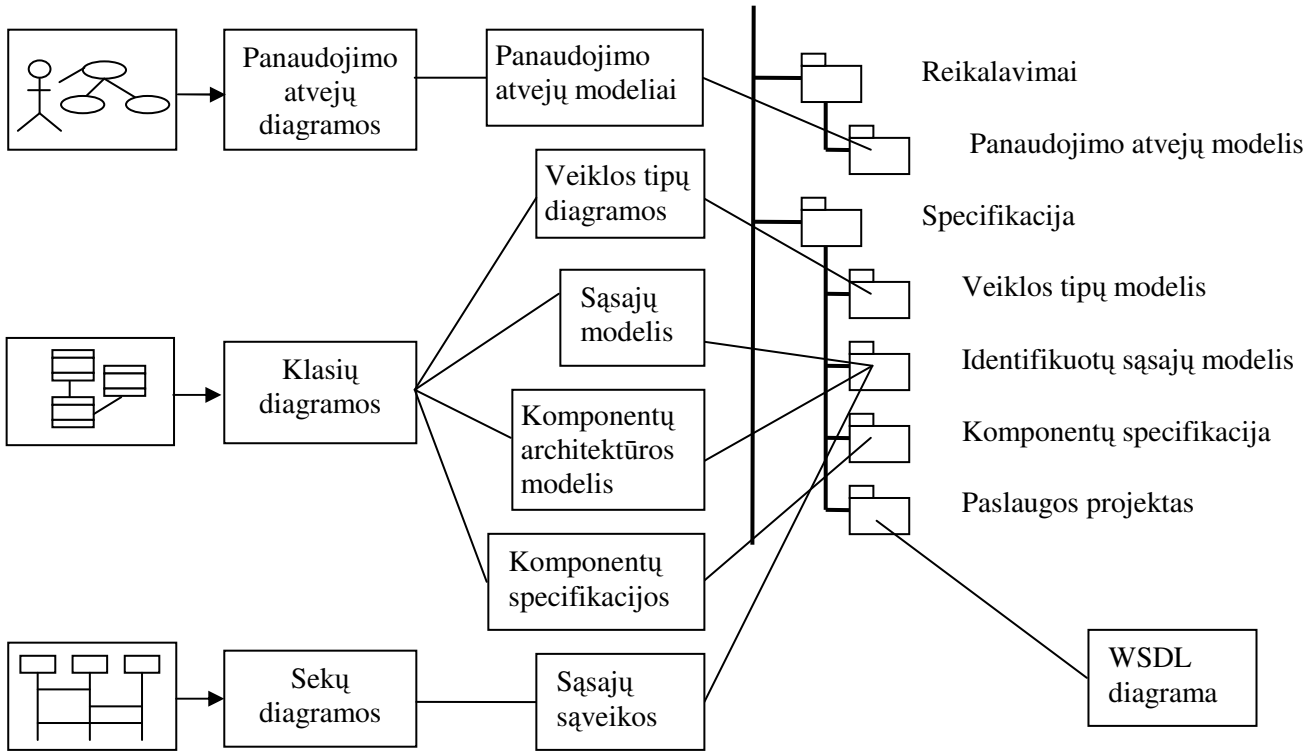
Sąsajų identifikacijos etape išskirti komponentai turi atlikti tam tikras operacijas. Šios operacijos identifikuojamos analizuojant sąsajų tarpusavio sąveikas. Tam kuriamos sekų diagramos ir specifikuojamos operacijos.

Kai sąsajų operacijos jau identifikuotos, reikia specifikuoti komponentus – išsiaiškinti kokią informaciją komponentai apdoroja. Specifikuojant komponentus, sudaromi veiklos sąsajų informaciniai modeliai. Jie sudaromi iš veiklos tipų modelio, įtraukiant sąsajų operacijose apdorojamus elementus. Sudarius sąsajų informacinius modelius, šias sąsajas galima apibendrinti ar specializuoti, projektuoti komponentų vidinę struktūrą. Paskutinis komponentų specifikavimo etapo žingsnis – realizacijos architektūros modelio sudarymas.

Specifikavus komponentus galima sudaryti paslaugos projektą – WSDL diagramą. Atlikus tiesioginę kodo inžineriją, iš šios diagramos galima sugeneruoti paslaugos aprašymo dokumentą WSDL kalba.

Siūlomą paslaugos projektavimo metodą iliustruoja 3.2 skyriuje pateiktas kalendoriaus paslaugos projektavimo pavyzdys.

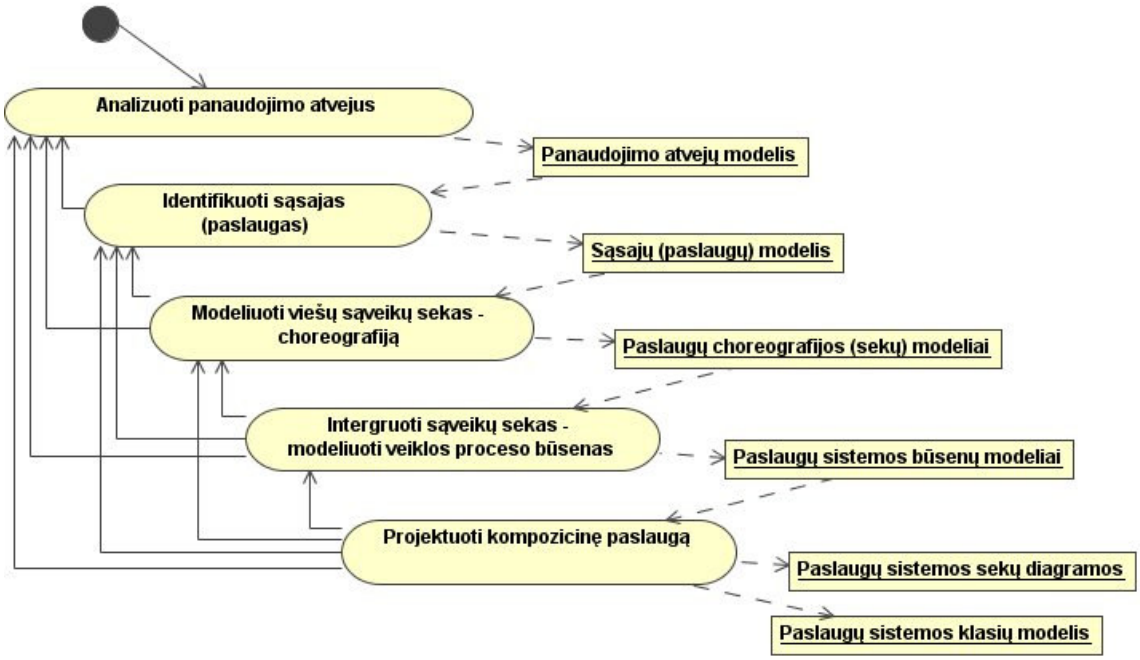
Paslaugos projektavimo modelių ir diagramų atvaizdavimo UML struktūra pateikta 2.11 paveiksle.



2.11 pav. Paslaugos projektavimo modelių ir diagramų atvaizdavimas UML

2.3 Paslaugų sistemos projektavimas

Paslaugos dažniausiai naudojamos jas jungiant su kitomis paslaugomis – kuriant paslaugų sistemas. Paslaugų sistemos projektavimo procesas – tai iteracinis procesas. Jis pavaizduotas 2.12 paveiksle.



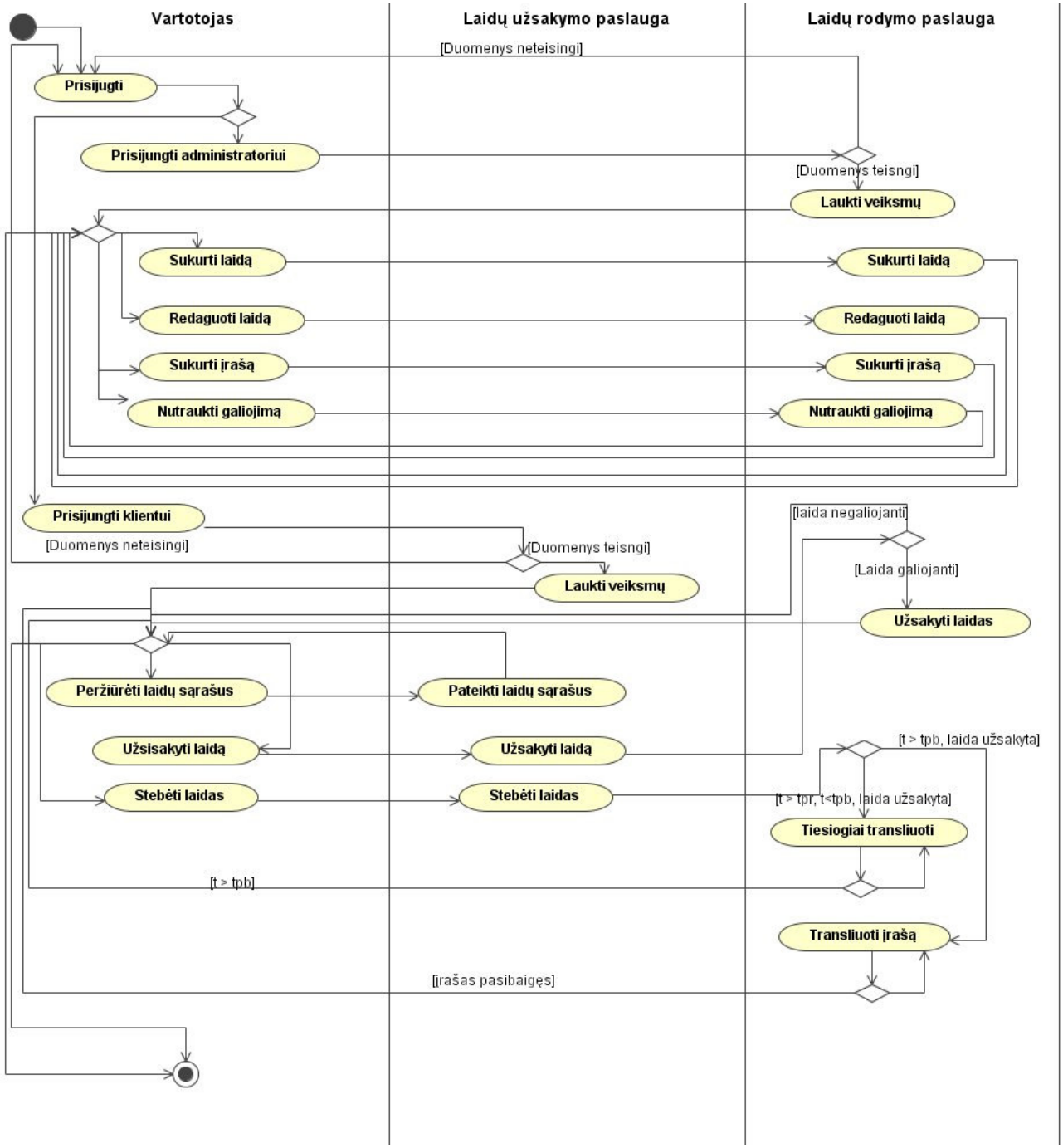
2.12 pav. Paslaugų sistemos projektavimo procesas

Projektuojant paslaugų sistemą, pirmiausiai nagrinėjami reikalavimai ir sudaromas sistemos panaudojimo atvejų modelis. Paslaugų sistemos projektavimo metu atskirų paslaugų projektavimas nenagrinėjamas – laikoma, kad paslaugos jau yra sukurtos arba, kad jos bus sukurtos pagal sąsajų apibrėžimus, sudarytus pagal panaudojimo atvejų specifikacijas.

Kiekvienas panaudojimo atvejis atitinka paslaugą, kuri abstrakčiai aprašoma sąsaja. Paslaugų architektūroje veiklos procesų vykdymą realizuoja paslaugų choreografija, kuri modeliuojama sekų diagramomis. Taigi, identifikavus paslaugas, sumodeliuojami jų bendradarbiavimo scenarijai – sudaromi paslaugų choreografijų (sekų) modeliai.

Sumodeliavus paslaugų scenarijus, reikia juos integruoti, t.y. modeliuoti veiklos procesą. Veiklos modeliavimo kalbos naudoja veiklos diagramas, kurios leidžia pavaizduoti proceso eigą. Šios diagramos labiau tinka veiklos procesams, kurie susideda iš daugelio žingsnių ir turi griežtai apibrėžtas veiklos taisykles. Tačiau, kai procesų vykdymo tvarka yra labai įvairi, o veiksmų vykdymo sąlygos priklauso tik nuo to, kokiose būsenose yra sistemos objektai (pavyzdžiui, vartotojas užsisakęs laidą gali ją žiūrėti tiesiogiai arba daug kartų žiūrėti įrašą), tuomet veiklos diagramos neduoda aiškaus veiklos vaizdo (2.13 paveikslas). Tą padaryti leidžia būsenų diagramos. Taigi, paslaugų sistemas siūloma projektuoti remiantis būsenų modeliu, kuris atitinka objektinio kūrimo principus. Būsenų mašina yra pripažintas sudėtinių paslaugų elgsenos modelis. Veiklos procesų valdymo programos irgi veikia būsenų mašinos principu, tačiau jos remiasi procedūriniu kūrimu, todėl kuriant sistemą objektiniais principais veiklos diagramas vis vien reikėtų atvaizduoti į būsenų diagramas. Internetu transliuojamų laidų sistemos (projektavimo pavyzdys aprašytas 3.3 skyriuje) veiklos diagramos

pavyzdys pateiktas 2.13 paveiksle. Matome, kad nors ši diagrama nėra pilna ir nevisi atvejai numatyti, ji jau yra pakankamai paini.



