INTERNETINĖS TECHNOLOGIJOS

Pasaulinio saityno paslaugų kompozicijų skaidymas į modulius

Jolanta Miliauskaitė

Vilniaus universiteto doktorantė
Vilnius University, Doctoral student
Akademijos g. 4, LT-08663 Vilnius
El. paštas: jolanta.miliauskaite@mii.vu.lt

Albertas Čaplinskas

Vilniaus universiteto Matematikos ir informatikos instituto vyriausiasis mokslo darbuotojas
Vilnius University, Institute of Mathematics and Informatics, Principal researcher
Akademijos g. 4, LT-08663 Vilnius
El. paštas: albertas.caplinskas@mii.vu.lt

Straipsnyje aptariami, analizuojami ir vertinami svarbiausi pasiūlymai, kaip spręsti pasaulinio saityno paslaugų kompozicijų skaidymo į modulius uždavinį. Tai yra vienas iš aktualiausių e. verslo sistemų kūrimo uždavinių, kurio neišsprendus neįmanoma adekvačiai apdoroti vykdymo meto įvykių, dinamiškai perplanuoti verslo procesų, atsižvelgti į paslaugų kokybės charakteristikas ir įgyvendinti daugelį kitų svarbių e. verslo sistemos funkcijų. Straipsnyje pateiktas esamos padėties vertinimas ir išryškintos svarbiausios dar neišspręstos problemos, su kuriomis susiduriama skaidant į modulius pasaulinio saityno paslaugų kompozicijas. Tikimasi, kad atlikta analizė bus naudinga visiems šioje srityje dirbantiems mokslininkams ir padės jiems planuoti savo tolesnius tyrimus.

Įvadas

Straipsnyje nagrinėjamas pasaulinio saityno paslaugų (PSP, angl. *Web service*) kompozicijų skaidymo į modulius uždavinys. PSP yra viena iš internetu teikiamų programinių paslaugų rūšių. Jos yra skirstomos į viešąsias ir verslo paslaugas. Viešosios paslaugos (pvz., kompiuterinio pašto paslaugos) su jokiu verslo procesu nesiejamos, jas gali gauti bet kuris interneto vartotojas. Verslo paslaugos (pvz., registruoti kliento duomenis) yra kokio nors verslo proceso (pvz., rezervuoti kambarį viešbutyje) dalis. Jos aprašomos vadinamosiomis kontrakto sąlygomis, apibūdinančiomis jų pobūdį, kokybę, gavimo sąlygas. Kontrakto sąlygos paprastai yra aprašomos WSDL kalba (Chinnici ir kt., 2007; Chinnici ir kt., 2007a), pridedant prie aprašo XML schemos (Fallside, Walmsley, 2004; Thompson ir kt., 2004; Biron, Malhotra, 2004) apibrėžtį ir galbūt kokių nors elgsenų taisyklių (angl. *policies*) aprašus WS-Policy kalba (Vedamuthu ir kt., 2007). Visos PSP turi sąsajas, per kurias jos yra užprašomos, ir programines realizacijas. Jos inkapsuliuoja kokius nors kompiuterinius išteklius (informaciją, programas ar kt.) ir sudaro galimybę standartizuotu būdu pasinaudoti tais ištekliais. Svarbiausias PSP trūkumas yra tai, kad jų aprašuose aprašomi tik sintaksiniai sąsajos ypatumai. Nei apie paslaugos semantiką ar pragmatiką, nei apie jos nefunkcines ar ekstrafunkcines savybes informacija

apraše neteikiama. Siekiant pašalinti šį trūkumą, sukurtos pasaulinio semantinio saityno paslaugos (PSSP, angl. *semantic Web service*). Jų aprašai papildyti semantinėmis anotacijomis ir galbūt nefunkcinių bei ekstrafunkcinių savybių specifikacijomis. Dėl to lengviau automatizuoti PSSP paiešką, komponavimą, aktyvavimą ir jų teikimo stebėseną.

Tarpusavyje komponuojant primityviąsias PSP ar PSSP, galima kurti vis sudėtingesnes jų kompozicijas ir sykiu vis sudėtingesnius verslo ar viešųjų paslaugų kompleksus. Kompozicijos gali būti *centralizuotos* ir *decentralizuotos*. Centralizuotos kompozicijos dažniausiai kuria verslo procesus. Jos turi koordinatoriumi vadinamą programą, koordinuojančią kuriamo verslo proceso veikimą. Koordinatorius nustato kompozicijai priklausančių paslaugų kvietimo tvarką. Informacijos, kad jos yra sukomponuotos viena su kita, kompoziciją sudarančios paslaugos neturi. Procesas gali būti vykdomas daugelyje organizacijų, bet yra koordinuojamas organizacijos, kuri yra jo savininkas, požiūriu. Centralizuota kompozicija gali būti išskirstyta, t. y. dekomponuota į kelis lygiagrečiai vykdomus paslaugų klasterius, koordinuojamus skirtingų koordinatorių. Tačiau ir šiuo atveju visi koordinatoriai vadovaujasi vienu visiems bendru verslo proceso savininko požiūriu. Decentralizuota kompozicija koordinatoriaus neturi. Jos veikimas koordinuojamas paslaugoms tarpusavyje keičiantis pranešimais. Kiekviena tokiai kompozicijai priklausanti paslauga turi informaciją, kokios kompozicijos sudėtyje ji veikia ir su kokiomis kitomis paslaugomis yra sukomponuota. Ši informacija nusako ir paslaugos vaidmenį bei jos indėlį į verslo procesą (jos vykdomas operacijas; maksimalią leidžiamą tų operacijų trukmę; su kuo, kada ir kokiais pranešimais keistis).

Kompozicijos kuriamos pagal specifikacijas, aprašančias jų pageidaujamas savybes. Kompozicija nebūtinai yra kuriama galutinio vartotojo poreikiams tenkinti. Paslaugų teikėjai gali paslaugos gavėjų prašomas paslaugas dinamiškai komponuoti iš smulkesnių paslaugų. Tarpinių kompozicijų skaičius neribojamas. Galų gale šitaip sukuriamos galutinio vartotojo poreikius tenkinančios verslo ar viešosios paslaugos. Šiame straipsnyje nagrinėjamas tik verslo paslaugų komponavimas siekiant sukurti dinamines e. verslo sistemas. Kitaip tariant, nagrinėjamos tik kompozicijos, realizuojančios kokį nors e. verslo sistemos procesą ar jo dalį. Tokios kompozicijos specifikuojamos kokia nors verslo procesų aprašymo kalba, pavyzdžiui, BPEL kalba (Andrews, 2003). Jei komponuojamos PSP, tai specifikacija rengiama rankiniu būdu, o jei PSSP – dažniausiai yra sudaroma automatiškai, naudojant dirbtinio intelekto teorijos planavimo metodus. Abiem atvejais specifikacija aprašo verslo proceso logiką, komponuojamas paslaugas ir jų vykdymo tvarką. Verslo procesas nebūtinai turi būti tiesinis, jame leidžiami ciklai, iššakojimai ir kitos netiesinės struktūros. Specifikacija kiekvienai paslaugai nurodo kompiuterio jungties (angl. *port*), per kurią galima paimti tos paslaugos WSDL aprašą, numerį. Konkretūs paslaugų teikėjai neįvardijami, komunikavimo protokolai nenurodomi. Tai daroma tam, kad kompozicijos egzempliorius būtų galima išskleisti (angl. *deploy*) skirtingose vykdymo aplinkose ir bendrauti su teikėjais skirtingomis komunikavimo technologijomis. Išskleidimo metu kompozicijos egzempliorius yra arba susiejamas su konkrečių teikėjų paslaugomis, arba dinaminio susiejimo atveju jam yra parenkama konkreti tokių paslaugų paieškos strategija.

