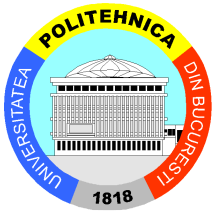


UNIVERSITATEA **POLITEHNICA** DIN BUCUREŞTI
Facultatea Automatică şi Calculatoare
Departamentul de Calculatoare

Teză de doctorat

Compunerea serviciilor web bazată pe calitatea serviciilor

Autor: Ing. Raluca Iordache

Conducător ştiinţific: Prof. Dr. Ing. Florica Moldoveanu

Bucureşti 2014

Cuprins

1. Introducere

1.1 Motivaţie. Obiectivul tezei.

În momentul actual, arhitecturile bazate pe servicii oferă oportunitatea dezvoltării de noi aplicații complexe prin compunerea unor aplicaţii deja existente, oferite ca servicii, pentru a reacţiona cerinţelor pieţei de a crea rapid aplicaţii noi. Arhitecturile bazate pe servicii permit selecţia dinamică şi integrarea automată a serviciilor componente oferind soluţii flexibile şi adaptabile la problemele care pot apărea în timpul execuţiei.

Un serviciu compus este descris de un model abstract de compunere care combină o serie de activităţi. Pentru a putea executa un serviciu compus, fiecărei activităţi trebuie să îi fie asociat un serviciu web concret care îndeplineşte funcţionalitatea cerută.

Existenţa unei oferte mari de servicii web ce implementează funcţionalităţi similare, sau chiar identice, face necesară investigarea unor posibilităţi de comparare a serviciilor web pe baza unor criterii legate de calitatea serviciilor (QoS). Compararea serviciilor trebuie sa ţină cont de cerinţele clientului, iar găsirea serviciului web optim pentru cerinţele date este o sarcină complexă. În practică există în mod curent situații în care nici o soluţie nu indeplineşte toate criteriile clientului, deoarece trebuie urmărite mai multe cerinţe, de obicei conflictuale. Un exemplu ar fi cazul des întâlnit în care se doreşte un serviciu web care să ofere un timp de răspuns mic, indisponibilitate a sistemului redusă şi în acelaşi timp un preţ mic. În această situaţie este posibil să nu existe o soluţie care să optimizeze toate criteriile, aşa că în acest caz soluţia cea mai bună este soluţia care conţine cel mai bun compromis. Clientul trebuie sa aibă posibilitatea de a specifica regulile pentru alegerea unei soluţii de compromis optime.

În cazul serviciilor web compuse, trebuie calculată valoarea agregată a calităţii serviciilor pentru modelul de orchestrare care defineşte compoziţia respectivă.

Serviciile web sunt componente dinamice: în orice moment un serviciu poate dispărea şi noi servicii similare pot apărea. Din acest motiv, caracteristicile nefuncţionale QoS se pot modifica frecvent. Astfel, se impune o abordare automată şi dinamică pentru compunerea serviciilor web.

În prezent există o serie de abordări care adresează problema compunerii serviciilor web ţinând cont de optimizarea parametrilor QoS, dar nu există nici o soluţie standard în această direcţie de cercetare, care să ţină cont atât de cerinţele cât şi de preferinţele clientului.

Obiectivul acestei teze de doctorat este de a furniza o metodă eficientă, dar în acelaşi timp intuitivă şi simplu de utilizat, de compunere a serviciilor web bazată pe cerinţele şi preferinţele clienţilor. Pentru specificarea preferinţelor clienţilor ne-am propus să concepem un limbaj expresiv, care să permită formularea de reguli complexe. Abordarea trebuie să fie

completă, pornind de la căutarea semantică a serviciilor şi continuând cu selecţia serviciilor optime pentru activităţile modelului de compunere. Optimizarea compunerii trebuie să ţină cont de preferinţele clientului, referitoare la parametrii de calitate ai serviciului compus rezultat.

1.2 Contribuţii personale

Teza de doctorat propune o abordare completă pentru compunerea serviciilor web bazată pe calitatea serviciilor [**1**]. Această abordare distinge două etape importante: descoperirea serviciilor web candidat şi alegerea serviciilor optime ţinând cont de cerinţele de calitate ale clientului. Contribuţiile abordării propuse sunt atât teoretice cât şi practice, fiind oferite ca proiecte open-source.

Descoperirea serviciilor pe baza unei căutări sintactice oferă rezultate cu o acurateţe scăzută. Această problemă este rezolvată folosind în etapa de descoperire a serviciilor o căutare semantică, bazată pe ontologii, pentru a identifica pentru fiecare activitate din modelul de compunere abstract o listă de servicii candidat care respectă atât cerinţele funcţionale cât şi cerinţele QoS. În etapa de selecţie a serviciilor, pentru fiecare activitate din modelul de orchestare este ales un serviciu web optim, din lista de servicii candidat oferită de etapa anterioară. Procesul de selecţie adoptă o strategie de optimizare globală pentru a realiza compunerea de servicii care respectă cel mai bine preferinţele QoS agregate specificate de client. O noutate a abordării noastre este integrarea preferinţelor QoS în cadrul procesului de descoperire a serviciilor, pentru a restrânge numărul de servicii candidat pentru fiecare activitate din modelul de orchestrare. În acest scop oferim o extensie a unei ontologii de preferinţe, care oferă posibilitatea specificării preferinţelor şi a regulilor de compromis şi folosim o strategie de optimizare locală în cadrul etapei de descoperire.

O altă contribuţie a tezei este propunerea unei metode de specificare a proprietăţilor QoS, denumită QoSPref [**2**], bazată pe un limbaj simplu şi intuitiv care oferă posibilitatea exprimării unor preferinţe complexe şi a unor reguli de compromis. În cadrul acestei metode, propunem un algoritm de selecţie a serviciilor web simplu, care nu necesită technici complexe de decizie multicriterială. Algoritmul nostru realizează o ordonare totală a alternativelor şi poate accepta şi preferinţe intranzitive.

Pentru a putea folosi şi testa metoda propusă, oferim o implementare a acestei metode, sub forma unui proiect open-source. Aplicaţia noastră oferă şi o interfaţă grafică pentru configurarea serviciilor candidat şi a caracteristicilor acestora, precum şi pentru configurarea cerinţelor şi preferinţelor clienţilor. Proiectul open-source este disponibil la adresa http://sourceforge.net/projects/qospref/.

Pentru a oferi o ontologie de preferinţe care să folosească flexibilitatea metodei noastre de specificare a preferinţelor (QoSPref) şi a regulilor de compromis, am extins ontologia de preferinţe existentă OWL-Q [**3**]. Această ontologie oferă posbilitatea specificării simetrice a cerinţelor clientului şi a caracteristicilor serviciilor oferite.

Folosind o căutare semantică bazată pe cerinţele funcţionale şi nefuncţionale ale clientului, obţinem o listă de servicii candidat pentru fiecare activitate din modelul abstract de orchestrare [**4**]. Pentru a alege pentru fiecare activitate serviciul concret care va îndeplini funcţionalitatea cerută trebuie să calculăm şi să comparăm valorile QoS agregate ale

serviciului compus. Calcularea valorii QoS agregate pentru un serviciu compus este o problemă complexă, iar majoritatea soluţiilor existente se limitează la modele de compunere care pot fi reprezentate ca workflowuri structurate. Soluţia oferită de noi pentru alegerea serviciilor concrete reprezintă o abordare *binding-as-a-service* (BaaS) [**5**] şi foloseşte metoda propusă de Yang et al. [**6**] pentru a elimina această restricţie.

Serviciile concrete care oferă soluţia optimă compusă sunt alese folosind un algoritm genetic [**7**] care evaluează fitness-ul unei soluţii pe baza valorii agregate QoS şi a preferinţelor specificate de client folosind metoda noastră QoSPref.

Abordarea noastră *binding-as-a-service* (BaaS) este oferită sub forma unui serviciu web care permite alegerea serviciilor concrete din cadrul unei mulţimi de servicii candidat, pe baza modelului abstract de compunere şi având in vedere optimizarea globală a serviciului compus. BaaS este implementat sub forma unui proiect open-source, disponibil la adresa: http://sourceforge.net/projects/baas/.

Metoda QoSPref poate fi aplicată nu numai pentru selecţia serviciilor pe baza caracteristicilor QoS, ci şi în cadrul general al oricărei probleme de optimizare multiobiectiv. Pentru a analiza utilitatea unei astfel de abordări, am adaptat metoda QoSPref şi am integrat-o într-un framework pentru studiul tehnicilor de incorporare a preferinţelor în algoritmi evolutivi de rezolvare a problemelor multiobiectiv [**8**].

1.3 Lucrări ştiinţifice în care au fost publicate contribuţiile originale cuprinse în teză

- Raluca Iordache and Florica Moldoveanu, "A conditional lexicographic approach for the elicitation of QoS Preferences," in *Lecture Notes in Computer Science vol. 7565, pp 182-193*, Rome, 2012 (rata de acceptare: 20%; indexat ISI).

- Raluca Iordache and Florica Moldoveanu, "QoS-aware web service semantic selection based on preferences," in *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, Zadar, Croatia, Procedia Engineering (2014), pp. 1152-1161.

- R. Iordache and F. Moldoveanu, "A web service composition approach based on QoS preferences," in *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013)*, Kauai, Hawaii, 2013, p. 220-224 (rata de acceptare: 27%).

- Raluca Iordache and Florica Moldoveanu, "A genetic algorithm for automated service binding," in *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, Zadar, Croatia, Procedia Engineering (2014), pp. 1162-1171.

- Raluca Iordache and Florica Moldoveanu, “An efficient approach for an end to end web service composition based on QoS preferences” în UPB Scientific Bulletin (în curs de apariţie).

- Raluca Iordache, Serban Iordache and Florica Moldoveanu, "A Framework for the Study of Preference Incorporation in Multiobjective Evolutionary Algorithms," in Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation, GECCO 2014, Vancouver, Canada, ACM (în curs de apariţie; indexat ISI).

1.4 Organizarea tezei

Teza de doctorat este formata din zece capitole. Primele şase capitole prezintă domeniul de studiu, descriu stadiul actual al cercetării şi problemele deschise existente. Următoarele trei capitole reprezintă contribuţii originale, menite să rezolve unele dintre problemele deschise menţionate în capitolele anterioare. Ultimul capitol prezintă concluziile activităţii de cercetare.

Capitolul 1 prezintă pe scurt problemele existente în prezent în cadrul direcţiei de cercetare a compunerii serviciilor web bazate pe cerinţele şi preferinţele clientului şi formulează obiectivele tezei de doctorat.

Capitolul 2 prezintă noţiuni introductive şi stadiul actual al cercetării în domeniul serviciilor web şi al arhitecturilor bazate pe servicii.

Capitolul 3 prezintă stadiul actual al cercetării dedicate compunerii serviciilor web, standardele existente şi cele mai importante abordări propuse în această direcţie.