2.13 pav. Interneto laidų transliavimo sistemos veiklos diagrama

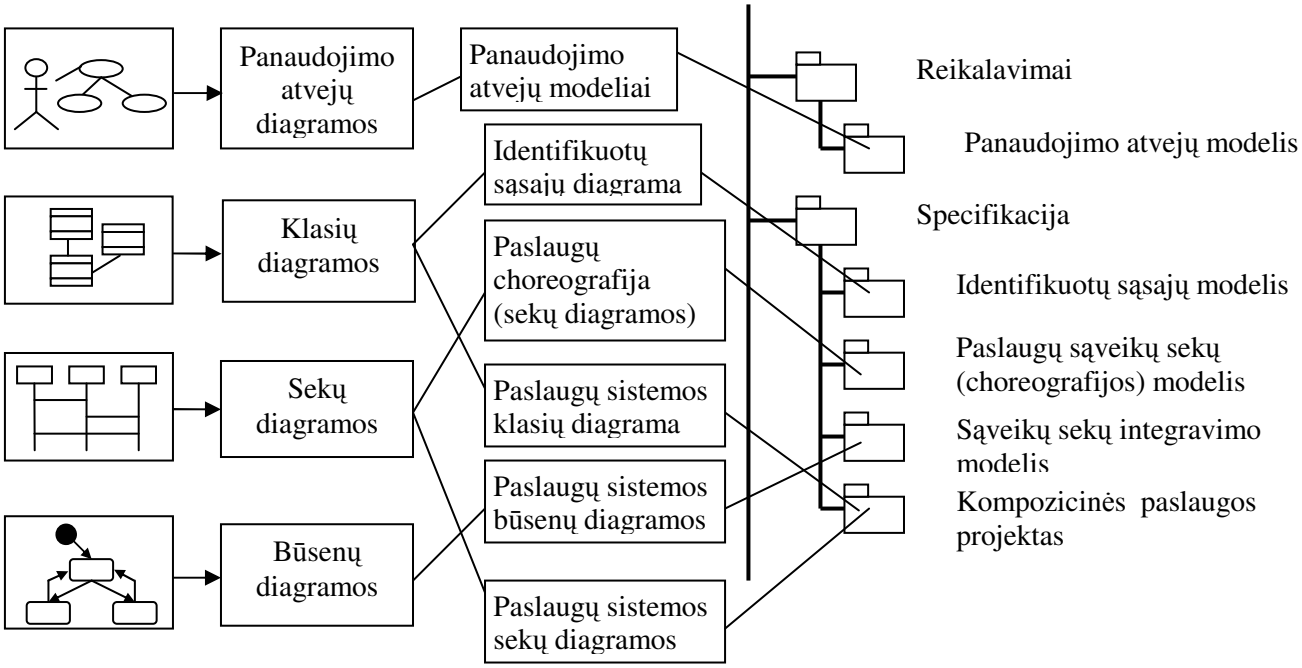
Projektuojant paslaugų sistemas remiantis būsenų modeliais, pagrindinės sistemos esybės gali turėti įvairias būsenas (pavyzdžiui, vartotojas gali būti registruotas arba neregistruotas), kurias gali keisti net keletas paslaugų. Problema kyla dėl to, kad paslaugos negali išsaugoti nei savo, nei sistemos esybių būsenų, todėl reikalingas tam tikras mechanizmas joms valdyti. Tokiu mechanizmu gali būti būsenų valdiklis, kuris įsimena sistemos esybių būsenas ir valdo į sistemą ateinančius pranešimų srautus, juos paskirstydamas reikiamoms paslaugoms atsižvelgiant į sistemos esybių būsenas.

Tokiu būdu paslaugų sistema (kompozicinė paslauga) sudaroma analizuojant choreografijos sekų diagramas ir sujungiant jas į būsenų modelį, kuris vaizduoja vykdomą procesą ir padeda suderinti sąveikas tarp paslaugų. Kitaip tariant, būsenų modelis padeda užtikrinti, kad proceso aprašas bus toks, kokio siekia paslaugų teikėjas.

Paskutinis paslaugų sistemos etapas – paslaugų sistemos projekto sudarymas. Išoriškai matoma kompozicinė paslauga aprašoma WSDL diagrama. Vidinį paslaugų sistemos projektą sudaro būsenų valdiklio klasė, sąveikaujanti su paslaugų klasėmis. Būsenų valdiklis priima užklausas ir tikrina jų vykdymo leistinumą, įvertindamas sistemoje saugomas įvykdytų veiklos procesų žingsnių būsenas. Jei užklausą pateikusiam klientui galima leisti naudoti paslaugą, būsenų valdiklis perduoda užklausą vidinėms arba išorinėms paslaugoms; jei ne, pateikia klientui pranešimą, jog sistema negali įvykdyti užklausos.

Siūlomą paslaugų sistemos projektavimo metodą iliustruoja 3.3 skyriuje pateiktas internetu transliuojamų laidų paslaugos projektavimo pavyzdys.

Paslaugų sistemos projektavimo modelių ir diagramų atvaizdavimo UML struktūra pateikta 2.14 paveiksle.



2.14 pav. Paslaugų sistemos projektavimo modelių ir diagramų atvaizdavimas UML

3 TINKLO PASLAUGŲ PROJEKTAVIMO METODIKA CASE ĮRANKIUOSE

3.1 Eksperimentinio pavyzdžio aprašymas

Kaip jau buvo minėta 2.2 skyriuje, vienos paslaugos projektavimo tikslas yra sudaryti paslaugos ir jos sąsajos modelį bei WSDL dokumentą, aprašantį paslaugą. Egzistuoja du pagrindiniai paslaugų projektavimo būdai:

- Modeliuojama paslauga ir, suprojektavus visą paslaugos funkcionalumą, kuriama WSDL diagrama, iš kurios generuojamas paslaugos WSDL dokumentas („UML – WSDL – Realizacija“ metodas).
- Sukuriamas paslaugos WSDL dokumentas ir iš jo sugeneruojama WSDL diagrama. Ja remiantis modeliuojama paslauga („Realizacija – WSDL“ metodas);

Iliustruojant šiame darbe nagrinėjamą tinklo paslaugų projektavimą, buvo išbandyti abu pavienių paslaugų projektavimo metodai (suprojektuota paslauga ir sudarytas WSDL modelis, bei realizuota paslauga ir sugeneruotas WSDL dokumentas) .

Projektavimui naudojamas Magic Draw paketas, kadangi jis palaiko atvirkštinę inžineriją - turint WSDL dokumentą, paketu MagicDraw galima atlikti atvirkštinę WSDL kodo inžineriją ir sugeneruoti WSDL diagramą. Tai gali būti naudinga tobulinant jau sukurtą paslaugą – sugeneravus WSDL diagramą, papildomas tinklo paslaugos funkcionalumas ir, panaudojus tiesioginę kodo inžineriją, gaunamas papildytas WSDL dokumentas.

Kad būtų galima atlikti atvirkštinę inžineriją, buvo sukurta eksperimentinė tinklo paslauga. Šiai paslaugai realizuoti pasirinkta PHP technologija. Tinklo paslaugos dažniausiai realizuojamos .NET arba Java technologijomis, todėl buvo įdomu išbandyti kaip tinklo paslaugos kuriamos naudojant šią technologiją, kuri retai pasirenkama paslaugoms realizuoti, tačiau plačiai naudojama kuriant įvairias sistemas.

Paslaugų realizacijos technologijos išsamiai nenagrinėjamos, kadangi šio darbo tikslas yra sudaryti tinklo paslaugų projektavimo metodiką, o realizuota paslauga reikalinga tik tam, kad gauti WSDL dokumentą.

Tinklo paslaugas kuriant PHP technologija, galima rinktis vieną iš trijų pagrindinių kelių – kurti tinklo paslaugas naudojančias SOAP, XML-RPC ar REST standartus. Naudojant XML-RPC standartą paslaugos sukuriamos per trumpesnį laiką, jas lengviau taisyti, o jų veiksmai vykdomi sparčiau. Naudojant REST tinklo paslaugas, iškvietimai perduodami HTTP protokolo pagalba, parametrai užrašomi užklausos pavidalu, o tinklo paslaugos atsakymas pateikiamas kaip XML failas. Plačiausiai naudojamas SOAP standartas. Daugelis programuotojų tinklo paslaugos sąvoką sieja su SOAP sąvoka. Tinklo paslaugos gali naudoti kitus standartus, tačiau beveik visi šie standartai turės galimybę apdoroti SOAP pranešimus [14]. PHP (iki PHP 5 versijos) neturi SOAP palaikymo, todėl reikalingos taip

vadinamos „trečiosios“ technologijos suteikiančios SOAP palaikymą. Populiariausios tokio tipo technologijos – nuSOAP [4] [14] ir Pear SOAP [14] paketai. Dėl paprastesnių programavimo konstrukcijų, paslaugos realizacijai pasirinktas nuSOAP paketas.

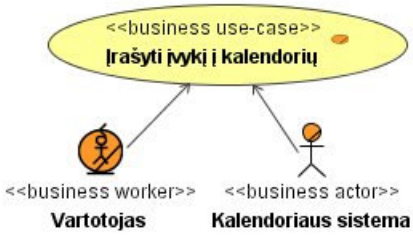
Iliustruojant pavienių paslaugų projektavimo metodiką, suprojektuota ir realizuota kalendoriaus tinklo paslauga, leidžianti vartotojams sudaryti įvykių grafikus (įrašyti įvykius ir juos peržiūrėti). Įvykiai gali būti labai įvairūs. Pavyzdžiui, toliau nagrinėjamame paslaugų sistemos projektavimo pavyzdyje ši paslauga naudojama internetu transliuojamų laidų planavimui ir valdymui.

Paslaugos dažniausiai naudojamos jungiant jas su kitomis paslaugomis – kuriant paslaugų sistemas. Problema kyla dėl to, kad paslaugos negali išsaugoti savo būsenų, todėl reikalingas tam tikras mechanizmas joms valdyti. Tokiu mechanizmu gali būti būsenų valdiklis. Būsenų valdikliu paremto projektavimo procesui iliustruoti, modeliuojama internetu transliuojamų laidų (pavyzdžiui, konferencijų, seminarų ir pan.) peržiūros sistema, kur prisijungęs vartotojas gali peržiūrėti laidų sąrašus ir užsiregistruoti norimos laidos peržiūrai. Užsisakytą laidą vartotojas gali peržiūrėti tiesiogiai jos transliavimo metu arba vėliau peržiūrėti vaizdo įrašą. Prie sistemos prisijungęs administratorius gali valdyti laidas (paskelbti naujas, redaguoti paskelbtas, peržiūrėti įvykusias, ar dar vykstančias laidas, praėjus tam tikram laikui po laidos, gali nutraukti jos galiojimą ir ši laida nebus skelbiama).

3.2 Kalendoriaus tinklo paslaugos projektas

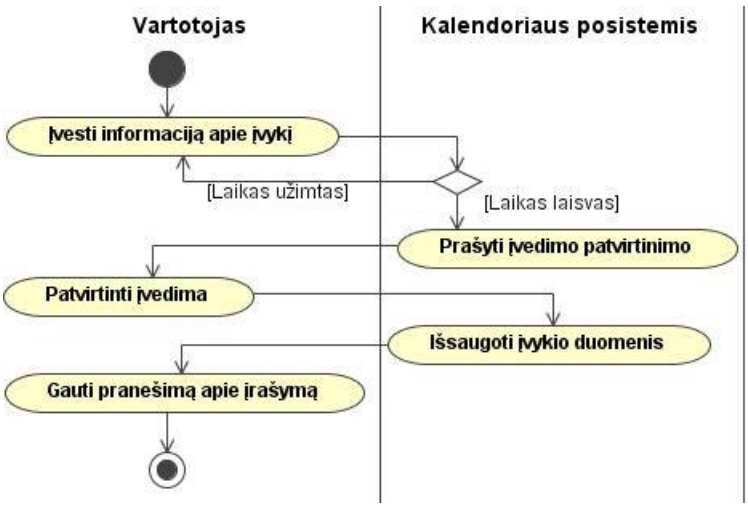
Veiklos modeliavimas

Projektuojant paslaugą, pirmiausiai modeliuojama jos veikla. Įvykio įrašymo veiklos panaudojimo atvejis parodytas 3.1 paveiksle.



3.1 pav. Įvykio įrašymo veiklos panaudojimo atvejis

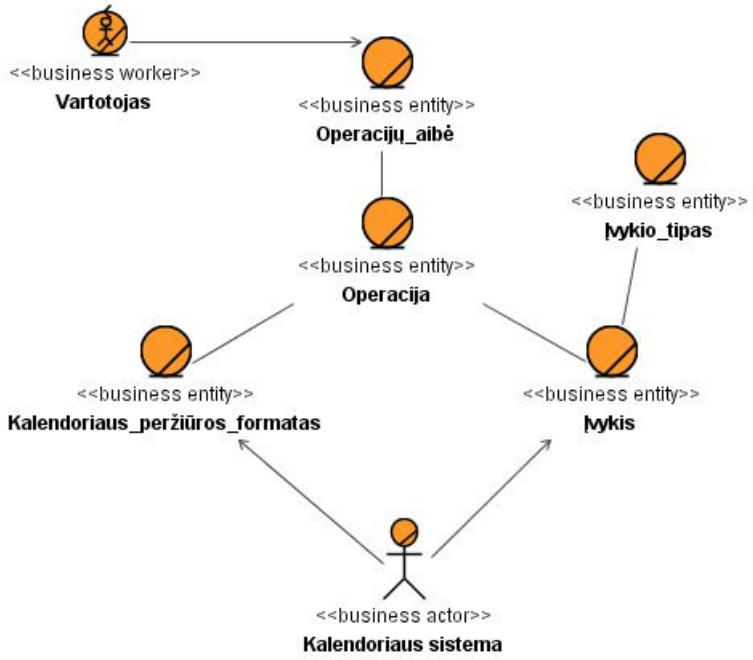
Šio proceso veiklos diagrama pateikta 3.2 paveiksle.



3.2 pav. Įvykio įrašymo proceso veiklos diagrama

Reikalavimų apibrėžimas

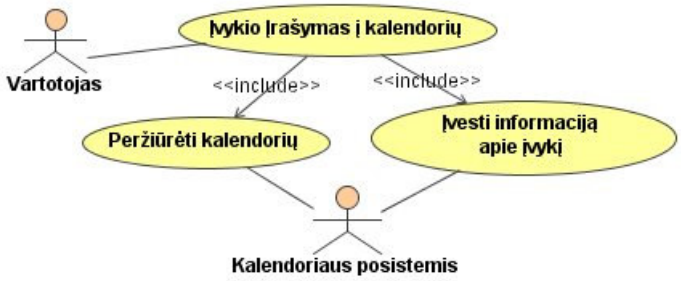
Reikalavimų etape sudaromas veiklos konceptų modelis (3.3 paveikslas) ir identifikuojami panaudojimo atvejai (3.4 paveikslas).



3.3 pav. Veiklos konceptų modelis

3.4 pav. Panaudojimo atvejų diagrama

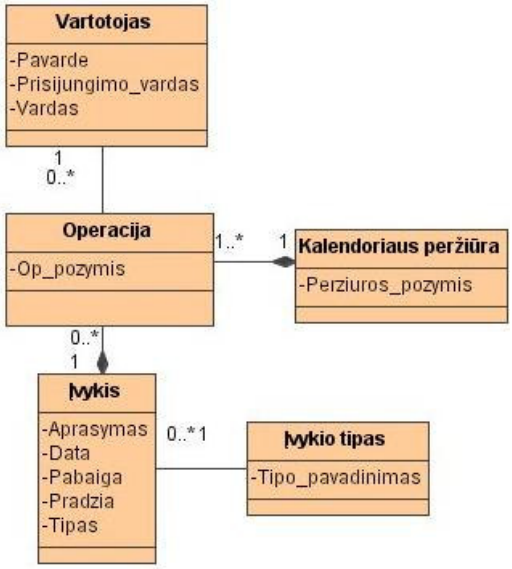
Jei įvykio įrašymas turi vykti kaip viena transakcija, tuomet panaudojimo atvejus reikėtų sujungti į vieną panaudojimo atvejį (3.5 paveikslas).