PSP kompozicijoms kurti dažniausiai naudojami arba orkestravimo, arba choreografijos metodai. Šitaip sukurtos kompozicijos negeba efektyviai apdoroti vykdymo meto įvykių, kurių dinamiškoje PSP vykdymo aplinkoje vyksta labai daug. Priežastis – kompozicijų realizavimas monolitinėmis, vadinamosiomis *kliento programomis*. Paslaugas realizuojantis programinis kodas į kliento programą neįtraukiamas. Jį dinamiškai kviečia tą programą interpretuojanti vadinamoji *vykdymo mašina.* Iš tiesų tokia programa yra statinė. Vykdymo metu jos arba apskritai neleidžiama keisti, arba geriausiu atveju tai galima daryti tik labai ribotai, pakeičiant suplanuotas konkrečias paslaugas joms ekvivalenčiomis paslaugomis. Negalima apdoroti situ-

acijų, kai teikiamos paslaugos ne visai sutampa su paslaugos gavėjo prašomomis paslaugomis, negalima suplanuotų PSP dinamiškai pakeisti kitomis arba jų sukomponuoti iš smulkesnių PSP. Taip pat negalima atsižvelgti į vartotojo suformuluotus paslaugos kokybės reikalavimus ir iš kelių tą patį funkcionalumą turinčių paslaugų vykdymo metu parinkti tinkamiausią, nes nėra kaip pasinaudoti vadinamosiomis *elgsenos taisyklėmis* (angl. *policy*), aprašančiomis paslaugos parinkimo strategiją, išskyrus atvejus, kai tos taisyklės yra suprogramuotos kliento programoje. Be to, aktyvuoti trečiųjų šalių teikiamas paslaugas yra daug sudėtingiau, negu aktyvuoti objektų lokaliuosius metodus objektinėse programų sistemose. Aktyvuojant tokias paslaugas, iškyla ypatingųjų situacijų apdorojimo, finansinio atsiskaitymo, žurnalizavimo, autentifikavimo, paslaugos vykdymo stebėsenos ir kiti klausimai. Jie vadinami *pagalbiniais turiniais*, papildančiais *pagrindiniu turiniu* vadinamą paslaugos dalykinį funkcionalumą. Monolitinėje kliento programoje pagrindinis turinys yra supintas su papildomais, jų negalima vieno nuo kito atskirti. Trumpai tariant, monolitinės kompozicijų realizavimo programos nepajėgios spręsti daugelio svarbių uždavinių, jas būtina skaidyti į modulius. Tačiau skaidyti kompozicijas į modulius nėra paprasta, nes minėti turiniai dažniausiai susikerta. Todėl juos reikia realizuoti vadinamaisiais *aspektais*, įvestais programų sistemų kūrimo aspektinėje paradigmoje (Lopes, 2005). Tradicinės kompozicijų specifikavimo kalbos neturi priemonių aspektams specifikuoti. PSSP šio uždavinio automatiškai taip pat neišsprendžia. Nepakanka tik semantizuoti paslaugų aprašus ir automatizuoti kompozicijų specifikacijų sudarymą. Norint skaidyti į modulius paslaugų kompozicijas, reikia išspręsti dar ir daugelį kitų problemų. Šio straipsnio tikslas – aptarti siūlomus skaidymo į modulius uždavinio sprendimo būdus, įvertinti esamą padėtį ir išryškinti svarbiausias dar neišspręstas problemas. Autoriai tikisi, kad atlikta analizė bus naudinga visiems šioje srityje dirbantiems mokslininkams, padės jiems planuoti tyrimus.

Dauguma šiuo metu pasiūlytų paslaugų kompozicijų skaidymo į modulius uždavinio sprendimo būdų (Charfi, Mezini, 2004; Charfi, Mezini, 2004a; d'Hondt, 2004; Cottenier, Elrad, 2005; Singh ir kt., 2005; Cibrán, Verheecke, 2005; Verheecke, 2006; Navarro ir kt., 2006; Akkawi ir kt., 2006; Braem ir kt., 2006; Charfi, 2007; Cibrán, 2007; Braem, Joncheere, 2007; Gheysels, 2007; Braem, D. Gheysels, 2007; Braem ir kt., 2007; Fraine, Braem, 2007; Charfi, 2007) yra grindžiami aspektinėmis technologijomis, kurias naudojant bandoma atskirti verslo proceso logiką nuo verslo taisyklių, ribojimų ir kitų pagalbinių turinių. Siūloma, kaip tiesiogiai skaidyti į modulius centralizuotas ir decentralizuotas PSP kompozicijas, apimant ir išskirstytąsias, kaip PSP kompozicijoms naudoti jų vykdymo infrastruktūrą arba kaip skaidyti į modulius PSP kompozicijas, dekomponuojant į aspektus visą išskirstytąją programų sistemą. Tolesnė šio straipsnio struktūra atitinka aptartąją pasiūlymų struktūrą. Antrajame straipsnio skyriuje nagrinėjama, kaip siūloma skaidyti į modulius centralizuotas PSP kompozicijas, trečiajame – kaip skaidyti į modulius decentralizuotas kompozicijas, ketvirtasis skyrius skirtas kompozicijų skaidymui į modulius naudojant infrastruktūrą problemų analizei, penktasis skyrius – siūlymams dekomponuoti į aspektus visą išskirstytąją programų sistemą ir pagaliau šeštajame skyriuje pateikiami viso darbo apibendrinimai ir išvados.

Centralizuotų PSP kompozicijų skaidymas į modulius

Pateikta keletas koncepcijų, kaip įvedant aspektus skaidyti į modulius centralizuotas PSP kompozicijas. Išsamiausios yra projektų AWED-WSML (Verheecke, 2006; Navarro ir kt., 2006; Navarro ir kt., 2006a), AO4BPEL (Charfi, Mezini, 2004; Charfi, Mezini, 2004a; Charfi, 2007), Padus (Braem ir kt., 2006; Braem, Joncheere, 2007; Gheysels, 2007; Braem, Gheysels, 2007; Braem ir kt., 2007; Fraine, Braem, 2007) ir AOWS (Singh ir kt., 2005) koncepcijos. Jos iš esmės apima svarbiausias ir kitų pasiūlymų idėjas. Aptarsime šias koncepcijas.