Capitolul 4 prezintă stadiul actual al cercetării în domeniul calităţii serviciilor web şi metodele propuse de adaptare a standardelor existente pentru a încorpora atributele de calitate QoS.

Capitolul 5 prezintă provocările aduse de compunerea serviciilor web bazată pe calitatea serviciilor şi prezintă metodele de calcul al calităţii pentru o compunere de servicii şi modalităţile de optimizare ale acesteia.

Selecţia bazată pe calitatea serviciilor QoS este o problemă de optimizare multiobiectiv, iar Capitolul 6 tratează metodele de rezolvare ale acestei probleme prezentând abordările tradiţionale precum şi metodele evolutive.

În Capitolul 7 propunem o metodă inovativă de specificare a preferinţelor denumită QoSPref,folosind un limbaj simplu şi intuitiv dar care permite în acelasi timp exprimarea unor preferinţe complexe. Aici este prezentată şi aplicaţia care implementează această metodă şi care demonstrează eficacitatea metodei propuse.

Capitolul 8 îşi îndreaptă atenţia asupra căutarii serviciilor web ţinând cont de atributele QoS şi prezintă contribuţia tezei în această direcţie. Abordarea propusăîn acest capitol este bazată pe extinderea ontologiei de preferinţe OWL-Q pentru a permite descrierea preferinţelor în limbajul QoSPref introdus în capitolul anterior.

Capitolul 9 tratează problema selecţiei serviciilor web în cazul modelelor complexe de compunere ţinând cont de calitatea globală a compunerii. Abordarea propusă combină algoritmul de selecţie QoSPref cu o metodă eficientă de calcul al valorii QoS agregate pe baza unor modele de orchestrare care pot conţine şi componente nestructurate. Implementarea pe care o oferim pentru metoda propusă foloseşte pentru alegerea soluţiei optime un algoritm genetic modificat.

Capitolul 10 prezintă concluziile activităţii de cercetare ale cărei rezultate sunt prezentate referitoare la contribuţiile propuse şi direcţiile viitoare de cercetare.

Pe lânga capitolele prezentate, lucrarea oferă şi o secţiune de Anexe care prezintă aplicaţiile practice realizate, detaliile de implementare precum şi paşii necesari pentru rularea acestor aplicaţii.

2.SOA

2.1 Definiții

Există multe definiții pentru SOA, majoritatea conțin elemente comune dar există şi diferențe în anumite puncte. Toate definițiile au un punct comun și anume acela că SOA este o paradigmă, un concept având ca scop creșterea flexibilității sistemelor complexe[**9**].

Una dintre cele mai des citate definiții este cea dată de OASIS (Organization for the Advancement of Structured Information Standards):

> *SOA (Service Oriented Architecture) este o paradigmă pentru organizarea şi utilizarea capabilităților distribuite care pot fi sub controlul unor domenii de proprietate diferite. Ea oferă un mecanism pentru descoperirea, interacțiunea şi folosirea unor resurse, cu scopul de a produce anumite efecte ținând cont de anumite precondiţii si efecte scontate.*

SOA propune câteva principii arhitecturale pentru dezvoltarea și integrarea aplicațiilor software, neimpunând adoptarea unei anumite tehnologii pentru implementare. SOA se bazează în prezent pe serviciile web, acestea fiind acceptate în mod universal ca un standard în industrie, dar SOA poate fi în principiu implementată folosind orice tip de tehnologie bazată pe servicii.

Serviciile de bază se împart in două categorii, în funcție de granularitatea lor:

- *servicii atomice* - implementează o funcționalitate de sine stătatoare și nu depind de alte servicii;
- *servicii compuse* - reprezintă servicii de afaceri compuse din mai multe servicii, coordonate spre exemplu folosind orchestrarea. Aceste tipuri de servicii sunt folosite pentru tranzacţii scurte.

În cadrul SOA există încă un tip de servicii folosit: serviciile workflow [**10**]. Aceste servicii sunt folosite pentru tranzacţii lungi şi pot fi privite ca un automat cu stări în cadrul căruia trecerea de la o stare la alta poate dura mai multe ore sau chiar zile.

2.2 Principii arhitecturale

Arhitectura orientată pe servicii se bazează pe mai multe principii arhitecturale, dintre care:

1. Încapsulare – Logica internă a unui serviciu nu este vizibilă pentru clienții acestuia.
2. Contract - Serviciile din cadrul unui depozit de servicii au un contract realizat folosind acelaşi standard.
3. Autonomie - Serviciile sunt autonome şi au un control mare asupra mediului de execuţie.
4. Interoperabilitate - Serviciile sunt cuplate slab şi sunt la rândul lor decuplate de mediul înconjurător.

5. Standardizare: descoperire, comunicare, invocare - Serviciile dispun de metadate pentru comunicare, astfel încât să poată fi descoperite şi interpretate cu uşurinţă de un agent software.
6. Reutilizare - Serviciile sunt componente ce pot fi reutlizate.
7. Compunere - Serviciile participă în mod efectiv la o compunere de servicii web, indiferent de complexitatea acesteia.

Despre o arhitectură se poate spune că este orientată pe servicii în momentul în care principiile arhitecturale de mai sus sunt folosite într-o mare măsură.

2.3 Cele patru caracteristici ale unei arhitecturi SOA

- Orientată spre procesul de afaceri – Arhitectura este creată pe baza proceselor de afaceri implementate într-un anumit moment şi este adaptată sincron cu evoluţia procesului de afaceri în timp.

- Independentă de o anumită companie/producător – Modelul architectural nu se bazează pe o anumită platformă proprietară, ci combină soluţii şi tehnologii diferite pentru a-şi atinge scopurile.

- *Bazată pe aplicaţii de întreprindere* – Arhitectura SOA este folosită majoritar pentru aplicaţii de tip întreprindere, încurajând refolosirea şi compunerea serviciilor pentru a extinde aplicaţiile tradiţionale.

- *Bazată pe compunere* – Arhitectura este bazată pe agregarea serviciilor şi pe metode agile de compunere a serviciilor.

Aceste caracteristici diferenţiază arhitectura SOA de alte modele arhitecturale și definesc cerinţele fundamentale ale unei arhitecturi ce se doreşte orientată pe servicii.

2.4 Modelul de interacțiune cu serviciile

Pentru realizarea unei arhitecturi orientate pe servicii este nevoie de următoarele elemente:

- Un model standardizat pentru reprezentarea şi accesul la aplicaţii sub formă de servicii;
- O infrastructură de conectare a serviciilor și de transport al datelor, ce poate fi realizată de un *Enterprise Service Bus*;
- O modalitate de compunere a funcționalităților oferite de aplicații pentru realizarea proceselor business.

Primul element este reprezentat de serviciile web, care în momentul actual constituie baza arhitecturii SOA. ESB este folosit pentru coordonarea serviciilor şi comunicarea dintre ele, iar pentru cel de-al treilea element, compunerea serviciilor web pentru realizarea

funcţionalităţii unor procese business, există o serie de propuneri, limbaje şi soluţii software, dar nu există o soluţie standard adoptată.

Figura următoare prezintă structura arhitecturii SOA privind conceptele de QoS.

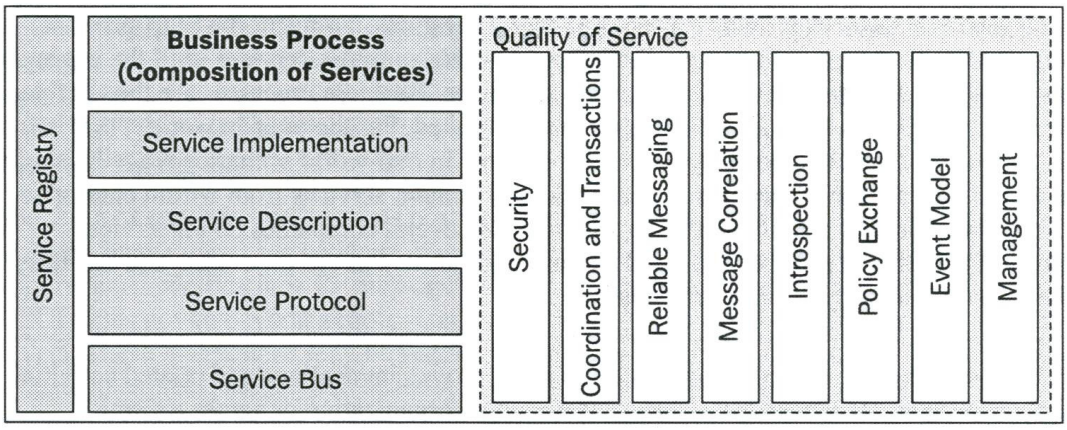


Fig. 1. Arhitectura SOA [11**]**

Pentru diversele blocuri componente ale SOA reprezentate în Fig. 1, există o serie de tehnologii ce le pot implementa pentru realizarea unei SOA funcționale. Spre exemplu, blocul de bază, nivelul de transport Service Bus, poate fi implementat de Enterprise-Service-Bus care oferă infrastructura pentru conectarea serviciilor. Ca protocol al serviciilor şi descriere a serviciilor pot fi folosite protocoalele de bază ale serviciilor web cum ar fi SOAP, respectiv WSDL. Similar, pentru registrul de servicii poate fi folosit UDDI. În principiu, orice limbaj de programare poate fi utilizat pentru implementarea serviciilor. În momentul de faţă, cel mai frecvent sunt folosite limbajele Java şi C#.

Pentru compunerea serviciilor sunt posibile o serie de abordări, care vor fi prezentate în capitolul 3.

2.5 Servicii Web

În prezent sistemele software trebuie să îndeplinească cerințe din ce in ce mai mari de scalabilitate si corectitudine. Din acest motiv, dezvoltarea software s-a îndreptat către un proces de reutilizare a componentelor software existente, în locul rescrierii complete a funcționalității cerute [McIlroy, 69] [**12**]. Serviciile Web reprezintă o paradigmă pentru dezvoltarea software bazată pe componente, folosind meta-limbajul de marcare standard XML şi protocolul HTTP.

2.5.1 Definiții

În literatură există o serie de definiții ale serviciilor web, definiţii ce poti fi văzute ca fiind complementare.

W3C (World Wide Web Consortium) definește serviciul web ca fiind “o aplicație software, definită printr-un URI (Uniform Resource Identifier), a cărei interfață poate fi definită, descrisă şi descoperită folosind artefacte XML și poate interacționa direct cu alte aplicații web prin intermediul mesajelor bazate pe XML, folosind protocoale bazate pe Internet”.

Serviciul web poate fi definit şi ca o funcționalitate business pusă la dispoziţia utilizatorilor prin intermediul internetului, ce poate fi accesată de clienţi umani sau de aplicaţii software[**13**]. Aplicații software scrise în limbaje de programare diferite și care rulează pe diverse platforme pot folosi serviciile web pentru a face schimb de date folosind o rețea (Internet), într-o manieră oarecum asemănătoare comunicării între procesele de pe un singur calculator. Interoperabilitatea se datorează folosirii unor standarde publice adecvate.

