VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Semantiniais web servisais pagrįstų sistemų analizė ir kūrimas

Analysis and Implementation of Semantic Web Services Based Systems

Magistro baigiamasis darbas

Atliko:
Karolis Lesickas (parašas)

Darbo vadovas:
a. Linas Būtėnas (parašas)

Recenzentas:
a. Mindaugas Plukas (parašas)

Vilnius
2009

Santrauka

Informacijos internete sparčiai daugėja, o esanti informacija greitai kinta. Susisteminti reikalingą informaciją ir ją panaudoti tokiomis sąlygomis yra sudėtinga. Daugėja ir web servisų teikiančių duomenis ar atliekančių veiksmus. Norint šiuos servisus efektyviai panaudoti jau reikia ne vien web servisų naudojamos sintaksės aprašo bet ir semantikos (prasmės) aprašo. Darbe nagrinėjamos galimybės kurti web servisų sistemas pasitelkiant veiklos srities formalius aprašymus – ontologijas. Nagrinėjamas ontologijos ir web serviso aprašo suderinimas – anotavimas. Taip pat pasiūlytos kelios skirtingos architektūros sistemų realizavimui, aprašomi šių architektūrų privalumai ir trūkumai, siūlomi patobulinimai. Pagrindinis dėmesys skiriamas web servisų paieškai ir jų iškvietimui naudojantis semantiniais aprašais.

Summary

Information on the Internet is growing fast, but the information is changing rapidly. Arrange the necessary information and its use in such conditions is difficult. A number of web services that provides data or perform actions is increasing rapidly. In order to effectively use these services it is needed to use not only syntactic description of web services but also a semantic description. The paper explores the possibilities to create Web services systems through the scope of using formal activities description - ontology. Examined ontology and web service mappings - annotation. In the paper it is also proposed a number of different systems realization architectures, describes the architectures advantages and disadvantages and proposed improvements. Main focus is on web services search and invocation.

Turinys

Įvadas

Informacijos internete sparčiai daugėja, o jau esanti informacija greitai kinta. Susisteminti reikalingą informaciją ir ją panaudoti tokiomis sąlygomis yra sudėtinga. Daugėja ir web servisų teikiančių duomenis ar atliekančių veiksmus. Web servisai jau spėjo įsitvirtinti kaip neoficialus standartas informacijos apsikeitimui su trečiomis šalimis internete. Taip įvyko todėl, kad web servisai nėra priklausomi nuo programavimo platformos ar juos palaikančios programinės įrangos, tačiau turi aiškiai apibrėžtą ir patogiai valdomą bendravimo procesą. Taip pat intensyviai plinta ir servisais paremtos sistemos (SOA – angl. Service oriented application). Kai šalys iš anksto susitaria, jog bendravimas tikrai vyks, kokia serviso paskirtis ir kokie saugumo reikalavimai, dabartiniu, nesemantinių web servisų visiškai pakanka norint įgyvendinti servisais paremtą sistemą. Tačiau norint sukurti globalią, dinamišką, adaptyvią sistemą dabar plačiai naudojamų serviso aprašymų WSDL (angl. WebService Definition Languge) nepakanka. Tam, kad SOA sugebėtų betarpiškai pasirinkti reikiamus servisus, nustatytų bendravimo ypatumus ir saugumo reikalavimus web servisai turėtų turėti semantinį aprašymą.

Plačiausiai naudojami, XML kalba paremti standartai, kurie skirti web servisų bendravimui, aprašo tik techninės web servisų detales, tačiau neaprašo semantinės pranešimų reikšmės. Pavyzdys galėtų būti WSDL, kuriuo galima aprašyti web serviso teikiamas operacijas, bei siunčiamų ir gaunamų duomenų struktūrą, tačiau negalima aprašyti duomenų semantikos. Tam, kad suskurti automatinę web servisų sistemą, reikia išankstinių susitarimų, o globalios sistemos tampa beveik neįmanomos. Semantiniai web servisai sprendžia šią problemą teikdami papildomą, aukščiausią web servisų aprašymo lygį web serviso semantiniai prasmei aprašyti.

Vienas iš pavydžių kaip galėtų būti naudojami semantiniai web servisai yra atostogų planavimo sistema. Sistema teiktų visą reikalingą informaciją atostogų suplanavimui įskaitant ir lėktuvo bilietų užsakymą, automobilio nuomą, viešbučio rezervacija ir t.t.. Ši sistema remiantis semantiniu web servisų aprašymu sugebėtų atrasti ir bendrauti su reikiamais servisais. Galėtų būti ne tik pateikiama informacija, bet ir realizuotas paslaugų įsigijimas[AZA+05].

Pagrindinis darbo tikslas – visapusiška semantinių web servisų ir jais pagrįstų sistemų analizė. Bus išnagrinėta kas yra semantiniai web servisai, kokiose situacijose jie naudojami, kaip sudaroma semantinių web servisų sistema, aptariami semantiniai web servisų aprašymai ir įrankiai naudojami jiems kurti. Bus aptariama web servisų anotavimo problema, servisų paieškos mechanizmas, bei serviso panaudojimo mechanizmas, naudojantis semantiniu aprašu. Atlikus analizę bus pasiūlyta kaip spręsti konkrečias su semantiniais web servisais susiėjusias problemas.

Darbo pradžioje pateikiama literatūros apžvalga, kuri sudaryta iš trijų skyrių: semantinių web servisų sandara; semantinių web servisų sąveikavimas; semantinių web servisų panaudojimas praktikoje. Pirmajame skyriuje bus aptariama, kaip yra aprašoma web serviso semantika.

Antrajame skyriuje bus aptariama kaip tarpusavyje bendrauja semantiniai servisai, kaip atliekama jų paieška ir koordinuojami veiksmai. Trečiajame skyriuje aptarsime kaip sukurti atskirus semantinių web servisų komponentus pasinaudojant METEOR-S karkasu.

Po literatūros apžvalga pateikiamos semantinių web servisų sistemų kūrimo ir tobulinimo galimybės. Aptariamos kelios skirtingos semantinių web servisų sistemų architektūros, pateikiami jų privalumai ir trūkumai. Taip pat aptariamos trūkumų šalinimo glimybės. Praktinėje dalyje daug dėmesio skiriama web servisų paieškos ir panaudojimo, pasitelkiant semantiką, galimybės.

1 Semantinių web servisų sandara

Šiame skyriuje pateiksime informaciją reikalingą sukurti semantinį web servisą. Skyriuje bus aprašoma tik savybės, kuriomis turi pasižymėti semantinis web servisas ir kaip tas savybes įgyvendinti. Šių savybių panaudojimas paieškos ar iškvietimo tikslais - tolimesnio skyriaus tema.

Skyrius pradedamas nuo metaduomenų – web servisų semantikos aprašymo pagrindo, tuomet pereinama prie šių duomenų panaudojimo, t.y. aprašo kūrimo, ir užbaigiama, konkrečių standartų, skirtų sukurti semantiniams web servisų aprašams, pateikimu.

1.1 Metaduomenys

Meta duomenys – tai „duomenys apie duomenis“. Metaduomenys reikalingi tam, kad galutinis vartotojas galėtų surasti norimus duomenis pagal jų prasmę ar kontekstą. Metaduomenų priskyrimas duomenims vadinamas semantika. Metaduomenų išgavimas – tai procesas kuomet identifikuojami metaduomenys dokumentams ar kitam turiniui. Šis procesas gali būti neautomatizuotas, pusiau automatizuotas ir visiškai automatizuotas. Semantinės aplikacijos yra kuriamos panaudojant metaduomenis ir ontologijas[CS06]. Yra trys metaduomenų tipai: sintaksės, struktūros ir semantiniai.

1.1.1 Sintaksės metaduomenys

Sintaksės meta duomenys – tai pats parasčiausias metaduomenų tipas. Šie duomenys aprašo nuo konteksto nepriklausančią informaciją apie turinį. Tai labai bedra informacija: dokumento galimas dydis, sukūrimo data ir t.t.. Sintaksės metaduomenys pridedami kaip antraštės. Pavyzdys:

<name> = "report.pdf"

<creation> = "30-09-2005"

<modified> = "15-10-2005"

<size> = 2048

Apie aprašomo dokumento turinį šie duomenys informacijos neteikia arba teikia labai mažai[CS06].

1.1.2 Struktūros metaduomenys

Struktūros metaduomenys aprašo informacijos turinio struktūrą. Yra aprašoma kaip duomenys turėtų būti pateikiami: jų išdėstymas, eiliškumas ir t.t.. Šių duomenų kiekis priklauso nuo

dokumento tipo. Tokių duomenys yra aprašomi DTD arba XSD aprašuose. DTD pavyzdys[CS06]:

<!ELEMENT contacts (contact*)>

<!ELEMENT contact (name, birthdate)>

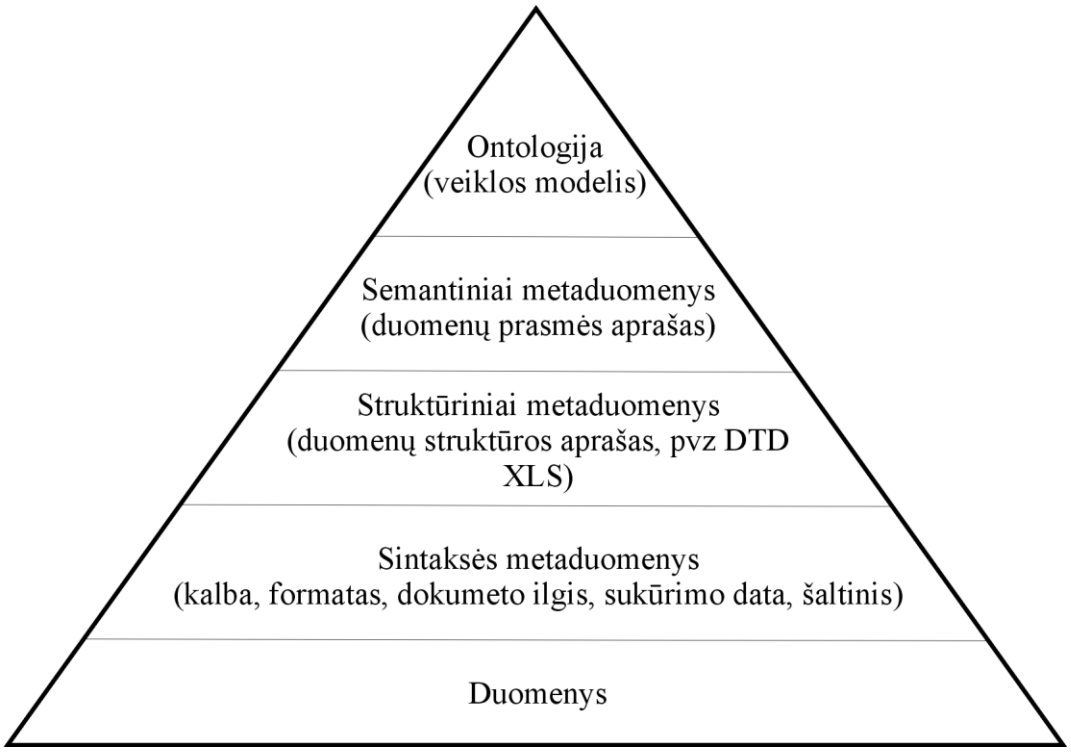
<!ELEMENT name (#PCDATA)>

<!ELEMENT birthdate {#PCDATA)>

Šiame pavyzdyje aprašyta, kad dokumentas turi bent vieną elementą *contacts*. Elemente *contacts* yra du elementai *name* ir *birthdate* ir tai, kad šiuose lementuose yra duomenys. Struktūriniai metduomenys yra pateikiami web servisų WSDL aprašuose.

1.1.3 Semantiniai metaduomenys

Semantiniai metaduomenys prie sintaksės ir struktūros metaduomenų prideda sąryšius, taisykles ir apribojimus. Semantiniai metaduomenys yra duomenys apie kontekstą arba duomenys apie turinio dalykinę sritį. Kitaip sakant tai yra duomenų reikšmė. Jei šie duomenys yra surišami su ontologijomis(veiklos modeliais), tai įgalina aukštesnį automatizavimo laipsnį. Semantinių duomenų pagalba galima atlikti aukšto tikslumo paieškas, o taip pat tai įgalina tarpveiksmingumą tarp heterogeninių duomenų šaltinių[CS06]. Metaduomenų hierarchija pavaizduota schemoje 1 pav..



1 pav. Duomenų hierarchija pagal jų semantiką.

1.1.4 Semantinių metaduomenų išgavimas

Tam, kad išgauti kuo informatyvesnius metaduomenis iš dokumento ir padaryti juos tinkamus vartoti, reikia efektyviai sužymėti dominančios srities semantiką dokumente. Gali būti naudojami keli būdai šiam tikslui pasiekti: naudojantis semantiniais žodynais; atliekant dokumento analizę ir ieškant išlanksto nustatytų šablonų; aprašant ontologijos ir dokumento sąryšius.

Semantikos sudarymas atsižvelgiant į žodžius ir jų sąryšius: žodžiai yra sujungiami su statiniu arba periodiškai atnaujinamu žodynu. Semantikos žodynas, toks kaip WordNet (Voorhees 1998), grupuoja anglų kalbos žodžius į komplektus sinonimų, pavadintų „synsets“ ir nustato ryšius tarp šių komplektų: nustato komplektus, kurie reiškia tą patį; yra bendresni; arba yra labiau specifiniai [WVH+07] [CS06].

Dokumento analizė: ieškoma šablonų, tam kad pritaikyti iš anksto nustatytos taisyklės. Reguliarios išraiškos ir santykiai tarp žodžių gali būti panaudoti, tam kad suprasi dokumentų reikšmę.

Ontologijos: sudaromi dalykinės srities ir dokumento sąryšiai. Sudarius tokius semantikos dokumentus gaunamas pats lanksčiausias ir naudingiausias semantinis aprašas.