3.5 pav. Panaudojimo atvejų diagrama, kai įvykio įrašymas vyksta kaip viena transakcija

Sąsajų identifikacija

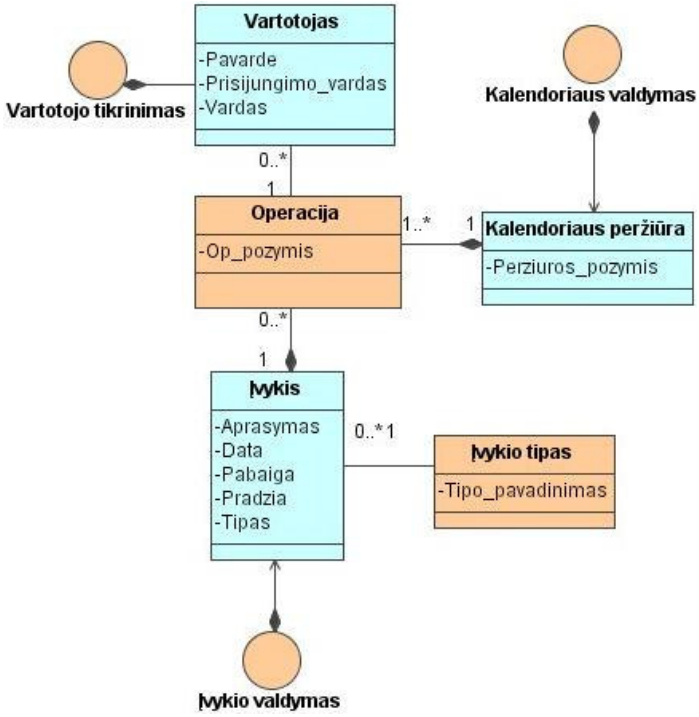
Norint identifikuoti sąsajas, pirmiausiai sudaromas veiklos tipų modelis (3.6 paveikslas).



3.6 pav. Veiklos tipų modelis

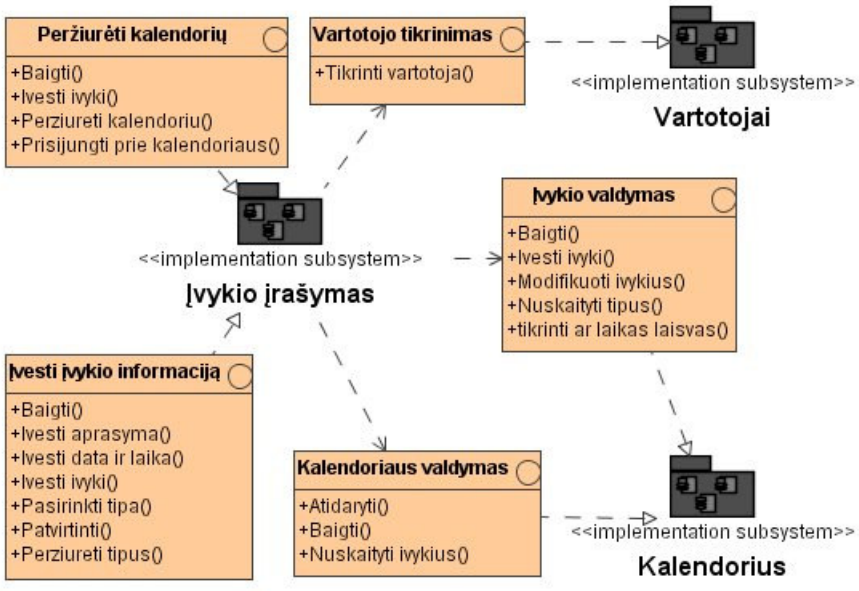
Išskiriami esminiai veiklos tipai (paveiksle jie parodyti kitokia spalva). Nagrinėjant panaudojimo atvejus, šiems esminiams veiklos tipams priskiriamos sąsajos, atsakingos už šių tipų informacijos apdorojimą (atsakomybę rodo kompozicijos ryšys). Priklausomi tipai sujungiami su esminiais tipais

kompoziciniais ryšiais. Esminius veiklos tipus ir sąsajų atsakomybes iliustruoja 3.7 paveiksle patekta diagrama.



3.7 pav. Sąsajų atsakomybės diagrama

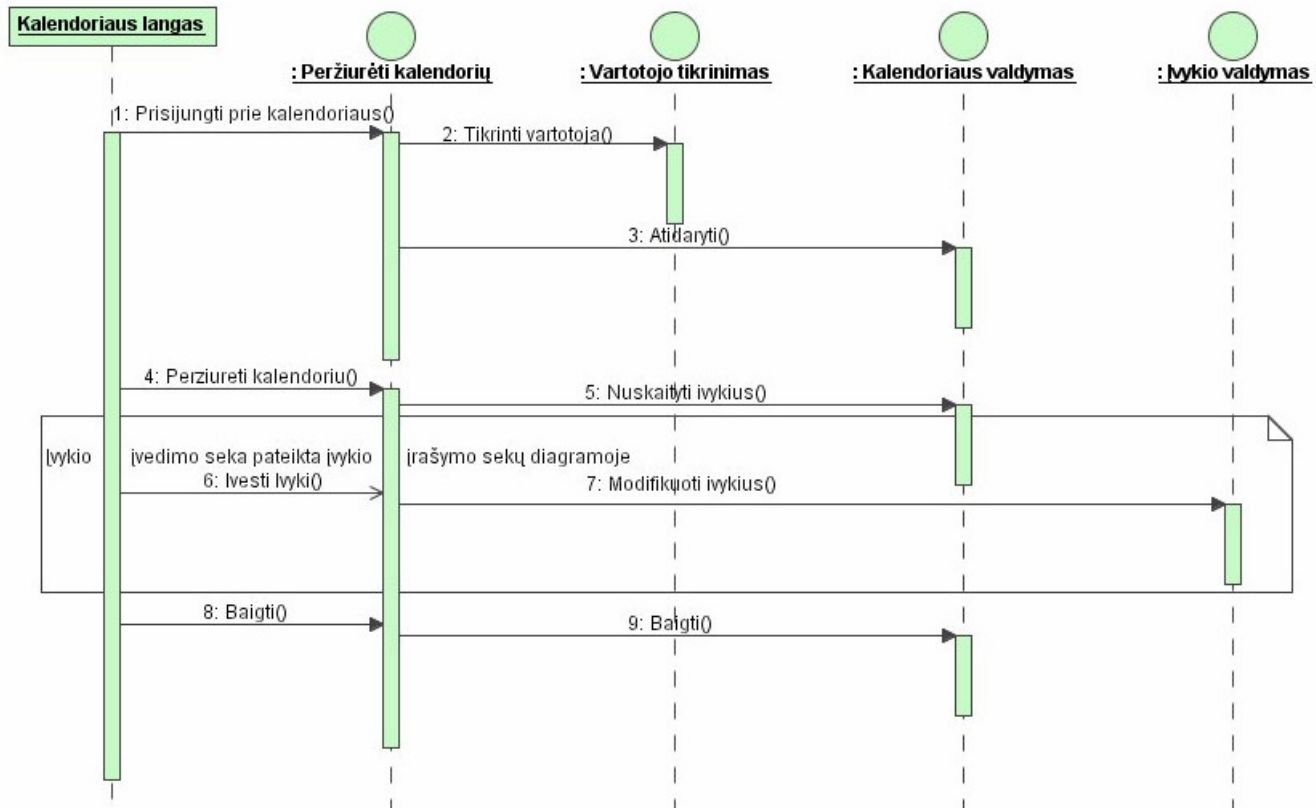
Sudarius veiklos tipų modelį ir išskyrus esminius veiklos tipus, sudaroma pradinė komponentų architektūros specifikacija (3.8 paveikslas):



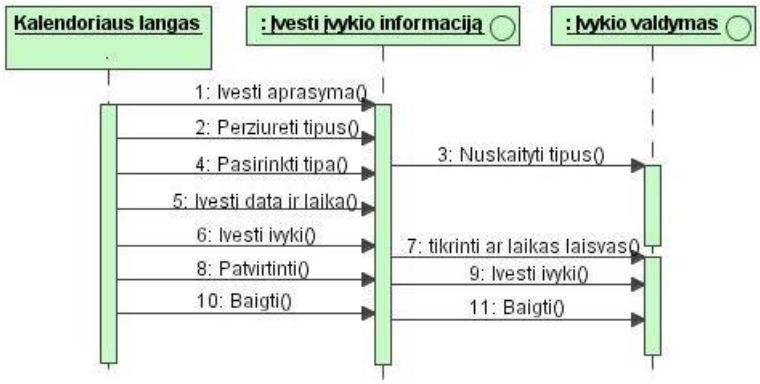
3.8 pav. Pradinė komponentų architektūros specifikacija

Sąsajų sąveikų analizė

Sąsajų sąveikų analizė reikalinga siekiant identifikuoti veiklos sąsajų operacijas. Tam sukuriamos sekų diagramos, specifikuojamos operacijos. Kalendoriaus peržiūros sekos diagrama pateikta 3.9 paveiksle, o įvykio įrašymo sekas diagrama – 3.10 paveiksle.



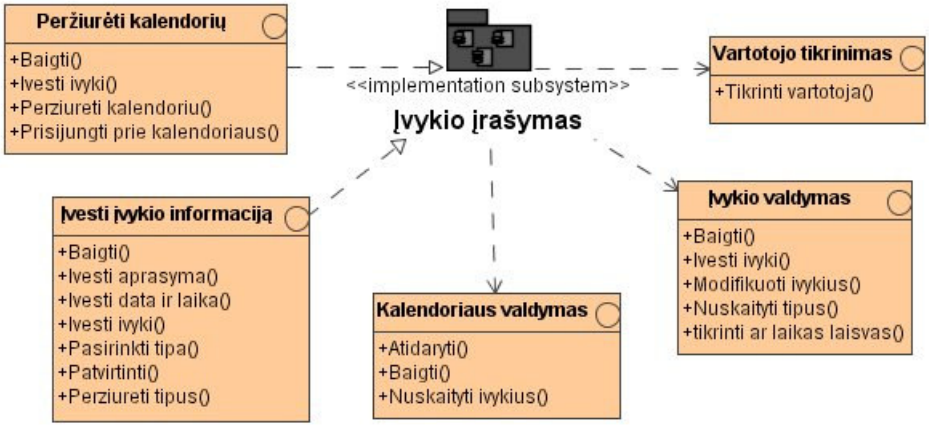
3.9 pav. Kalendoriaus peržiūros sekos diagrama



3.10 pav. Įvykio įrašymo sekos diagrama

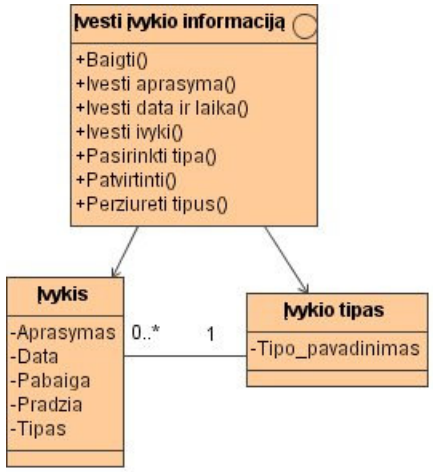
Komponentų specifikacija

Kai sąsajų operacijos jau identifikuotos, reikia išsiaiškinti kokią informaciją komponentai apdoroja, t.y. reikia specifikuoti komponentus (3.11 paveikslas).

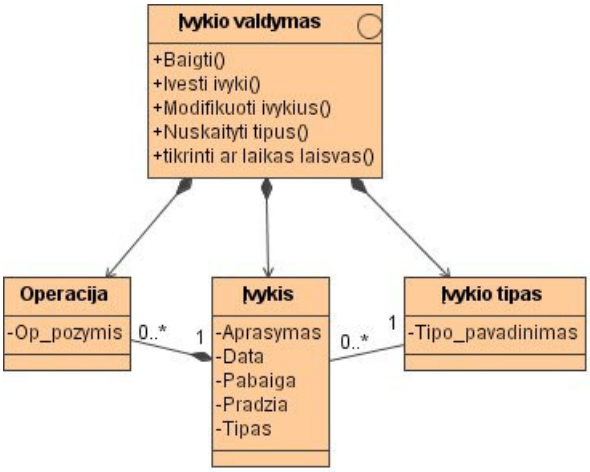


3.11 pav. Komponento specifikacija

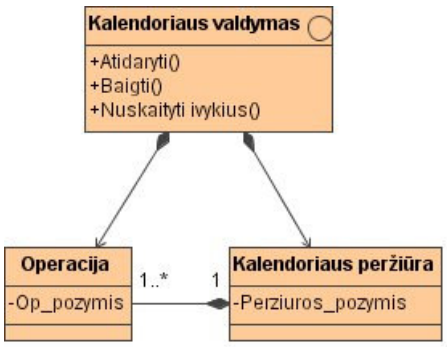
Kitas žingsnis – sudaryti veiklos sąsajų informacinius modelius. Jie sudaromi iš veiklos tipų modelio, įtraukiant sąsajų operacijose apdorojamus elementus. Įvykio įvedimo sąsajos informacinis modelis pateiktas 3.12 paveiksle, įvykio valdymo sąsajos informacinis modelis – 3.13 paveiksle, kalendoriaus valdymo sąsajos informacinis modelis – 3.14 paveiksle.