AWED-WSML projektas grindžiamas tarpininkavimo idėja. Kompozicijas siūloma specifikuoti AWED (angl. *Aspects with Explicit Distribution*) kalba (Navarro ir kt., 2006a). Kadangi AWED yra aspektinė kalba, skirta išskirstytosioms programoms programuoti, ja rašomos tokios kliento programos, kurios dinamiškai kuria aspektus ir juos išskirsto po specialaus daugiasluoksnio architektūrinio karkaso WSML (angl. *Web Services Management Layer*) sluoksnius. Svarbiausi AWED kalbos ypatumai yra šie: 1) nuotoliniai pjūviai (angl. *remote pointcuts*), leidžiantys susieti skirtinguose kompiuteriuose vykstančius vykdymo meto įvykius; 2) kompiuterių grupės sąvoka, leidžianti daryti pjūviuose nuorodas į tokias grupes ir manipuliuoti jomis pagalbinių turinių kode (angl. *advice*), įterpiamame į pagrindinius turinius; 3) galimybė tiek sinchroniškai, tiek asinchroniškai vykdyti po skirtingus kompiuterius išbarstytą pagalbinių turinių kodą. WSML karkasas ir yra tarpininkas. Be kita ko, jis yra fasadas, sukuriantis kliento programai abstrakčią sąsają ir paslepiantis nuo jos konkrečių PSP sąsajų detales. WSML įterpiamas tarp kliento programos ir PSP valdymo infrastruktūros. Kliento programoje lieka tik verslo proceso logiką aprašantis kodas (pagrindinis turinys), o kodas, skirtas valdyti ir tvarkyti PSP kliento pusėje, ir kiti papildomi turiniai yra paverčiami aspektais, išskirstomais po atitinkamus WSML karkaso sluoksnius (Verheecke, 2006; Navarro ir kt., 2006; Navarro ir kt., 2006a). Karkasas perima visus kliento programos prašymus suteikti jos reikalaujamas paslaugas, žiūri, kokie aspektai reikalingi teikiant tas paslaugas, ir susieja aspektus su papildomų turinių kodu, išbarstytu po PSP kompozicijai priklausančias paslaugas realizuojančias programas. Visos tos programos privalo būti aspektinės. Jei vykdant konkrečioje vykdymo aplinkoje išskleistą kompozicijos egzempliorių susidaro situacija, kurioje prašymas gauti paslaugą dėl kokių nors priežasčių yra nesuderinamas su jau suplanuotos panaudoti konkrečios PSP kontrakto sąlygomis, karkasas gali peradresuoti tą prašymą kitam, ekvivalentų funkcionalumą turinčią paslaugą teikiančiam paslaugų teikėjui. Jei kurio nors prašymo netenkina nė viena konkreti PSP, karkasas gali bandyti sukomponuoti prašomą paslaugą iš turimų PSP. Karkaso sukurtos kompozicijos yra centralizuotos. Jas koordinuoja tas pats koordinatorius, kaip ir verslo procesą įgyvendinančią kompoziciją. Taigi, karkasas gali parinkti reikiamas PSP vykdymo metu ir parinkdamas atsižvelgti ne tik į jų funkcionalumą, bet ir į kokybės charakteristikas bei nurodytas elgsenos taisykles. Tai reiškia, kad AWED-WSML projekte paslaugų komponavimas bent jau iš dalies yra dinaminis, jis vyksta vykdymo metu.

AO4BPEL (Charfi, Mezini, 2004) – tai aspektinis BPEL kalbos variantas. Rašant kompozicijų specifikacijas šia kalba, išlieka galimybė į kompozicijas įterpti ar iš jų pašalinti aspektus vykdymo metu. Kadangi BPEL kalba grindžiama prielaida, jog verslo procesas yra sudarytas iš verslo veiklų, tai aspektus natūralu įterpti prieš tų veiklų vykdymą arba po jo. Kitaip tariant, šioje kalboje galima griežtai apibrėžti turinių sankirtos taškus (angl. *join point*), nes bet kuri verslo veikla yra suvokiama kaip pagrindinis turinys, su kuriuo supinami aspektais apibrėžti PSP tvarkymo ir valdymo turiniai. Konkrečiai veiklai turinių sankirtos taškai parenkami pagal parinkimo momentu turimas verslo proceso ir tos veiklos atributų reikšmes. Ši idėja ir yra įgyvendinta AO4BPEL projekte (d'Hondt, 2004; Cibrán, 2007; Charfi, 2007). Panašiomis idėjomis grindžiamas ir *Padus* projektas (Braem ir kt., 2006; Braem, Joncheere, 2007; Gheysels, 2007; Braem, Gheysels, 2007; Braem ir kt., 2007; Fraine, Braem, 2007). Abiejuose projektuose aspektai įterpiami į kompozicijas, specifikuojamas aspektine BPEL kalba. Tačiau juose pasiūlytos idėjos tinka ne tik BPEL kalbai, bet ir bet kuriai kitai verslo procesų aprašymo kalbai, grindžiamai prielaida, jog verslo procesai yra sudaryti iš verslo veiklų.

Projekte AOWS (angl. *Aspect-Oriented Web Service*) (Singh ir kt., 2005) pasiūlyta dar viena koncepcija, kaip panaudoti aspektus centralizuotoms PSP kompozicijoms skaidyti į modulius, ir sukurta tą koncepciją įgyvendinanti architektūra. Ji pasiūlyta kaip galima alternatyva tradici-

nei <SOAP, UDDI, WSDL> architektūrai, pakeičiant joje UDDI registrą AO-UDDI registru ir paslaugų specifikavimo kalbą WSDL – aspektine paslaugų specifikavimo kalba AO-WSDL. Aprašai AO-WSDL kalba yra semantiškai turtingesni negu aprašai WSDL kalba. Juose galima tiksliau specifikuoti paslaugos funkcines ir nefunkcines savybes, kitaip tariant, PSP beveik priartėja prie PSSP. Tačiau yra daroma prielaida, kad PSP susijusių operacijų grupės yra realizuojamos komponentais ir charakterizuojami ne PSP, o komponentų susikertantys turiniai. Taigi, AOWS projekte <SOAP, UDDI, WSDL> architektūra yra keičiama <SOAP, AO-UDDI, AO-WSDL> architektūra, kuri už pirmąją yra kai kuriais požiūriais tobulesnė, o kai kuriais požiūriais mažiau tobula. Kaip jau minėta, šioje architektūroje galima specifikuoti susikertančius turinius. Taigi PSP aprašai tampa išsamesni, dėl to galima tobulinti dinaminę PSP paiešką ir taip komponuoti PSP, kad jos būtų integruojamos nepaliekant jokių „siūlių“. Projekto terminais, tokios kompozicijos vadinamos *aspektinėmis*. Komponuoti galima tik specifikuotas AO-WSDL kalba ir registruotas AO-UDDI registre paslaugas. Dėl to apribojamos praktinės <SOAP, AO-UDDI, AO-WSDL> architektūros naudojamos galimybės, nes nebegalima pasinaudoti trečiųjų šalių teikiamomis paslaugomis.