1.2 Ontologijos

Ontologija yra formalus veiklos srities sudedamųjų dalių ir ryšių tarp šių dalių aprašymas. Ontologijos susideda iš šių elementų: individų, klasių, atributų ir ryšių [Ont08].

Individai. Individai yra ontologijos pagrindas. Individai – tai konkretūs ontologijos objektai, tokie kaip žmonės, automobiliai, namai (gali būti ir abstraktesnės esybes: skaičiai, žodžiai). Ontologijose individai tiesiogiai nėra aprašomi, tačiau ontologijos paskirtis įgalinti individų klasifikaciją.

Klasės. Klasės – tai abstrakčios objektų grupės. Objektai gali būti individai ir kitos klasės. Pavyzdžiui, individas „žmogus“ priklauso klasei „žmonija“. Klasė priklausanti kitai klasei vadinama subklase.

Atributai. Atributai, tai klases elementai skirti charakterizuoti klasės objektams. Atributai susideda iš atributo pavadinimo ir reikšmės. Pavyzdžiui, automobilio „Ford Mondeo“ individo aprašymas:

Pavadinimas: Ford Mondeo

Durų-skaičius: 4

Variklis: {2.0L, 2.2L}

Transmisija: 5-speed

Reikšmė gali būti ir sudėtinis duomenų tipas, kaip parodyta atribute *Variklis*.

Ryšiai. Formaliai ryšiai yra atributai, kurių reikšmės yra kiti objektai. Pavyzdžiui, individas „Ford Sierra“ galėtų turėti atributą „Najesnis-modelis“, kuris rodytų į individą „Ford Mondeo“. Iš tokio sąryšio galėtume pasakyti, kad gamyboje automobilį Ford Sierra pakeitė automobilis Ford Mondeo.

Kuriant semantinius web servisus, ontologija naudojama semantiškai aprašyti web serviso priimamus ir siunčiamus duomenis, bei nusakyti operacijų prasmę. Ontologijos įgalina aplikacijas „suprasti“ servisus be žmogaus įsikišimo. Čia kyla problema, kad semantinių servisų vystytojai dažnai ontologijas aprašo vieni nuo kitų nepriklausomai ir tos pačios veiklos ontologijos tampa šiek tiek skirtingos. To pasekoje, skirtingos ontologijos neleidžia tarpusavyje bendrauti servisams, kurie, turėdami ta pačią ontologiją, tai daryti galėtų. Problemos iškyla servisų paieškoje ir sąveikavime. Vienas iš siūlomų sprendimų yra sukurti bendras ontologijas. Tam, kad išvengti standartizacijos ir žmogaus įtakos problemų ontologijoms kurti siūlomos automatinės priemonės. Dažniausiai tam pasitelkiami semantiniai žodynai [WVH+07].

1.3 Web servisų semantinės anotacijos

Kontekstinė informacija, kuri nustato santykius tarp duomenų ir realiojo pasaulio aspektų, vadinama semantiniais metaduomenimis. Pagrindinis aspektas realizuojant semantinius web servisus yra tokių metaduomenų pateikimas ir jų surišimas su servisu[GK05]. Procesas sujungiantis metaduomenis su resursu (mūsų atveju resursas tai web servisas) vadinamas anotacija.

1.3.1 Esybių identifikavimas

Esybės identifikavimo proceso metu, naudinga informacija iš dokumentų yra išgaunama gramatika paremtomis taisyklėmis, natūralia kalbos apdorojimo technika ir vartotojo nustatytais šablonais. Taip pat galima naudoti ir dalykinės srities ontologijas[CS06]. Gramatikos taisyklių pagalba surandami žodžiai. Kadangi žodžiai gali būti su priesagomis, priešdėliais, sudurtiniai ar kaip kitaip suformuoti yra naudojama natūrali kalbos apdorojimo technika nustatyti esybėms. Vartotojo šablonai reikalingi tam, kad nustatyti esybes, kurių nėra žodynuose arba surišimas pagal žodžius nėra prasmingas. Esybes galima identifikuoti ontologijomis. Turėdami dalykinės srities modelį galime ieškoti atitikmenų šaltinyje. Taigi, šio proceso eigoje mes nustatome mus dominančias esybės konkrečiame resurse.

1.3.2 Esybės dviprasmiškumo pašalinimas

Dažnai pasitaiko tokia situacija, kad nustatytai esybei egzistuoja daug atitikmenų žodyne ar duomenų bazėje. Pavyzdžiui, John Smith gali būti paminėtas kaip finansų analitikas ir kaip

vadovaujantis kompanijos pareigūnas. Taip pat dokumente esanti informacija gali tiesiogiai nenusakyti kas yra John Smith. Tačiau jei dokumento tema apie kompanijos strategijos vystymą, tikėtina, kad kalbama apie John Smith kaip apie vadovaujantį kompanijos pareigūną. Tam, kad automatiškai pašalinti tokias dviprasmybes reikia pažangių metodų. Metodo parinkimas priklauso resurso kilmės ir nuo reikiamo tikslumo. Semantiniuose web servisuose dažniausiai remiasi ontologijomis ir aplink esybe esančiu kontekstu. Tačiau šis procesas dažniausiai atliekamas neautomatiniu būdu.

1.3.3 Anotavimas

Nustačius esybes ir pašalinus dviprasmybes iš semantikos, sekantis žingsnis yra semantinių duomenų surišimas su esybėmis – anotavimas. Jei web servisas turi WSDL aprašą ir ontologija aprašyta OWL schema, tuomet anotavimo uždavinys yra rasti atitikmenis tarp WSDL ir OWL, ir tai aprašyti. Yra atliekami du veiksmai: atitikmens suradimas (angl. matching) ir formalus aprašymas (angl. mapping). Šie veiksmai gali būti dalinai arba visiškai automatizuoti. Suradimas gali būti atliekamas dalinai automatizavus. Žmogaus įsikišimas reikalingas tik sudėtingesnėse situacijose. Tuo tarpu formalų aprašymą galima generuoti automatiškai visada.

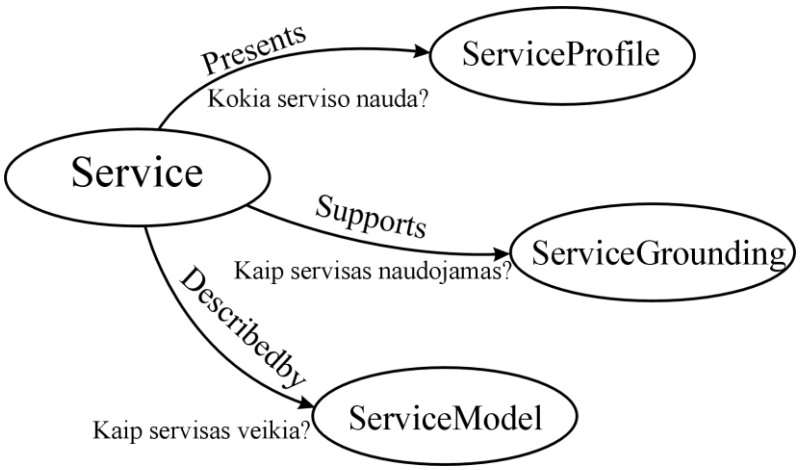
1.3.4 Semantinių web servisų aprašai

Populiariausios web servisų ontologijų aprašų kalbos yra OWL-S (OWL-S, OWL-based Web Service Ontology), WSMO (WSMO, Web Services Modeling Ontology) and WSDL-S (WSDL-S, Web Service Semantics). WSMO ir OWL-S turi savo semantinį modelį web servisams, tuo tarpu WSDL-S tik papildo WSDL aprašą semantine informacija.

1.3.4.1 OWL-S

OWL-S teikia web servisams žymių kalbos konstruktus (angl. markup language constructs), šių servisų savybių ir galimybių aprašymui nedviprasmiška forma, kurią gali interpretuoti kompiuteris [OWL06]. OWL-S web servisų žymės palengvina Web servisų užduočių automatizavimą: serviso paiešką, vykdymą, sąveikavimą, komponavimą ir vykdymo stebėjimą. OWL-S yra paremtas sluoksnių idėja ir realizuoja aukštesnį OWL sluoksnį. Aukščiausio sluoksnio schema vaizduojama 2 pav..

OWL-S aprašo serviso profilį (Kokią naudą servisas duoda klientui), serviso modelį (Kaip jis yra naudojamas) ir serviso pagrindą (kaip jis veikia).



2 pav. OWL-S aukščiausias, semantinis sluoksnis

Kiekviena web serviso esybė aprašoma klase ‚Service'. ‚Service' klasė turi tris savybes: ‚presents' kas atitinka serviso profilį, ‚describedBy' kas atitinka serviso modelį ir ‚supports' kas atitinka pagrindą. Savybė ‚presents' teikia informaciją reikalingą surasti norimą servisą, o savybės ‚describedBy' ir ‚supports' nusako kaip servisu naudotis

1.3.4.2 WSMO

WSMO – tai W3C patvirtintas standartas. WSMO sudarytas iš pagrindinių semantinių web servisų elementų, aprašytų formalia kalba (WSML) (WSML, Web Services Modeling Language). Taip pat WSMO turi savo vykdymo aplinką WSMX (WSMX, Web Service Execution Environment). WSMO yra paremtas web servisų modeliavimo karkasu WSMF (Web Service Modelling Framework) [WSM08].

WSMO visi servisai aprašomi dvejomis savybėmis: ‚Goals' aprašo web serviso tikslą (OWL-S atitikmuo būtų ‚presents') ir ‚Mediators' aprašo bendravimo tarp dviejų WSMO elementų ypatumus. WSMO numato tris ‚Mediators' lygius: duomenų lygis nustato kaip sąveikauja skirtingi duomenų šaltiniai; protokolo lygis nustato komunikavimo šablonus; proceso lygis nustato loginius dalykinės srities bendravimo ypatumus.

1.3.4.3 WSDL-S

Tiek OWL-S, tiek WSMO semantinis aprašas yra atskiras dokumentas, reikalaujantis nemenko sistemos praplėtimo, be to naudojantis skirtingą nei sintaksės aprašymo modelį. Tuo tarpu WSDL-S aprašą galima sukurti jau esamiems web servisams naudojantis jau esamu sintaksės aprašu. Taip yra todėl, kad WSDL-S tik praplečia WSDL standartą[WSD05].

WSDL-S tai dar vienas W3C patvirtintas standartas, leidžiantis aprašyti web servisų galimybes ir reikalavimus. Anotavimas yra įvykdomas pasitelkiant WSDL išplečiančiais elementais ir atributais. WSDL-S yra patrauklus dėl keleto priežasčių: daugelis sistemų kūrėjų,

dirbančių su web servisais, jau pažysta WSDL, todėl išmokti praplėtimus nėra sudėtinga; dalykinės srities modelį galima aprašyti bet kokia pasirinkta kalba (nebūtinai OWL kaip reikalaujam OWL-S standarte), dėl to galima naudoti daugiau jau esančių dalykinės srities modelių[CS06]. Taip pat WSDL-S web servisų anotavimui turi patogų grafinį įrankį „Radiant“, kuris gali atlikti formalų aprašymą automatizuotai (atitikmenų suradimas atliekamas pusiau automatizuotai) Dėl šių priežasčių WSDL-S yra dažniausiai naudojamas semantinių web servisų standartas.

WSDL praplėtimai skirti semantiniams web servisams pateikti lentelėje:

1 lentelė. WSDL-S atributai.

Papildantis elementas/atributas	**Aprašymas**
modelReference	Atributas elementams Input ir Output. Anotuoja Input ir Output pranešimų tipus į semantinį modelį.
modelReference	Atributas elementui Operation. Nustato operacijos semantiką
schemaMapping	Atributas elementams Input ir Output. Suriša WSDL pranešimų struktūrą ir sintaksę su semantiniu modeliu.
pre-conditions	Elementas. Tėvinis elementas Operation. Semantinių sąlygų rinkinys, reikalingas tam, kad operacija galėtų būti sėkmingai įvykdyta.
Effects	Elementas. Tėvinis elementas Operation. Semantinių sąlygų rinkinys, kurios privalo būti patenkintos po operacijos atlikimo.
Category	Elementas. Tėvinis elementas Operation. Operacijos kategorizavimo informacija, kuri gali būti panaudojama pateikiant servisą web servisų registre, tokiame kaip UDDI

2 Semantinių web servisų sąveikavimas

Šiame skyriuje bus aptariama kaip naudojantis semantiniu aprašu web servisus pateikti paieškai, juos surasti, teisingai iškviesti. Taip pat bus aprašyta semantinių web servisų sąveikavimo veiksmų seka.

2.1 Semantinių web servisų paieška

Kadangi web servisų paieška vykdoma UDDI (Universal Discovery, Description, and Integration) registrų pagalba, pirmiausia aptarsime jų veikimą.

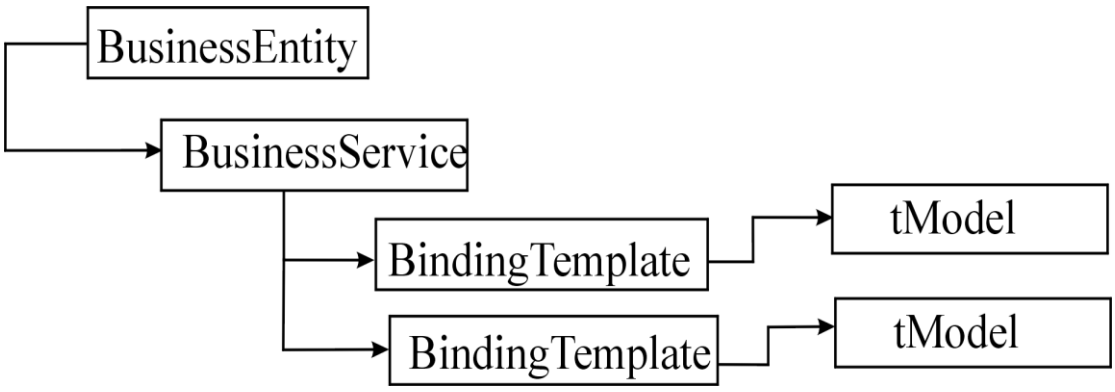
2.1.1 UDDI

UDDI yra specifikacija skirta sukurti paskirstytą žiniatikliu paremtą registrą web servisams. UDDI nusako kaip servisai publikuojami registre, kaip daromos užklausos norint rasti servisą ir kaip prisijungti norimą servisą.