3.12 pav. Įvykio įvedimo sąsajos informacinis modelis

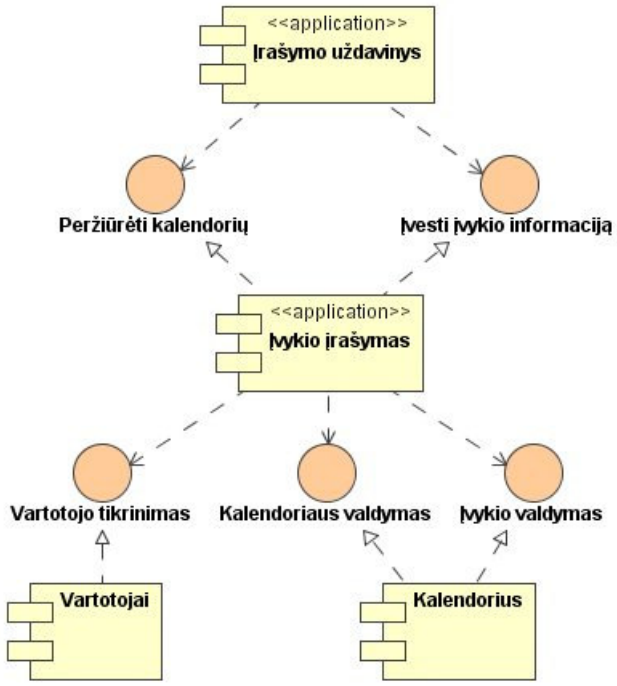


3.13 pav. Įvykio valdymo sąsajos informacinis modelis



3.14 pav. Kalendoriaus valdymo sąsajos informacinis modelis

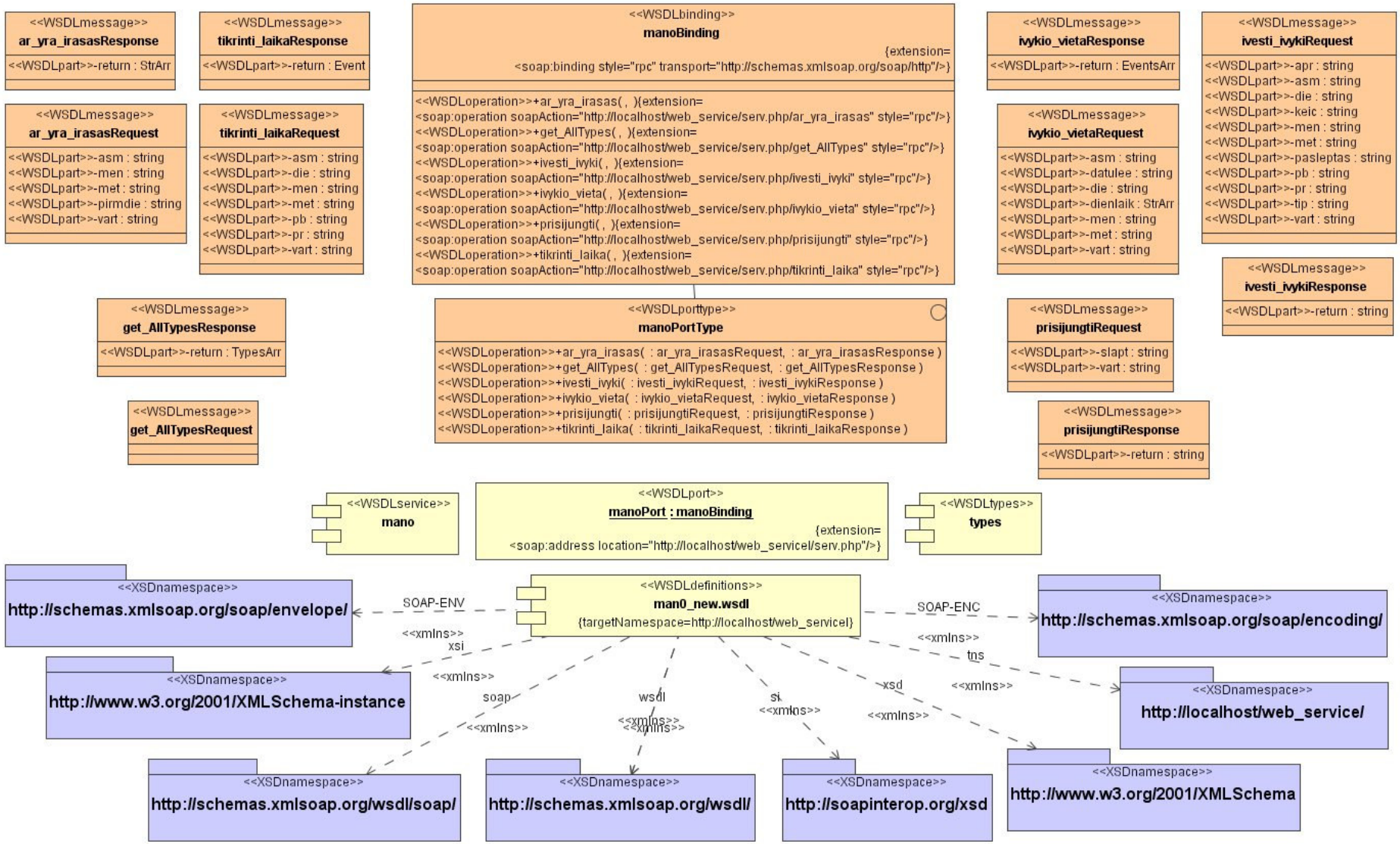
Tolesnis žingsnis – sąsajas galima apibendrinti ar specializuoti, projektuoti komponentų vidinę struktūrą. Paskutinis komponentų specifikavimo etapo žingsnis – sudaryti realizacijos architektūros modelį (3.15 paveikslas).



3.15 pav. Programinės realizacijos architektūra

Paslaugos projektas

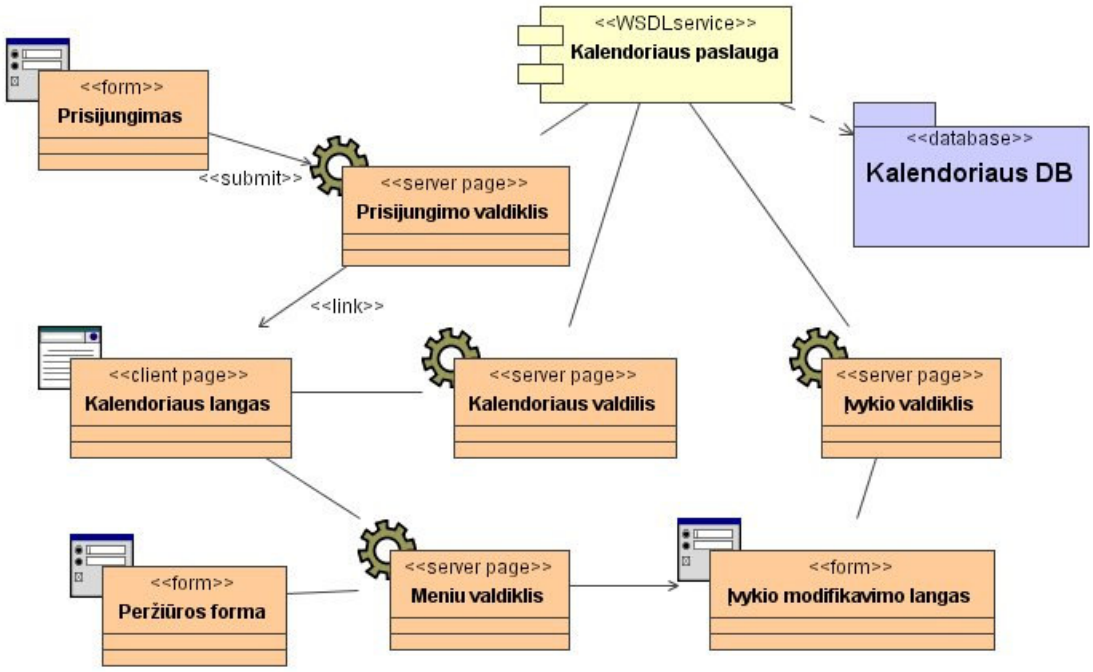
Kalendoriaus paslaugos projektas pateiktas (3.16 paveiksle).



3.16 pav. Kalendoriaus paslaugos WSDL diagrama

Kalendoriaus paslaugos panaudojimo modelis

3.17 paveiksle pateikiamas kalendoriaus paslaugos panaudojimo modelis.



3.17 pav. Kalendoriaus paslaugos panaudojimo modelis

Paslaugos realizacija

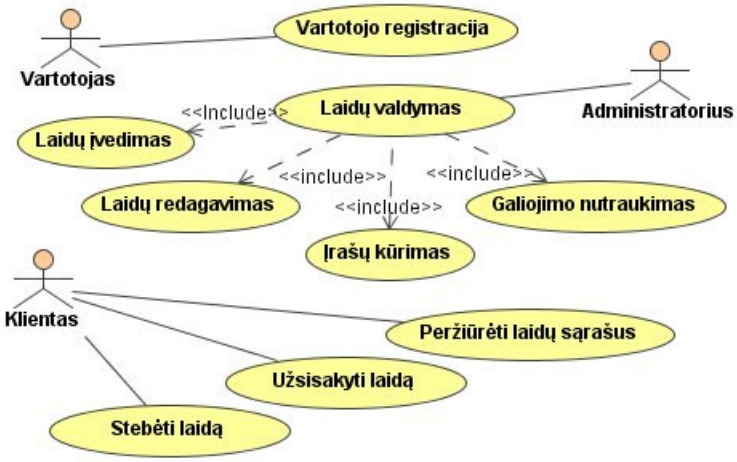
Kad būtų galima išbandyti atvirkštinę inžineriją (paslaugų projektavimo metodą, kai pirmiausiai turimas WSDL dokumentas, arba projektavimo metodą Realizacija–WSDL), buvo sukurta eksperimentinė kalendoriaus tinklo paslauga, leidžianti vartotojams sudaryti įvykių grafikus (įrašyti įvykius ir juos peržiūrėti). Šios paslaugos WSDL dokumentas pateiktas 1 priede (žr. 1 Priedas). Atlikus atvirkštinę kodo inžineriją, iš WSDL dokumento sugeneruotas WSDL modelis, kuris nepateikiamas, nes jis sutampa su WSDL modeliu, gautu taikant UML – WSDL – Realizacija metodą.

3.3 Interneto laidų transliavimo paslaugų sistemos projektas

Panaudojimo atvejai

Projektuojama internetu transliuojamų laidų (pavyzdžiui, konferencijų, seminarų ir pan.) peržiūros sistema, kur prisijungęs vartotojas gali peržiūrėti laidų sąrašus ir užsiregistruoti norimos laidos peržiūrai. Užsisakytą laidą vartotojas gali peržiūrėti tiesiogiai jos transliavimo metu arba vėliau peržiūrėti vaizdo įrašą. Prie sistemos prisijungęs administratorius gali valdyti laidas (paskelbti naujas, redaguoti paskelbtas, peržiūrėti įvykusias, ar dar vykstančias laidas, praėjus tam tikram laikui po laidos, gali nutraukti jos galiojimą ir ši laida nebus skelbiama).

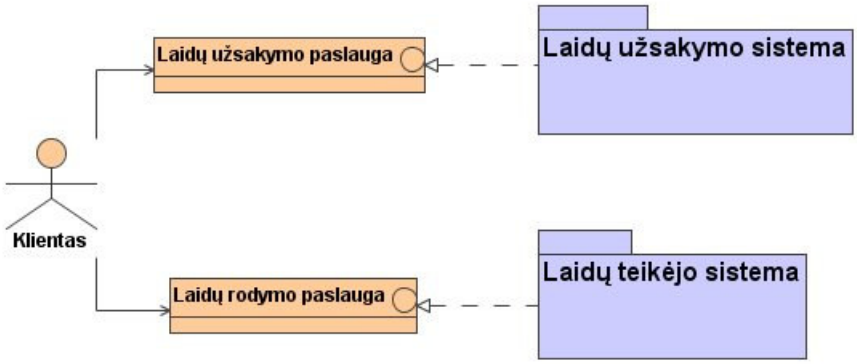
Sistemos panaudojimo atvejų diagrama pateikta 3.18 paveiksle.



3.18 pav. Interneto laidų transliavimo sistemos panaudojimo atvejų diagrama

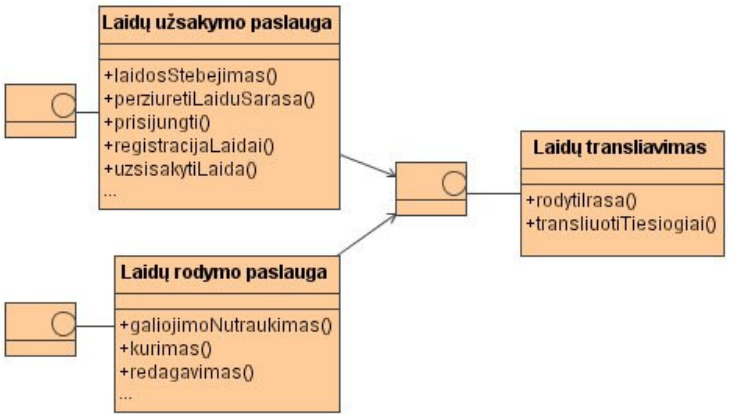
Sąsajų (paslaugų) identifikavimas

Kiekvienas panaudojimo atvejis atitinka paslaugą, kuri abstrakčiai aprašoma sąsaja. 3.19 paveiksle pateiktas modelis vaizduoja abstrakčią klasių diagramą („juodos dėžės“ principas), kuri parodo ryšius tarp laidų užsakymo ir laidų teikėjo sistemų. Paketai („Laidų užsakymo sistema“ ir „Laidų tekėjo sistema“) apibendrina sistemų klases. Rodomos tik išorinės šių sistemų sąsajos („Laidų užsakymo paslauga“ ir „Laidų rodymo paslauga“).



3.19 pav. Laidų užsakymo ir laidų teikėjo paslaugos bei jų ryšiai

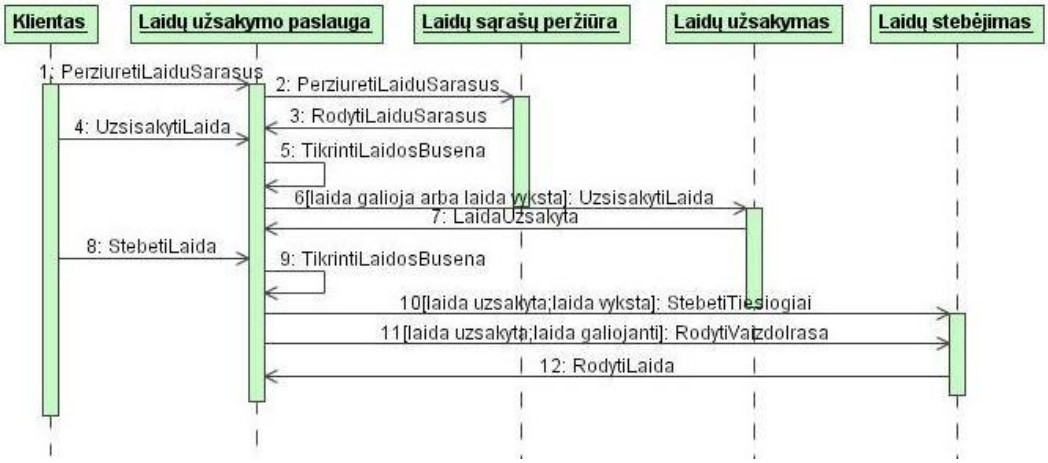
Identifikuotų paslaugų modelis (3.20 paveikslas) vaizduojamas klasių diagrama. Priešingai nei objektiniame projektavime, šis modelis neatspindi funkcionalumo. Čia nėra jokių informacijos srautų, veiklos įvykių ar taisyklių aprašymų.