Esminis <SOAP, AO-UDDI, AO-WSDL> architektūros komponentas yra vadinamasis AO jungiklis (angl. *AO connector*), dinamiškai sujungiantis kliento programą su jos prašomų paslaugų teikėjais. Visas kitas komunikavimas taip pat vyksta tik per jį. Tai leidžia tradicines aspektinės paradigmos technologijas keisti komponentinėmis. Aspektai nėra paverčiami savarankiškais moduliais, jie lieka paslaugas įgyvendinančiuose komponentuose ir yra suvokiami kaip jų operacijos. Specifikuojant kompoziciją nurodoma, kokie aspektai (apsauga, transakcijų valdymas ar pan.) su ja siejami. Kitaip tariant, nurodomos nefunkcinės kompozicijos savybės, kurias AO jungiklis automatiškai prijungia prie prašymų kompoziciją sudarančioms paslaugoms gauti. Gavęs šitaip papildytus prašymus ir turėdamas PSP specifikacijas AO-WSDL kalba, AO-UDDI registras gali parinkti paslaugų teikėjus, teikiančius paslaugas, turinčias ne tik reikiamą funkcionalumą, bet ir pageidaujamas nefunkcines bei ekstrafunkcines savybes. Be to, turint išsamius AO-WSDL aprašus, galima sukomponuoti ne tik reikalaujamą funkcionalumą, bet iš dalies ir reikiamas nefunkcines savybes.

Decentralizuotų PSP kompozicijų skaidymas į modulius

Pasiūlymų, kaip skaidyti į modulius decentralizuotas PSP kompozicijas, pateikta gana nedaug. Kita vertus, decentralizuotos kompozicijos yra labai svarbios, nes centralizuotoms kompozicijoms dažnai nepavyksta pasiekti reikiamo našumo. Taip yra todėl, kad centralizuotų kompozicijų vykdymo mašinos, net ir išskirstytų kompozicijų atveju, yra nepajėgios iki galo lygiagretinti pranešimų mainų. Decentralizuotose kompozicijose tai padaryti įmanoma, nes jose pranešimų mainai vyksta naudojant P2P sąveikas. Tokia kompozicija yra išskirstoma po skirtingų organizacijų kontroliuojamus kompiuterių tinklo domenus, nes nėra kokios nors vienos organizacijos, kuriai priklausytų ja realizuojamas verslo procesas ir kuri jį kontroliuotų. Decentralizuotą kompoziciją galima išskirstyti po organizacijų domenus dviem būdais. Išskirsčius dekompoziciją pirmu būdu, ją sudarančios paslaugos yra dinamiškai išskleidžiamos visų organizacijų domenuose dar prieš aktyvuojant tą kompoziciją. Išskirsčius kompoziciją antru būdu, paslaugos domene išskleidžiamos jau vykdant kompoziciją, kai tų paslaugų yra paprašoma. Informaciją apie tai, kokias paslaugas reikia išskleisti, įtraukiama į prašymus tas paslaugas suteikti. Šie būdai papildo vienas kitą ir gali būti naudojami kartu. Kadangi verslo proceso dalis, kurią vykdo konkreti organizacija, kartais gali būti valdoma kitos organizacijos, tai kompiuterių tinklų administratoriai ne visada nori diegti dinamiškai aktyvuojamus P2P sąveikos mechanizmus. Taip yra todėl, kad, įdiegus tokius mechanizmus, domene gali pradėti veikti iš išorės aktyvuojamos programos, administratoriui nedavus tam leidimo ir netgi apskritai apie

tai nežinant. Abu pirmiau minėti kompozicijų išskirstymo būdai verčia administratorius tai daryti. Tuo ir yra grindžiami pasiūlyti decentralizuotų kompozicijų skaidymo į modulius būdai. Viena iš išsamiausių decentralizuotų kompozicijų skaidymo į modulius koncepcijų pateikta (Cottenier, Elrad, 2005; Akkawi ir kt., 2006) ir kituose Ilinojaus technologijos instituto (JAV) lygiagrečiojo programavimo tyrimų grupės darbuose. Ši koncepcija apibendrina ir daugelį kitų reikšmingesnių pasiūlymų. Joje pasiūlyta aspektams jautrių PSP (angl. *Aspect-Sensitive Services* – CASS) samprata. Ją realizuoja universalus (t. y. tinkantis bet kuriai platformai) išskirstytas aspektinis kompiuterinės platformos plėtinys ECF (angl. *Executable Choreography Framework*), leidžiantis inkapsuliuoti koordinavimą, veiklos gyvavimo ciklą, transakcijų valdymą ir kitus panašaus pobūdžio papildomus turinius. Šiuo plėtiniu papildomos kompozicijos apimamų organizacijų domenų kompiuterinės platformos. Jei prašymas suteikti paslaugą arba atsakymas į jį tenkina verslo veiklos viduje aprašyto įvykio šabloną, platforma pradeda valdyti su tuo prašymu arba su atsakymu į jį susijusią valdymo srauto giją ir prieš šį įvykį arba po jo įterpia nurodytą verslo proceso elementą arba, kitaip tariant, modulį. Taigi, platforma turi mechanizmus, panašius į aspektų aktyvavimo mechanizmus, kuriuos naudojant vykdymo metu galima supinti vienas nuo kito atskirtus susikertančius turinius. Kompozicijos yra skaidomos į modulius specifikacijų lygmeniu. Tam vartojama nuo konkrečios platformos nepriklausoma ECL (angl. *Executable Choreography Language*) specifikavimo kalba. Šia kalba parašytas specifikacijas galima vykdyti, t. y. ją galima interpretuoti panašiai, kaip centralizuotų kompozicijų atveju tai daro vadinamoji vykdymo mašina. Kalbos ideologija inspiruota aspektinės ideologijos. Be to, ji priklauso vadinamajai XML kalbų kategorijai. ECL kalba yra aprašomi moduliai ir taisyklės, nusakančios, prieš kokius įvykius arba po jų reikia tuos modulius aktyvuoti. Decentralizuotai kompozicijai specifikuoti vien ECL specifikacijos nepakanka. Ši specifikacija tiktai papildo pagrindinę CDL (angl. *Choreography Description Language*) kalba parašytą paties verslo proceso specifikaciją. Galima sakyti, kad šitaip verslo proceso darbų srautas yra papildomas aspektais arba, kitaip tariant, ECL specifikacija patikslina tą srautą. ECL specifikaciją interpretuoja atitinkamos platformos ECF plėtinys. Šis plėtinys įgyvendina tris pagrindines funkcijas: gaunamų pranešimų perėmimą, verslo veiklos konteksto sklaidą ir dinaminį taisyklių aktyvavimą. ECF perima į domeną gaunamus ir iš jo siunčiamus pranešimus, tikrina, ar įvykis tenkina kurį nors iš domene aprašytų įvykių šablonų, ir jeigu tenkina – įterpia į perimtą pranešimą su tuo šablonu siejamą taisyklę, kuri vėliau dinamiškai išskleidžiama ir aktyvuojama. Kadangi taisyklėms reikalinga informacija apie tai, kliento ar kurios nors iš organizacijų kontekste yra jos vykdoma verslo veikla, tai iš domeno į domeną ši informacija yra perduodama kartu su pranešimais. Tai ir vadinama verslo veikos konteksto sklaida. Taigi, <SOAP, CDL, ECL, ECF> architektūra sudaro visas prielaidas decentralizuotoms kompozicijoms skaidyti į modulius. Joje naudojamos idėjos daug kuo primena projekto AWED-WSML idėjas, pritaikytas decentralizuotų kompozicijų atvejui. Tačiau, skirtingai negu AWED-WSML, <SOAP, CDL, ECL, ECF> architektūra tiesioginio aspektų išskirstymo nepalaiko.