UDDI registras yra suskirstytas į Baltus, Geltonus ir Žalius puslapius[UDDI08]. Baltuose puslapiuose talpinama informacija apie web serviso dalykinę sritį, rūšiuojant abėcėlės tvarka. Geltonuose puslapiuose taip pat pateikiama informacija apie dalykinę sritį, tačiau rūšiavimas yra daromas pagal dalykinės srities kategoriją. Žaliuose puslapiuose yra pateikiama techninė informacija apie web servisus.

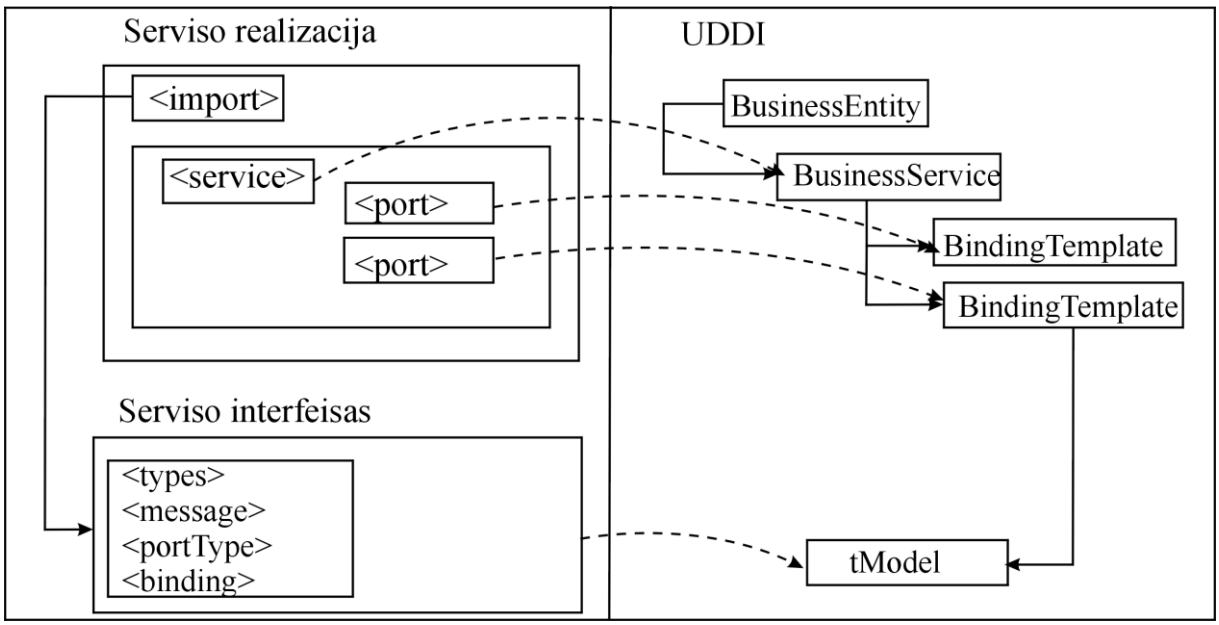
UDDI turi keturias skirtingas duomenų struktūras, tam kad apibrėžti registro įrašą (3 pav.). Šios struktūros aprašomos XML dokumente.

1. <businessEntity>. Šioje struktūroje laikoma informacija apie serviso dalykinę sritį ir ryšį su kitomis dalykinėmis sritimis.
2. <businessService>. Laikomas sąrašas servisu, priklausančių kategorijai nurodytai <businessEntity>
3. <bindingTemplate>. Po to kai servisas yra randamas, čia laikoma informacija reikalinga iškviesti servisą.
4. <tModel>. Šioje struktūroje laikoma serviso specifikacija. Tai yra neformalizuotas serviso aprašymas, gali būti pateikta nuoroda, kur galima rasti detalų serviso aprašymą.



3 pav. UDDI duomenų struktūra.

WSDL aprašas UDDI registre atvaizduojamas pagal schemą pavaizduotą 4 pav.



4 pav. WSDL atvaizdavimas į UDDI

2.1.2 Semantikos naudojimas paieškoje

Standartizavus visas servisų sąsajas, automatiška paieška vis vien nebūtų įmanoma naudojant tik WSDL aprašymus. Tai yra dėl kelių priežasčių

1. Sudėtinga standartizuoti visus, visų web servisų sąsajas ir tokio standartizacijos laikymąsi.
2. Vien tik standartizacija neužtikrina, kad web servisai bendraus teisingai: gali būti sukurtas servisas *PirkinioUzsakymas*, su dviem *float* tipo parametrais *svoris* ir *pinigai*, tačiau šių parametrų matavimo vienetai gali būti skirtingi.
3. Tam, kad kompiuteris suprastų ką operacija atlieka, reikia didelio kiekio semantinės informacijos. Jei šios informacijos ieškosime pagal standartizuotas sąsajas dideliame UDDI registre, eigos metu (angl. run time) prijungimas užtruktų per ilgai.
4. Esant duomenų tipų nesutapimui, būtų sudėtinga nustatyti, kaip reikėtų naudotis servisais.

Tam, kad išvengti šių apribojimų naudojamos keturių tipų semantikos: duomenų, funkcinė, nefunkcinė, vykdymo[CS06].

2.1.2.1 Duomenų semantika

Duomenų semantika formaliai aprašo duomenis, siunčiamus įėjimo ir išėjimo žinutėmis (angl. input and output messages). WSDL-S apraše duomenų semantika gali būti pridėta naudojant *modelReference* išplėtimą:

```
<wsdl:message name="PurchaseOrderRequestMessage">
<wsdl:part name=i"PORequest type="tns:PORequest"
wssem:modelReference="POOntology#PurchaseOrderRequest"/>
</wsdl:message>
```

Šiame apraše matome kad *PurchaseOrderRequestMessage* priklauso *POOntology* ontologijai ir atitinka jos elementą *PurchaseOrderRequest*.

2.1.2.2 Funkcinė semantika

Funkcinė semantika nustato web serviso galimybes. Funkcinę semantika galima nusakyti dviem būdais: anotuojant operacijas arba nustatant kokios sąlygos turi būti patenkintos prieš įvykdant servisą ir po jo vykdymo. WSDL-S funkcinę semantiką galima aprašyti pridedant *ModelReference, Category, Pre-Conditions* ir *Effecs*.

```
<operation name="GetOneQuote"
wssem: modelReference="Ontology1#FinancialTranisacstion>
<wssem:category categoryName="Stock quotation services"
taxonomyURL"http://www.census.gov/epod/naics02/"
taxonomyCode="523999"/>

<input message="s0:getOneQuoteSoapln"/>
<wssem:precondition name="stockSymbol"
wssem:modelReference="Ontology0#stockSymbol"/>

<output message="s0:GetOneQuoteSoapOut"/>
<wssem:effect name="price"
wssem:modelReference="Ontology1#price"/>
```

2.1.2.3 Nefunkcinė semantika

Nefunkcinė semantika nusako serviso kokybės QoS (angl. quality of service) reikalavimus, tokius kaip pristatymo trukmė ir strateginius reikalavimus, tokius kaip žinutės saugumas. Nefunkciniai reikalavimai gali būti kiekybiniai (tokie kuriuos galima išmatuoti, pvz. atsako laikas) ir kokybiniai (tokie kurie nusakomi Taip/Ne, pvz. kodavimas RSA). QoS atributai: atsako laikas, reakcijos laikas, vienu metu apdorojamų užklausų kiekis, prieinamumas (laiko dalis kai servisas pilnai funkcionuoja), patikimumas, tikslumas (serviso klaidų įvertis), prisitaikymas prie neteisingų parametrų, stabilumas (kaip dažnai keičiasi serviso sąsaja), kaina, saugumo reikalavimai[ZB06].

2.1.2.4 Vykdymo semantika

Vykdymo semantika tai formalus sistemos funkcijų vykdymo aprašymas[Ore04]. Vykdymo semantika susiėjusi su vidiniu serviso veikimu, t.y. kaip yra atliekami įvairūs patikrinimai ar valdomos netikėtos situacijos (angl. execptions). Šios semantikos pagalba galima nustatyti serviso veikimo logiką tam tikromis aplinkybėmis.

2.1.3 Semantinių web servisų publikavimas ir paieška

Skirtingai nei paprasti web servisai, semantiniai web servisai neturi visuotinai priimto nusistovėjusio registrų standarto tokio kaip UDDI [AZA+05]. Tai vis dar yra aktyvių tyrimų sritis. Karkasus, skirtus publikavimui ir paieškai yra sukūrusios kelios kūrėjų grupės: OWL-S, WSMO ir METEOR-S. Toliau pateiksime METEOR-S servisų paieškos ir publikavimo karkasą MWSDP (Web Service Discovery and Publication framework). MWSDP pasirinktas todėl, kad palaiko WSDL-S standartą, kurio pavyzdžius mes jau nagrinėjome.

METEOR-S turi grafinę sąsają semantinių web servisų pateikimui registre. Norint patalpinti servisą registre, pakanka turėti tik WSDL aprašą. Grafinė sąsaja nurodo kokios informacijos trūksta, tam kad servisą galima būtų pateikti semantiškai. Pateikus visą reikiamą informaciją METEOR-S transformuoja WSDL į WSDL-S ir patalpina jį registre.

Taip pat METEOR-S leidžia atlikti automatinę paieška registruose pagal iš anksto nustatytą šabloną. Toliau pateikiamas šablono pavyzdys:

```
Semantic Template
IndustryCategory = NAICS:Electronics
ProductCategory = DUNS:RAM
Location = Athens, GA
        Operation = Rosetta#requestPurschaseOrder
                        Input = Rosetta#PurchaseOrderDetails
                        Output = Rosetta#PurshaseConfirmation
                        Non-Funcional Requirements
                            Encrition = RSA
                            ResponseTime < 5 sec
        Operation = Rosetta#QueryOrderStatus
                        Input = Rosetta# PurchaseOrderStatusQuery
                        Output = Rosetta# PurchaseOrderStatusResponse
```

Šis semantikos šablonas nurodo, kad ieškomas servisas turi atitikti duomenų, funkcinę ir nefunkcinę semantiką iš Ontologijos *Rosetta*. Operacijų tipai turi būti *requestPurschaseOrder, QueryOrderStatus*. Tai yra funkciniai serviso reikalavimai. Pranešimų tipai *PurchaseOrderDetails, PurshaseConfirmation, PurchaseOrderStatusQuery* ir *PurchaseOrderStatusResponse*. Tai yra duomenų semantiniai reikalavimai. Nefunkcinis

reikalavimas yra atsako trukmė mažesnė už 5 sekundes (ResponseTime < 5 sec), tai yra kiekybinis reikalavimas. Kokybinis nefunkcinis reikalavimas yra operacijos pranešimus šifruoti RSA (Encription = RSA).

Suradus reikiamą web servisą reikia atlikti veiksmus tam servisui panaudoti. Serviso kompozicijos problemą galima palikti vartotojui, t.y. apsiriboti tuo, kad semantiką naudojame tik paieškai. Jei priimtas sprendimas automatizuoti kompoziciją, automatizacijos sudėtingumas priklauso nuo to kiek šis mechanizmas turi būti adaptyvus servisų pokyčiams.

Atlikus paiešką gaunamas serviso aprašas, pvz. WSDL. Automatizuoti reikiamų funkcijų iškvietimą nėra sudėtinga. Sudėtingiau yra realizuoti QoS reikalavimus.

Kaip matome, tam, kad rasti web servisą, reikia suformuoti semantinę užklausą serviso savybėms. Dažnai tokį aprašą sukurti yra sudėtinga. Vienas iš šios problemos sprendimo būdų – sukurti tikslų talpyklą [FB06]. Šioje talpykloje būtų galima atlikti paiešką pagal raktinius žodžius arba panašiomis nesemantinėmis procedūromis pvz. naudojantis katalogais. Ši talpyklą gražintų panašų į mūsų aprašytą semantinį šabloną, kurio pagalba rastume reikiamus servisus.

Dar viena problema, kurią numato semantinių web servisų kūrėjai, ta, jog servisų paieškai naudojamas „serverio-kliento“ modelis. Išaugus semantinių web servisų skaičiui, reikėtų didelių ir galingų talpyklų, kas nėra itin efektyvu. Šios problemos sprendimui siūlomas P2P tinklas (tinklas kuriame bendraujama ne su serveriu, o kiekvienas klientas bendrauja su kiekvienu klientu) [BCS04]. Toks tinklas padėtų vartotojams tiesiogiai bendrauti ieškant reikiamų servisų, be to tai įgalintų registrų tarpusavio bendradarbiavimą. Tam, kad įgyvendinti P2P, siūloma pridėti komunikavimo sluoksnį, kuriame būtų aprašomas klientų bendravimas. Taip pat siūloma praplėsti ir ontologiją, tam, jog apimtume ir komunikacijos sluoksnį[VSS+05].

2.2 Web servisų choreografija

Veikla, kuri įtraukia keletą nepriklausomų procesų, kurie naudoja web servisų technologiją, gali būti sėkminga tik tuo atveju jei ji yra tinkamai koordinuojama [WCM04]. Tai reiškia, kad siuntėjas ir gavėjas turi iš anksto susitarti dėl:

1. Dėl žinučių struktūros ir formato;
2. Žinučių apsikeitimo sąlygų;
3. Žinučių apsikeitimo sekos.

Su pirmuoju punktu susitvarko paprasčiausias WSDL aprašas. Antrajam punktui tinkami semantiniai aprašai, pavyzdžiui WSDL-S. Tačiau nė viena iš anksčiau išvardintų technologijų nenustato sekos, arba kitaip – choreografijos, kuria turėtų būti atliekami veiksmai norimam tikslui pasiekti. Sakysim, jei servisas priima parametrą metrais, kitas servisas teikia duomenis coliais ir egzistuoja servisas, konvertuojantis colius į metrus. Tuomet norint pateikti parametrą

metrais iš pradžių reikia kreiptis į konvertavimo servisą ir gautą reikšmę norimam servisui. Naudojant semantinį aprašą galima tokį veikimą sumodeliuoti. Tai teikia keletą privalumų: galima naudoti daugiau servisų nepaisant jų heterogeniškumo; integravimas naujų aplikacijų į jau egzistuojančias aplikacijas gerokai supaprastėja.