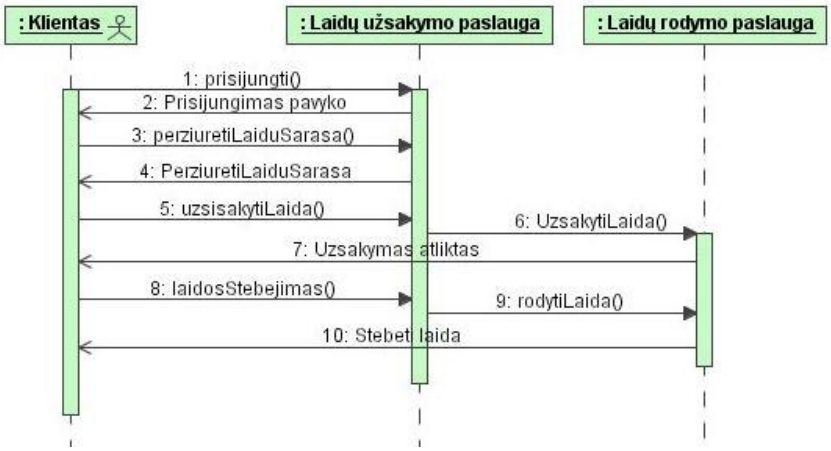
3.20 pav. Identifikuotų paslaugų modelis

Viešųjų sąveikų sekų (choreografijos) modeliavimas

Paslaugų architektūroje, veiklos procesų vykdymą realizuoja paslaugų choreografija, kuri modeliuojama sekų diagramomis. Taigi, identifikavus paslaugas, modeliuojamos jų viešų sąveikų sekos – sudaromi paslaugų choreografijų (sekų) modeliai. 3.21 paveiksle parodyta pradinė situacija – pageidaujamas sistemos funkcionalumas, o 3.22 paveiksle – paslaugų choreografija.



3.21 pav. Pageidaujamas sistemos funkcionalumas

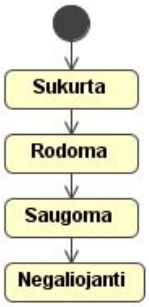


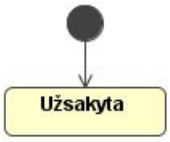
3.22 pav. Paslaugų choreografija

Veiklos modeliavimas

Sumodeliavus paslaugų sąveikų scenarijus, reikia juos integruoti, t.y. modeliuoti veiklos procesą. Veiklos modeliavimo kalbos naudoja veiklos diagramas, kurios labiau tinka veiklos procesams, tačiau kai reikia sukurti sistemą, kur yra keli tokie procesai, tada šią sistemą galima pavaizduoti būsenų diagrama.

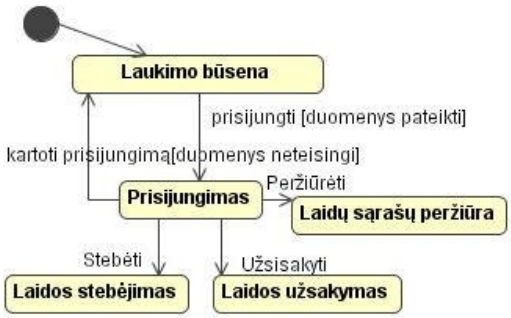
Pagrindinės sistemos esybės gali įgyti įvairias būsenas (jos pateiktos 3.23 – 3.25 paveiksluose), kurias gali keisti tiek laidų užsakymo, tiek laidų rodymo paslauga, todėl reikalingas paslaugų valdymo mechanizmas. Tokiu mechanizmu gali būti būsenų valdiklis, kuris prisimintų pagrindinių sistemos esybių būsenas ir valdytų į sistemą ateinančius pranešimų srautus, juos paskirstydamas reikiamoms paslaugoms. Būsenų valdikliais pagrįsta paslaugų sistema sudaroma analizuojant choreografijos sekų diagramas ir sujungiant jas į būsenų modelį, kuris vaizduoja vykdomą procesą ir padeda suderinti sąveikas tarp paslaugų.



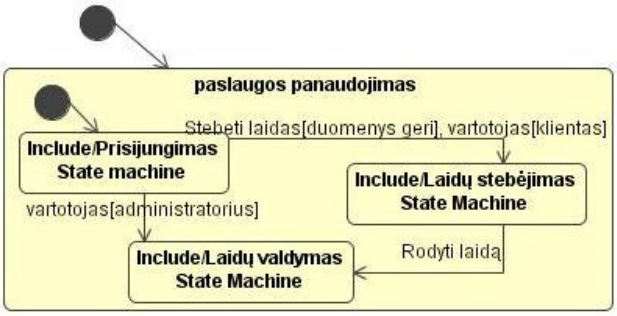


3.23 pav. Laidos būsenos 3.24 pav. Laidos užsakymo būsenos 3.25 pav. Kliento būsenos

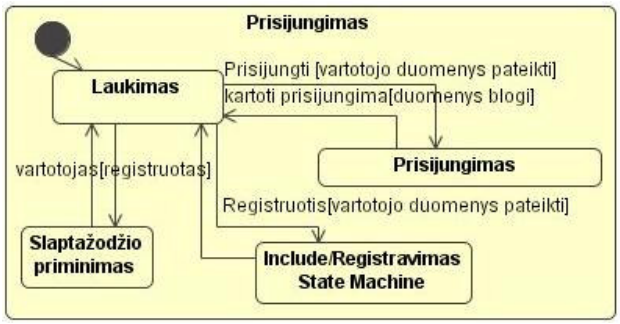
Laidų užsakymo paslaugos būsenų diagrama pateikta 3.26 paveiksle, o laidų rodymo paslaugos būsenų modelis – 3.27 – 3.33 paveiksluose.



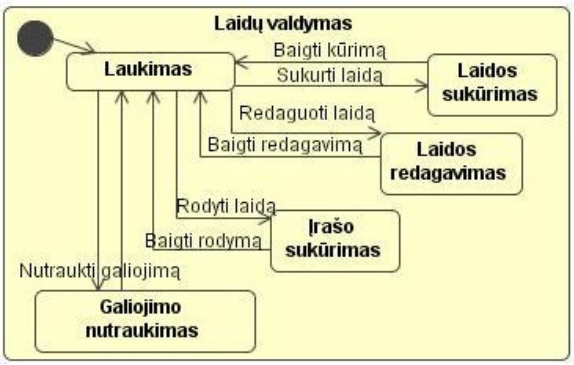
3.26 pav. Laidų užsakymo paslaugos būsenų modelis



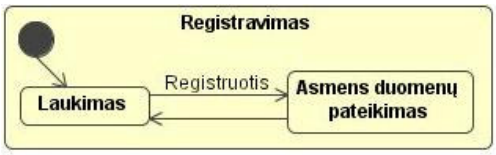
3.27 pav. Laidų rodymo paslaugos panaudojimo būsena



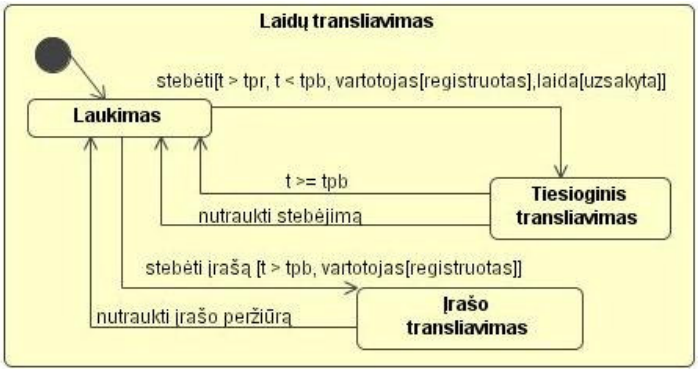
3.28 pav. Prisijungimo būsena



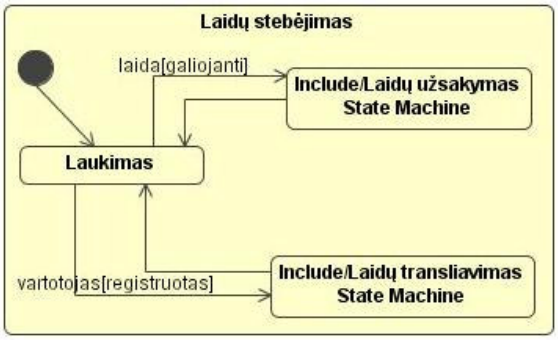
3.30 pav. Laidų valdymo būsena



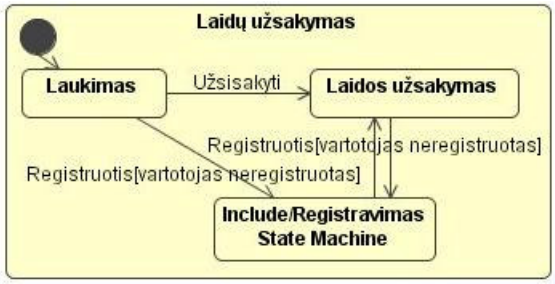
3.29 pav. Registravimo būsena



3.31 pav. Laidų transliavimo būsena



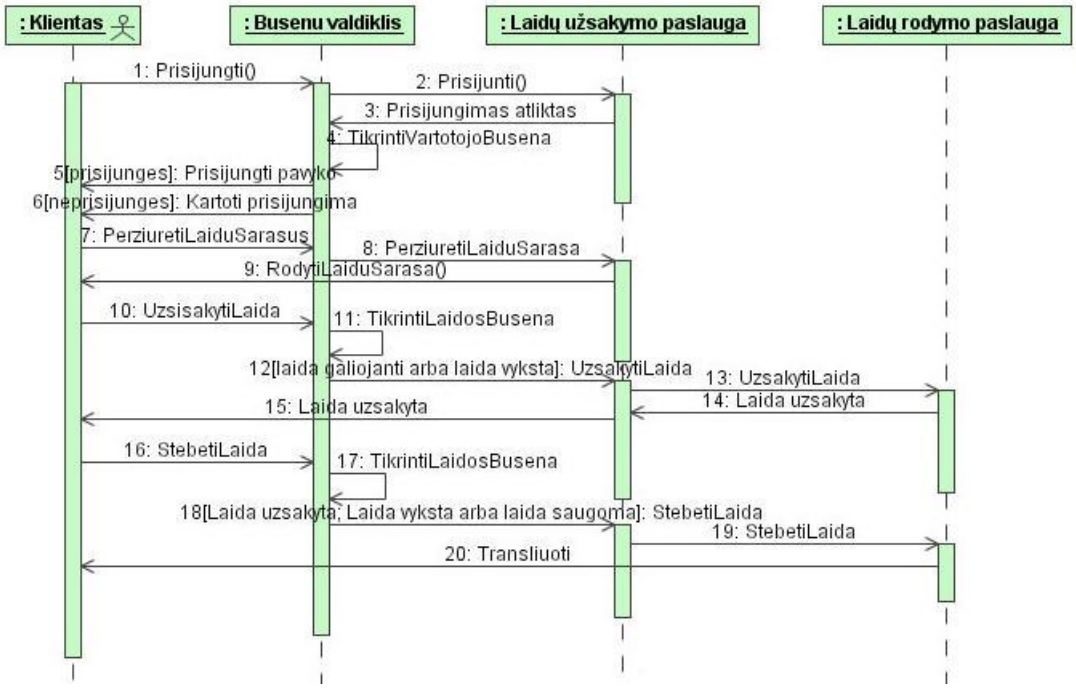
3.32 pav. Laidų stebėjimo būsena



3.33 pav. Laidų užsakymo būsena

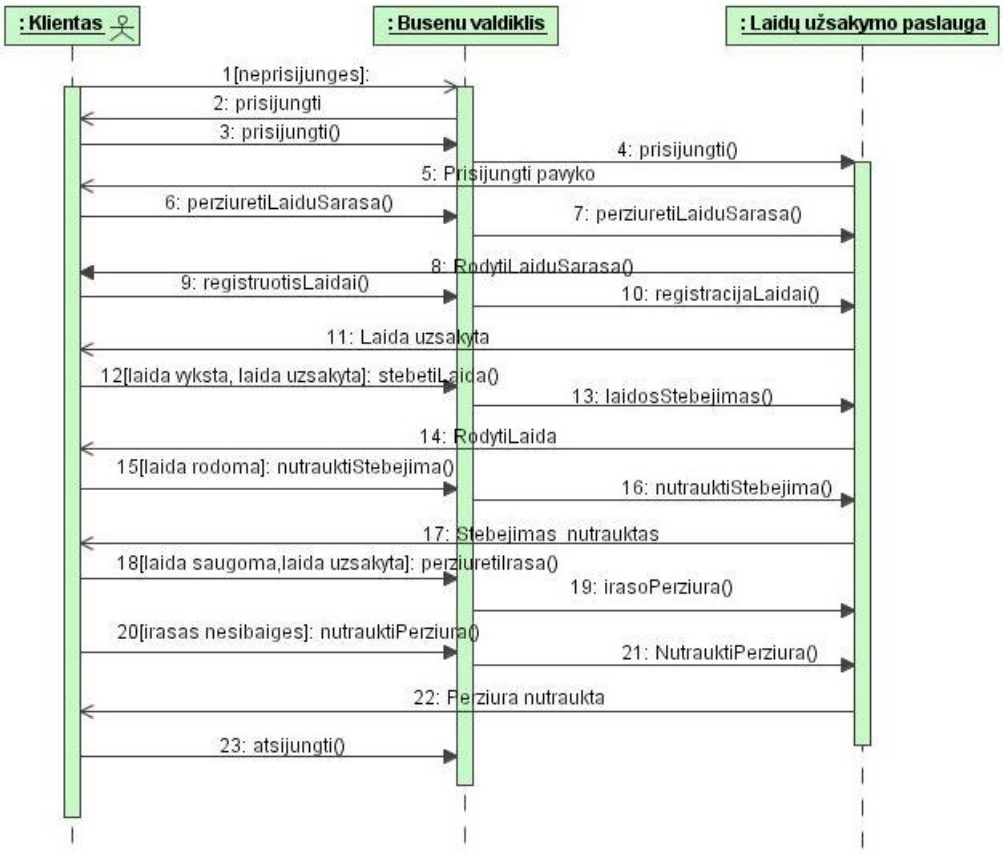
Paslaugos projektavimas

Vidinį paslaugų sistemos projektą sudaro būsenų valdiklio klasė, sąveikaujanti su laidų užsakymo ir laidų rodymo paslaugų klasėmis Paslaugų sistemos sekų diagrama, kai paslaugoms valdyti naudojamas būsenų valdiklis, parodyta 3.34 paveiksle.

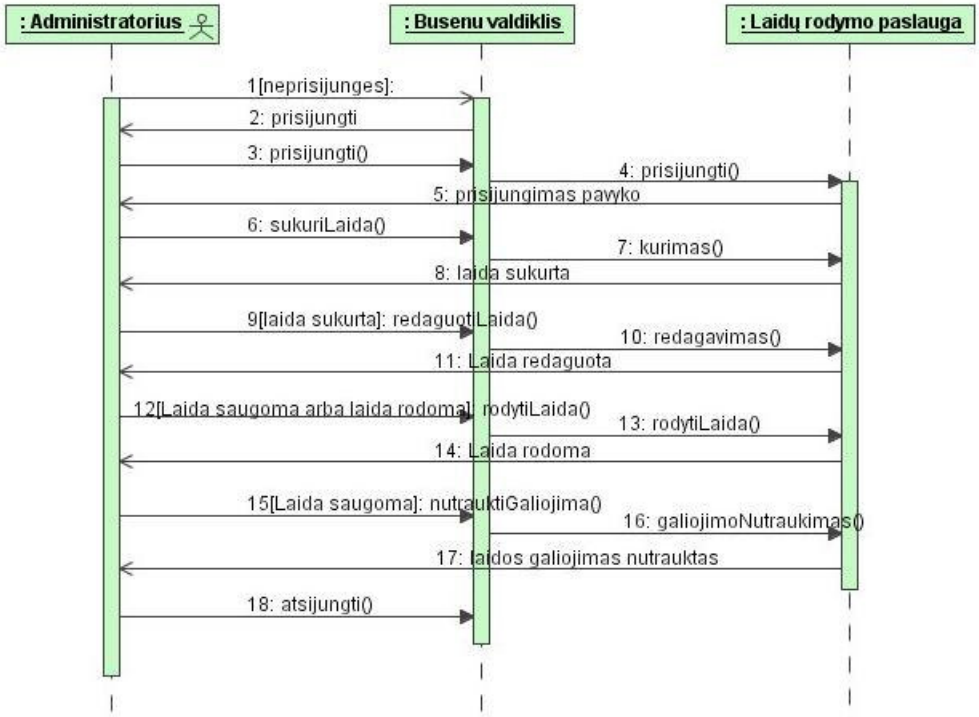


3.34 pav. Paslaugų sistemos sekų diagrama

Laidų užsakymo paslaugos sekų diagrama pateikta 3.35 paveiksle, o laidų rodymo paslaugos sekų diagrama – 3.36 paveiksle.