Serviciile web sunt în mod general folosite de către alte servicii sau aplicaţii web, utilizând protocoale cum ar fi HTTP sau SMTP.

2.5.2 Descoperirea serviciilor UDDI

UDDI furnizează un mecanism pentru înregistrarea şi descoperirea serviciilor Web. Componenta de bază a UDDI este un registru XML care conține informații despre serviciile pe care le oferă, astfel încât acestea să poată fi localizate de un client care interoghează registrul UDDI. Informația salvată în registrul UDDI este în mod conceptual organizată în “pagini albe” – informaţii legate de contractul serviciului, “pagini aurii” – informații legate de categoria industrială din care face parte procesul realizat de serviciul web şi “pagini verzi” ce conțin informații tehnice despre serviciu. Registrul UDDI permite descoperirea serviciilor în funcție de anumite aspecte funcționale, neavând în vedere proprietățile non-funcționale QoS ale serviciilor.

2.5.3 Descrieri ale serviciilor Web folosind WSDL

WSDL (Web Service Description Language) este un vocabular XML pentru descrierea operațiilor unui serviciu web. Un document WSDL descrie funcționalitatea oferită de un serviciu web, felul în care comunică şi adresa unde poate fi accesat. Descrierile WSDL conțin definiții ale interfeței serviciului, ale operațiilor și parametrilor de intrare și ieșire ale acestor operații, precum si informațiile necesare pentru accesarea serviciului.

WSDL nu conține - și asupra acestui fapt vom reveni în cadrul capitolului “Probleme ale tehnologiei Web Service” - date legate de calitatea serviciului sau ontologii pentru descrierea semantică a serviciului.

2.5.4 Comunicarea folosind SOAP

SOAP este un protocol simplu, bazat pe XML, folosit pentru schimbul de informaţii în Internet. Acest protocol nu defineşte vreun model de programare, deci poate fi utilizat atât în sistemele bazate pe schimbul de mesaje, în cele care utilizează RPC (Remote Procedure Call), cât şi în cele bazate pe obiecte distribuite. SOAP poate fi folosit cu mai multe protocoale de transport cum ar fi HTTP, SMTP si FTP. Un mesaj SOAP are o structură foarte

simplă, având un element XML numit *envelope* care defineşte conținutul mesajului și reprezintă principalul element utilizat la transmiterea informaţiei. SOAP *envelope* conține la rândul lui două elemente: un *SOAP* header, care conţine informații legate de securitate şi tranzacții, şi un *SOAP body,* care conține datele schimbate între servicii.

2.5.5 Semantica serviciilor Web

Standardele XML actuale folosite pentru descrierea serviciilor web oferă doar posibilitatea unei descrieri sintactice a serviciului, în timp ce semantica mesajelor lipseşte. Din această cauză, nu este posibilă o interpretare automată a serviciilor web, iar compunerea automată a serviciilor web este dificilă.

Serviciile web semantice SWS (Semantic Web Service) integrează ontologiile la serviciile web clasice pentru a ușura selecția, compunerea si monitorizarea automată a serviciilor web [**14**]. Reprezentarea bazată pe ontologii adaugă serviciului informații semantice ce pot fi interpretate automat.

Ontologia este o specificare formală explicită a unor concepte dintr-un anumit domeniu și a relațiilor dintre acestea. Ea implică un set de cunoștinte, legăturile semantice între acestea şi câteva reguli de inferenţă şi logică. O ontologie defineşte un vocabular comun pentru un anumit domeniu, fiecare termen fiind definit explicit şi posedând o semantică procesabilă de către calculator. Ontologiile facilitează comunicarea și oferă accesul la informaţie bazat pe conținut.

Semantica serviciilor web oferă o descriere a serviciilor la nivelul procesului implementat, ce conține pe lângă informații funcționale și alte informații cum ar fi precondiții şi postcondiții ale operațiilor serviciului, precum şi efectele executării unei anumite operații.

Pentru implementarea web semantic este nevoie să fie definite metadate semantice, care să fie adăugate la descrierea serviciului, în așa fel încât folosind aceste informații semantice, un calculator să poată prelucra datele în mod efectiv. Înţelegerea automată a datelor se bazează pe formatul comun stabilit de ontologie. Următoarele tehnologii semantic web / framework-uri sunt folosite în acest scop: RDF/RDFS , OWL/OWL-S, WSMF/WSMO.

2.5.5.1 RDF /RDFS (Resource Descripton Format - Schema)

RDF [**15**] este un cadru pentru reprezentarea informațiilor in internet. RDF oferă procesarea metadatelor resurselor disponibile pe internet, independent de domeniul de folosire a datelor. Scopul RDF este specificarea semantică a datelor printr-o metodă standardizată bazată pe XML, independent de domeniul folosirii datelor. O resursă este definită de un URI (ex. o pagină web). Pe lângă resurse, cadrul RDF include proprietăți și declarații. O proprietate reprezintă un aspect specific, un atribut, o relație ce descrie resursa. O anumită resursă împreună cu o proprietate a sa având asignată o valoare formează o declarație. Această valoare poate fi o altă resursă sau o valoare de tip literal (text). Descrierea RDF este astfel o listă de 3-upluri {resursă, proprietate, valoare}. În RDF nu este specificat niciun

mecanism de definire a relațiilor dintre atribute şi resurse. Acest lucru este realizat de către RDFS (Resource Descripton Format Schema), care oferă un mecanism pentru a descrie proprietăți specifice unui anumit domeniu de folosire a datelor și clase de resurse ce conțin aceste proprietăți. RDFS definește un sistem de clase similar celui întâlnit la programarea orientată pe obiecte. Clasele sunt organizate ierarhic prin extinderea subclaselor. Proprietățile şi subproprietățile construiesc de asemenea ierarhii de proprietăți. În RDFS sunt definite domeniul de folosire a datelor şi plaja de valori în care își găsesc valorile clasele și proprietățile. Astfel RDF definește noțiuni de bază și RDFS reprezintă extensia semantică a RDF și definește un limbaj de descriere a vocabularului folosit.

Având o infrastructură semantică, aplicații complexe pot fi construite prin compunerea datelor din diferite aplicații, compunere care să țina cont de semnificația datelor.

2.5.5.2 OWL-S

OWL-S (Ontology Web Language for Services) [**16**], denumită anterior DARPA Agent Markup Language for Services (DAML-S), este o ontologie de servicii care își propune descrierea serviciilor astfel încât un agent software să poată interpreta serviciul respectiv.

Structura ontologiei este prezentată în figura de mai jos.

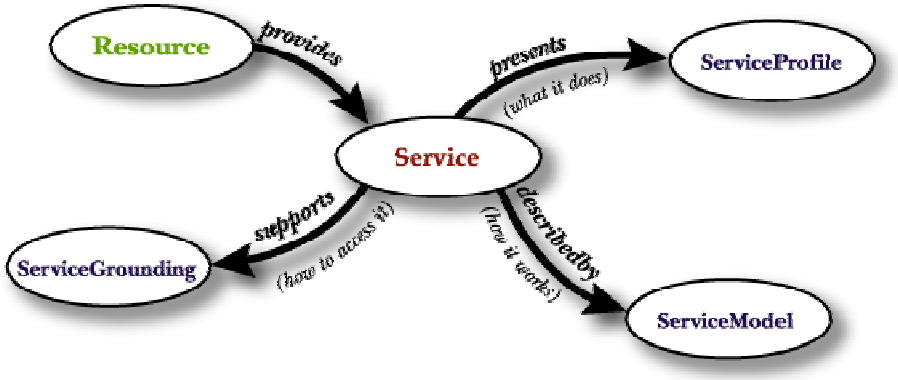


Fig. 2. Structura ontologiei OWL-S

OWL-S este organizată în așa fel încât să dea un răspuns la următoarele trei întrebări:

1) Ce oferă serviciul pentru viitorii clienţi?

   Răspunsul la această întrebare este dat de “profilul” serviciului. Fiecare serviciu web definește un *ServiceProfile*, care descrie funcționalitatea serviciului, limitările existente în aplicarea acestuia, precum și atributele sale de calitate (QoS - Quality of Service). Această descriere este realizată în așa fel încât un agent software de căutare să poată stabili dacă serviciul îndeplinește criteriile sale de căutare şi realizează funcționalitatea căutată.

2) Cum poate fi folosit serviciul?

   Răspunsul este dat de modelul procesului, definit de clasa *ServiceModel*. Modelul serviciului descrie modul în care serviciul poate fi folosit, semantica pe care trebuie să o aibă o cerere trimisă serviciului şi posibilele răspunsuri/reguli oferite ca rezultat.

3) Cum poate fi apelat serviciul?

   Răspunsul este dat în partea de "grounding" a serviciului. Aici sunt definite detalii despre protocoalele de comunicare cu serviciul şi modalitățile de apelare ale acestuia. Aceste atribute sunt definite în clasa *ServiceGrounding*.

Structura generală a acestei ontologii are trei părţi principale:

- *profilul* serviciului, pentru descrierea şi descoperirea serviciului;
- *modelul proces*, care oferă o descriere detaliată a operaţiilor serviciului;
- *implementarea,* care oferă detalii despre modalitatea de interoperare cu serviciul, prin mesaje.

OWL-S extinde OWL pentru a suporta descoperirea serviciilor în funcție de proprietăți, compunere și invocare automată şi monitorizarea execuției. OWL-S nu înlocuiește standardele curente pentru servicii web, ci încearcă să adauge capacități semantice folosind conceptele definite anterior.

2.5.5.3 WSMF / WSMO

WSMO (Web Service Modelling Ontology) oferă un model conceptual pentru descrierea aspectelor relevante ale serviciilor web în scopul descoperirii automate, compunerii şi invocării serviciilor. Descrierea elementelor WSMO este realizată prin utilizarea WSML (Web Service Modeling Language) – o familie de limbaje pentru definirea ontologiilor, ce constă într-un număr de variante bazate pe formalisme logice şi diferite niveluri de expresivitate semantică. WSMO defineşte un model conceptual pentru WSMX (Web Service Modeling Execution). În acest fel WSMO, WSML şi WSMX oferă un cadru coerent pentru modelarea, descrierea şi execuţia serviciilor web semantice [**17**].

WSMF definește patru categorii: scopuri, ontologii, mediatori și servicii web.

Scopul definește problema implementată de serviciul web.

Ontologia definește semantica formală a termenilor folosiți de WSMF.

Conceptul de *mediator* în cadrul WSMO adresează problema interoperabilităţii elementelor eterogene. Rolul mediatorului este de a rezolva conflictele care pot apărea atunci când se folosesc terminologii diferite la nivel de date, la nivel de comunicare între servicii (nivel protocol) și la nivel de proces de afaceri. Mediatorul oferă posibiliatea cuplării unor resurse ce folosesc terminologii diferite, într-o manieră slab cuplată, rezolvând astfel problemele de interoperabilitate existente într-un mediu distribuit.