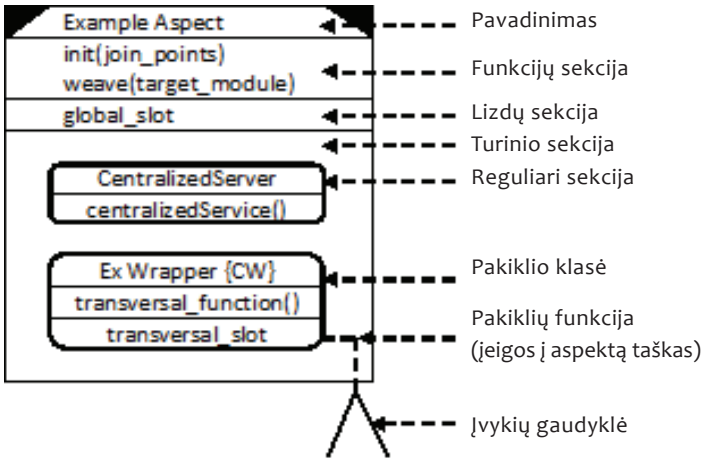
Reikšmingi pasiūlymai decentralizuotų PSP kompozicijų skaidymo į modulius klausimu pateikti Ponnalagu ir bendraautorių darbe (2007). Skaidymo į modulius problemai spręsti siūloma naudoti atvirojo kodo aspektinio programavimo platformą PROSE (Nicoră, Alonso, 2005), praplečiant ją aspektų išskirstymo galimybėmis. PROSE platformoje galima įgyvendinti specialius metodus, vadinamuosius kablius (angl. *hooks*), gebančius tuose taškuose, kuriuose reikia aktyvuoti aspektus, perimti iš Java virtualiosios mašinos kreipinius į klasėse aprašytus metodus. Naudodami šiuos PROSE platformos ypatumus, Ponnalagu ir bendraautoriai pasiūlė pirmąją PSP kompozicijų skaidymo į modulius technologiją, naudojant kurią kompoziciją sudarančias paslaugas realizuojančių programų kode nereikia daryti jokių pakeitimų. Technologija grindžiama vadinamuoju sąryšių modeliu (angl.

relationship model). Šis modelis susieja PSP kompozicijos nefunkcinius reikalavimus su ją sudarančių PSP nefunkcinėmis savybėmis ir kompoziciją sudarančių PSP aspektus su kompozicijos aspektais. Šio modelio pagrindu ir yra atliekamas programų kodų nekeičiantis PSP kompozicijų skaidymas. Taigi, sistema veikia šitaip (Nicoră, Alonso, 2005): 1) gavus reikalavimą sukurti PSP kompoziciją, turinčią reikalaujamas nefunkcines savybes, nustatoma, kurie aspektai (jie saugomi centralizuotai) yra reikalingi šioms savybėms užtikrinti; 2) nustatoma, su kurių kompoziciją sudarančių PSP pagrindiniais turiniais kertasi šiais aspektais įgyvendinami turiniai; 3) kompoziciją sudarančioms PSP, kurių pagrindiniai turiniai kertasi su reikalaujamais nefunkciniais turiniais, pasiunčiami prašymai supinti pagrindinius jų turinius su atitinkamais nefunkciniais turiniais; 4) jei dėl kokių nors priežasčių PSP negali to padaryti, susidaro ypatingoji situacija, kurią sistema turi apdoroti specialiu būdu. Pagrindinis šio pasiūlymo trūkumas yra tas, kad iš tiesų ne visas kompozicijos nefunkcines savybes lemia ją sudarančių PSP nefunkcinės savybės.

Išskirstytų PSP kompozicijų infrastruktūros naudojimas

Pasiūlyta tik keletas aspektinių kalbų ar sistemų, palaikančių papildomų turinių tiesioginį išskirstymą. Vienas iš pirmųjų tokio tipo siūlymų buvo D – kalbinis išskirstytų sistemų programavimo karkasas (Lopes, 1997; Lopes, Kiczales, 1997). Programuojant išskirstytas sistemas, labai svarbu lygiagrečiai vykdomų gijų sinchronizavimas ir dalykinio lygmens duomenų mainai tarp viena nuo kitos nutolusių programų. Kitaip tariant, tai yra svarbiausi išskirstytų programų sistemų papildomi turiniai. Neaspektinės išskirstyto programavimo kalbos nepalaiko savarankiško papildomų turinių programavimo. Todėl sinchronizavimo ir nuotolinių duomenų mainų turinius tenka supinti su funkcinių komponentų kodu *ad hoc* būdu. D karkasas leidžia atskirti tuos turinius. Jis numato trijų rūšių modulius (jie įvardyti D karkaso terminais): 1) klases, naudojamas funkciniams komponentams programuoti; 2) koordinatorius, saugančius gijų sinchronizavimo kodą; 3) portalus, saugančius dalykinio lygmens nuotolinius duomenų mainus įgyvendinantį kodą. Karkase numatytos dvi moduliams programuoti skirtos aspektinės kalbos: COOL (programuoti koordinatorius) ir RIDL (programuoti portalus). Specialiai tam skirtas įrankis Aspect Weaver™, derindamas šiomis kalbomis ir Java ar kita klasėms ir baziniam funkcionalumui aprašyti skirta kalba parašytas programas, sukuria vieną vykdomąją programą. Karkasui veikti reikia labai nedidelių laiko sąnaudų, todėl ir jo poveikis sistemos efektyvumui yra nedidelis.

Pirmiau aptartos idėjos buvo plėtojamos JAC projekte (Pawlak ir kt., 2000; 2001; 2001a; 2002; 2004; 2002; Pawlak), bei *DJcutter* kalboje (Nishizawa ir kt., 2004). Šiame projekte sukurta refleksyvi tarpinė aspektinė programinė įranga A-TOS (Pawlak ir kt., 2000), grindžiama vadinamojo *aspektinio komponento* idėja (žr. pav.).