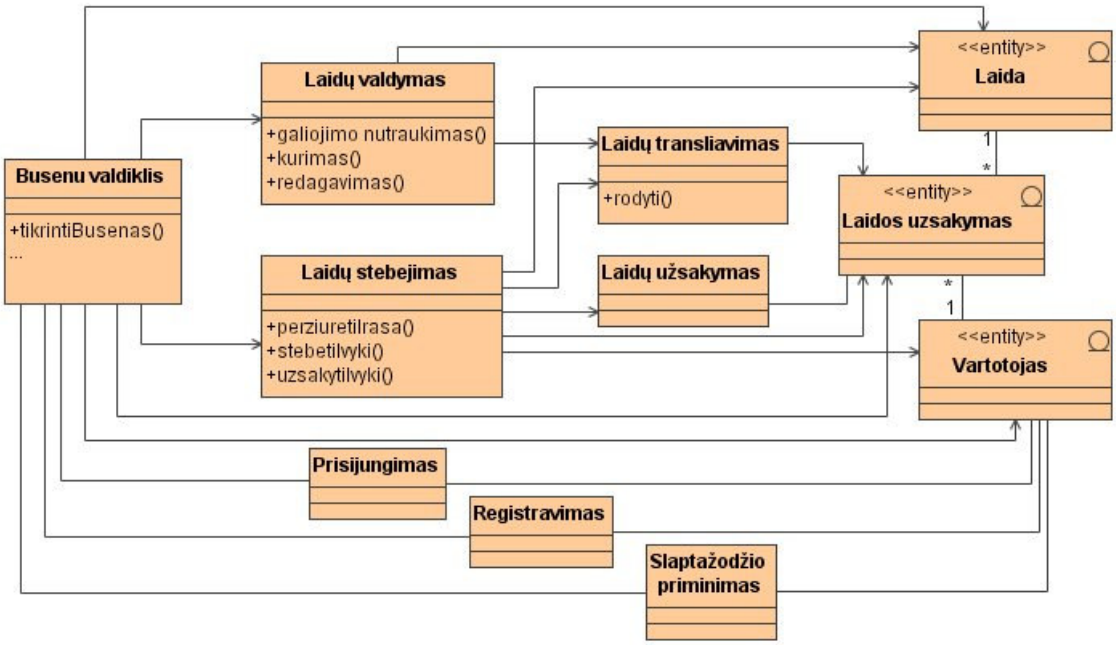


3.35 pav. Laidų užsakymo paslaugos sekų diagrama



3.36 pav. Laidų rodymo paslaugos sekų diagrama

Paslaugų sistemos (kompozicinės paslaugos) klasių diagrama parodyta 3.37 paveiksle.



3.37 pav. Paslaugų sistemos klasių diagrama

3.4 Reikalavimai ir rekomendacijos tinklo paslaugų projektavimo CASE įrankiams

Siūlomai metodikai realizuoti reikalingas CASE įrankis. Šis įrankis turėtų palaikyti RUP išplėtimus ir turėti galimybę projektuoti įprastas UML diagramas:

- panaudojimo atvejų (angl. *Use Case*),

- klasių (angl. *Class*),
- sekų (angl. *Sequence*),
- būsenų (angl. *State*),
- veiklos (angl. *Activity*),
- realizacijos (angl. *Implementation*).

Be galimybės kurti pagrindines diagramas, CASE įrankis turėtų palaikyti paslaugoms projektuoti reikalingus UML išplėtimus ir suteikti galimybes projektuoti paslaugas – kurti WSDL diagramas. Paslaugų projektavimo „Realizacija -WSDL“ bei „WSDL - Realizacija“ metodams reikalinga, kad projektavimo įrankis turėtų galimybę atlikti tiesioginę (WSDL dokumento generavimui iš turimos WSDL diagramos) ir atvirkštinę (WSDL diagramos generavimui iš turimo WSDL dokumento) kodo inžineriją.

Rekomenduojama, kad projektavimo įrankis palaikytų naujausią UML notaciją, kuri suteikia daugiau projektavimo galimybių. Šiuo metu naujausia ir kai kuriuose įrankiuose palaikoma yra UML 2.0 notacija.

Kadangi paslaugos dinamiškai komponuojamos atsižvelgiant į sistemos esybių būsenas, todėl būtų pravartu atlikti būsenų mašinų patikrinimą, tačiau šiame darbe būsenų mašinų verifikavimas nenagrinėjamas.

Geras kandidatas paslaugoms projektuoti yra MagicDraw įrankis, kuris pasižymi visomis anksčiau minėtomis savybėmis, turi duomenų bazių ir XML schemų generavimo galimybes, plėtimo API, be to, yra nuolat tobulinamas.

IŠVADOS

1. Išnagrinėti reikalavimai paslaugų projektavimo metodui.
2. Suformuota paslaugų projektavimo metodika:
    a. Vienos paslaugos projektavimo etapai:
        i. veiklos modeliavimas,
        ii. reikalavimų analizė,
        iii. sąsajų identifikavimas,
        iv. sąsajų sąveikų analizavimas,
        v. komponentų specifikavimas,
        vi. paslaugos modelio (WSDL diagramos) sudarymas.
    b. Paslaugų sistemos projektavimo etapai:
        i. panaudojimo atvejų modeliavimas,
        ii. sąsajų identifikavimas,
        iii. viešų sąveikų sekų (choreografijos) modeliavimas,
        iv. veiklos modeliavimas,
        v. kompozicinės paslaugos modelio sudarymas.
3. Siūlomos paslaugų projektavimo metodikos etapams realizuoti panaudotas standartinės UML notacijos klasių, sekų, būsenų diagramų rinkinys, šias diagramas suderinant tarpusavyje ir papildoma WSDL diagrama.
4. Sistemai sudaryti įvestas būsenų valdiklis, kuris pagal aprašytus paslaugų kontraktus leidžia arba draudžia naudoti paslaugas.
5. Siūloma metodika sujungia objektinio projektavimo, veiklos modeliavimo, organizacijų bei paslaugų architektūros principus. Ji leidžia aprašyti paslaugoms būdingus modelius UML diagramomis ir suderinti jas tarpusavyje.
6. Pateikta metodika buvo pritaikyta MagicDraw įrankyje, kuris teikia galimybes kurti WSDL modelius, generuoti WSDL dokumentus. Šią metodiką galėtų naudoti projektuotojai, kuriantys sistemas, kur naudojamos paslaugų technologijos.

LITERATŪRA

[1] *About UDDI* [interaktyvus] Oasis UDDI. 2004, [žiūrėta 2004 05 20]. Prieiga per internetą: <www.uddi.org>.

[2] ACHOUR, M.; BETZ, F.; DOVGAL, A.; *PHP manual* [interaktyvus] PHP: Hypertext preprocessor. 2005, [žiūrėta 2005 04 12]. Prieiga per internetą: <http://www.php.net/>.

[3] ARSANJANI, A. *Service-oriented modelling and architecture* [interaktyvus] IBM developerWorks. 2004, [žiūrėta 2005 04 20]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/library/ws-soa-design1/>.

[4] ASADUZZAMAN, *A. Building XML Web Services with PHP NuSOAP* [interaktyvus] Dev Articles. 2003, [žiūrėta 2004 04 10]. Prieiga per internetą: <http://www.devarticles.com/c/a/PHP/Building-XML-Web-Services-with-PHP-NuSOAP>.

[5] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. The Unified Modeling Language User Guide. Addison Wesley, 2000.

[6] BOOTH, D.; HAAS, H.; NEWCOMER, E. *Web Services Architecture* [interaktyvus] World Wide Web Consortium. W3C Working Group Note 11 February 2004. 2004 vasaris, [žiūrėta 2004 11 6]. Prieiga per internetą: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.

[7] CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. *Web Services Description Language (WSDL) 1.1* [interaktyvus] World Wide Web Consortium. W3C Note 15 March 2001. 2001 kovas, [žiūrėta 2004 03 03]. Prieiga per internetą: <http://www.w3.org/TR/wsdl>.

[8] CLARK, J.; MAKOTO, M. *RELAX NG Specification* [interaktyvus] RELAX NG home page. 2001, [žiūrėta 2004 03 20]. Prieiga per internetą: <http://www.relaxng.org/>.

[9] COLAN, M. *Service-Oriented Architecture expands the vision of Web services* [interaktyvus] IBM developerWorks. 2004, [žiūrėta 2004 12 02]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/library/ws-soaintro.html>.

[10] ENDREI, M.; ANG, J.; ARSANJANI, A. *Patterns: Service-Oriented Architecture and Web Services* [interaktyvus] IBM RedBooks. 2004, [žiūrėta 2005 04 20]. Prieiga per internetą: <http://www.redbooks.ibm.com/redbooks/SG246303>.

[11] *Extensible Markup Language (XML)* [interaktyvus] World Wide Web Consortium. 2004, [žiūrėta 2004 03 06]. Prieiga per internetą: <http://www.w3.org/XML/>.

[12] HE, H. *What is Service-Oriented Architecture?* [interaktyvus] O'Reilly WebServices.xml.com. 2002, [žiūrėta 2004 03 10]. Prieiga per internetą: <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>.

[13] *Introducing MagicDraw* [interaktyvus] No Magic Inc. 2005, [žiūrėta 2005 04 29]. Prieiga per internetą: <http://www.magicdraw.com>.

[14] KAUFMAN, S. *Application Development: Consuming Web services in PHP* [interaktyvus]. 2004, [žiūrėta 2005 04 29]. Prieiga per internetą: <http://builder.com.com/5100-6389_14-5164884.html>.

[15] *MySQL Reference Manual* [interaktyvus] MySQL. [žiūrėta 2004 04 02]. Prieiga per internetą: <http://www.mysql.com>.

[16] *New to SOA and Web services* [interaktyvus] IBM developerWorks. 2004, [žiūrėta 2004 11 28]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/webservices/newto/>.

[17] PARIKH, A.; PRADHAN, R.; SHAH, N. *Modeling of Web Services: A Standards –Based Approach* [interaktyvus] SOFTWARE MAG.COM The IT Software Journal. Application Development Feature. 2004 gegužė, [žiūrėta 2005 04 20]. Prieiga per internetą: <http://www.softwaremag.com/L.cfm?Doc=2004-05/2004-05modeling-web-svcs>.

[18] RIMAVIČIŪTĖ, L.; NEMURAITĖ, L. *Interneto paslaugų sistemų projektavimo metodika*: Konferencijos „Informacinės technologijos 2005“ pranešimų medžiaga. Kaunas, Technologija, 2005, p. 83 – 86.

[19] *Simple Object Access Protocol (SOAP) 1* [interaktyvus] World Wide Web Consortium. W3C Note 08 May 2000. 2000 gegužė, [žiūrėta 2004 03 06]. Prieiga per internetą: <http://www.w3.org/TR/soap/>.

[20] *UML Products by Company* [interaktyvus] Object By Design. 2004, [žiūrėta 2005 01 26]. Prieiga per internetą: <http://www.objectsbydesign.com/tools/umltools_byCompany.html>.

[21] WESSBERG, M. *RUP Introducing the IBM Rational Unified Process essentials by analogy* [interaktyvus] IBM developerWorks. 2005, [žiūrėta 2005 04 29 d.]. Prieiga per internetą: <http://www-128.ibm.com/developerworks//rational/library/05/wessberg/>.

[22] *XML Schema* [interaktyvus] World Wide Web Consortium. 2004, [žiūrėta 2004 03 20]. Prieiga per internetą: <http://www.w3.org/XML/Schema>.

[23] YANK, K. *Web Services Demystified* [interaktyvus] SitePoint. 2002, [žiūrėta 2004 03 10]. Prieiga per internetą: <http://www.sitepoint.com/article/web-services-demystified>.

[24] ZIMMERMANN, O.; KROGDAHL, P.; GEE, E. *Elements of Service-Oriented Analysis and Design* [interaktyvus] IBM developerWorks. 2004, [žiūrėta 2005 04 20]. Prieiga per internetą: <http://www-106.ibm.com/developerworks/webservices/library/ws-soad1/>.