Serviciul web defineşte elementele ce descriu serviciul, cum ar fi: precondiţii, postcondiţii şi alte elemente de control ale procesului[**13**].

Ontologia WSMO oferă o specificaţie ontologică a elementelor definite de WSMF. Definiţia WSMO a unui serviciu este bazată pe următoarele patru concepte: *proprietăţile nefuncţionale*, *mediatorii folosiţi, capacitatea serviciului* şi *interfaţa*.

Proprietăţile nefuncţionale descriu caracteristici ale serviciului, cum ar fi: costul serviciului, performanţa, securitatea şi accesibilitatea. Acestea sunt folosite în principal de agenţii software în momentul căutarii şi selectării unui serviciu web.

Mediatorii WSMO reprezintă mediatorii din WSMF folosiţi de serviciul respectiv.

Capacitatea serviciului defineşte funcţionalitatea şi comportarea serviciului şi poate ajuta agenţii software să localizeze serviciul căutat pentru îndeplinirea unui anumit scop.

Interfaţa serviciului defineşte modul în care serviciul respectiv poate fi apelat. WSMO descrie în interfaţa unui serviciu comportarea acestuia din două perspective:

- *comunicare*: modul în care serviciul comunică cu clientul → *coregrafie*;
- *cooperare*: modul în care serviciul foloseşte alte servicii pentru îndeplinirea scopului său → orchestrare [**18**].

WSMX (Web Service Modelling Execution Environment) - este un mediu de execuţie care permite descoperirea, selecţia, medierea şi invocarea serviciilor Web semantice. WSMX este bazat pe un model conceptual furnizat de WSMO şi demonstrează fiabilitatea abordării WSMO în realizarea interoperabilităţii dinamice a serviciilor Web[**19**].

2.5.5.4 SWRL- Semantic Web Rule Language

SWRL este un limbaj pentru Semantic Web, propus de W3C, care oferă posibilitatea exprimării unor reguli logice folosind OWL-DL (Ontology Web Language Description Logic), OWL Lite şi RML (Rule Markup Language).

Regulile sunt definite sub forma unei implicaţii între una sau mai multe condiţii specificate în cadrul unui bloc “antecedent” şi una sau mai multe condiţii specificate în cadul unui bloc de “concecinţe” şi pot fi interpretate sub forma: de fiecare dată când condiţiile specificate în blocul antecedent sunt îndeplinite, condiţiile specificate în cadrul blocului de consecinţe sunt deasemenea îndeplinite.

SWRL îmbogăţeşte limbajul OWL prin posibilitatea exprimării unor condiţii şi reguli de validare. O serie de ontologii pentru preferinţe integrează regulile SWRL pentru a permite conversia unor metrici diferite, permiţând astfel compararea unor atribute exprimate în unităţi diferite de măsură.

Există o serie de editoare şi de motoare de execuţie pentru SWRL.

2.5.5.5 Servicii REST

Serviciile web REST (Representational State Transfer) pot fi considerate o versiune mai simplă a serviciilor web SOAP şi o alternativă a acestora. Serviciile web REST folosesc HTTP pentru transferul de informații dintre servicii. REST foloseşte protocolul HTTP nativ oferind astfel un acces rapid la date şi resurse cu ajutorul serverelor web. REST folosește metode HTTP cum ar fi GET, POST, UPDATE, DELETE şi OPTIONS. În acest moment nu există nici o metodă standard care să ofere un cadru pentru reprezentarea aspectelor de securitate a serviciilor, a atributelor nefuncţionale sau pentru compunerea serviciilor REST.

Pentru un acces rapid de citire asupra unor resurse, REST este o alternativă la serviciile clasice, ce poate fi luată in considerare. REST este folosit în momentul actual şi de aplicaţii web mari cum ar fi cele oferite de Google sau Amazon.

2.5.5.6 Problemele tehnologiei Web Service

Tehnologia Web Service are o serie de probleme care îngreuneaza interoperabilitatea dintre servicii. Fişierele WSDL nu au o "durată de viaţă" / stare, lucru necesar anumitor procese, şi de asemenea nu permit salvarea unor proprietăți importante ce definesc serviciul. Direcţia de cercetare a acestei lucrări adresează aceste aspecte prin prisma necesității definirii proprietăților non-funcţionale, de calitate a serviciilor (QoS - Quality of Service). Definirea unor proprietăți cum ar fi disponibilitatea, accesibilitatea, securitatea sau scalabilitatea este necesară pentru o alegere corectă, conformă cerințelor detaliate ale unui client.

Procesele care apelează serviciile web sunt dinamice, într-o schimbare continuă datorată modificării continue a cerințelor de afaceri, iar o compunere statică a servicilor web nu poate face față acestor cerințe. Pentru a îndeplini aceste cerințe și a face față mediului dinamic actual, serviciile web trebuie să poata fi compuse dinamic, ținând cont de cerințele de calitate ale clientului consumator de servicii.

Dintre toate limbajele SWS existente, OWL-S este cel mai matur şi des folosit de frameworkurile de căutare a serviciilor web, bazate atât pe funcţionalitatea şi proprietăţile nefuncţionale ale serviciilor web [**20**]. În cadrul lucrării [**21**] autorii analizează, pe baza unei scenariu propus practibilitatea specificaţiei OWL-S pentru scenariile de căutare, invocare, compunere, orchestrare şi monitorizare automată a serviciilor web şi prezintă dezavantajele şi limitările acestei specificaţii. Lucrarea prezintă un prototip de funcţionare a unui scenariu bazat pe OWL-S şi prezintă problemele masive întâmpinate în cadrul implementării. OWL-S nu oferă posibilitatea reprezentării unor procese şi cunoştinţe procedurale bazat pe semantică prin lipsa posibilităţii de a definii variabile pentru a exprima condiţiile şi parametri existenţi în cadrul proceselor modelate. Deasemenea procese complexe, care au activităţi ciclice şi condiţii de execuţie nu pot fi reprezentate formal folosind OWL-S.

Pentru compunerea serviciilor web, OWL-S a fost folosit cu succes în scenarii de planificare AI (artificial Intelligence) pentru generarea şi rularea automată a planurilor de compunere[**22**]. Pentru scenarii de orhestrare OWL-S are următoarele dezavantaje majore: nu este un model de orchestrare şi nici un model de conversaţie, ci doar descrie comportarea serviciilor web. În majoritatea scenariilor proceselor business practice

comunicarea este de tip asincron, în timp ce OWL-S descrie execuţia unui serviciu web ca fiind o serie de remote procedure calls. WSMO oferă posibilitatea comunicării asincrone, în cadrul compunerii de servicii web, deoarece conţine un model de coreografie bazat pe ASM (Abstract State Machines) ale cărei principii includ şi modelarea schimbarilor de stare folosind reguli de tranziţie. WSMO oferă din punct de vedere semantic aceleaşi avantaje ca şi OWL-S pentru cautarea serviciilor web şi există o serie de frameworkuri bazate pe el. În ceea ce priveşte modelul de orchestrare oferit de WSMOl, acesta este foarte primitiv şi poate fi folosit numai în scenarii simple de interacţiune a serviciilor web.

3. Compunerea serviciilor Web

3.1 Context şi motivație

Existența unei oferte mari de servicii web aduce cu sine necesitatea dezvoltării unui mecanism de compunere a serviciilor pentru realizarea unui proces de afaceri complex ce îmbină funcționalitatea mai multor servicii existente, în scopul obținerii funcţionalității dorite. Rezultatul obținut se numește compunere a serviciilor, iar în momentul în care acest serviciu este oferit ca un serviciu de sine stătător, el se numește serviciu compus.

Compunerea serviciilor web ridică numeroase probleme legate de posibilitatea descoperirii serviciilor web necesare şi de asigurarea corectitudinii unei soluţii de compunere. Pentru a "măsura" corectitudinea unei soluţii se pot folosi metode formale. Aceste metode formale pentru specificarea şi verificarea sistemelor informatice constituie o nouă direcţie de cercetare ce este folosită în multe domenii informatice [**23**].

Compunerea serviciilor web se poate clasifica în funcție de mai multe criterii, cum ar fi: perspectiva din care este realizată compoziţia, momentul realizării ei și în funcție de gradul de automatizare.

Compunerea serviciilor web poate fi realizată folosind o abordare top-down sau bottom-up[**20**]. În cazul abordării top-down compunerea este modelată la un nivel înalt, iniţial este definit un proces business care este apoi detaliat şi transformat într-o compunere de servicii prin alegerea serviciilor componente care pot alcătui procesul definit. Abordarea bottom-up încearcă să combine servicii web existente pentru a realiza un proces abstract.

În cadrul diferențierii folosind drept criteriu momentul realizării compoziției, putem defini două momente: *faza de proiectare* a serviciului (*design-time*) şi *faza de rulare* a acestuia (*run-time*).

Diferențierea în funcție de gradul de automatizare definește trei grade de automatizare: compunere *manuală*, *asistată* şi *automată* a serviciilor.

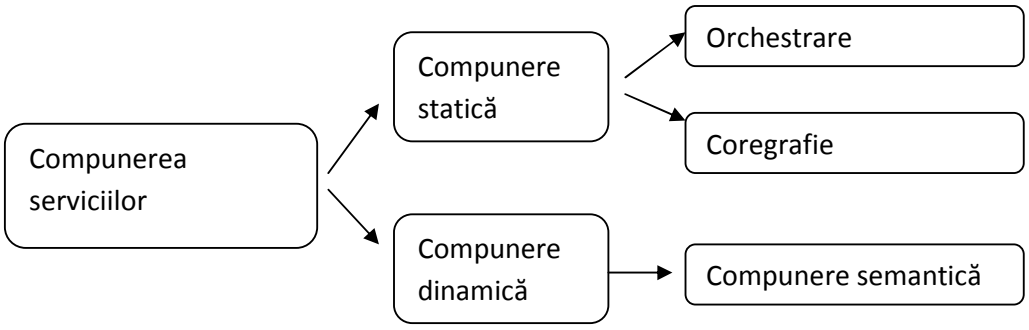
Compunerea serviciilor realizată în momentul proiectării implică o compunere statică, realizată în faza de dezvoltare a aplicaţiei sau sistemului. O compunere realizată în faza de rulare este o compunere dinamică, creată şi optimizată pentru cererea din momentul respectiv.

Diferențierea unei compuneri în funcție de gradul de automatizare este un proces complex, granița dintre cele trei categorii (manuală, asistată sau automată) fiind în unele situaţii neclară. În cazul compunerii manuale, o persoană realizează compunerea serviciilor. La o compunere asistată, persoana poate folosi o aplicație de modelare ce o ajută în alegerea şi combinarea serviciilor. În cazul compunerii automate, o aplicație realizează compunerea serviciilor, fără intervenție umană.

Compunerea serviciilor se realizează în următorii paşi:

- Identificarea scopului compunerii
- Selectarea serviciilor
- Ordonarea corectă a serviciilor (controlul intrărilor şi ieşirilor şi al datelor transmise)
- Execuţia compunerii
- Verificarea compunerii obţinute, monitorizarea soluţiei
- Adaptarea, modificarea, în funcţie de rezultatele verificării şi montorizării.