P a v. ***Aspektinis komponentas (Pawlak ir kt., 2000)***

Aspektinis komponentas – tai specialus paprasto komponento plėtinys. Aspektiniai komponentai skirti išskirstytų sistemų aspektams, pavyzdžiui, saugai, patikimumui ar transakcijoms programuoti. Aspektinių komponentų fragmentai gali būti išskirstyti po kompiuterių tinklą, kuriame veikia išskirstytoji programų sistema, ir yra supinami su tos sistemos funkciniais turiniais. Pagrindinė JAC projekto idėja yra dinaminių pakiklių (angl. *wrapper*) naudojimas (Pawlak ir kt., 2001). Panašiai, kaip paprasti komponentai aprašomi klasėmis, aspektiniai komponentai aprašomi *aspektinių komponentų klasėmis.* Tokios klasės yra globalią semantiką turinčios paprastųjų klasių specializacijos. Aspektinio komponento klasėje galima apibrėžti lizdais vadinamus laukus (angl. *slots*) ir funkcijas, kurie kartu aprašo tos klasės globaliąsias savybes bei elgsenas ir yra įterpiami į dalykinę programą. Aspektiniams komponentams supinti su bazine programa reikia, kad laukų reikšmės apibrėžtų ir turinių sankirtos taškus. Aspektinio komponento klasė gali būti konteineriu, nes, kaip parodyta paveiksle, į ją galima įdėti kitas klases bei objektus. Taigi, globaliosios klasės ir pakiklių klasės, apibrėžiančios aspekto semantiką, yra aprašomos aspektinės klasės viduje. Aspektinė klasė privalo turėti funkcijas **init(join_point)** ir **weave(target_module)**. Pirmoji aktyvuoja kiekvieną naujai sukurtą aspektinio komponento egzempliorių, o antroji supina bazinę programą su aspektiniu komponentu. Supynimas padaromas apgaubiant atitinkamus bazinės programos objektus aspektinėje klasėje apibrėžtais pakikliais. Pakikliai veikia vykdymo metu. Jie keičia bazinės programos semantiką, pridėdami reikiamą kodą prieš apgaubiamą objektą ir po jo. Tai padaroma pasinaudojant A-TOS. Pakavimo technologija yra grindžiama įvykiais, todėl pakikliai gali būti pridedami ir šalinami dinamiškai vykdymo metu.

Aspektinio komponento klasės yra aprašomos *DJcutter* kalba (Nishizawa ir kt., 2004), kuri taip praplečia programavimo kalbą *AspectJ* (Kiczales ir kt., 2001), kad ja būtų galima programuoti aspektus išskirstytose programų sistemose. Tam reikia padaryti, kad aspektų fragmentus būtų galima išbarstyti po daugelį išskirstytos programų sistemos kompiuterių tinklo kompiuterių ir kad tie aspektai galėtų išreikštiniu būdu komunikuoti tarpusavyje. Svarbiausioji *DJcutter* kalbos naujovė yra *nutolę pjūviai* (angl. *remote pointcuts*). Nutolę pjūviai – tai funkcijos, programos vykdymo metu identifikuojančios turinių sankirtos taškus nutolusiuose kompiuteriuose. Šis konstruktas yra objektinių kalbų nutolusio metodų kvietimo (angl. *remote method invocation*) analogas. *Djcutter* kalboje aspektai supinami su pagrindine programa, kai ši įkeliama į kompiuterio atmintį.

D karkasas, projektas JAC ir *DJcutter* kalba yra grindžiami panašiomis idėjomis. Jos nesunkiai pritaikomos PSP kompozicijoms skaidyti į modulius. Tačiau visais trimis atvejais gauname mažesnes galimybes negu AWED-WSML projekte. Nei D, nei JAC, nei Djcutter neleidžia asinchroniškai vykdyti įterpiamų kodo fragmentų.

Yra dar vienas dėmesio vertas pasiūlymas. Tai *ReflexD* (Tanter, Toledo, 2006) – branduolys, palaikantis visą diapazoną aspektinių kalbų išskirstytoms sistemoms programuoti. Iš tiesų, *ReflexD* gali būti suvokiama kaip lanksti infrastruktūra išskirstytais aspektais manipuliuoti. Naudojant šią infrastruktūrą, galima kurti įvairias aspektines kalbas ir įvairius aspektinius karkasus. Branduolys įgyvendintas Java kalba. *ReflexD* pateikia metaobjekto protokolą kaip interfeisą, kurį naudojant realizuojami pjūviai, įterpiamo kodo fragmentai ir susiejimas (angl. *binding*). Branduolys pats savaime (t. y. nesukūrus kokios nors kalbos) negali būti panaudotas PSP kompozicijoms skaidyti į modulius. Be to, nėra iki galo aišku, ar gerai *ReflexD* integruojamas su kitomis jau esamomis PSP infrastruktūromis.

Išskirstytų sistemų aspektizavimas

Daugelis plačiai naudojamų išskirstytoms programų sistemoms kurti skirtų pramoninių technologijų, pavyzdžiui, *JBoss* (Jamae, Johnson, 2009) ar *Spring* (Laddad, 2010), turi aspektinius analogus (JBoss AOP, Spring AOP). Yra ir panašių akademinio pobūdžio technologijų, pavyzdžiui, DAOP (Pinto ir kt., 2003). Iš esmės visos jos leidžia skaidyti į modulius PSP

kompozicijas. Kaip tai daroma, parodyta, pavyzdžiui, Sirbi, Kulkarni darbe (2010). Tokios technologijos numato karkasus, paprastai realizuotus kaip tarpinės programinės įrangos bibliotekos, be kita ko, teikiančius ir aspektinio programavimo paslaugas. Tai palengvina programuotojo darbą, o kartais, pavyzdžiui, *JBoss* AOP atveju, jis supaprastėja dar ir todėl, kad aspektus galima programuoti kaip Java kalbos klases ir programuotojams nebereikia papildomai mokytis kurios nors aspektų programavimo kalbos, tarkime, *AspectJ*. Naudojant perėmimo (angl. *interception*) ir konfigūravimo mechanizmus, papildomus turinius galima supinti su pagrindiniais turiniais dinamiškai, t. y. vykdymo metu. Tačiau šios technologijos vis vien yra blogai pritaikytos paslaugų kompozicijoms skaidyti į modulius. Pagrindinis trūkumas yra tas, kad pagalbiniai turiniai, beje, kaip ir pagrindiniai, turi būti iš anksto programuojami rankiniu būdu ir tik paskui visi kartu išskirstomi po kompiuterių tinklą. Dinamiškai kuriamoms paslaugų kompozicijoms toks skaidymo į modulius būdas tiesiog yra nepritaikomas.

Išvados

Mokslinėje spaudoje paskelbtų PSP kompozicijų skaidymo į modulius metodų analizė parodė, jog dominuoja aspektinėmis technologijomis grindžiami šios problemos sprendimo būdai. Nepaisant tokios paradigminės vienovės, patys metodai yra gana skirtingi. Kita vertus, iškeltų idėjų nėra daug. Galima išskirti tokias pagrindines idėjas: 1) kompozicijų specifikavimo kalbų aspektizavimas (AWED, AO4BEL, AO-WSDL, ECL, COOL, RIDL, *DJcutter* kalbos) ir jomis specifikuotų papildomų turinių supynimas su pagrindiniu turiniu; 2) specialaus tarpininko, organizuojančio dinaminį papildomų turinių supynimą su pagrindiniu, įterpimas tarp kompoziciją realizuojančios programos ir tradicinės PSP valdymo infrastruktūros (WSML, PROSE, D, A-TOS karkasai); 3) kompiuterinės platformos aspektinis praplėtimas, inkapsuliuojantis papildomus turinius ir dinamiškai terpiantis juos į pagrindinį turinį (ECF); 4) PSP valdymo infrastruktūros aspektizavimas (<SOAP, AO-UDDI, AO-WSDL> ir *ReflexD* architektūros); 5) išskirstytų sistemų aspektizavimas (*JBoss* AOP, *Spring* AOP, DAOP technologijos). Straipsnyje visos šios idėjos išanalizuotos kritiškai. Kol kas nė viena nėra galutinai atmesta, bet taip pat nė viena idėja ar koks nors jų derinys nepateikė galutinio PSP kompozicijų skaidymo į modulius uždavinio sprendimo ir tyrinėjama toliau. Manytina, kad mažiausiai perspektyvi yra išskirstytų sistemų aspektizavimo idėja, bet ji kol kas taip pat nėra galutinai atmesta.