Yra keletas bandymų sukurti standartus choreografijos aprašymui: BPEL4WS (WSDL-S pagrindu), WS-CDL (aukštesnis sluoksnis virš WSDL-S), WSCI (aukštesnis sluoksnis virš WSDL-S), WSMO – Choreography (skirta WSMO standartui). Tačiau nė vienas standartas nepateikia pilno sprendimo:

- Visi standartai priklausomi nuo technologijos (WSDL arba WSMO);
- Trūksta aiškaus modelio, kuris atskirtų struktūrinius, elgesio ir operavimo aspektus;
- Semantikos galimybės nėra pilnai išnaudojamos;
- Heterogeniškose sistemose nėra pilno automatizavimo.

3 Semantinių web servisų panaudojimas praktikoje

Iki šiol darbe yra aptarta semantinių web servisų sandara ir jų tarpusavio sąveikavimas. Web servisų aprašai sudaryti iš meta duomenų, kurių aukščiausias lygis yra semantiniai metaduomenys. Apjungus šiuos duomenis į bendrą sistema, gauname dalykinės srities ontologiją. Norint web servisui suteikti semantinį aprašą reikia identifikuoti savybes, pašalinti dviprasmybes ir atlikti anotacijas. Turėdami štai tokį aprašą galėsime atlikti automatinę paiešką pagal web serviso prasmę. Taip pat galėsime nustatyti, kaip reikia atlikti veiksmus norint pasiekti rezultatą. Tai yra pagrindiniai semantiniais web servisais paremtos sistemos kūrimo žingsniai. Toliau bus aptariama kaip kuriuos iš šių žingsnių realizuoti. Šiame skyriuje aprašysime, kokios yra taikymo sritys semantiniams web servisam ir kokią funkciją šiose srityje jie atlieka, taip pat aprašysime praktinį semantinių web servisų kūrimo procesą.

3.1 Kūrimo procesas

Aprašomas kūrimo procesas remsis METEOR-S karkasu[MSWS08]. Šis karkasas pasirinktas todėl, kad jo pagrindą sudaro populiariausios technologijos web servisų sistemoms (WSDL, BPEL) su semantiniu praplėtimu. Technologijos paremtos OWL-S ar WSMO turi privalumų, kuriant semantines web servisų sistemas pradedant nuo antologijų: t.y. pirmiausia kuriama veiklos srities ontologija iš kurios gaunamas sąveikavimo procesas, servisų semantika, technologinė realizacija. Toks kūrimo procesas būtų neblogas, jei web servisų sistemas norėtume kurti nuo pagrindų. Tačiau praktikoje dažniausiai pasitaiko situacija, kai jau yra web servisai turintys WSDL aprašus ir netgi UDDI registrą. Šias sistemas norma praplėsti, jų pilnai

neperdarant. Dėl to naudosime semantinį WSDL praplėtimą- WSDL-S, ir kitas technologijas, nėra visiškai kitokios nei nesemantinių web servisus sistemų. Dažniausiai šios technologijos yra tik praplėstos semantikai.

3.1.1 WSDL-S anotacija

MWSAF yra įrankis skirtas pusiau automatiniam WSDL failų anotavimui. Šis įrankis sugeba išnagrinėti (angl. parse) WSDL failus ir ontologijas. Palaikomos RDF-S , DAML+OIL and OWL standartais paremtos ontologijos. Įrankis automatiškai parenka geriausius ontologijos atitikmenis WSDL aprašui. Vartotojas gali pasirinkti kurie atitikmenis buvo surasti teisingai ir pridėti nerastus atitikmenis. Sekantis žingsnis sugeneruoti WSDL-S failą. Šis įrankis tai daro automatiškai. Šis įrankis taip pat turi galimybę surasti veiklos sritį, kuriai priklauso WSDL aprašas. Tam skirti du algoritmai. Pirmasis paremtas WordNet semantiniu žodynu, antrasis – mokymosi algoritmu, kuris dirba su ontologijų rinkiniu. Pradinis rinkinys instaliuojamas automatiškai. Vėliau šį rinkinį galima praplėsti savo paties sukurtomis arba iš interneto parsiustomis ontologijomis.

3.1.2 Publikavimas

Semantinių web servisų publikavimui METEOR-S karkase yra skirtas SWSP (Semantic Web Service Publishing) įrankis. Publikavimas vykdomas UDDI standartu paremtame registre su semantiniu praplėtimu.

Norint publikuoti semantinį web servisą reikia užpildyti keletą formų. Registrui pateikiami duomenys:

- Veiklos pavadinimas – laisvai pasirenkamas. Gali būti įmones publikuojančios servisą pavadinimas.
- Serviso pavadinimas – semantinio web serviso vardas.
- Veiklos sritis – nurodoma veiklos srities ontologija
- Dislokacijos vieta – nurodoma serviso buvimo vietos ontologija. Pvz. Europa.
- Operacijų sąrašas – nurodomos visos publikuojamos registre operacijos. Operacijoms nurodomos ontologijos elementai. Taip pat nurodomi įvedamų ir išvedamų elementų ontologijos elementai.
- Paieškos sąrašas - paieškai pateikiamų operacijos duomenų sąrašas.
- WSDL vieta – nuoroda, kuria galima rasti WSDL aprašą.

3.1.3 Paieška

METEOR-S karkase yra grafinis įrankis atlinkti semantinę paiešką registruose. Tai labiau skirta pažintiniams tikslams, nes paieška yra vienkartinė, o serviso prijungimas nėra realizuotas. Paieška leidžiama pagal visus kriterijus kuriuos nurodėme registruojant servisą publikavimui: veiklos srities ontologijas, dislokacijos ontologijas, operacijų, bei jų parametrų prasmes. Paieškos rezultatai pateikiami lentelėje su detaliu semantinio web serviso aprašymu.

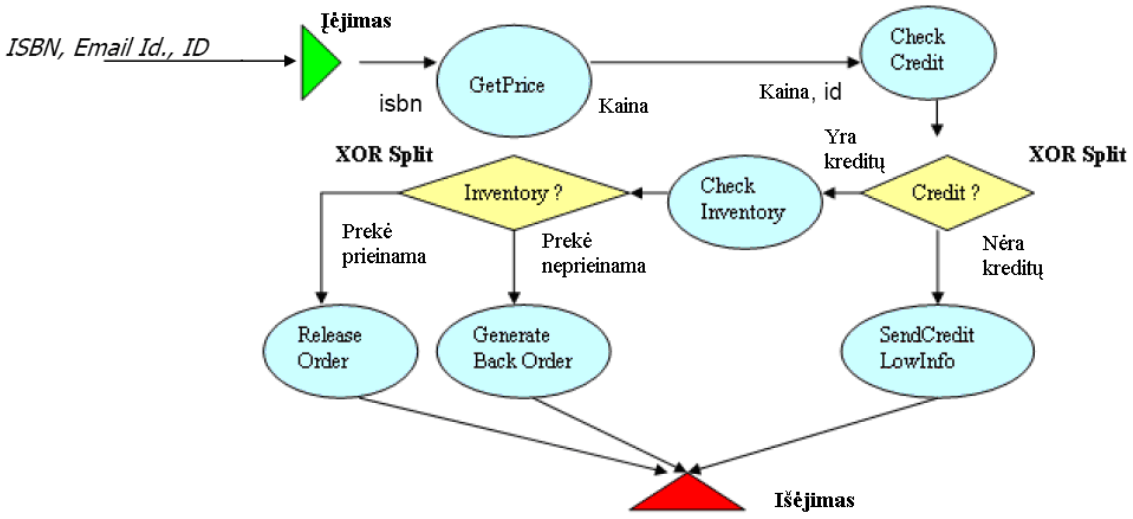
Panaudojus tokį paieškos mechanizmą galima būtų sukurti taikomąją programą, kuri pagal įvestus semantinius reikalavimu automatiškai prijungtų web servisą.

3.1.4 Choreografija

Choreografija aprašoma verslo procesų vykdymo kalba BPEL (Business Process Execution Language). Verslo procesai aprašomi BPWS (Bussines Process for Web Services) failuose. Šių failų generavimui ir redagavimui METEOR-S siūlo naudotis Eclipse įskiepu BPWS4J. Šio įrankio pagalba verslo procesų partnerių nuorodas, procesų susiejusių su partneriais eigos eiliškumą.

Choreografijos pavyzdys būtų prekės užsakymas. Tam kad užsakyti prekę reikia įvykdyti tam tikrus web servisus, tam tikra tvarka. Surasti reikiamą prekę, pervesti pinigus, nurodyti adresą (eiliškumas ir procesų skaičius gali skirtis). Būtent tokius reikalavimus reikia nurodyti BPWS.

Toliau įvairiems verslo procesams nurodomi atitikmenys ontologijose. Taip pat nurodomi QoS reikalavimai. Atlikti šiai užduočiai METEOR-S turi aplikaciją BPELSA (BPEL Semantic Annotator). Naudojant šią aplikaciją verslo procesams pridedama semantinė informacija – vykdoma anotacija.



6 pav. Knygos pirkimo verslo procesai.

BPELSA prie BPWS failų prideda tokią informaciją: veiklos srities ontologija, lokacijos ontologija, verslo procesų atitikmenis ontologijose, QoS informaciją (kokybinę ir kiekybinę).

6 pav.vaizduojama verslo procesų schema knygos pirkimui. Apskritimuose vaizduojami web servisai, o veiksmų seka aprašoma BPWS faile.

4 Praktinė dalis

Iki šio išanalizuotas semantinių web servisų kūrimo ir jų sąveikavimo teorinis pagrindas, bei panagrinėti egzistuojantys įrankiai padedantys sistemos kūrimo procese. Taip pat aptarti keletas alternatyviu standartų ir pasirinktas vienas iš jų.

Literatūros analizės metu buvo analizuojamos teorinės galimybės panaudoti semantiką servisų aprašymui, publikavimui, paieškai bei panaudojimui. Atlikus literatūros analizę padarytos išvados, kad yra sukurti standartai semantiniam web servisų aprašymui, tačiau kol kas nėra pavyzdinių sistemų, kurios būtų pagrįstos web servisais ir bendrautų semantiniu pagrindu. Yra atskiros dalys web servisų anotavimui, pusiau automatiniam publikavimui bei pusiau automatinei paieškai.

Kadangi praktinių pavydžių nėra daug, literatūroje beveik neanalizuojami architektūriniai semantinių web servisų sprendimai, nėra analizuojamos silpnos ar stiprios skirtingų architektukų pusės.

Literatūros analizės metu buvo priimtas sprendimas toliau naudotis WSDL-S standartu ir šiuo standartu paremtomis technologijomis. Šio sprendimo priežastys: galima naudoti servisus turinčius WSDL aprašą; yra automatinio anotavimo priemonės; yra įrankiai verslo procesams aprašyti; tai labiausiai naudojamas ir geriausiai aprašytas standartas.

4.1 Praktinės dalies tikslas ir numatomas rezultatas

Praktinės dalies darbo tikslas - sukurti web servisais paremtos sistemos prototipą, kurio pagalba galėtume išbandyti automatinę naujų servisų bei naujų servisų metodų paiešką ir jų panaudojimą. Taip pat išbandyti automatinį prisitaikymą prie esamų web servisų pasikeitimų.

Darbe bus bandoma sukurti semantiniais web servisais pagrįstos sistemos sprendimo kelias galimas architektūras, išanalizuoti jų privalumus bei trūkumus, parinkti tinkamą platformą realizacijai (t.y. reikalingą programinę aplinka, web serverį, sql serverį), realizuoti pasiūlytas architektūras programiškai, palyginti jas tarpusavyje, pasiūlyti ir įgyvendinti įmanomus patobulinimus.

Laukiamas rezultatas – web servisų veiklos srities aprašymas – ontologija; keletas web servisų ir jų aprašo semantinės anotacijos; keletas alternatyvių architektūrų semantinių web servisų sistemų realizacijai.

4.2 Sprendimo aplinka

Web servisų programinei realizacijai populiariausios yra dvi platformos: Sun Microsystem kompanijos Java programavimo platforma ir Microsof Visal Studio ASP .NET platforma. Vis dėlto daugiausia semantikos anotavimo ir paieškos pavyzdžių pateikiama Java platformai. Tyrimus šioje srityje atlieka Jungtinių Amerikos Valstijų Džordžijos (angl. Georgia) universitetas, vykdantis Meteor-S projektą ir kompanija IBM, išleidusi „Semantic Tools for Web Services“ įrankių komplektą skirtą „Websphere“ produktų grupei („Websphere“ tai produktų grupė skirta realizuoti servisais paremtas aplikacijas). Abi šios institucijos naudoja Eclipse programinės įrangos kūrimo platformą. Eclipse – tai nemokama programinės įrangos kūrimo platforma, kuri susideda iš praplečiamo programavimo karkaso, įrankių ir įvairių bibliotekų.

Realizuoti kuriamą sistema tikriausiai nebūtų kliūčių ir ASP .NET platformoje naudojantis Visual Studio, tačiau, kadangi Java ir daugelis šiai platformai skirtų įrankių yra nemokami, tyrinėjimo tikslams pasirinksime būtent ją. Taip pat naudosime Eclipse programinės įrangos kūrimo platformą.

Web servisų pasiekimas bus realizuojamas naudojantis Tomcat web serveriu. Šis web serveris pasirinktas dėl aukšto suderinamumo su Eclipse platforma.