Realizarea unei compuneri de servicii se împarte în următoarele categorii:



3.2 Compunere manuală / statică

În momentul de față, compunerea statică este abordarea cea mai frecvent întâlnită. Compunerea creată realizează toate funcţionalitățile definite de un anumit proces, definește serviciile implicate în compunere, ordinea în care acestea sunt apelate și fluxul datelor dintre servicii.

Compunerea statică a mai multor servicii poate fi realizată prin orchestrare, coregrafie, coordonare şi asamblare.

În următoarele secţiuni ale acestui capitol voi descrie cele mai importante abordări ale compunerii statice a serviciilor web.

3.2.1 Orchestrare

Orchestrarea serviciilor presupune existența unui proces/serviciu central, care așa cum se observă și în Fig. 3, are rolul de coordonare explicită a serviciilor. Serviciile apelate sunt slab cuplate și nu cunosc date despre serviciul compus în care sunt implicate. Coordonatorul central / orchestratorul pare a dispune de o inteligență și un control autonom, acest lucru fiind de fapt rezultat din automatizarea sistemelor implicate și din utilizarea elementelor de control al fluxului dintre servicii. Orchestratorul cunoaște procesul și este responsabil de realizarea corectă a acestuia. Datorită coordonării centrale, schimbări în cadrul procesului se realizează într-un singur punct și erorile apărute pot fi rezolvate central.

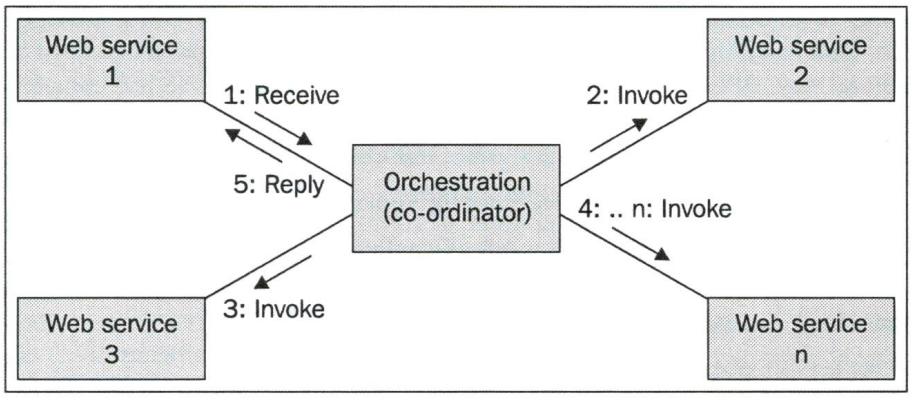


Fig. 3. Orchestrare [11**]**

WS-BPEL (WS-Business Process Execution Language) [**24**] este un limbaj de orchestrare, bazat pe XML, care permite modelarea de procese business abstracte sau concrete ale căror activităţi sunt bazate pe servicii web. Procesele business abstracte au un rol descriptiv şi pot fi implementate de unul sau mai multe use-case-uri. Procesele business concrete sunt executabile şi descriu comportarea concretă a unui participant la interacţiunea business. WS-BPEL defineşte un model de integrare interoperabil care facilitează execuţia automată a unor procese business complexe atât din cadrul unei corporaţii, cât şi din afara ei.

3.2.2 Coregrafie

În cadrul coregrafiei serviciilor nu există un coordonator central; fiecare serviciu web ştie când să se execute şi cu cine interacţionează (Fig. 4) - efort colaborativ bazat pe sincronizare şi schimbarea de mesaje.

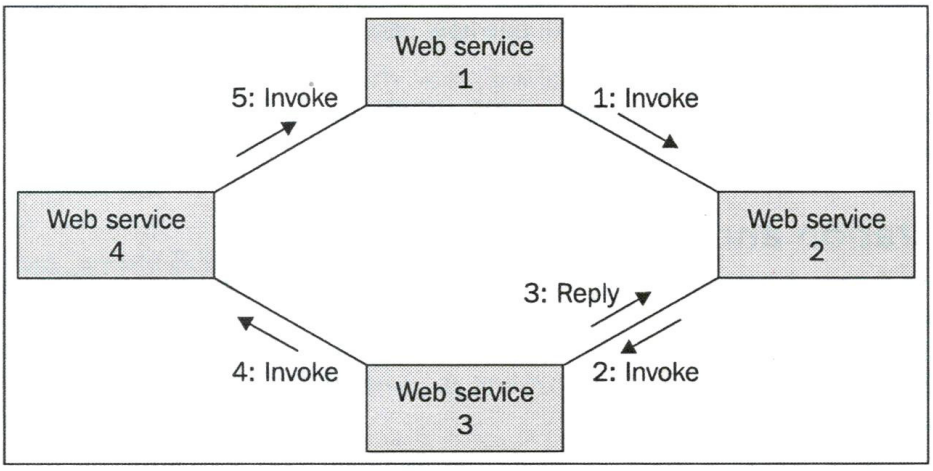


Fig. 4. Coregrafia serviciilor [11**]**

Coregrafia este în general folosită în cadrul unui sistem, în timp ce orchestrarea serviciilor este folosită pentru cereri de ansamblu, ce implică mai multe sisteme.

WS-CDL (WS-Choreography Description Language) [**25**] este un limbaj de coregrafie, bazat pe XML, care descrie comunicarea *peer-to-peer* dintre diferite servicii web, dintr-un punct de vedere global (nu a unui serviciu web implicat în comunicare), comportarea serviciilor web participante. Schimbul de mesaje dintre servicii conduce la îndepinirea unui anumit scop comun.

3.2.3 Coordonare

Coordonarea serviciilor oferă un model în care o serie de instanţe de servicii web execută o activitate distribuită urmând un protocol de coordonare. Un coordonator decide dacă un protocol a avut succes sau nu şi informează participanţii. Serviciile participante la un protocol de coordonare nu comunică între ele, ci direct cu serviciul coordonator care conduce protocolul de coordonare.

În prezent, există două specificaţii similare pentru coordonarea serviciilor: WS-Coordination [**26**] şi WS-CAF (Web Service Composite Application Framework) [**27**].

WS-Coordination oferă un framework extensibil pentru coordonarea interacţiunilor dintre serviciile web. Interacţiunile coordonate poartă numele de *activităţi*, iar un serviciu care ia parte la o activitate se numeşte *participant*. Pentru a coordona activităţi tranzacţionale sunt definite specificaţiile WS-Atomic Transaction şi WS-BusinessActivity. Un serviciu coordonator reprezintă o agregare a următoarelor trei servicii: înregistrare, activare şi protocolul de coordonare. Înainte de începerea unei activităţi, un serviciu se înregistrează la serviciul de înregistrare al coordonatorului. Activarea unei activităţi se realizează atunci când serviciul iniţiator solicită un context de coordonare de la serviciul de activare. Protocolul de coordonare defineşte protocolul specific care trebuie urmat.

WS-CAF este un framwework similar, care foloseşte specificaţiile WS-CTX (WS-Context), WS-CF (WS-Coordination Framework) şi WS-TXM (WS-Transaction Management) pentru a coordona servicii web în scopul realizării unei compuneri de servicii.

3.2.4 Asamblare

O compunere de servicii poate fi privită ca o asamblare de servicii existente [**20**]. Pentru a crea un serviciu compus executabil, o serie de servicii trebuie sa fie *asamblate* astfel încât să poată funcţiona împreună. O astfel de *asamblare de servicii* constituie un artefact care poate să fie instalat într-o ESB (Enterprise Service Bus) şi astfel poate fi apelat prin protocoale cum ar fi SOAP sau HTTP. O astfel de asamblare se servicii poate la rândul ei sa fie folosită, ca parte atomică, într-o altă asamblare de servicii.

SCA (Service Component Architecture) este o tehnologie care oferă un model pentru compunerea de aplicaţii bazată pe principiile SOA. Această tehnologie are la bază principiul SOA, în care o aplicaţie este alcătuită dintr-o serie de servicii web care pot fi compuse pentru a oferi o nouă funcţionalitate business. SCA acceptă multe tehnologii diferite şi metode de comunicare între componentele sale. SCA Assembly Model [**28**] este o specificaţie care defineşte configuraţia unui sistem SCA. Specificaţia cuprinde o serie de artefacte care sunt

definite de elemente descrise în fişiere XML. O componentă SCA oferă o funcţionalitate şi implementarea acestei funcţionalităţi într-una din tehnologiile acceptate de SCA cum ar fi Java, BPEL, C++, EJB, etc.

O soluţie SCA poate rula în cadrul unui Service Component Runtime care este o parte a Enterprise Service Bus.

Atributele nefuncţionale QoS ale unei asamblări de servicii, precum şi cerinţele QoS pot fi descrise în cadrul SCA folosind Policy Framework.

3.2.5 Model driven service composition

În cadrul compunerii statice a serviciilor web putem să observăm modele de compunere bazate pe idei din cadrul MDD (Model Driven Development). Modelul de compunere este generat automat plecând de la un proces modelat abstract. Un limbaj de modelare de nivel înalt, care poate fi folosit şi pentru a modela orchestrări de servicii, este UML, prin intermediul diagramelor de activitate. Procesul abstract este definit folosit diagrame de activitate, care pot fi transformate în reprezentări executabile. Autorii [**29**] propun o metodă în care pe baza diagramelor de activitate UML care descriu compunerea serviciilor web, folosind metodologia MDA (Model Driven Architecture), prin transformări XSLT, generează specificaţii executabile pentru o serie de limbaje de compunere, cum ar fi WS-BPEL şi WorkSCo. O contribuţie a metodei este transformarea descrierilor WSDL în UML. Metoda foloseşte pe lângă construcţiile UML standard şi un set de extensii pentru servicii web. Această metodă care oferă şi un prototip implementat este extinsă de către autorii lucrării [**30**] care iau în considerare în cadrul compunerii serviciilor şi atributele QoS şi efectuează o alocare dinamică a serviciilor alese pentru a implementa o activitate. Metoda nu ia în considerare posibilitatea realizării unei soluţii de compromis, prin exprimarea preferinţelor, în alegerea soluţiei şi necesită intervenţie umană pentru a modifica criteriile QoS în cazul existenţei unui număr mari de soluţii. De asemenea, metoda pleacă de la premisa existenţei unui registru semantic ce conţine servicii care au proprietăţile QoS definite folosind WS-QoS, un limbaj nestandard, complementar descrierii WSDL, introdus de [**31**].