TERMINŲ IR SANTRUMPŲ ŽODYNAS

XML (*Extensible Markup Language*)	Išplėstoji žymių kalba
OOAD (*Object-Oriented Analysis and Design*)	Objektinės analizės ir projektavimo metodas
EA (*Enterprise Architecture*)	Organizacijų architektūra.
BPM (*Business Process Modelling)*	Veiklos procesų modeliavimas
URI (*Unified Resource Identifier*)	Visuotinis resurso identifikatorius
UDDI (*Universal Description, Discovery, and Integration*)	Universalus aprašymo, atradimo ir integravimo protokolas
WSDL (*Web Service Description Language*)	Tinklo paslaugų aprašymo kalba
SOAP (*Simple Object Access Protocol*)	Pasikeitimo pranešimais protokolas
HTTP (*Hypertext Transport Protocol*)	Protokolas, naudojamas dokumentų perdavimui pasauliniame žiniatinklyje
TCP/IP (*Transmission Control Protocol/Internet Protocol*)	Standartinis tinklo protokolas, leidžiantis kompiuteriams pasiekti internetą.
SOA (*Service-Oriented Architecture*)	Paslaugų architektūra
SOAD (*Service-Oriented Analysis and Design*)	Paslaugų analizės ir projektavimo metodas
MagicDraw	Projektavimo įrankis
UML (*Unified Modelling Language*)	Universali modeliavimo kalba
PHP (*Hypertext Pre-processor*)	Programavimo kalba
W3C (*WWW Consortium*)	Pasaulinio žiniatinklio konsorciumas
OMG (*Object Management Group*)	Objektų valdymo grupė
REST (*Representational State Transfer*)	Tinklo programų kūrimo architektūrinis stilius
ESB (*Enterprise Service Bus*)	Organizacijos paslaugų magistralė
QoS (*Quality of Service*)	Paslaugų kokybė
BPEL (*Business Process Execution Language for Web Services*)	Veiklos procesų vykdymo kalba skirta tinklo paslaugoms
RUP (*Rational Unified Process*)	Programinės įrangos kūrimo (projektavimo, testavimo ir t.t.) procesas
MySQL	Duomenų bazių valdymo sistema

1 PRIEDAS. WSDL dokumentas

Realizavus eksperimentinę sistemą sugeneruotas WSDL dokumentas:

```
    <?xml version="1.0" encoding="ISO-8859-1" ?>
    <definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:si="http://soapinterop.org/xsd"
xmlns:tns="http://localhost/web_service" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://localhost/web_service">
       <types>
          <xsd:schema targetNamespace="http://localhost/web_service">
             <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
             <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
             <xsd:complexType name="StrArr">
                <xsd:complexContent>
                   <xsd:restriction base="SOAP-ENC:Array">
                      <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string[]" />
                   </xsd:restriction>
                </xsd:complexContent>
             </xsd:complexType>
             <xsd:complexType name="Event">
                <xsd:all>
                   <xsd:element name="pradzia" type="xsd:string" />
                   <xsd:element name="pabaiga" type="xsd:string" />
                   <xsd:element name="aprasymas" type="xsd:string" />
                   <xsd:element name="v_id" type="xsd:string" />
                   <xsd:element name="asmeninis" type="xsd:string" />
                   <xsd:element name="t_id" type="xsd:int" />
                   <xsd:element name="pavadinimas" type="xsd:string" />
                   <xsd:element name="paslepimas" type="xsd:string" />
                   <xsd:element name="keiciamas" type="xsd:string" />
                </xsd:all>
             </xsd:complexType>
             <xsd:complexType name="EventsArr">
                <xsd:complexContent>
                   <xsd:restriction base="SOAP-ENC:Array">
                      <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:Event[]" />
                   </xsd:restriction>
                </xsd:complexContent>
             </xsd:complexType>
             <xsd:complexType name="EventType">
                <xsd:all>
                   <xsd:element name="t_id" type="xsd:int" />
                   <xsd:element name="pavadinimas" type="xsd:string" />
                </xsd:all>
             </xsd:complexType>
             <xsd:complexType name="TypesArr">
                <xsd:complexContent>
                   <xsd:restriction base="SOAP-ENC:Array">
                      <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:EventType[]" />
                   </xsd:restriction>
                </xsd:complexContent>
             </xsd:complexType>
          </xsd:schema>
       </types>
       <message name="ar_yra_irasasRequest">
          <part name="asm" type="xsd:string" />
          <part name="vart" type="xsd:string" />
          <part name="met" type="xsd:string" />
          <part name="men" type="xsd:string" />
          <part name="pirmdie" type="xsd:string" />
       </message>
       <message name="ar_yra_irasasResponse">
          <part name="return" type="tns:StrArr" />
       </message>
       <message name="ivykio_vietaRequest">
          <part name="asm" type="xsd:string" />
          <part name="vart" type="xsd:string" />
          <part name="met" type="xsd:string" />
          <part name="men" type="xsd:string" />
          <part name="die" type="xsd:string" />
          <part name="datulee" type="xsd:string" />
          <part name="dienlaik" type="tns:StrArr" />
       </message>
       <message name="ivykio_vietaResponse">
          <part name="return" type="tns:EventsArr" />
```

```
        </message>
        <message name="get_AllTypesRequest" />
        <message name="get_AllTypesResponse">
           <part name="return" type="tns:TypesArr" />
        </message>
        <message name="tikrinti_laikaRequest">
           <part name="asm" type="xsd:string" />
           <part name="vart" type="xsd:string" />
           <part name="met" type="xsd:string" />
           <part name="men" type="xsd:string" />
           <part name="die" type="xsd:string" />
           <part name="pr" type="xsd:string" />
           <part name="pb" type="xsd:string" />
        </message>
        <message name="tikrinti_laikaResponse">
           <part name="return" type="tns:Event" />
        </message>
        <message name="ivesti_ivykiRequest">
           <part name="asm" type="xsd:string" />
           <part name="vart" type="xsd:string" />
           <part name="met" type="xsd:string" />
           <part name="men" type="xsd:string" />
           <part name="die" type="xsd:string" />
           <part name="tip" type="xsd:string" />
           <part name="apr" type="xsd:string" />
           <part name="pr" type="xsd:string" />
           <part name="pb" type="xsd:string" />
           <part name="pasleptas" type="xsd:string" />
           <part name="keic" type="xsd:string" />
        </message>
        <message name="ivesti_ivykiResponse">
           <part name="return" type="xsd:string" />
        </message>
        <message name="prisijungtiRequest">
           <part name="vart" type="xsd:string" />
           <part name="slapt" type="xsd:string" />
        </message>
        <message name="prisijungtiResponse">
           <part name="return" type="xsd:string" />
        </message>
        <portType name="manoPortType">
           <operation name="ar_yra_irasas">
              <input message="tns:ar_yra_irasasRequest" />
              <output message="tns:ar_yra_irasasResponse" />
           </operation>
           <operation name="ivykio_vieta">
              <input message="tns:ivykio_vietaRequest" />
              <output message="tns:ivykio_vietaResponse" />
           </operation>
           <operation name="get_AllTypes">
              <input message="tns:get_AllTypesRequest" />
              <output message="tns:get_AllTypesResponse" />
           </operation>
           <operation name="tikrinti_laika">
              <input message="tns:tikrinti_laikaRequest" />
              <output message="tns:tikrinti_laikaResponse" />
           </operation>
           <operation name="ivesti_ivyki">
              <input message="tns:ivesti_ivykiRequest" />
              <output message="tns:ivesti_ivykiResponse" />
           </operation>
           <operation name="prisijungti">
              <input message="tns:prisijungtiRequest" />
              <output message="tns:prisijungtiResponse" />
           </operation>
        </portType>
        <binding name="manoBinding" type="tns:manoPortType">
           <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
           <operation name="ar_yra_irasas">
              <soap:operation soapAction="http://localhost/web_service/serv.php/ar_yra_irasas"
style="rpc" />
              <input>
                 <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
              </input>
              <output>
                 <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
              </output>
           </operation>
```

```
          <operation name="ivykio_vieta">
             <soap:operation soapAction="http://localhost/web_service/serv.php/ivykio_vieta"
style="rpc" />
             <input>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </input>
             <output>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </output>
          </operation>
          <operation name="get_AllTypes">
             <soap:operation soapAction="http://localhost/web_service/serv.php/get_AllTypes"
style="rpc" />
             <input>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </input>
             <output>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </output>
          </operation>
          <operation name="tikrinti_laika">
             <soap:operation soapAction="http://localhost/web_service/serv.php/tikrinti_laika"
style="rpc" />
             <input>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </input>
             <output>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </output>
          </operation>
          <operation name="ivesti_ivyki">
             <soap:operation soapAction="http://localhost/web_service/serv.php/ivesti_ivyki"
style="rpc" />
             <input>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </input>
             <output>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </output>
          </operation>
          <operation name="prisijungti">
             <soap:operation soapAction="http://localhost/web_service/serv.php/prisijungti"
style="rpc" />
             <input>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </input>
             <output>
                <soap:body use="encoded" namespace="http://localhost/web_service"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
             </output>
          </operation>
       </binding>
       <service name="mano">
          <port name="manoPort" binding="tns:manoBinding">
             <soap:address location="http://localhost/web_service/serv.php" />
          </port>
       </service>
    </definitions>
```

2 PRIEDAS. Straipsnis „Interneto paslaugų sistemų projektavimo metodika“

INTERNETO PASLAUGŲ SISTEMŲ PROJEKTAVIMO METODIKA

Laura Rimavičiūtė, vadovė doc. Lina Nemuraitė
Kauno technologijos universitetas, Informacijos sistemų katedra
Studentų g. 50 – 308, LT-51368, Kaunas

Vis daugiau programinės įrangos kuriama interneto paslaugų pavidale. Tokios paslaugos plačiai naudojamos bankininkystėje, finansų, draudimo, turizmo industrijoje, elektroniniame versle ir pan. Tinklo paslaugos (angl. Web Services) atsirado palyginti neseniai, todėl šioje srityje susiduriama su tam tikromis problemomis, būdingomis naujoms technologijoms: idėjų plėtojimas vyksta technologiniame lygmenyje, o modeliavimo bei projektavimo metodai atsiranda vėliau. Straipsnyje pateikiama tinklo paslaugų projektavimo metodika, skirta pavienių paslaugų projektavimui bei jau esamų paslaugų panaudojimui kuriant informacinių sistemų projektus.

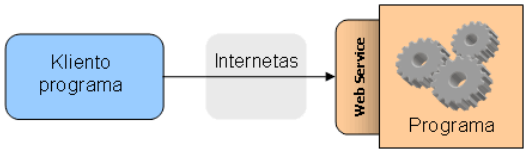
1 Įvadas

Vis daugiau programinės įrangos kuriama interneto paslaugų pavidale. Tokios paslaugos plačiai naudojamos bankininkystėje, draudimo, turizmo industrijoje, elektroniniame versle ir pan. Prognozuojama, kad ateities programos jau neturės didelių sukompiliuotų paleidžiamųjų kodų (EXE, DLL ir kt). Programos bus padalintos į mažesnes paslaugas, kurias paprasčiau sukurti, palaikyti ir valdyti.

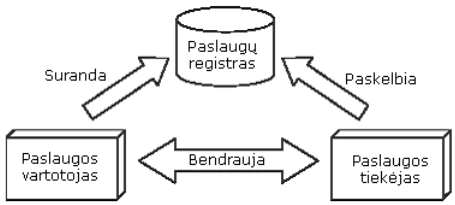
Kadangi tinklo paslaugų (angl. Web Service) architektūra yra gana nauja, todėl nėra vieningos modeliavimo metodikos. Straipsnyje pateikiama siūloma tinklo paslaugų projektavimo metodika, skirta pavienių paslaugų projektavimui bei jau esamų paslaugų panaudojimui kuriant informacinių sistemų projektus.

2 Tinklo paslaugų technologijos

Tinklo paslauga – tai programinės įrangos dalis, prieinama per internetą (1 paveikslas). Išsamų tinklo paslaugos apibrėžimą pateikia WWW Consortium (W3C): tinklo paslauga yra programa, identifikuojama URI (Unified Resource Identifier). Jos sąsajos ir ryšiai aprašomi bei randami panaudojant XML [5] artefaktus. Ji tiesiogiai sąveikauja su kitomis programomis, naudodama XML pranešimus ir interneto protokolus [6].



1 pav. Supaprastintas tinklo paslaugos modelis



2 pav. Paslaugų architektūra

Tinklo paslaugos leidžia programoms dalintis duomenimis bei pasinaudoti kitų programų teikiamomis galimybėmis nepriklausomai nuo to, kaip šios programos buvo sukurtos, kokioje operacinėje sistemoje ar platformoje jos dirba ir kokie įrenginiai reikalingi toms programoms pasiekti.

Tinklo paslaugos yra neatsiejamos nuo paslaugų architektūros, kuri pateikia konceptualų tinklo paslaugų modelį, padeda suprasti tinklo paslaugas, nustato ryšius tarp konceptualaus modelio komponentų, tačiau nepateikia specifikacijų tinklo paslaugos realizacijai ir neapriboja jų tarpusavio sujungimo galimybių.

Tinklo paslauga – tai resursas, apibūdinamas abstrakčia teikiamo funkcionalumo aibe. Paslaugą realizuoja agentas – konkreti programinės ar aparatūrinės įrangos dalis, siunčianti ir gaunanti pranešimus. Įvykus pokyčiams agente (pavyzdžiui, jį suprogramavus kita kalba), tinklo paslauga nepakinta.

Tinklo paslaugos pranešimų pasikeitimo mechanizmas dokumentuojamas tinklo paslaugos aprašymu tinklo paslaugų aprašymo kalba WSDL (Web Service Description Language) [3]. Ji nustato pranešimų formatus bei šablonus, duomenų tipus, transportavimo protokolus, kurie turi būti naudojami bendravimui tarp paslaugos vartotojo ir paslaugos tiekėjo realizuoti. Specifikacija taip pat nurodo vieną ar kelis tinklo adresus, kur paslaugos tiekėjas gali būti iškviestas. Be WSDL aprašo, tinklo paslauga turi apibrėžtą semantiką – tai bendrai aprašytas laukiamas paslaugos elgesys atsakant į vartotojo siųstus pranešimus.

Paslaugų architektūra (angl. Service-Oriented Architecture – SOA) [4] nėra naujas dalykas, tai alternatyva labiau įprastiems, stipriai susietiems objektiniams modeliams. Paslaugų architektūra pagrįstos sistemos gali būti sudarytos iš pakartotinai naudojamų paslaugų, sukurtų remiantis objektiniais projektavimo principais, tačiau ji pati nėra objektinė. Skirtumas tarp šių dviejų architektūrų – tai sąsaja, jungianti skirtingo funkcionalumo programos elementus. Sąsaja neturi priklausyti nuo naudojamos techninės įrangos, operacinės sistemos ir programavimo kalbos, kuria paslauga yra realizuota. Toks neutralumas vadinamas laisvu paslaugų susiejimu. Sistemos, kurių komponentai jungiami laisvai susiejant, pasižymi

lankstumu, joms lengviau pritaikomi įvairūs struktūriniai bei vidiniai realizacijos pakeitimai. Paslaugų architektūra pateikta 2 paveiksle.

Kuriant paslaugų sistemos architektūros modelį nepakanka tik paslaugos aprašymo. Turi būti apibrėžta, kaip sistemoje bus valdomi darbų srautai tarp įvairių paslaugų.

Pagrindiniai paslaugų architektūros privalumai: paprastesnis integravimas ir valdymas, trumpesnis sukūrimo laikas, mažesnė kaina ir didesnės pakartotino naudojimo galimybės, lengvas keitimas ir tobulinimas.

Paslaugų architektūros realizacijos problemos

Pagrindinė paslaugų architektūros realizacijos problema yra ta, kad esami modeliavimo metodai (objektinė analizė ir projektavimas, organizacijų architektūros karkasai bei veiklos procesų modeliavimas) teikia dideles galimybes identifikuojant bei apibrėžiant reikiamas architektūrines abstrakcijas, tačiau paslaugų architektūros realizacijai to nepakanka.

Objektinė analizė ir projektavimas (angl. Object-Oriented Analysis and Design – OOAD) [7] – tai galingas metodas, į kurį paslaugų analizė ir projektavimas turėtų atsižvelgti. Pagrindinė objektinio projektavimo problema yra ta, kad jo išskaidymas pernelyg smulkus paslaugų modeliavimui. Stiprūs sąryšiai tvirtai susieja modelio dalis tarpusavyje. Tuo tarpu paslaugų projektavime, naudojant laisvą susiejimą, siekiama lankstumo bei judrumo. Šios aplinkybės rodo, kad objektinį projektavimą sunku tiesiogiai pritaikyti paslaugų architektūros kūrimui. Tačiau objektinis projektavimas naudingas paslaugos struktūros (klasių bei komponentų) modeliavimui.