Reikia pastebėti, kad visi šios aplinkos komponentai yra suderinami su daugeliu populiariausių operacinių sistemų. Visi toliau nurodomi pavyzdžiai buvo realizuoti Microsoft Windows operacinėje sistemoje, tačiau aprašomos architektūros ir jas palaikantys komponentai gali būti diegiami tiek Microsoft operacinėse sistemose, tiek ir atviro kodo, pavyzdžiui Linux, operacinėse sistemose.

Web servisų semantiniam aprašui naudosime WSDL-S standartą. Šis standartas pranašesnis už OWL-S tuo, kad WSDL-S yra dažniausiai web servisų naudojamo WSDL standarto praplėtimas. Dėl šios priežasties semantinį aprašymą nesunku pridėti jau egzistuojantiems web servisams.

4.3 Web servisas su WSDL

Kadangi remsimės WSDL-S standartu, reikia sukurti web servisus su WSDL aprašu. Sistemoje bus naudojami du skirtingo tipo web servisai: parduotuvės ir banko. Reikia pastebėti, kad praktikoje šie web servisai negalėtu pilnai užtikrinti banko ar elektroninės parduotuvės veikimo, tačiau šio darbo tikslas nėra įgyvendinti pilnavertę sistemą ir šie web servisai bus naudojami tik kaip priemonė iliustruoti semantikos naudojimą web servisuose.

Banko web servisas turės du metodus: *GetBankoId()* ir *Pirkimas(int saskaita, float suma)*. *GetBankoId()* metodas gražina banko identifikavimo numerį, kuris yra int tipo. Metodas

Pirkimas(int saskaita, float suma) priima du parametrus: *saskaita* – sąskaitos numeris ir *suma* – pinigų sumą litais kurią reikia pervesti į nurodytą sąskaitą. Wsdl aprašymo fragmentas, kuri vėliau anotuosime:

```
  <wsdl:types>
    <schema  elementFormDefault="qualified"  targetNamespace="http://bank.org"
xmlns="http://www.w3.org/2001/XMLSchema">
     <element name="GetBankoId">
      <complexType/>
     </element>
     <element name="GetBankoIdResponse">
      <complexType>
       <sequence>
        <element name="GetBankoIdReturn" type="xsd:int"/>
       </sequence>
      </complexType>
     </element>
     <element name="Pirkimas">
      <complexType>
       <sequence>
        <element name="saskaita" type="xsd:int"/>
        <element name="suma" type="xsd:float"/>
       </sequence>
      </complexType>
     </element>
     <element name="PirkimasResponse">
      <complexType/>
     </element>
    </schema>
   </wsdl:types>
```

Iš pateikto fragmento matome, kokie galimi metodai, kokie parametrai reikalingi jiems iškviesti ir kokie parametrai bus gražinami. Tai yra struktūrinis aprašymo lygis.

Atitinkamai realizuojame ir parduotuvės web servisą. Naudosime tris metodus *GetPrekes()*, *getSaskNr()* ir *Pirkti(String Preke)*. *GetPrekes()* gražins String tipo masyvą su prekių sąrašu. *getSaskNr()* pateiks int tipo reikšmę - banko sąskaitos numerį. *Pirkti(String Preke)* priims String tipo kintamąjį su prekės pavadinimu ir gražins int tipo kintamąjį – šios prekės kainą.

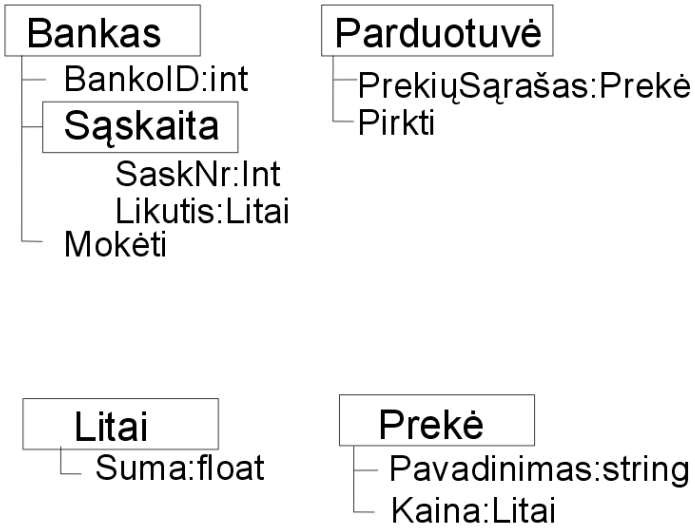
Kliento programoje bus panaudojami abu web servisai. Tam, kad klientas apsipirktų bus atliekami tokie veiksmai: peržiūrimas prekių sąrašas, išsirenkama prekė ir gaunama prekės kaina, gaunamas banko sąskaitos numeris, tuomet naudojantis banko web servisu atliekamas mokėjimas.

4.4 Ontologija

Semantinis aprašas turi remtis kokia nors ontologija, todėl norint atlikti semantinę anotaciją prieš tai teks aprašyti veiklos sritį. Šiam tikslui naudosime OWL kalbą ir įrankį „Protégé“. Tai grafinis įrankis skirtas generuoti OWL veiklos aprašymo failus.

Veiklos sritį pavadinsime *Prekyba*. Ši veiklos sritis susidės iš 4 pagrindinių klasių: *Bankas, Parduotuvė, Prekė, Pinigai*. Klasė *Bankas* turi subklasę *Saskaita*. Klasė *Bankas* turi „dataType“ atributą *BankoID*, kurio leistinos reikšmės yra integer tipo. Pagal šį atributą galima nustatyti kokiame banke yra sąskaita. Klasė *Pinigai* turi „dataType“ atributą *Suma*, kuris yra float tipo. Klasė *Prekė* turi „Object“ atributą *Kaina,* kuris sudarytas iš klasės *Pinigai* objektų. Klasė *Parduotuvė* turi „Object“ atributą *PrekiuSarsas*, kuris susideda iš klasės *Prekė* objektų. Veiklos srities *Prekyba* grafinė schema vaizduojama 1 pav..

Matome, jog egzistuoja du atributai be jokio tipo: *Bankas* klasėje – *Mokėti*, ir klasėje *Parduotuvė – Pirkti*. Taip yra todėl, kad šie veiklos aspektai gali būti realizuojami skirtingai. Pavyzdžiui, web servise galime aprašyti metodą *Mokėti* ir jam perduoti tik *SaskNr* ir *Suma*. Tačiau įmanomas variantas, kad bankas gali reikalauti ir daugiau informacijos apie atsiskaitymą, pvz. *Prekė->Pavadinimas*. Ontologija neapriboja realizacijos ypatumų. Tam, kad išspręsti kokie parametrai bus paduodami, pasinaudosime WSDL-S galimybėmis ir pridėsime prie perduodamų ir gražinamų parametrų anotacijas iš ontologijos.



1 pav. Ontologijos „Prekyba“ schema

4.5 Web servisas su WSDL-S

Turėdami web serviso WSDL aprašymą ir veiklos ontologiją, galime atlikti anotaciją, t.y. aprašyti web serviso ir ontologijos sąryšius. Šiuos sąryšius aprašysime WSDL-S standartu.

WSDL-S standartas paremtas WSDL 2.0 standarto praplėtimais, t.y. naudojantis elementais, kuriuos WSDL 2.0 standartas leidžia nustatyti pačiam vartotojui [WSD05]. Pagal WSDL-S standartą šie elementai yra sudaryti iš „wssem“ priešdėlio ir vieno iš WSDL-S standarto raktinio žodžio (papildančio elemento). Tam, kad nustatyti web serviso ir ontologijos sąryšius mums

užteks pasinaudoti papildančiu elementu „modelReference“. Šis elementas nurodo WSDL elemento sąryšį su ontologijos klase arba klasės atributu. Kiti WSDL-S elementai skirti orchestracijai sudaryti ir šiame darbe nagrinėjami nebus.

Toliau pateikiama, kaip buvo anotuotas Banko web serviso aprašo fragmentas. WSDL-S elementai paryškinti.

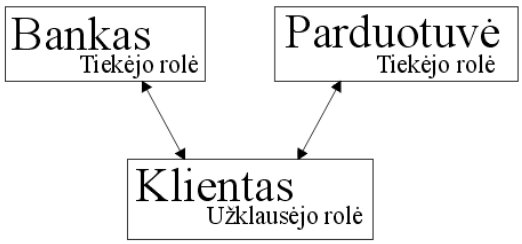
```
      <element name="GetBankoId">
       <complexType/>
      </element>
      <element name="GetBankoIdResponse">
       <complexType>
        <sequence>
         <element            name="GetBankoIdReturn"            type="xsd:int"
wssem:modelReference="BankoID"/>
        </sequence>
       </complexType>
      </element>
    <element name="Pirkimas" wssem:modelReference="Pirkti">
       <complexType>
        <sequence>
         <element name="saskaita" type="xsd:int" wssem:modelReference=" SaskNr"/>
         <element name="suma" type="xsd:float" wssem:modelReference=" #Litai"/>
        </sequence>
       </complexType>
      </element>
      <element name="PirkimasResponse">
       <complexType/>
      </element>
     </schema>
    </wsdl:types>
```

Ženklas „#“ reiškia, jog tai ontologijos klasė. Likę elementai anotuojami analogiškai.

4.6 Dviejų servisų semantinė sistema

Iki šiol sukūrėme du web servisus, parašėme veiklos srities aprašymą ir įvykdėme anotaciją. Visos šios informacijos pakanka sukurti dinamišką servisais paremtą sistemą. Toliau pateiksime keletą galimų architektūrų semantinių web servisų sistemų įgyvendinimui, bei palyginsime jas tarpusavyje.

Tikriausiai, paprasčiausia semantinę sistemą galėtų sudaryti du web servisai su WSDL-S aprašu, kur semantinį apdorojimą atlieka web serviso klientas. Kliento programoje turėsime kreipinius į *Banko* ir *Parduotuves* web servisus. Kliento programa atliks semantiškai anotuoto wsdl failo apdorojimą, nustatys kokios reikalingos funkcijos yra šiuose web servisuose ir pagal tai sudarys SOAP užklausas. Sistemos veikimo schema vaizduojama 2 pav..



2 pav. Dviejų servisų panaudojimo schema

Šiai sistemai bus reikalingas komponentas iš anotuoto wsdl failo sudarantis ontologijos ir metodų atitikmenis. Kadangi WSDL failai yra aprašomi XML kalba rekomenduojama naudotis XML apdorojimo bibliotekomis pvz. JAXP [JAX08]. Vienas iš galimų šio komponento įgyvendinimo būdų būtų sukurti struktūrą, kurioje iš pradžių būtų surašomos visos klasės ir klasių atributai iš ontologijos, kuriuos naudos užklausėjo rolės web servisas. Tuomet komponentas iš anotuoto WSDL failo parenka atitinkančius metodus ir parametrus. Vėliau ši struktūra naudojama sugeneruoti SOAP užklausas.

Norint sukurti šią sistemą galima sugeneruoti kliento kodą pagal neanotuotą WSDL failą pasitelkiant Eclipse funkcionalumą. Sugeneruotame kode turėsime statiškai įrašytus kviečiamų metodų pavadinimus. Šiuos pavadinimus reikia pakeisti naudojantis prieš tai aprašytos struktūros atitikmenimis.

Ši sistema sugebės prisitaikyti prie naudojamų web servisų pakeitimų, pvz. jei pasikeistų metodo pavadinimas arba jei metodas tuo pačiu pavadinimu pradėtų atlikti kitas funkcijas. Taip pat galima būtų praplėsti ontologiją, bei aprašius veiksmų logiką, įgyvendinti web servisą, kuris prisitaiko prie perduodamų parametrų pasikeitimo (pvz. jei atsiskaitymai būtų pradėti daryti eurais o ne litais). Ši sistema negali prisitaikyti prie iš anksto nenumatytų būtinos veiksmų sekos pasikeitimų (pvz. jei iš anksto nebuvo numatyti, kad prieš perkant prekę reikia ją rezervuoti ir tokia funkcija pridėta jau veikiančiai sistemai). To priežastis – veiksmų seka yra suprogramuota kliento programoje, norint šią seką padaryti dinamišką reikalingas dar ir orchestracijos aprašymas. Tačiau ir orechestrcijos aprašymai neišspręstų visų praktinių problemų, nes vartotojai dažniausiai nori žinoti veiksmų seką prieš leidžiant ją atlikti (pvz. vartotojams svarbu kaip atliekamos operacijos su jų pinigais), todėl bet kokiu atveju visiškas automatizavimas nebūtų įmanomas.

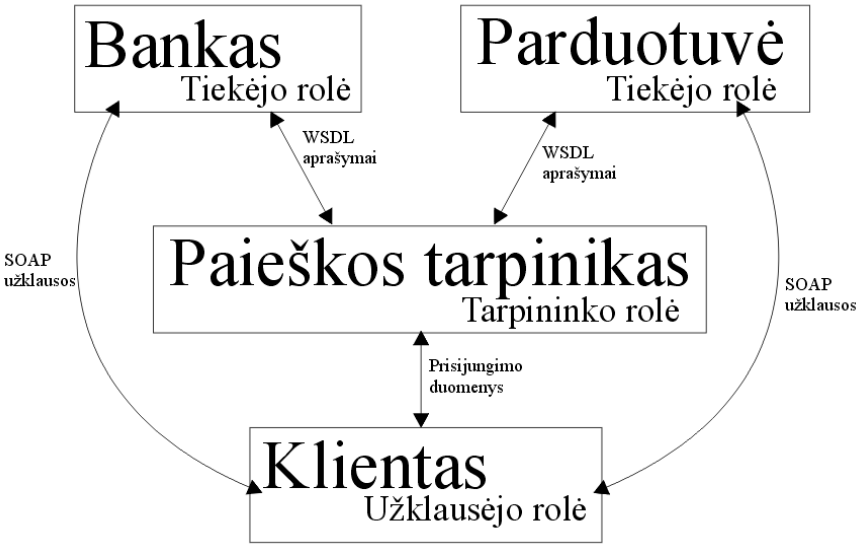
Sistemos, kurioje semantinį apdorojimą atlieka pats klientas, privalumas – ji nepriklauso nuo jokių išorinių resursų, realizaciją galima pasirinkti tinkamą konkrečiai situacijai. Vienintelis reikalavimas standartizacijai WSDL-S specifikacijos laikymasis. Sistemos silpnosios pusės – nestandartizuota realizacija gali sukelti palaikymo problemų; sistema turi savyje laikyti visų žinomų web servisų aprašymus (bent jau nuorodas į aprašymus). Dar vienas šios sistemos

ypatumas (priklausomai nuo situacijos tai nebūtinai yra trūkumas) kiekvienai sistemai reikia realizuoti atskirą paieškos algoritmą, o paieška vykdoma kliento resursais.