O altă direcţie din cadrul compunerii de servicii bazată pe modelare este folosirea limbajului BPMN pentru a reprezenta modelele business, în locul reprezentării UML. Autorii lucrării [**32**] susţin că în mediul analiştilor care proiectează procese business, BPMN este mult mai des întâlnit decât UML, fiind folosit şi de multe aplicaţii pentru proiectare de procese. Din experienţa practică în cadrul unor proiecte software complexe, susţin această afirmaţie, de aceea metoda propusă în această lucrare porneşte de la modele BPMN pentru a realiza compunerea de servicii web. În cadrul metodei [**32**], autorii realizează o transformare de la BPMN la WS-BPEL, complexitatea transformării find dată de structura diferită a limbajelor, BPMN fiind un limbaj bazat pe grafuri, în timp ce limbajul WS-BPEL este structurat pe blocuri executabile. Metoda propusă înlatură limitările existente în definirea workflowurilor care

conţin activităţi ciclice şi structuri paralele. Voi descrie această metodă, pe care se bazează şi contribuţia BaaS , prezentată în cadrul secţiunii 8.2.

3.2.6 BPM, BPMN, BPEL şi BPEL4SWS

BPM (Business Process Management) este o abordare pentru gestionarea proceselor, având ca scop alinierea tuturor aspectelor unei organizații cu cerințele clienților. BPM are în vedere optimizarea activităților curente ale unei organizații și a interacțiunilor umane din interiorul și cu exteriorul firmei, operând cu acestea sub formă de procese. Un proces de afaceri reprezintă o serie de activități. Împreună, activitățile realizează un anumit scop. BPMS (Business Process Management Systems) coordonează procesele de afaceri provenind dintr-un mediu inter-organizaţional.

Pentru a putea gestiona procesele de afaceri, este nevoie ca acestea să poată fi descrise și documentate. Pentru descrierea proceselor complexe și a tuturor aspectelor relevante (cum ar fi fluxul de date, structurile de control, evenimentele) este nevoie de utilizarea unei notații.

Pentru modelarea proceselor de afaceri s-a evidențiat şi este folosit pe scară largă Business Process Management Notation (BPMN) [**33**]. Acest limbaj de modelare a proceselor pune accentul pe controlul fluxului și oferă un limbaj comun de modelare, ce poate fi folosit atât de către utilizatorul tehnic, cât şi de utilizatorul de afaceri, păstrând în acelaşi timp nealterată semantica procesului de afaceri [**34**]. Cu alte cuvinte, BPMN creează o punte standardizată pentru lacuna dintre procesul de proiectare a afacerilor şi procesul de implementare. Acest standard pentru modelarea proceselor s-a impus pe scară largă, existând în prezent un număr mare de aplicaţii care oferă posibiltatea de modelare BPMN.

Pentru descrierea procesului executabil este nevoie de un limbaj de compunere. Printre abordările existente se remarcă limbajul de compunere BPEL (WS-BPEL - Web Services Business Process Execution Language), un limbaj executabil ce specifică acțiunile serviciilor web din cadrul unui process de afaceri. BPEL este un standard OASIS (Organization for the Advancement of Structured Information Standards). BPEL este folosit pentru compunerea statică a serviciilor web.

BPEL definește comportarea unui proces de afaceri ţinând cont de interacțiunile acestuia cu alte procese partenere. Este definită o gramatică bazată pe XML pentru descrierea proceselor, a logicii de control şi a formatelor mesajelor. Pentru definirea serviciilor participante la compunere și a serviciilor de care este nevoie pentru realizarea unui proces, este folosit WSDL. Funcționalitatea unui proces BPEL este oferită clienților la fel ca şi în tehnologia serviciilor web, folosind WSDL.

Un principiu important ce stă la baza BPEL este separarea aspectelor interne ale procesului de cele externe. Pentru această separare sunt definite două variante: o variantă abstractă și o variantă executabilă a unui proces. Procesul abstract definește aspectele publice ale

interacțiunilor de afaceri, neincluzând detalii interne ale procesului. Execuția procesului se bazează pe logica procesului și specifică interacțiunea serviciilor implicate.

Un process de afaceri este realizat în mai mulți pași, numiți activități. Activitățile sunt clasificate în două categorii: activități de bază şi activități structurale. Activitățile de bază se ocupă de interacțiunile procesului cu resursele externe. Exemple de *activități de bază* sunt apelarea serviciilor web, recepția și transmiterea de date, specificarea problemelor și excepțiilor ce apar pe parcurs. Activitățile de bază pot fi combinate în algoritmi complecşi ce formează *activități structurale*. Exemple de activități structurale sunt: *sequence* (secvența în care sunt apelate anumite activități), *flow* (definește setul de activități ce pot fi apelate paralel), *switch-case* (elemente de control al fluxului BPEL) și *loop* (pentru implementarea iteraţiilor). BPEL oferă mecanisme pentru paralelizarea operațiilor și pentru tranzacții (mai multe activități pot fi agregate într-o singură tranzacție).

Pentru rularea unui proces BPEL este nevoie de un mediu de rulare, un server BPEL. Există o serie de implementări de servere BPEL, cele mai folosite fiind bazate pe Java EE, cum ar fi: Oracle BPEL Process Manager, IBM WebSphere Business Integration Server Foundation, BEA WebLogic Integration și AquaLogic. Există și o serie de implementări open source, cum ar fi: Apache Orchestration Director Engine, BPEL SE (parte integrantă din Enterprise Service Bus OpenESB de la Sun Microsystems). Folosind un server BPEL, toate procesele se află într-un anumit loc central, simplificând astfel mentenanța.

BPEL nu oferă posibilitatea interacțiunii umane în cadrul procesului, orchestrarea serviciilor putând fi realizată doar automat. Acest lucru s-a dovedit în practică a fi un mare dezavantaj, fapt ce a dus la propunerea (în cooperare de către IBM și SAP) unei extensii care rezolvă parţial această problemă: BPEL4People. Datorită acestei posibilități de integrare manuală a anumitor părți dintr-un proces, se pot realiza complet procese de afaceri complexe.

În practică, pentru a realiza procese complexe este necesară realizarea compunerii de servicii în mod dinamic, lucru care nu poate fi realizat doar pe baza descrierilor WSDL. Descrierea sintactică WSDL limitează posibilităţile de integrare a unor alte servicii care au o descriere sintactică diferită, dar sunt compatibile din punct de vedere semantic. Serviciile web semantice permit descrierea semantică a funcţionalităţii oferite, oferind soluţia pentru interpretarea automată a capabilităţilor oferite. În acest scop, un nou limbaj, BPEL4WS[**35**] oferă o extensie a limbajului BPEL, ce permite realizarea unei anumite funcţionalităţi bazându-se pe ontologii. BPEL4WS foloseşte limbajul BPELLight, care permite specificarea unor procese independent de tehnologia Web Service, interacţiunea dintre procese bazându-se pe schimbul de mesaje simple care nu folosesc niciun limbaj de descriere a interfeţelor existent. Acest limbaj oferă flexibilitate pentru definirea şi reutilizarea proceselor modelate. BPELLight permite ataşatea descriilor semantice la mesajele folosite, astfel încât se pot folosi frameworkuri semantice bazate pe OWL-S sau WSMO pentru a selecta serviciile care oferă o anumită funcţionalitate cerută. BPEL4WS poate folosi într-un proces atât servicii web semantice, cât şi servicii web convenţionale, ce pot intercomunica. Pentru a facilita această intercomunicare în cadrul unui proces, se foloseşte SAWSDL

(Semantic Annotations for WSDL) pentru a asocia reprezentării în XML a datelor reprezentările realizate folosind ontologiile. BPEL4WS poate oferi procesul modelat sub forma unui nou serviciu web semantic sau convenţional.

3.2.7 Electronic Business Extensible Markup Language (ebXML)

ebXML, numită în mod uzual și e-business XML, este o altă metodă de compunere bazată pe processe. Metoda a fost propusă de UN-CEFACT (United Nations Centre for Trade Facilitation and Electronic Business) și de OASIS (Organization for the Advancement of Structured Information Standards).

ebXML reprezintă o alternativă la tehnologiile pentru comerțul electronic deja existente, cum ar fi EDI (*electronic data interchange)*, tehnologie folosită de concernele mari, care implică costuri crescute. Principiul de bază al metodei ebXML este acela că "electronic Business XML facilitează colaborarea în afaceri oricui, oriunde și cu oricine".

În cadrul acestei metode procesul de afaceri poate fi specificat atât în UML cât și în XML. Specificarea în UML nu este folosită pentru a genera direct procesul, dar oferă posibilitatea definirii tuturor elementelor și relațiilor necesare. ebXML oferă o serie de specificații pentru a defini procese standard, specificații adoptate global de multe domenii din industrie. În cadrul acestei metode interacțiunea dintre servicii se realizează folosind coregrafia.

Collaborative Partner Profile Agreements (CPPA) reprezintă documente bazate pe XML, care definesc condițiile / termenii în care două companii pot colabora, definind un contract între cele două instituții. Fiecare instituție are un document propriu în care sunt descrise caracteristici cum ar fi protocoalele pe care le acceptă sau condițiile de securitate oferite. Modul în care două companii colaborează este descris de documentul CPA (Collaborative Protocol Agreements), un protocol care definește informații de identificare, securitate, comunicare şi reguli.

3.3 Compunere asistată

Crearea unor compuneri de servicii complexe este o sarcină dificilă și deseori predispusă la erori. Găsirea serviciilor optime, cuplarea lor şi verificarea funcționalității serviciului final reprezintă o secvență de operații complexe ce trebuie realizate. În momentul de față există o serie de aplicații care oferă "asistență" pentru realizarea acestor paşi, spre exemplu pentru căutarea şi localizarea serviciilor adecvate sau pentru verificarea soluției finale obținute. În cele ce urmează, voi prezenta pe scurt câteva dintre aceste aplicaţii.

Web Service Composer

Web Service Composer [**36**] este un prototip care asistă un utilizator în compunerea dinamică a serviciilor web. WSC oferă un proces semi-automat care realizează căutarea şi prezentarea de servicii adecvate, la fiecare pas al compunerii, filtrând serviciile pe baza descrierii lor semantice. Aplicația oferă posibilitatea compunerii serviciilor web având o specificație semantică în OWL-S și a executării compunerii alese. Pentru a realiza o

compunere de servicii utilizatorul foloseşte un mecanism de înlănțuire înapoi, căutând un serviciu care să ofere ca rezultat o valoare căutată. Aplicația Web Service Composer îşi propune şi implementarea unei înlănțuiri înainte, metodă în care se pleacă de la prima acțiune ce trebuie îndeplinită. De asemenea, aplicația oferă posibilitatea filtrării serviciilor în funcție de proprietățile lor ne-funcționale.

Compunerea generată poate fi executată folosind WSDL-urile serviciilor alese.