Organizacijų architektūros karkasai (angl. Enterprise Architecture (EA) frameworks) [7] naudojami siekiant integruoti skirtingai realizuotas sistemos dalis, tačiau jie nepaliečia projektavimo srities.

Veiklos procesų modeliavimas (angl. Business Process Modeling – BPM) [7] teikia pilną funkcinių vienetų vaizdą, bet nenagrinėja architektūrinės bei realizacijos srities; veiklos procesų modeliai ne visada būna sinchronizuoti su panaudojimo atvejų modeliais. Be to, nei vienas iš veiklos procesų modeliavimo metodų nenurodo kaip pritaikyti esamas programas paslaugų architektūrai. Egzistuojančios sistemos paprastai turi didelį kiekį kritiškai svarbių duomenų bei saugo veiklos logiką, todėl jos negali būti taip paprastai pakeistos.

Paslaugų projektavimui siūloma naudoti paslaugoms skirtą analizės ir projektavimo metodą (angl. Service-Oriented Analysis and Design – SOAD) [1][7], sudarytą iš reikiamų objektinio projektavimo, organizacijų architektūros karkasų bei veiklos procesų modelio elementų, sujungiant juos su naujais elementais. Objektinis projektavimas programos lygyje nagrinėja klases ir objektus, o veiklos procesų modelis – įvykiais pagrįstus procesų modelius. Paslaugų analizės ir projektavimo metodas visa tai jungia. Tokie metodai priklauso ne nuo panaudojimo atvejų, o nuo veiklos įvykių bei procesų. Paslaugų analizės ir projektavimo modelis susideda iš trijų etapų: paslaugų identifikavimo, paslaugų specifikavimo bei paslaugų realizacijos.

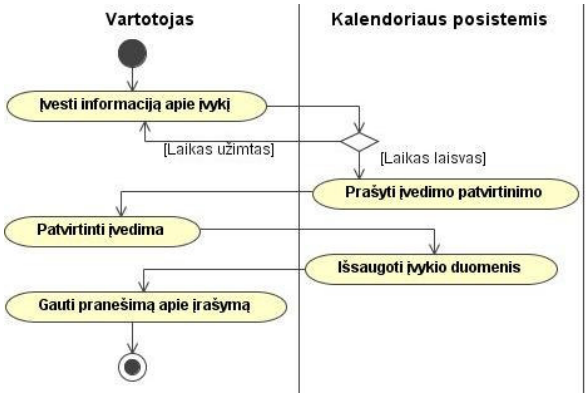
3 Tinklo paslaugų projektavimas

Vienos paslaugos projektavimas

Paslaugos projektavimo tikslas yra sudaryti paslaugos ir jos sąsajos modelį bei WSDL dokumentą, aprašantį paslaugą. Egzistuoja du paslaugų projektavimo būdai: sukuriamas paslaugos WSDL dokumentas, iš jo sugeneruojama WSDL diagrama ir ja remiantis modeliuojama paslauga; modeliuojama paslauga ir suprojektavus visą paslaugos funkcionalumą, kuriama WSDL diagrama, iš kurios generuojamas paslaugos WSDL dokumentas.

Panagrinėkime atvejį, kai pirmiau modeliuojama paslauga. 4 paveiksle parodytas kalendoriaus paslaugos modelis. Ši paslauga leidžia vartotojams sudaryti įvykių grafikus. Įvykiai gali būti labai įvairūs. Pavyzdžiui, toliau nagrinėjamame pavyzdyje ši paslauga naudojama internetu transliuojamų laidų planavimui ir valdymui.

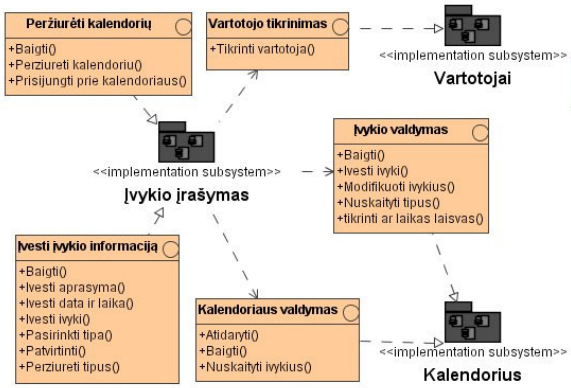
Projektuojant paslaugą, pirmiausiai modeliuojama jos veikla (3 paveiksle pateikta įvykio įrašymo veiklos diagrama), analizuojami reikalavimai, sudaromas veiklos konceptų modelis bei identifikuojami panaudojimo atvejai (4 paveikslas).



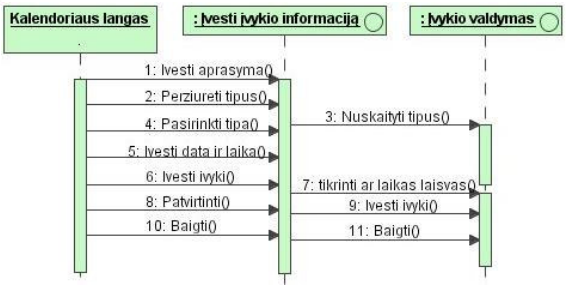
3 pav. Įvykio įrašymo proceso veiklos diagrama

4 pav. Panaudojimo atvejų diagrama

Išanalizavus veiklą bei panaudojimo atvejus, identifikuojami komponentai. Šiame etape pirmiausiai sudaromas veiklos tipų modelis, išskiriami esminiai veiklos tipai, jiems priskiriamos sąsajos, atsakingos už šių tipų informacijos apdorojimą. Pradinė komponentų architektūros specifikacija pateikta 5 paveiksle.



5 pav. Pradinė komponentų architektūros specifikacija



6 pav. Įvykio įrašymo sekos diagrama

Sudarius pradinį komponentų architektūros modelį, analizuojamos sąsajos, kad butų galima identifikuoti veiklos sąsajų operacijas. Tam sukuriamos sekų diagramos, specifikuojamos operacijos. Įvykio įrašymo sekos diagrama patekta 6 paveiksle.

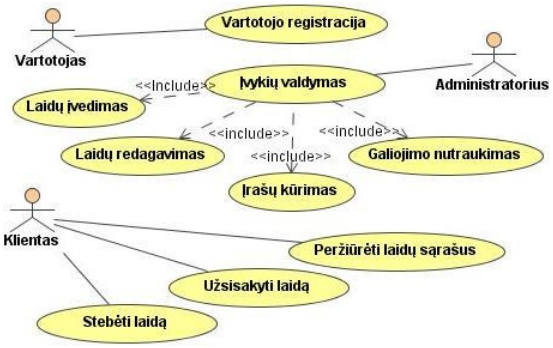
Identifikavus veiklos sąsajų operacijas, specifikuojami paslaugos komponentai, iš veiklos tipų modelio sudaromi sąsajų informaciniai modeliai. Tolesni etapai: sąsajų apibendrinimas ar specializavimas, komponentų vidinės struktūros projektavimas, realizacija, WSDL diagramos projektavimas, WSDL dokumento generavimas.

Kitas paslaugų projektavimo būdas – kai pirmiau turimas WSDL modelis. Šiuo atveju nagrinėjami WSDL modelyje aprašomi pranešimai, kurių pagalba klientas gali bendrauti su tinklo paslauga ir sudaroma sistemos specifikacija (veiklos, klasių modeliai ir t.t).

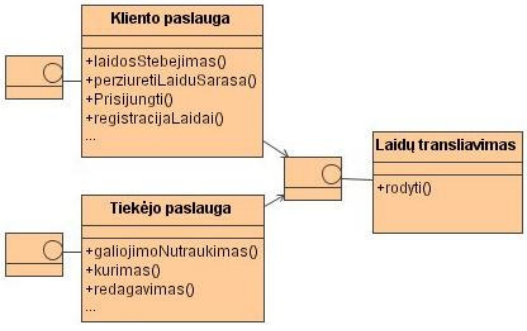
Paslaugų sistemos projektavimas

Paslaugos dažniausiai naudojamos jungiant jas su kitomis paslaugomis – kuriant paslaugų sistemas. Problema kyla dėl to, kad paslaugos negali išsaugoti savo būsenų, todėl reikalingas tam tikras mechanizmas joms valdyti. Tokiu mechanizmu gali būti būsenų valdiklis. Būsenų valdikliu paremto projektavimo proceso iliustravimui bus modeliuojama laidų (pavyzdžiui, konferencijų, seminarų ir pan.) transliuojamų internetu peržiūros sistema, kur prisijungęs vartotojas gali peržiūrėti laidų sąrašus ir užsiregistruoti norimos laidos peržiūrai. Užsisakytą laidą vartotojas gali stebėti jos transliavimo metu arba vėliau peržiūrėti vaizdo įrašą. Prie sistemos prisijungęs administratorius gali valdyti laidas (paskelbti naujas, redaguoti paskelbtas, peržiūrėti įvykusias, ar dar vykstančias laidas, praėjus tam tikram laikui po laidos, gali nutraukti jos galiojimą ir ši laida nebus skelbiama).

Projektuojant tinklo paslaugų sistemą visų pirma modeliuojami panaudojimo atvejai (7 paveikslas). Kitas žingsnis – paslaugų identifikavimas: sudaromas valdančių klasių (komponentų iš kurių konstruojamos paslaugos) modelis; iš valdančių klasių modelio identifikuojamos galimos paslaugos (8 paveikslas).



7 pav. Panaudojimo atvejų diagrama



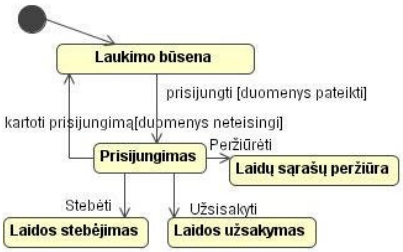
8 pav. Paslaugų modelis

Priešingai nei objektiniame projektavime, paslaugų sistemos modelis neatspindi funkcionalumo – nėra informacijos srautų, įvykių ar taisyklių aprašymų. Veiklos procesų vykdymą vaizduoja paslaugų choreografija.

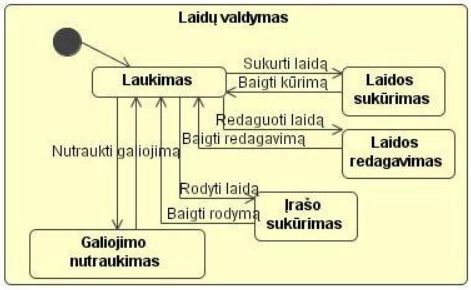
Veiklos modeliavimo kalbos naudoja veiklos diagramas, labiau tinkančias veiklos procesams, tačiau paslaugų sistemos veikimą geriau atspindi būsenų diagrama. 9 paveiksle pateikiama kliento paslaugos būsenų diagrama, o 10 paveiksle – vienos tiekėjo paslaugos būsenos modelis.

Pagrindinės sistemos esybės gali turėti įvairias būsenas, pavyzdžiui, laida gali būti sukurta, rodoma, saugoma, negaliojanti. Tam tikras esybių būsenas gali keisti tiek kliento, tiek tiekėjo paslauga, todėl būtinas sistemos elementas kuris prisimintų kitų esybių būsenas ir valdytų į sistemą ateinančių pranešimų srautus.

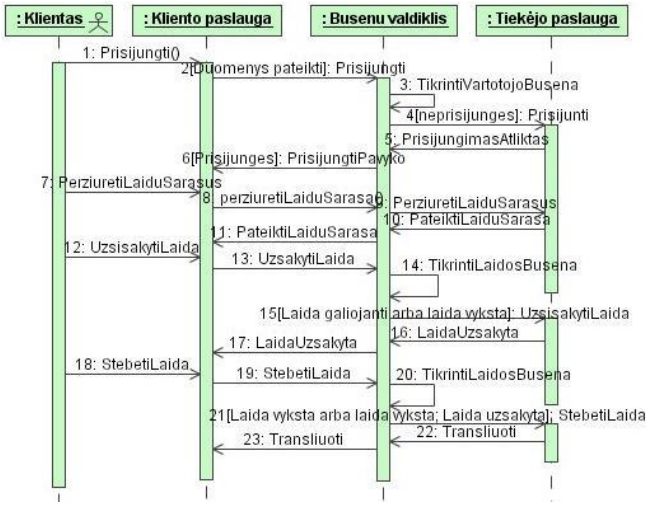
Paslaugų choreografija modeliuojama sekų diagramomis, supaprastintas modelis pateiktas 11 paveiksle.



9 pav. Kliento paslaugos būsenų diagrama



10 pav. Tiekėjo paslaugos laidų valdymo būsena



11 pav. Supaprastinta sistemos choreografija

4 Išvados

Straipsnyje pateikiama reikalavimų paslaugų projektavimo metodui analizė ir suformuota paslaugų projektavimo metodika, kuri susideda iš šių žingsnių:

- Vienos paslaugos projektavimo etapai: veiklos modeliavimas; reikalavimų analizė; komponentų identifikavimas; sąsajų sąveikų analizavimas; komponentų specifikavimas; paslaugos modelio sudarymas.
- Paslaugų sistemos projektavimo etapai: panaudojimo atvejų modeliavimas; paslaugų identifikavimas; veiklos modeliavimas; choreografijos modeliavimas.

Siūloma metodika sujungia objektinio projektavimo, veiklos modeliavimo, organizacijų karkasų bei paslaugų architektūros principus.

Pateikta metodika buvo pritaikyta MagicDraw įrankyje, kuris teikia WSDL modelių kūrimo, WSDL dokumentų generavimo galimybes. Ją galėtų naudoti projektuotojai kuriantys sistemas, kur naudojamos paslaugų technologijos.

Literatūros sąrašas

[1] **A. Arsanjani**. "Service-oriented modelling and architecture". Prieiga per internetą: http://www-106.ibm.com/developerworks/library/ws-soa-design1/

[2] **G.Booch**, **J.Rumbaugh**, **I.Jacobson**. The Unified Modeling Language User Guide, Addison Wesley, 2000.

[3] **E. Christensen**, **F. Curbera**, **G. Meredith**, **S. Weerawarana**. "Web Services Description Language (WSDL) 1.1". Prieiga per internetą: http://www.w3.org/TR/wsdl

[4] **M. Endrei**, **J. Ang**, **A. Arsanjani**. "Patterns: Service-Oriented Architecture and Web Services" Prieiga per internetą: http://www.redbooks.ibm.com/redbooks/SG246303/

[5] "Extensible Markup Language". Prieiga per internetą: http://www.w3.org/XML/

[6] "Web Services Architecture". Prieiga per Internetą: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[7] **O. Zimmermann**, **P. Krogdahl**, **C. Gee**. "Elements of Service-Oriented Analysis and Design". Prieiga per internetą: http://www-106.ibm.com/developerworks/webservices/library/ws-soad1/

Method for development of systems comprised of services

Web services are widely used in banking, finance, insurance, tourism, e-business sectors and so on. Web services have emerged only recently, so the issues are typical for new technologies: ideas had arisen on technological level, and methods for modelling and development are coming only later. In this article the methodology is proposed for design of stand-alone services and combining them into larger system.