4.7 Semantinė sistema su paieškos tarpininku

Kai kurias problemas iš dviejų servisų sistemos architektūros galėtų išspręsti sistema su paieškos tarpininku. Šioje sistemoje naudojamas dar vienas web servisas – paieškos tarpininkas. Kliento programa kreipiasi į tarpininką su užklausomis ir gauna reikiamus prisijungimo prie web serviso parametrus. Tarpininkas gavęs šias užklausas atlieka semantinę paiešką *Prekės* ir *Banko* aprašymuose ir gražina rezultatus. Naudodamas šiuos rezultatus klientas jungiasi prie reikiamų servisų. Semantinės sistemos su tarpininku architektūra vaizduojama 3 pav..

Paieškos tarpininko web serviso aprašo galima semantiškai neanotuoti, nes jo metodai ir gražinami parametrai keistis neturėtų. Tačiau visi web servisai, o taip pat ir kliento programa turėtų naudotis ta pačia ontologija. Tiesa tarpininkas ontologijos aprašo gali ir neturėti, nes nustatyti servisų elementų ir ontologijos sąryšius galima ir nežinant pačios ontologijos logikos. Pakanka naudoti tik anotuotame WSDL apraše rastus klasių ir jų atributų pavadinimus.



3 pav. Semantinių web servisų su paieškos tarpininku sistemos architektūra

4.7.1 Realizacija

Paieškos tarpininkas nebūtinai turi būti web servisas, tačiau naudojant web servisą iš kliento pusės naudojamos tos pačios technologijos, kaip ir atliekant veiksmus su galutiniais web servisais. Taip pat tarpininką galima patalpinti bet kokiame serveryje – svarbu turėti interneto prieigą.

Tarpininko vidinis mechanizmas turėtų atlikti tuos pačius veiksmus kaip ir kliento komponentas iš anotuoto wsdl failo sudarantis ontologijos ir metodų atitikmenis. Galimi keli kliento ir paieškos tarpininko veikimo scenarijai. Klientas galėtų perduoti ontologijos pavadinimą. Pagal ontologijos klases ir jų atributus būtų sudaroma atitikmenų struktūra. Sukurtą atitikmenų struktūrą galima būtų perduoti kliento programai. Tačiau tokiu atveju ir klientas ir paieškos tarpininkas turėtų naudoti tokią pačią struktūrą. Kita problema kurią reikėtų spręsti tai kaip paieškos tarpininkas atskirtų galutinių web servisų ontologijas. Viena iš išeičių – laikyti visas reikiamas ontologijas paieškos tarpininko web servise, kita – anotuojant galutinių web servisų WSDL failus įrašyti ir ontologijos pavadinimą.

Galimas ir kitas scenarijus – paieškos tarpininkui pateikti tik ontologijos klasės arba jos atributo pavadinimą ir tokiu būdu gauti reikiamus parametrus. Šio scenarijaus silpnoji vieta ta, kad bus reikalingas intensyvus bendravimas tarp paieškos tarpininko ir kliento, nes kiekvienam metodui ar perduodamam parametrui teks atlikti po užklausą. Kartais tai gali būti gera savybė - jei reikia panaudoti tik keletą web serviso metodų iš ontologijos. Pirmojo scenarijaus atveju vis viena bus siunčiama visa struktūra, nors naudosime tik jos dalį. Antru atvejų yra galimybė reikalauti tik reikiamų metodų. Tačiau tokiu atveju reikėtų realizuoti dinamišką SOAP užklausų kūrimą prieš atliekant kiekvieną veiksmą. Tai vėl gali sukelti didesnį bendravimo tarp kliento ir paieškos tarpininko srautą,nes kiekvieną kartą generuosime po užklausą prieš kiekvieną veiksmą. Išeitis galėtų būti išsaugoti gautą rezultatą kliento pusėje po pirmos užklausos ir daugiau į paieškos tarpininką nebesikreipti.

4.7.2 RDBVS naudojimas paieškos tarpininko servise

Kaip jau minėta, paieškos tarpininkas turi atlikti tuos pačius veiksmus kaip ir kliento komponentas iš anotuoto wsdl failo sudarantis ontologijos ir metodų atitikmenis. Tikėtina, jog tarpininkas bus skirtas laikyti ir apdoroti dideliems kiekiams web servisų aprašų.

Kadangi teksto paieškos ir apdorojimo operacijos yra gana lėtos, kiekvieną kartą kreiptis į web servisus ir apdoroti jų semantiškai anotuotus WSDL aprašus gali prireikti daug laiko ir resursų. Tai gali tapti didėle greitaveikos problema dėl kurios paprasčiausias XML apdorojimas tarpininko sistemoje būtų neįmanomas.

Kaip šios problemos sprendimą galima pasiūlyti reliacinės duomenų bazės valdymo sistemos(RDBVS) naudojimą. Užklausos duomenų bazėse įvykdomos greičiau, be to jos turi daug kitų naudingų funkcijų: atsarginės kopijos, duomenų korektiškumo palaikymas, saugumo palaikymas. Tokioje duomenų bazėje išsaugotume sąryšius tarp ontologijos, ontologijos klasių, šių klasių atributų, web servisų metodų ir perduodamų parametrų. Pačios duomenų bazės

struktūrą galima pasirinkti įvairią. Svarbu, kad teisingai būtų atspindėti šie sąryšiai. Tuomet naudodamiesi SQL (Simple Query Language) užklausomis galėtume greitai gauti norimą informaciją apie web servisus. Kai kurios iš duomenų bazių valdymo sistemos (pvz. Microsoft SQL Server 2005) turi galimybę talpinti XML kalbos duomenis. Tokiu atveju galima būtų prieš talpinant į duomenų bazę WSDL failų net neapdoroti.

Problema, kurią reikėtų išspręsti tai duomenų atnaujinimas. Kadangi apdoroję WSDL failus informaciją apie juos laikytume duomenų bazėje ją reikėtų atnaujinti priklausomai nuo pasikeitimų web servisuose. Galima keli šios problemos sprendimo būdai. Įvykus serviso pasikeitimui, šis servisas inicijuotų kreipimąsi į paieškos tarpininką, informuodamas apie tai, jog WSDL aprašymas yra atnaujintas. Tam reikėtų tarpininko web servise sukurti metodą, kuris priimtu serviso pavadinimą. Tačiau tokiu atvejų realizuojant sistemą kiekviename galutiniame web servise turėtume įgyvendinti tokio metodo iškvietimą. Jei norime turėti tarpininką kuris nepriklauso nuo galutinių web servisų realizacijos, galime kas nustatytą laiko tarpą atnaujinti visus web servisų aprašymus duomenų bazėje. Neigiamos tokios realizacijos savybės: kiekvienas atnaujinimas reikalautų nemažai resursų ir duomenų bazė tarp atnaujinimų vis vien nebūtų visiškai sutampanti su esamais WSDL aprašymais. Taigi pirmąjį sprendimo būdą reikėtų rinktis tuo atveju jei galima realizuoti web servisus taip, jog būtų informuojama apie WSDL aprašo pasikeitimus, kai svarbu sutaupyti paieškos tarpininko veikimo resursus ir kai būtina turėti dabartinį aprašymo atvaizdą duomenų bazėje. Antrąjį sprendimo būdą reikėtų rinktis tuo atveju jei web servisų realizacijai įtakos neturime ir galime leisti neatitikimus tarp aprašymų ir duomenų bazės tarp atnaujinimų. Taip pat reikėtų užtikrinti, jog yra pakankamai resursų atlikti visus atnaujinus (akivaizdu, kad kuo dažniau vykdomi atnaujinimai tuo daugiau resursų reikės, tačiau tuo tikslesnį atvaizdą apie WSDL aprašymus turėsime duomenų bazėje).

4.7.3 Paieškos tarpininko nauda

Paieškos tarpininko naudojimas naudingas tuo, jog galima patalpinti vienoje vietoje daugiau web servisų. Tokiu atveju kliento programai pakanka žinoti tik prisijungi prie tarpininko ypatybes ir nereikia laikyti visų naudojamų web servisų nuorodų.

Dar vienas privalumas tas, jog semantinis apdorojimas vykdomas vienoje vietoje t.y. visą apdorojimo kodą laikome tarpininko web servise. Tai supaprastina semantinės sistemos palaikymą, be to realizuoti kodą reikia tik vieną kartą. Tokioje sistemoje galime naudoti sudėtingesnes technologijas ar algoritmus (pvz. RDBVS panaudojimas). Taip pat yra standartizuojamas semantinio apdorojimo procesas (naudojame arba struktūrą arba atskirų metodų gražinimą). Tai supaprastina naujų klientų realizavimą.

4.8 Web servisų paieška pagal ontologiją

Kol kas aptarėme tik web servisų metodų dinaminį iškvietimą. Tačiau semantikos pagalba galima atlikti ir web servisų paiešką ir jų dinaminį panaudojimą. Tokią paiešką gali inicijuoti klientas, bet labiausiai tikėtina, kad ji bus atliekama paieškos tarpininke (web servisų paieška nebūtų tikslinga jei atliktumėme ją kliento programoje, nes jei naudojame dviejų web servisų architektūrą visus web servisus žinome iš anksto).

Kurdami semantinių web servisų sistemos klientą turime ontologiją ir veiksmus, kuriuos reikia atlikti. Gali būti tokių situacijų kai nesvarbu koks web servisas atliks veiksmus (pvz. mylių vertimas į kilometrus) arba, kai reikalinga pasitelkti kiek įmanoma daugiau web servisų tai pačiai užduočiai atlikti (pvz. norime gauti kuo didesnį prekių sąrašą kad galėtume išrinkti tinkamiausią). Tokiais atvejais galima ieškoti web servisų, kurie yra tos pačios ontologijos. Web servisų galima ieškoti arba tik pagal funkcijas kurias jie atlieka arba ir pagal web servisų charakteristikas, pvz. saugumą, efektyvumą.

Paieška iš kliento pusės galėtų atrodyti panašiai kaip ir tuo atveju jei norime gauti duomenis apie jau žinomus web servisus. Taip pat yra dvi galimybės realizuoti bendravimą tarp paieškos tarpininko serviso ir kliento. Galima gražinti pilną struktūrą apie surastą web servisą arba galima teikti duomenis apie norimą ontologijos klasę arba jos atributą. Pirmuoju atveju paieškos tarpininkui tereikia pateikti tik ontologijos pavadinimą, antruoju reikia pateikti ontologijos pavadinimą ir ieškomo parametro pavadinimą.

Galima ieškoti web serviso tik pagal ontologijos pavadinimą (neatsižvelgiant į tai kokios veiklos sritys realizuotos web servise). Tokiu atveju bus gražinamas arba web serviso adresas arba semantiką aprašanti struktūra. Jei paieškos tarpininko pusėje realizuota semantikos apdorojimo funkcija, būtų patogiau gražinti aprašančią struktūrą, nes norint servisą panaudoti automatiškai šia struktūrą kliento pusėje kurti vis vien tektų.

Jei randami keli tinkami variantai turi būti sprendžiama parinkimo problema. Ją galima spręsti kliento arba paieškos tarpininko pusėje. Jei parinkimo problema sprendžiama kliento pusėje tuomet naudojamas didesnis informacijos srautas tarp paieškos tarpininko ir kliento, nes gražinami visų rastų web servisų parametrai. Tačiau tokiu būdų galima naudoti visus web servisus (jei reikia rasti kuo daugiau tą pačią funkciją atliekančių web servisų) arba galima atlikti parinkimą pagal įvairias serviso charakteristikas. Pavyzdžiui, galima rinki servisą pagal saugumo parametrus t.y. kokia autentikacija naudojama (vartotojo vardas ir slaptažodis; trečios šalies išduotas sertifikatas; Kerberos bilietas), koks naudojamas perduodamų duomenų šifravimas arba galima rinkti servisą pagal tai kaip prieš tai vykdytose sesijose servisas greitai veikė (tam reikia

rinkti statistiką). Dar vienas privalumas sprendžiant parinkimo problemą kliento pusėje – galima naudoti prieš tai buvusių serviso naudojimo atvejų statistikas. T.y. klientas gali kaupti norimus parametrus apie bendravimo su web servisais sesijas. Pagal šiuos parametrus galima rinkti greičiausią, efektyviausią, informatyviausią ar kitą charakteristiką turintį web servisą. Tačiau parinkimą pagal statistiką reikėtų nagrinėti detaliau, nes kyla problemų su statistikų nebeatitikimu esamai situacijai. T.y. jei mes sukaupiame statistiką kuri rodo, kad kažkurie servisai yra geri(pagal charakteristiką) toliau mes naudosime tik tuos servisus. Tokiu būdu neberasime naujų arba atnaujintų, galbūt, geresnių web servisų.

Parinkimo problemą sprendžiant paieškos tarpininko pusėje taip pat įmanoma paieška pagal serviso charakteristikas, tačiau neįmanoma naudoti statistikos, nes paieškos tarpininkas servisų nenaudoja (tik apdoroja servisų aprašus) ir tokios statistikos kaupti negali. Be abejo, būtų galima realizuoti statistikos paėmimą iš klientų, tačiau būtų prarastas sistemos universalumas (reikėtų išankstinių susitarimų dėl paieškos tarpininko kliento realizacijos). Parinkimo problemos sprendimo tarpininko pusėje privalumai: reikia mažesnio srauto tarp paieškos tarpininko ir kliento; kliento programoje nereikia realizuoti jokio papildomo kodo (kadangi kliento programų gali būti daug, nereikia kiekvieną karta realizuoti parinkimo kodo).