PASSAT

PASSAT (Plan Authoring System based on Sketches, Advice, and Templates) [**37**] este o aplicație interactivă care permite utilizatorului să definească planuri, bazate pe experiența anterioară stocată în template-uri și adaptate la preferința utilizatorului. PASSAT nu este prevăzut pentru realizarea de compuneri de servicii, dar conceptele pe care se bazează pot fi folosite în contextul compunerii de servicii. PASSAT este bazat pe HTN (Hierarchical Task Network). În HTN, sarcinile ce trebuie realizate sunt divizate în sarcini din ce în ce mai mici, până în momentul în care se obțin sarcini primitive ce pot fi realizate direct de către operatorii de planificare. Utilizatorul poate începe planificarea adăugând sarcini la un plan şi apoi divizând sarcinile folosind template-uri HTN. Aceste template-uri constau într-un set de sarcini care înlocuiesc sarcina inițială, împreună cu precondițiile acestora. PASSAT oferă și funcționalitatea de planificare automată, situație în care sistemul divizează singur sarcinile folosind anumite template-uri ale sistemului. Utilizatorul poate defini anumite constrângeri care restricționează setul de acțiuni folosite pentru a genera un anumit plan. Pe lângă abordarea de planificare top-down bazată pe folosirea rețelelor HTN, PASSAT oferă si posibilitatea de a “schița” un plan. După ce un plan a fost schițat, sistemul încearcă să găsească template-uri potrivite pentru a realiza planul respectiv şi îi oferă utilizatorului mai multe posibilități dintre care acesta poate să aleagă varianta potrivită. PASSAT verifică planul rezultat și informează utilizatorul în legătură cu cerințele și sarcinile ce necesită rezolvare pentru ca planul să fie complet și executabil.

CAT

CAT (Composition Analysis Tool) [**38**] este o aplicație care asistă utilizatorul în realizarea compunerii de workflow-uri interactive. CAT analizează workflow-ul, semnalează posibilele erori şi oferă sugestii pentru a ajuta utilizatorul să construiască workflow-uri complete şi consistente. La fel ca şi PASSAT, această aplicație nu este prevăzută direct pentru compunerea de servicii web, dar conceptele pe care le prezintă pot fi adaptate pentru acest lucru. CAT folosește baze de cunoștințe ce conțin multe reprezentări de componente și constrângeri, precum și tehnici de planificare bazate pe inteligență artificială pentru a coordona relațiile și constrângerile dintre diverse componente. Autorii au definit un algoritm independent de domeniu, numit ErrorScan, care verifică corectitudinea unui workflow. Acest algoritm verifică existența tuturor atributelor necesare și oferă în cazul unor erori sugestii

pentru corectarea problemei apărute. CAT folosește euristici pentru a stabili afișarea erorilor şi a sugestiilor oferite în funcție de importanța acestora.

3.4 Compunere automată

Pentru realizarea unei compuneri automate a serviciilor este nevoie ca serviciile să aibă o descriere semantică. Necesitatea automatizării procesului de compunere a *serviciilor Web* poate fi uşor motivată printr-un exemplu simplu din domeniul turistic. Pentru vacanţa de vară, o familie caută o excursie în Portugalia, excursie ce constă dintr-un pachet de servicii – bilet de avion, hotel şi maşină de închiriat pe toată peroada şederii. Preţul excursiei nu trebuie să depaşească 1500 EUR, iar data de plecare trebuie să fie într-un interval de trei zile dinainte stabilit. În cazul compunerii clasice a serviciilor Web, clientul trebuie să efectueze toate task-urile în mod manual, neautomatizat. Problema devine și mai complicată atunci când în cadrul compunerii sunt implicate mai multe servicii care oferă aceeași funcționalitate, dar care diferă prin parametrii de QoS. În aceste condiții, utilizatorul va trebui sa alegă din fiecare categorie de servicii pe acelea care compuse împreună satisfac cel mai bine nevoile sale.

Automatizarea procesului de compunere a serviciilor Web ar ușura enorm rezolvarea scenariului prezentat anterior, prin reducerea timpului necesar pentru realizarea și executarea procesului. În același timp sistemul realizat ar fi flexibil, permiţând actualizarea rapidă a procesului în cazul apariției unor servicii mai performante sau mai ieftine. De asemenea, în cadrul execuției unui proces complex, există riscul ca una dintre componente să eșueze în îndeplinirea funcționalității sale. În acest caz este nevoie de un mecanism care să asigure rularea procesului, prin înlocuirea rapidă şi eficientă a serviciului indisponibil. Una dintre posibilitățile de adaptare a compunerii la noua situație este schimbarea rapidă a cursului compunerii prin folosirea unei alte căi, predefinite pentru back-up. O altă posibilitate este reconstruirea unui nou proces, eliminând componenta cu eroare.

În [**39**] sunt identificate două direcţii principale de cercetare în domeniul compunerii automate a serviciilor: compunerea bazată pe workflow-uri şi compunerea folosind tehnici de planificare bazate pe inteligenţa artificială.

3.4.1 Compunerea serviciilor bazată pe workflow-uri

Abordările din această categorie se bazează pe similarităţile existente între structura unui serviciu compus şi cea a unui workflow: definiţia unui serviciu compus include un set de servicii atomice împreună cu fluxul de control şi de date între aceste servicii.

Soluţiile oferite diferă prin modul în care este generat workflow-ul corespunzător serviciului compus: în mod static sau dinamic. În cazul static, clientul construieşte un model abstract al procesului, care include o serie de sarcini, precum şi dependenţele între datele acestora. Fiecare sarcină conţine o clauză de interogare care este folosită pentru a căuta un serviciu web atomic adecvat. Deoarece modelul abstract al procesului este oferit de client, numai selecţia şi încorporarea serviciilor atomice se realizează automat. În schimb, în cazul

compunerii dinamice, automatizarea are loc atât la nivelul generării modelului abstract, cât şi la nivelul selecţiei serviciilor atomice.

Una dintre platformele care permit compunerea automată a serviciilor bazată pe workflow-uri este EFlow [**40**], care foloseşte o metodă statică de generare. Un serviciu compus este modelat ca un graf care defineşte ordinea de execuţie în cadrul nodurilor din proces. Graful este creat manual, dar poate fi actualizat în mod dinamic. El include noduri pentru servicii, decizii şi evenimente. Nodurile serviciilor reprezintă invocarea unui serviciu atomic sau compus, nodurile de decizie specifică alternativele şi regulile care controlează fluxul de execuţie, iar nodurile de evenimente permit proceselor corespunzătoare serviciilor să declanşeze şi să detecteze diverse tipuri de evenimente. Arcele grafului indică dependenţele în execuţia nodurilor. Deşi graful trebuie specificat manual, EFlow asociază în mod automat servicii concrete nodurilor din graf. Definiţia unui nod al unui serviciu conţine o procedură de căutare (*search recipe*) exprimată într-un limbaj de interogare. Atunci când nodul unui serviciu este invocat, procedura sa de căutare este executată pentru a selecta un serviciu adecvat.

O altă abordare bazată pe workflow-uri este oferită de Polymorphic Process Model (PPM) [**41**], care combină compunerea statică şi dinamică. Pentru realizarea compunerii statice, se consideră procese de referinţă care sunt compuse din sub-procese abstracte, adică sub-procese a căror funcţionalitate este clar definită, dar pentru care nu se specifică o implementare. Un serviciu care să ofere o astfel de implementare va fi selectat automat în momentul rulării, printr-o metodă asemănătoare celei din EFlow. Compunerea dinamică este obţinută în PPM prin intermediul proceselor bazate pe servicii. În acest caz, un serviciu este modelat ca o maşină de stare, care specifică stările posibile ale unui serviciu şi tranziţiile între acestea. Tranziţiile sunt determinate de invocări ale operaţiilor unui serviciu sau de tranziţii interne ale acestuia. Compunerea dinamică este realizată prin inferenţe logice aplicate maşinii de stare.

3.4.2 Compunerea serviciilor folosind tehnici de planificare bazate pe inteligență artificială

O nouă direcție la modă pentru compunerea automată a serviciilor este folosirea tehnicilor de planificare bazate pe inteligență artificială. O problemă de planificare specifică o stare inițială, o stare finală dorită, o mulțime de posibile acțiuni, precum şi relaţiile de tranziţie care definesc precondiţiile şi efectele execuţiei fiecărei acţiuni. Scopul unei probleme de planificare este de a găsi o secvenţă de acţiuni care aplicată unui sistem aflat în starea inițială să îl aducă în starea finală dorită. În cazul compunerii serviciilor, starea inițială şi cea finală sunt indicate de clientul serviciului web, iar mulţimea acţiunilor posibile este dată de mulţimea serviciilor atomice disponibile. Relaţiile de tranziţie sunt date de funcţiile de modificare a stării corespunzătoare fiecărui serviciu atomic.

Există o gamă variată de pachete software pentru rezolvarea problemelor de planificare folosind metode bazate pe inteligenţa artificială, iar diverşi cercetători au propus metode de utilizare a unora dintre aceste planificatoare în scopul compunerii automate a serviciilor.

Un exemplu în acest sens este abordarea propusă în [**42**], care foloseşte planificatorul SHOP2 [**43**]. Acesta este un sistem de planificare independent de domeniu, bazat pe metoda HTN (descrisă în secţiunea dedicată aplicaţiei PASSAT). SHOP2 folosește o bază de cunoștinte care conține *operatori* şi *metode.* Un *operator* reprezintă o descriere a acțiunilor ce trebuie efectuate pentru a îndeplini o sarcină de bază. O *metodă* descrie procesul prin care o anumita sarcină este descompusă în sarcini de bază. Pentru a putea utiliza acest planificator la compunerea automată a serviciilor, autorii lucrării [**42**] introduc un algoritm capabil să translateze descrierea OWL-S a serviciilor în domeniul lui SHOP2.

Alte abordări includ: utilizarea planificatorului Optop, care foloseşte *regresia estimată* [**44**], realizarea planificării folosind limbajul Golog, care este bazat pe *calculul situaţional* [**22**], utilizarea pachetului software ASTRO [**45**], care foloseşte tehnici de planificare bazate pe paradigma "*Planning via Model Checking*", sau realizarea planificării folosind procese de decizie Markov [**46**][**47**]. În [**48**] autorii compară astfel de abordări bazate pe tehnici de planificare ce folosesc inteligența artificială. Comparația este realizată folosind criterii prestabilite şi are în vedere doar compunerea serviciilor în sine, neluând în considerare problema descoperirii serviciilor web. În urma acestui studiu, autorii concluzionează că folosirea unui planificator HTN este superioară altor tehnici de planificare.

3.5 Descoperirea si selecția serviciilor Web

În viața de zi cu zi oamenii aleg furnizori de servicii bazându-se pe recomandări de la prieteni, familie, cunoștinte sau experți, sau căutând într-un director de servicii cum ar fi spre exemplu „Pagini aurii". Inspirate de această atitudine de selecție a serviciilor, există o serie de abordări pentru selecția serviciilor într-un mediu electronic.

În prezent, descoperirea serviciilor este în principal realizată folosind o căutare sub forma unui text liber. Acest lucru presupune ca numele şi descrierea serviciului să fie corecte şi sugestive. Acest proces este îngreunat de faptul că furnizorii de servicii folosesc de obicei terminologii diferite.