LITERATŪRA

ANDREWS, T.; CURBERA, F.; DHOLAKIA, H.; GOLAND, Y.; KLEIN, J.; LEYMANN, F.; LIU, K.; ROLLER, D.; SMITH, D.; THATTE, S.; TRICKOVIC, I.; WEERAWARANA, S. (2003). *Business Process Execution Language for Web Services*. BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems. Prieiga per internetą: <http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp>.

AKKAWI, F.; FLETCHER, D. P.; COTTENIER, T.; DUNCAVAGE, D. P.; ALENA, R. L.; ELRAD, T. (2006). An Executable Choreography Framework for Dynamic Service-Oriented Architectures. In *Proceedings of the 2006 , IEEE Aerospace Conference,* Big Sky, Montana, USA, March 4–11, 2006. IEEE.

BIRON, P. V.; MALHOTRA, A. (eds.) (2004). *XML Schema.* Part 2: *Datatypes.* Second Edition. W3C Recommendation 28 October, 2004. Prieiga per internetą: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html>.

BRAEM, M.; VERLAENEN, K.; JONCHEERE, N.; VANDERPERREN, W.; Van der STRAETEN, R.; TRUYEN, E.; JOOSEN, W.; JONCKERS, V. (2006). Isolating process-level concerns using Padus. In *Proceedings of the 4th International Conference on Business Process Management (BPM ´2006),* Vienna, Austria, Sept., 2006. LNCS 4102. Springer-Verlag, p. 113–128.

BRAEM, M.; JONCHEERE, N. (2007). Requirements for Applying Aspect-Oriented Techniques in

Web Service Composition Languages. In M. Lumpe, W. Vanderperren, eds. *Software Composition, 6th International Symposium, SC 2007*, Braga, Portugal, March 24–25, *Revised Selected Papers,* LNCS 4829, Springer, p. 152–159.

BRAEM, M.; GHEYSELS, D. (2007). History-Based Aspect Weaving for WS-BPEL Using Padus. In *Fifth IEEE European Conference on Web Services (ECOWS 2007),* 26–28 November, 2007, Halle (Saale), Germany. IEEE Computer Society, p. 159–167.

BRAEM, M.; JONCHEERE, N.; VANDERPERREN, W.; VAN DER STRAETEN, R.; JONCKERS, V. (2007). Concern-Specific Languages in a Visual Web Service Creation Environment. *Electronic Notes in Theoretical Computer Sciences*, vol. 163(2), p. 3–17.

CHARFI, A. (2007). *Aspect-Oriented Workflow Languages: AO4BPEL and Applications*. Dissertation von Diplom-Informatiker, Technischen Universität, Darmstadt. Prieiga per internetą: <http://www.st.informatik.tu-darmstadt.de/pages/projects/AO4BPEL/diss.pdf>.

CHARFI, A.; MEZINI, M. (2004). Aspect Oriented Web Service Composition with AO4BPEL. In L.-J. Zhang; M. Jeckle, (eds.). *Proceedings of the European Conference on Web Services (ECOWS '2004)*. LNCS 3250. Springer, p. 68–182.

CHARFI, A.; MEZINI, M. (2004a). Hybrid Web Service Composition: Business Processes Meet Business Rules. In *Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04)*, p. 30–38. Prieiga per internetą: <http://www.cs.ubbcluj.ro/~amarcus/cbpf04/papers/Charfi04.pdf>.

CHINNICI, R.; MOREAU, J.-J.; RYMAN, A.; WEERAWARANA, S., (eds.). (2007). *Web Services Description Language (WSDL)*, Version 2.0, Part 1: *Core Language*. W3C Recommendation 26 June, 2007. Prieiga per internetą: <http://www.w3.org/TR/2007/REC-wsdl20-20070626>.

CHINNICI, R.; MOREAU, J.-J.; RYMAN, A; WEERAWARANA, S., eds. (2007a). *Web Services Description Language (WSDL)*, Version 2.0, Part 2: *Adjuncts*. W3C Recommendation 26 June, 2007. Prieiga per internetą: <http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/wsdl20-adjuncts.pdf>.

CIBRÁN, M. A. (2007). *Connecting High-Level Business Rules with Object-Oriented Applications: An approach using Aspect-Oriented Programming and Model-Drive Engineering*:. PhD Thesis. Vrije Universiteit, Brussel.

CIBRÁN, M. A.; VERHEECKE, B. (2005). Dynamic Business Rules for Web Service Composition. In *Proc. of the Dynamic Aspects Workshop (DAW) in conjunction with AOSD*, p. 13–18.

COTTENIER, T.; ELRAD, T. (2005). *Executable Choreography Processes with Aspect-Sensitive Services*: Technical Report, Computer Science Department, Illinois Institute of Technology. Prieiga per internetą: <http://mypages.iit.edu/~concur/ecf/ecfass.pdf>.

D'HONDT, M. (2004). *Hybrid Aspects for integrating Rule-based Knowledge and Object-Oriented Functionality*: PhD Thesis. Vrije Universiteit, Brussel, May. Prieiga per internetą: <http://ssel.vub.ac.be/research/PhD/thesisMaja.pdf>.

FALLSIDE, D. C.; WALMSLEY, P., eds. (2004). *XML Schema.* Part 0: *Primer.* Second Edition. W3C Recommendation 28 October. Prieiga per internetą: <http://www.w3.org/TR/xmlschema-0/>.

FRAINE, B.; BRAEM, M. (2007). Requirements for Reusable Aspect Deployment. In: M. Lumpe; W. Vanderperren, (eds.) *Software Composition, 6th International Symposium, SC 2007,* Braga, Portugal, March 24–25, 2007, Revised Selected Papers. LN CS 4829. Springer, p. 176–183.

GHEYSELS, D. (2007). *Implementation of Stateful Aspects in Padus*: PhD thesis. Vrije Universiteit, Brussel, 2006–2007. Prieiga per internetą: <http://soft.vub.ac.be/~njonchee/theses/thesis_dimitri.pdf>.

JAMAE, J.; JOHNSON, P. (2009). *JBoss in Action*. Manning Publications.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. G. (2001). An overview of Aspectj. Iš: *European Conference on Object-Oriented Programming 2001 (ECOOP '2001),* LNCS 2072, Springer, p. 327–353.

LADDAD, R. (2010). *AspectJ in action. Enterprise AOP with Spring applications.* 2nd edition. Manning Publications.

LOPES, C. V. (1997). *D: A Language Framework for Distributed Programming*: PhD thesis, College of Computer Science, Northeastern University.