Galimas dar vienas web servisų parinkimo problemos sprendimo variantas – sukurti parametrituotą užklausą paieškos tarpininkui, kurioje būtų galima nurodyti ar reikia paieškos tarpininkui atlikti parinkimą ir jei reikia pagal kokius parametrus tai daryti. Tokiu būdu galėtume atlikti parinkimą pagal serviso charakteristikas paieškos tarpininko pusėje, tuomet gauti visų servisų, atitinkančių šiuos kriterijus, sąrašą kliento pusėje ir tuomet, jei reikia, atlikti parinkimą pagal statistiką. Pavyzdžiui, klientas paieškos tarpininkui pateikia užklausą surasti visus web servisus kurių ontologija yra *Parduotuvė*, kuris autentikacijos nenaudoja, o duomenis perduoda šifruotus SSL protokolu. Klientas gavęs tokių web servisų sąrašą pasirenka tą kuris, pagal statistiką, turi pigiausių prekių sąrašą arba veikia greičiausiai. Tokio sprendimo varianto privalumas tas, jog sumažinamas duomenų srautas tarp kliento ir paieškos tarpininko, nes siunčiami tik charakteristikas atitinkančių web servisų duomenys, ir taip pat yra galimybė panaudoti statistiką apie web serviso veikimą kliento pusėje.

4.9 Paieškos slenkstis

Iš anksčiau aptartų semantinių web servisų sistemų matome, kad bet koks dinaminis servisų panaudojimas reikalauja papildomų resursų. Visų prima reikalingas tinklo srautas: jei tai dviejų web servisų sistema šis srautas reikalingas parsisiųsti semantiškai anotuotus WSDL aprašus, jei tarpininko sistema - reikia atlikti užklausas ir priimti rezultatus. Tai pat reikalingi resursai

semantiniam apdorojimui (atitikmenų tarp serviso ir ontologijos sudarymui). Resursų sunaudojame ir atlikdami dinamiška SOAP užklausų formavimą.

Dar viena priežastis, kodėl dažnas servisų semantikos peržiūrėjimas gali būti problema, yra ta, jog paieškos tarpininko paslauga gali būti mokama. Šiuo metu įvairių web servisų panaudojimas būna apmokestintas, todėl tikėtina, kad ateityje gali atsirasti web servisų paieškos centrai (atliekantys paiešką semantiniu pagrindu), kurie gali būti apmokestinti. Tokie centrai veiktų paieškos tarpininko rolėje. Jei mokestis imamas už kiekvieną užklausą, sistemos veikimas tampa brangus.

Tačiau nėra būtina atlikti paiešką kiekvieną kartą naudojant servisus. Daugeliu atveju paieškos su tais pačiais parametrais rezultatas bus tas pats (nebent paieškos centras labai dinamiškai yra atnaujinamas). Galima būtų naudoti jau surastus web servisus daugiau į paieškos tarpininką nesikreipiant.

Jei atliktume paiešką tik kartą ir daugiau semantikos nenagrinėtume prarastume nemažai gerų semantinių sistemų savybių. Jei servise įvyktų pasikeitimas sistema prie jo nebeprisitaikytu. Taip pat atsiradus naujiems, geriau funkcijas atliekantiems, web servisams, jų panaudojimas nebūtų vykdomas.

Šias problemas galėtų išspęsti paieškos slenkstis. Paieškos slenkstis tai skaitinė reikšmė. Viršijus šią reikšmę klientas kreipiasi į paieškos trapininką su užklausa. Priešingu atveju naudojamasi jau turimais web servisais.

Paieškos slenkstis gali reikšti įvairias veikimo savybes, pavyzdžiui laiko tarpą, kuris praėjo nuo paskutinio servisų atnaujinimo (paskutinio kreipimosi į paieškos tarpininką) arba serviso panaudojimo atvejų skaičius. Gali būti naudojami keli slenksčiai. Slenksčio prasmė turi būti parenkama pagal sistemos kontekstą, t.y. reikia atsižvelgti kas ir kaip keičiasi paieškos tarpininko turimame servisų sąraše, kokios sritys laikomos svarbiomis kliento pusėje.

Skaitinė slenksčio reikšme gali būti statinė arba dinaminė. Statinė reikšmė gali būti parenkama iš konfigūracinių nustatymų. Šią reikšmę gali parinkti sistemos naudotojas atsižvelgdamas į patirtį arba numatydamas sistemos ypatybes. Dinaminė slenksčio reikšmė galėtų būti parenkamai priklausomai nuo įvairių su sistemos veikimu susiejusių savybių. Pavyzdžiui, turime sistemą, kurioje slenkstis, tai laiko tarpas nuo paskutinio kreipimosi į paieškos tarpininką. Pradžioje slenkstis parenkamas santykinai mažas. Atlikus užklausą ir gavus rezultatus, klientas patikrina ar užklausa gavo kokius nors naujus rezultatus lyginant su prieš tai buvusia užklausa. Jei ne - slenksčio reikšmė padidinama. Jei naujų rezultatų yra - slenksčio reikšmė sumažinama. Tokiu būdu sistema gali prisitaikyti prie paieškos tarpininko web servisų sąrašo pasikeitimų dinamikos (jei tarpininko turimų servisų sąraše vyks mažai pasikeitimų,

slenksčio reikšmė taps didelė ir kreipimaisi į tarpininką bus vykdomi retai, o jei pasikeitimai bus dažni slenksčio reikšmė bus maža ir paiešką atliksime dažniau).

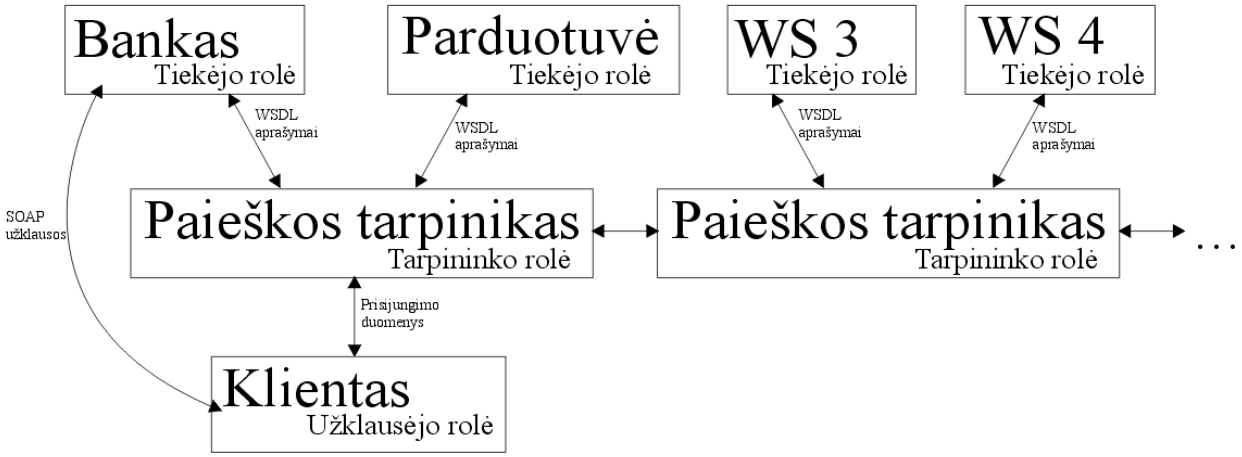
Atnaujinimą visada teks vykdyti jei servisuose įvyko pakeitimas ir sugeneruotomis SOAP užklausomi užduoties atlikti nebeįmanoma. Tai nesukelia didelių problemų jei SOAP užklausa netinkama sintaksiškai (pvz. metode atsirado trečias parametras). Serviso kvietimas naudojant netinkamą sintaksę iššauks veikimo klaidą, todėl visada žinosime, kad veikimas negalimas ir atliksime kreipimąsi į paieškos tarpininką tam, kad gautume pasikeitusį serviso aprašą. Tačiau galimas atvejis kai sintaksiškai užklausa galima, tačiau semantiškai ji nebetinka. Deja būdo kaip aptikti tokį neatitikimą neatlikus semantinio apdorojimo nėra. Dėl šios priežasties slenksčio naudojimas netinkamas, tokiose sistemose, kuriose reikalingas aukštas semantikos korektiškumo užtikrinimas. Tačiau daugeliu atveju slenkstį naudoti praktikoje būtų galima, nes pagal semantiką ieškomi web servisai neturėtų atlikti svarbių užduočių, pvz. bankinių pavedimų, nes tokius servisus saugiau aprašyti statiškai. Be to mažai tikėtina, kad metodas tuo pačiu pavadinimu bus pradėtas naudoti kitoms ontologijos funkcijoms atlikti.

Slenksčio privalumai: sumažina srautą tarp kliento ir paieškos tarpininko; pagretina tarpininko veikimą; jei paieška mokama – kliento programos išlaikymas tampa pigesnis.

Slenksčio trūkumai: nėra galimybės visiškai užtikrinti, kad klientas naudojasi ontologiją atitinkančiais web servisais; slenksčio veikimo algoritmas turi būti aprašomas kliento programoje, todėl turint daug klientų sistemos palaikymas tampa sudėtingesnis.

4.10 Paieškos tarpininkų tinklas

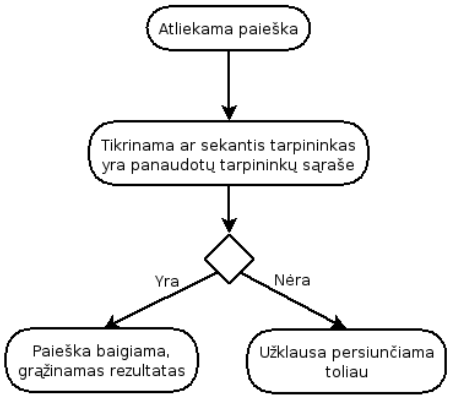
Norėdami praplėsti paieškos apimtį arba sumažinti paieškos tarpininko apkrovimą galima panaudoti paieškos tarpininkų tinklą. Tokios sistemos architektūroje paieškos tarpininkų naudotume ne vieną, o keletą. Tam, kad nekeisti kliento realizacijos klientas galės jungtis ir atsakymą gaus tik iš vieno paieškos tarpininko. Tarpininkų tinklas bus pasiekiamas tik paieškos tarpininkams. Paieškos tarpininkų tinklas vaizduojamas 4 pav..



4 pav. Paieškos tarpininkų tinklo architektūra

Pagal 4 pav. pateiktą schemą matome, kad paieškos tarpininkai bendrauja nuosekliai, t.y. kiekvienas paieškos tarpininkas žino tik vieną sekantį tarpininką. Neradęs duomenų pagal pateiktą užklausą šią užklausą tarpininkas persiunčia vieninteliam žinomam tarpininkui iš tinklo. Tam, kad nereikėtų duomenis siųsti visa grandine atgal, persiunčiamoje užklausoje galima nurodyti pradinio užklausėjo adresą.

Paieškos tarpininkų grandinėse gali susidaryti begaliniai ciklai. Tokie ciklai susidarys jei esančioje grandinėje paskutinis tarpininkas vėl rodo į pirmąjį. Šia problemą galima spręsto keliais būdais. Galima su užklausa siųsti visų paieškos tarpininkų adresus kurie nieko neradę užklausą persiuntė. Tokiu atveju paieškos tarpininko veiksmai turėtų būti tokie: pagal gautą užklausą atliekama paieška; tarpininkas patikrina ar adresas tarpininko, kurį šis žino kaip sekantį iš tinklo, nėra jau sąraše; jei yra - paieška baigiama ir grąžinami visi paieškos rezultatai; jei nėra - paieškos tarpininkas įtraukia savo adresą į sąrašą ir persiunčia užklausą su sąrašu toliau. Paieškos rezultatai keliauja kartu su užklausa iki tol ko visi paieškos tarpininkai iš tinklo įtraukiami į panaudotųjų sąrašą. Veiklos diagrama vaizduojamas 5 pav.



5 pav. Paieškos tarpininko veiklos diagrama

Jei tarpininkų tinklas didelis, galima nustatyti kiek tarpininkų mes norime įtraukti į paiešką arba kokio dydžio turi būti grąžinamas rezultatas. Tokiu atveju taip pat išsprendžiama ir begalinio ciklo problema. Taip pat gali būti, kad reikiamas tik vienas servisas, todėl rezultatas turėtų būti grąžinamas iš kart po sėkmingos paieškos.

Paieškos tarpininkų tinkle bendravimas galėtų būti realizuotas ir nenuosekliai. Tuomet kiekvienas paieškos tarpininkas turėtų žinoti visų tinkle esančių tarpininkų adresus. Tokį bendravimą yra gana paprasta realizuoti, nes tarpininko atsakymą į užklausą galima realizuoti taip kaip realizuotas atsakymas į kliento užklausą (čia reikėtų pridėti tik parametrą kuris nurodytų, jog nereikia atlikti paieškos tinkle, nes priešingu atveju paieškos tarpininkas vėl ieškos tame pačiame tinkle). Tokios architektūros trūkumas – reikia palaikyti visų tarpininkų sąrašą. Taip pat gali būti problema užtikrinat didelio tinklo srauto apdorojimą.

Taigi paieškos tarpininkų tinklas naudingas tuo, jog padidėja duomenų paieškai kiekis, paiešką galima paskirstyti po atskirus paieškos tarpininkus. Šio tinklo trūkumai: jei paieška realizuota nuosekliai - ji gali trukti ilgiau, jei nenuosekliai - reikia palaikyti visų paieškos tarpininkų esančių tinkle sąrašą.

Išvados

Šiame darbe išanalizavome semantinių web servisų sistemų kūrimo ypatumus. Buvo pateiktas sistemos dalykinės srities aprašymas – ontologija; keletas web servisų realizuojančių šią ontologiją; ontologijos ir web servisų sąryšių aprašymas – anotacija; kliento programa naudojanti semantiškai anotuotus servisus.

Semantinių web servisų sistemai buvo pasiūlytos trys skirtingos architektūros. Buvo išnagrinėti šių architektūrų privalumai ir trūkumai. Buvo aptartos architektūrų realizacijos ypatybės, bei pasiūlyti patobulinimai.

Priežastys kodėl semantinių web servisų sistemos kol kas retai įgyvendinamos praktikoje yra kelios. Yra technologinių sunkumų įgyvendinant šias sistemas: naudojant WSDL-S standartą web serviso semantikai aprašyti reikalingas WSDL 2.0 standarto palaikymas, tačiau toks standartas retai naudojamas, įrankiai, skirti apdoroti šiuo standartu parašytus web servisų aprašus, yra neseniai atsiradę ir mažai išvystyti; nėra gerai išvystytų semantinio apdorojimo ir dinaminio serviso iškvietimo įrankių, todėl visa tai tenka programuoti kiekvienoje sistemoje. Yra ir netechnologinių problemų: jei sistema pakankamai maža, kurti jos semantinį aprašą nėra tikslinga (paprasčiau yra sistemą išsamiai dokumentuti), o didelių viešai prieinamų web servisų paieškos sistemų nėra daug, be to reikia kad šios sistemos palaikytų semantiką, web servisai būtų semantiškai anotuotoji. Tačiau tikėtina kad ateityje tokios sistemos atsiras: kompanijos Google ir Microsoft kuria web servisais paremtų dedikuotų serverių sistemas, o kompanija IBM turi viziją, jog vartotojai turėtų naudotis tik terminalais, kurie sudėtingus skaičiavimus atliktų centralizuotose sistemose. Jie visi šie projektai būtų sėkmingi, atsirastų didelis kiekis viešai prieinamų web servisų, kurių žmogus panaudoti nebesugebėtų, todėl prireiktų automatizavimo, o automatizavimo problemas galėtų spręsti semantika. Dar viena netechnologinė kliūtis ta, jog reikia kurti ontologijų registrus, nes šiuo metu kyla problema kai tos pačios veiklos ontologijos skirtingų autorių aprašomos skirtingai ir nors sistemos atlieka tas pačias funkcijas, dėl ontologijų aprašymo skirtumų, sistemų tarpusavio sąveikavimą neįmanomas.

Darbe pasiūlytos trys skirtingos architektūros semantinių web servisų realizacijai: dviejų web servisų, paieškos tarpininko ir paieškos tarpininkų tinklo. Paprasčiausia sistema sudaro du semantiškai anotuoti web servisai ir klientas. Sistemos, kurioje semantinį apdorojimą atlieka pats klientas, privalumas – ji nepriklauso nuo jokių išorinių resursų, realizaciją galima pasirinkti tinkamą konkrečiai situacijai. Vienintelis reikalavimas standartizacijai WSDL-S specifikacijos laikymasis. Sistemos silpnosios pusės – nestandartizuota semantikos apdorojimo realizacija gali sukelti palaikymo problemų, sistema turi savyje laikyti visų žinomų web servisų aprašymus (bent jau nuorodas į aprašymus). Dar vienas šios sistemos ypatumas (priklausomai nuo situacijos tai

nebūtinai yra trūkumas) kiekvienai sistemai reikia realizuoti atskirą paieškos algoritmą, o paieška vykdoma kliento resursais. Kai kurias iš šių problemų padeda išspręsti paieškos tarpininkas. Paieškos tarpininko naudojimas naudingas tuo, jog galima patalpinti vienoje vietoje daugiau web servisų. Tokiu atveju kliento programai pakanka žinoti tik prisijungi prie tarpininko ypatybes ir nereikia laikyti visų naudojamų web servisų nuorodų. Dar vienas privalumas tas, jog semantinis apdorojimas vykdomas vienoje vietoje t.y. visą apdorojimo kodą laiko tarpininko web servise. Tai supaprastina semantinės sistemos palaikymą, be to realizuoti kodą reikia tik vieną kartą. Tokioje sistemoje galime naudoti sudėtingesnes technologijas ar algoritmus (pvz. RDBVS panaudojimas). Taip pat yra standartizuojamas semantinio apdorojimo procesas (naudojame arba struktūrą arba atskirų metodų gražinimą). Tai supaprastina naujų klientų programinį įgyvendinimą. Paieškos tarpininkų tinklas naudingas tuo, jog padidėja duomenų paieškai kiekis, paiešką galima paskirstyti po atskirus paieškos tarpininkus. Šio tinklo trūkumai: jei paieška realizuota nuosekliai ji gali trukti ilgiau, jei nenuosekliai reikia palaikyti visų paieškos tarpininkų esančių tinkle sąrašą.

Taip pat šiame darbe buvo pasiūlytas paieškos slenkstis. Slenksčio privalumai: sumažina srautą tarp kliento ir paieškos tarpininko; pagretina tarpininko veikimą; jei paieška mokama – kliento programos išlaikymas tampa pigesnis. Slenksčio trūkumai: nėra galimybės visiškai užtikrinti, kad klientas naudojasi ontologiją atitinkančiais web servisais; slenksčio veikimo algoritmas turi būti aprašomas kliento programoje, todėl, turint daug klientų, sistemos palaikymas tampa sudėtingesnis.

Šaltinių sąrašas

[AZA+05] I. Budak Arpinar, Ruoyan Zhang, Boanerges Aleman-Meza, Angela Maduko. Ontology-driven Web services composition platform. Springer-Verlag 2005. 25 pages.

[BCS04] Farnoush Banaei-Kashani, Ching-Chien Chen, and Cyrus Shahabi. WSPDS: Web Services Peer-to-peer Discovery Service. 2004. 7 pages.

[CS06] Jorge Cardoso, Amit P. Sheth. Semantic Web Services, Processes and Applications, Springer Science+Business Media 2006. 383 pages.

[FB06] Andreas Friesen, Egon Borger. A HighLevel Specification for Semantic Web Service Discovery Services. ACM New York, USA 2006. 8 pages.

[GK05] Venkat N Gudivada, Mrunalini Kalavala. Semantic Web Services, Consortium for Computing Sciences in Colleges. 2005, 14 pages.

[JAX08] The Java API for XML Processing
[žiūrėta 2008-11-19]. Prieiga per Internetą:
<http://www.w3.org/Submission/WSDL-S/>

[MSWS08] METEOR-S: Semantic Web Services and Processes. 2008.
[žiūrėta 2008-02-19]. Prieiga per Internetą:
< http://lsdis.cs.uga.edu/projects/meteor-s/>

[Ont08] Ontology (information science). 2008.
[žiūrėta 2008-03-21]. Prieiga per Internetą:
< http://en.wikipedia.org/wiki/Ontology_(computer_science) >

[Ore04] Eyal Oren. WSMX Execution Semantics:Executable Software Specification. Digital Enterprise Research Institute (DERI), Galway, Ireland, 2004, 11 pages

[OWL06] OWL-S 1.2 Pre-Release. 2006.
[žiūrėta 2008-02-15]. Prieiga per Internetą:
< http://www.daml.org/services/owl-s/>

[UDDI08] Universal Description Discovery and Integration. 2008.
[žiūrėta 2008-03-21]. Prieiga per Internetą:
<http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration>

[VSS+05] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, John Miller. METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services, University

of Georgia 2005, 24 pages.

[WCM04] WS Choreography Model Overview. W3C Working Draft 24 March 2004. [žiūrėta 2008-04-03]. Prieiga per Internetą: <http://www.w3.org/TR/ws-chor-model/>

[WSD05] Web Service Semantics - WSDL-S. W3C Member Submission 7 November 2005. [žiūrėta 2008-02-16]. Prieiga per Internetą: <http://www.w3.org/Submission/WSDL-S/>

[WSM08] ESSI WSMO working group. [žiūrėta 2008-02-16]. Prieiga per Internetą: <http://www.wsmo.org/>

[WVH+07] Xia Wang, Tomas Vitvar, Manfred Hauswirth, and Doug Foxvog. Building Application Ontologies from Descriptions of Semantic Web Servines. IEEE/WIC/ACM International Conference on Web Intelligence, 2007, 7 pages

[ZB06] Diego Zuquim Guimarães Garcia, Maria Beatriz Felgar de Toledo. Semantics-enriched QoS Policies for Web Service Interactions. WebMedia. 2006, p. 35-43.

Sąvokų apibrėžimai ir santrumpų sąrašas

Anotacija - procesas sujungiantis metaduomenis su resursu.

BPEL – angl. Business Process Execution Language. Veiklos procesų vykdymo kalba

BPWS - Bussines Process for Web Services. Web servisų veiklos procesai.

DTD – angl. Document Type Definition. Dokumento tipo apibrėžimas.

Kerberos bilietas – autentikacijos užtikrinimo priemonė, pasitelkianti trečios šalies patvirtinimą – bilietą.

Ontologija yra formalus veiklos srities sudedamųjų dalių ir ryšių tarp šių dalių aprašymas.

OWL – angl. Web Ontology Language. Web ontologijų kalba.

OWL-S – OWL skirta semantiniams web servisams.

QoS – angl. Quality of Service. Paslaugos kokybė.

SOA – angl. Service oriented application . Servisais paremta aplikacija.

SOAP – angl. Simple Object Access Protocol. Priėjimo prie paprastų objektų protokolas

UDDI – angl. Universal Discovery, Description, and Integration. Universali paieška, aprašymas ir integracija. UDDI – tai web servisų registrai skirti web servisų paieškai.

W3C – pasaulinio žiniatinklio konsorciumas

WSDL – angl. Web Services Description Language. Web servisų apibūdinimo kalba.

WSDL-S – WSDL kalba praplėsta semantiniam aprašymui.

WSMO – angl. Web Service Modeling Ontology. Web servisų modeliuojama ontologija.

XML - angl. eXtensible Markup Language. Praplečiama žymėjimo kalba.

XSD - angl. XML Schema Definition. XML schemos aprašymas.

1 priedas. Ontologija „Prekyba“

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:p1="http://www.owl-ontologies.com/assert.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://localhost/Prekyba.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://localhost/Prekyba.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Bankas">
  </owl:Class>
  <owl:Class rdf:about="#Preke">
  </owl:Class>
  <owl:Class rdf:ID="Saskaita">
    <rdfs:subClassOf rdf:resource="#Bankas"/>
  </owl:Class>
  <owl:Class rdf:about="#Litai">
  </owl:Class>
  <owl:Class rdf:about="#Parduotuve">
  </owl:Class>
  <owl:ObjectProperty rdf:ID="PrekiusSarasas">
    <rdfs:range rdf:resource="#Preke"/>
    <rdfs:domain rdf:resource="#Parduotuve"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Kaina">
    <rdfs:domain rdf:resource="#Preke"/>
    <rdfs:range rdf:resource="#Litai"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Likutis">
    <rdfs:domain rdf:resource="#Saskaita"/>
    <rdfs:range rdf:resource="#Litai"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="SaskNr">
    <rdfs:domain rdf:resource="#Saskaita"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Pavadinimas">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="BankoID">
    <rdfs:domain rdf:resource="#Bankas"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Suma">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:domain rdf:resource="#Litai"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="Pirkti">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Parduotuve"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="Moketi">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

2 priedas. Anotuotas web serviso „Bankas“ aprašas

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://bank.org" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://bank.org" xmlns:intf="http://bank.org" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://bank.org" xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="GetBankoId">
    <complexType/>
   </element>
   <element name="GetBankoIdResponse">
    <complexType>
     <sequence>
      <element name="GetBankoIdReturn" type="xsd:int" wssem:modelReference="BankoID"/>
     </sequence>
    </complexType>
   </element>
   <element name="Pirkimas" wssem:modelReference="Pirkti">
    <complexType>
     <sequence>
      <element name="saskaita" type="xsd:int" wssem:modelReference=" SaskNr"/>
      <element name="suma" type="xsd:float" wssem:modelReference=" #Litai"/>
     </sequence>
    </complexType>
   </element>
   <element name="PirkimasResponse">
    <complexType/>
   </element>
  </schema>
 </wsdl:types>
  <wsdl:message name="GetBankoIdResponse">
    <wsdl:part element="impl:GetBankoIdResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="PirkimasResponse">
    <wsdl:part element="impl:PirkimasResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="GetBankoIdRequest">
    <wsdl:part element="impl:GetBankoId" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="PirkimasRequest">
    <wsdl:part element="impl:Pirkimas" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="Bankas">
    <wsdl:operation name="GetBankoId">
      <wsdl:input message="impl:GetBankoIdRequest" name="GetBankoIdRequest"/>
      <wsdl:output message="impl:GetBankoIdResponse" name="GetBankoIdResponse"/>
    </wsdl:operation>
    <wsdl:operation name="Pirkimas">
      <wsdl:input message="impl:PirkimasRequest" name="PirkimasRequest"/>
      <wsdl:output message="impl:PirkimasResponse" name="PirkimasResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BankasSoapBinding" type="impl:Bankas">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetBankoId">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="GetBankoIdRequest">
```

```
            <wsdlsoap:body use="literal"/>
          </wsdl:input>
          <wsdl:output name="GetBankoIdResponse">
            <wsdlsoap:body use="literal"/>
          </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Pirkimas">
          <wsdlsoap:operation soapAction=""/>
          <wsdl:input name="PirkimasRequest">
            <wsdlsoap:body use="literal"/>
          </wsdl:input>
          <wsdl:output name="PirkimasResponse">
            <wsdlsoap:body use="literal"/>
          </wsdl:output>
        </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="BankasService">
        <wsdl:port binding="impl:BankasSoapBinding" name="Bankas">
          <wsdlsoap:address location="http://localhost:8080/Bankas/services/Bankas"/>
        </wsdl:port>
      </wsdl:service>
    </wsdl:definitions>
```