LOPES, C. V.; KICZALES, G. (1997). *D: A Language Framework for Distributed Programming*. Technical report SPL97-010, P9710047, Xerox Palo

Alto Research Center. Prieiga per internetą: <http://www2.parc.com/csl/groups/sda/publications/papers/PARC-AOP-D97/for-web.pdf>.

LOPES, C. V. (2005). Aspect-Oriented Programming: A Historical Perspective (What's in a Name?). In *Aspect-Oriented Software Development*. Addison-Wesley, p. 97–122.

NAVARRO, L. D. B.; SÜDHOLT, M.; VANDERPERREN, W.; VERHEECKE, B. (2006). Modularization of distributed web services using Aspects with Explicit Distribution (AWED). In R. E. Filman (ed.). *Proceedings of the 5th International Conference on Aspect-Oriented Software Development,* AOSD 2006, Bonn, Germany, March 20–24, 2006. ACM, p. 51–62.

NAVARRO, L. D. B.; SÜDHOLT, M.; VANDERPERREN, W.; DE FRAINE, B.; SUVÉE, D. (2006a). Explicitly distributed AOP using AWED. In *Proceedings of the 5th international conference on Aspect-oriented software development (AOSD '06),* New York, NY, USA, March, 2006. ACM Press, p. 51–62.

NICORĂ, A; ALONSO, G. (2005). Dynamic AOP with PROSE. In J. Castro, E. Teniente (eds.). *Advanced Information Systems Engineering*, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13–17, 2005, Proceedings of the CAiSE'05 Workshops, vol. 2. FEUP Edições, Porto, p. 125–138.

NISHIZAWA, M.; CHIBA, S.; TATSUBORI, M. (2004). Remote pointcut: a language construct for distributed AOP. In *Proc. of the 3rd Int. Conf. on Aspect-Oriented Software Development (AOSD '04)*, New York, NY, USA, 2004. ACM Press, p. 7–15.

PAWLAK, R.; DUCHIEN, L.; FLORIN, G.; MARTELLI, L.; SEINTURIER, L. (2000). Distributed Separation of Concerns with Aspect Components. In *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS '00)*, IEEE Computer Society, p. 276–287.

PAWLAK, R.; DUCHIEN, L.; FLORIN, G.; SEINTURIER, L. (2001). Dynamic Wrappers: Handling the Composition Issue with JAC. In *TOOLS-USA 2001,* August, 2001, Santa-Barbara CA, USA, IEEE Computer Society, p. 56–65.

PAWLAK, R.; SEINTURIER, L.; DUCHIEN, L.; FLORIN, G. (2001a). JAC: A flexible solution for aspect-oriented programming in Java. In A. Yonezawa, S. Matsuoka (eds.). *Reflection, LNCS 2192*. Springer, p. 1–24.

PAWLAK, R. (2002). *CEDRIC Research Report: A Notation for Aspect-Oriented Distributed Software Design*. Laboratoire CEDRIC-CNAM, 55 rue Turbigo, 75003 Paris, France. Prieiga per internetą: <http://jac.ow2.org/docs/papers/uml/index.html>.

PAWLAK, R.; DUCHIEN, L.; FLORIN, G.; LEGOND-AUBRY, F.; SEINTURIER, L.; MARTELLI, L. (2002). *JAC: An Aspect-Based Distributed Dynamic Framework*. Technical Overview. ObjectWeb Consortium, December 5, 2002. Prieiga per internetą: <http://jac.ow2.org/docs/JAC.pdf>.

PAWLAK, R.; SEINTURIER, L.; DUCHIEN, L.; FLORIN, G.; LEGOND-AUBRY, F.; MARTELLI, L. (2004). JAC: An aspect-based distributed dynamic framework. *Software – Practice and Experience*, vol. 34(12), p. 1119–1148. Prieiga per internetą: <http://hal.inria.fr/docs/00/03/06/49/PDF/SPERXP.pdf>.

PINTO, M.; FUENTES, L.; TROYA, J. M. (2003). DAOP-ADL: an architecture description language for dynamic component and aspect-based development. In F. Pfenning, Y. Smaragdais (eds.). *Proceedings of the second international conference on Generative programming and component engineering (GPCE '03)*. LNCS 2830. Springer-Verlag, p. 118–137.

PONNALAGU, K.; NARENDRA, C. N.; KRISHNAMURTHY, J.; RAMKUMAR, R. (2007). Aspect-oriented Approach for Non-functional Adaptation of Composite Web Services. In *2007 IEEE International Conference on Services Computing – Workshops (SCW 2007), Workshop on Web Service Composition and Adaptation (WSCA07),* 9–13 July, 2007, Salt Lake City, Utah, USA, IEEE Computer Society, p. 284–291.

SINGH, S.; GRUNDY, J.; HOSKING, J.; SUN, J. (2005). An architecture for developing aspect-oriented web services. In *Proceedings of the third European Conference on Web Services (ECOWS),* Vxjo, Sweden. IEEE Computer Society, p. 72–82.

SIRBI, K.; KULKARNI, P. J. (2010). Modularization of enterprise application security through Spring AOP. *International Journal of Computer Science & Communication*, vol. 1(2), p. 227–231.

TANTER, E.; TOLEDO, R. (2006). A versatile kernel for distributed AOP. In *Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2006),* Bologna, Italy, 2006. LNCS 4025. Springer-Verlag, p. 316–331.

THOMPSON, H. S.; BEECH, D.; MALONEY, M.; MENDELSOHN, N., eds. (2004). *XML Schema*. Part 1: *Structures*. Second Edition, W3C Recommendation 28 October, 2004. Prieiga per internetą: <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.

VEDAMUTHU, A.S.; ORCHARD, D.; HIRSCH, F.; HONDO, M.; YENDLURI, P.; BOUBEZ, T.; YALÇINALP, Ü., eds. (2007). *Web Services Policy 1.5 – Framework*. W3C Recommendation, 04 September, 2007. Prieiga per internetą: <http://www.w3.org/TR/2007/REC-ws-policy-20070904/>, <http://www.w3.org/TR/ws-policy/ws-policy-framework.pdf>.

VERHEECKE, B. (2006). *Dynamic Integration, Composition, Selection and Management of Web Services in Service-Oriented Applications. An Approach using Aspect-Oriented Programming*: PhD Thesis. Vrije Universiteit, Brussel. Prieiga per internetą: <http://ssel.vub.ac.be/Members/BartVerheecke/PhDThesis/PhDVerheeckeBart.pdf>.

MODULARIZATION OF WEB SERVICE COMPOSITION

Jolanta Miliauskaitė, Albertas Čaplinskas

Summary

The paper discusses, analyzes and evaluates the most important scientific propositions on how to solve the problem of web service composition modularization. This problem is very important in the context of eBusiness systems because it is impossible in such systems to process the run-time events in a flexible way, reconfigure business processes dynamically, to take into account in the service discovery process the quality of services, and to implement effectively many other eBusiness system features without knowing an efficient solution of this problem. The paper discusses the current state of affairs and highlights most important open issues. The authors hope that it will be useful for all researchers involved in the web service studies and helpful when planning their further research.