Pentru descoperirea serviciilor necesare pentru îndeplinirea unui anumit scop, un client poate căuta manual sau automat folosind numele și descrierea WSDL a serviciului, sau registrul UDDI. În cadrul soluțiilor de afaceri actuale se obișnuieste căutarea în mai multe registre UDDI şi apoi agregarea rezultatelor folosind tehnici de filtrare şi de ordonare în funcție de relevanță.

Serviciile web pot fi descoperite şi folosind WS-Discovery, o specificație OASIS ce defineşte un protocol multicast de căutare, bazat pe SOAP, pentru descoperirea în mod dinamic a locației unui serviciu într-o rețea locală.

Noțiunea de web semantic simplifică acest proces, oferind unui agent sau unei aplicații software posibilitatea descoperirii dinamice și a compunerii serviciilor web. În secţiunea 2.5.5.2 este prezentată ontologia OWL-S, ontologie ce poate fi folosită pentru a descrie din punct de vedere semantic serviciile web din mai multe perspective (spre exemplu: descoperire, invocare, compunere). Această abordare are însă anumite îngrădiri stricte [**49**]. Spre exemplu, în cadrul SOA utilizatorii interoghează directorul de servicii în care furnizorii își promovează serviciile folosind cuvinte cheie din cadrul ontologiilor folosite de ei. Din cauza faptului că serviciile sunt oferite de furnizori independenți, dintr-un mediu distribuit, este foarte probabil ca ontologia folosită de utilizator pentru interogări să difere de ontologia pe care se bazează descrierea serviciilor.

Framework-ul WSMO prezentat in capitolul 2.5.5.3 realizează o „traducere" a unei ontologii prin intermediul mediatorilor care rezolvă problemele de interoperabilitate apărute în cadrul cuplării serviciilor web.

Descoperirea serviciilor adecvate pentru realizarea unui anumit scop în funcţie de anumite preferințe reprezintă o problemă foarte importantă pentru utilizator. Un criteriu de care trebuie ținut cont în alegerea unui serviciu este calitatea acestuia, caracteristică descrisă de către atributele de calitate QoS prezentate pe larg în secţiunea 4.

În lucrarea [**49**], autorii prezintă o abordare interesantă de căutare a serviciilor web semantice, care definește o posibilitate de mapare a ontologiilor utilizatorului la ontologiile domeniului din care provine serviciul web, precum și un serviciu de definire și de stabilire a priorităţilor criteriilor de căutare, bazate pe aspecte funcționale şi nefuncționale indicate de utilizator. De obicei, rezultatele căutării sunt evaluate și priorizate în funcție de „gradul de potrivire" dintre interogare şi descrierea serviciului. În această abordare, prioritizarea este realizată şi în funcție de gradul de importanță pentru utilizator al diferitelor criterii din cadrul interogării. Pentru a exemplifica, putem da un exemplu din domeniul turistic: printre criteriile de căutare ale unui utilizator se află atât data de plecare, cât și prețul excursiei, iar utilizatorul poate acorda o importanţă mai mare prețului pachetului turistic decât plecării la o dată dorită. Într-o situație similară, obținând aceleași rezultate ale căutarii, un alt utilizator, neavând un program flexibil, preferă ca excursia să înceapă la data aleasă, indiferent de prețul acesteia. Astfel, este necesară existența unei posibilități de acordare a unui „rang" criteriilor pe baza cărora se face selecția. Această abordare oferă și un prototip pentru testarea conceptului prezentat.

Există o serie de abordări pentru extensia WSDL şi a UDDI pentru a conține informații semantice despre servicii. Capitolele 4.1.1 şi 4.1.2 prezintă o serie de asemenea abordări, folosite pentru adnotarea serviciilor cu atribute QoS, dar aceste abordări pot fi folosite şi pentru adnotarea serviciilor cu informații care să ajute la descoperirea lor.

METEOR-S (Web Service Annotation Framework) [**50**] este un framework ce adresează problema căutării serviciilor web în situații în care ontologiile utilizatorilor diferă de cele ale domeniul serviciului furnizat. Aplicaţia definește algoritmi pentru a adnota descrierile WSDL

ale serviciilor cu ontologii aparținând diferitelor domenii. Această clasificare şi adnotare este realizată automat, efectuând o conversie la o structură proprie numită *SchemaGraph* atât a descrierii WSDL a unui serviciu, cât și a ontologiei și folosind algoritmi specifici pentru a detecta corespondența dintre acestea. După ce fiecare concept din WSDL este comparat cu toate conceptele ontologiei, este aleasă cea mai bună corespondență. Această abordare definește patru categorii de informații semantice ce pot fi adnotate în cadrul serviciilor web: semantica datelor (parametrii de intrare şi ieșire), semantica funcțională (ce oferă serviciul) , semantica execuției (corectitudinea acesteia) şi semantica atributelor de calitate QoS.

4. Calitatea serviciilor Web (QoS)

În momentul actual, existenţa unei oferte mari de servicii web ce implementează funcţionalităţi similare sau chiar identice face necesară investigarea unor posibilităţi de comparare a serviciilor web pe baza unor criterii legate de calitatea serviciilor (QoS).

QoS este definit în ISO 8402 [**51**] ca “totalitatea atributelor și caracteristicilor unui produs sau serviciu, care influențează capacitatea acestuia de a satisface anumite cereri stabilite sau implicite”. QoS este un concept larg, ce cuprinde atribute importante, funcţionale şi nefuncţionale, cum ar fi: metrici de performanţă, atribute de securitate, integritate tranzacţională, scalabilitate, fiabilitate, disponibilitate - caracteristici ce sunt definite pentru fiecare serviciu în parte.

Un atribut de calitate nu este însă o măsură cantitativă. Spre exemplu, durata de execuţie a unui serviciu nu are o valoare constantă, ci variază de la un apel la altul. De aceea durata de execuție a unui serviciu poate fi specificată prin intermediul unor sub-atribute agregate cum ar fi durata medie de execuție şi durata maximă de răspuns. Aceste sub-atribute sunt cunoscute sub numele de *metrici de calitate.* În [**52**] o metrică de calitate este definită ca “o măsură cantitativă a gradului în care un element deține un atribut de calitate”. Astfel, atributul de calitate “disponibilitate” poate fi definit folosind următoarele metrici: “disponibilitate medie”, reprezentând timpul în care un serviciu este accesibil, definit in procente, şi “interval între erori ”, reprezentând intervalul de timp mediu dintre două eșuări ale serviciului dat [**53**].

Un vocabular QoS, constând dintr-o serie de atribute generale QoS și subsetul lor, este prezentat în Fig. 5.

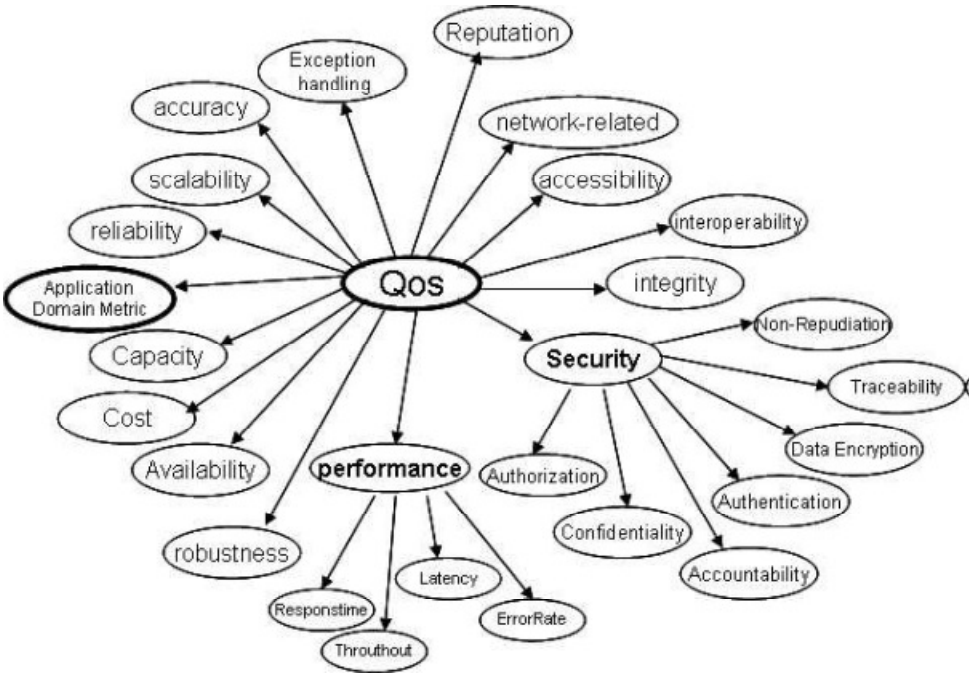


Fig. 5. Model de calitate - definit în [54**]**

Calitatea unui serviciu compus din alte servicii web este dată de calitatea fiecărui serviciu implicat în compunere, în parte. QoS reprezintă un factor important pentru asigurarea satisfacţiei clientului, însă atributele de calitate catalogate drept importante de un anumit client sunt specifice nevoilor sale şi sunt diferite de cele ale altor clienţi. Spre exemplu, pentru un client, atribute cum ar fi durata execuţiei şi securitatea pot fi cruciale, în timp ce pentru un alt client aspectele legate de preţ sunt mai importante decât durata execuţiei. Astfel, se impune necesitatea realizării unei metode de compunere a serviciilor web ţinând cont de calitatea serviciilor cerută de un anumit client.

Atributele QoS ale unui serviciu nu pot fi definite utilizând descrierea WSDL a acestuia. Pentru definirea acestor atribute ale serviciului există o serie de propuneri pentru extinderea standardului WSDL sau a UDDI. O altă abordare propusă într-o serie de lucrări este realizarea unui modul *broker* care să gestioneze aceste informații. [**53**]

4.1 Abordări de reprezentare a preferinţelor

În prezent nu există un standard stabilit pentru descrierea proprietăţilor QoS. Deoarece descrierea proprietăţilor nefuncţionale este o direcţie de cercetare actuală, există o serie de abordări de reprezentare a atributelor QoS atât teoretice cât şi practice pe care le vom analiza în această secţiune. Abordările existente adresează standardele WSDL şi UDDI, oferind extensii ale acestora pentru a permite definirea calităţii serviciilor sau adresează latura semantică a serviciilor prin definirea de ontologii QoS.

4.1.1 Extensii WSDL pentru QoS

Pentru extinderea descrierii WSDL a unui serviciu cu scopul definirii atributelor QoS există mai multe propuneri. Aceste propuneri prezintă abordări diverse, majoritatea în formă de draft, viitorul în privința standardizării şi adoptării lor fiind incert. În prezent, nu există nici o abordare care să se detașeaze de celelalte în ceea ce privește acceptarea / adoptarea pe o scară largă.

În [**55**] este propusă folosirea adnotărilor (annotations) pentru descrierea atributelor WSDL. Adnotările sunt folosite în WSDL în principal pentru documentație.

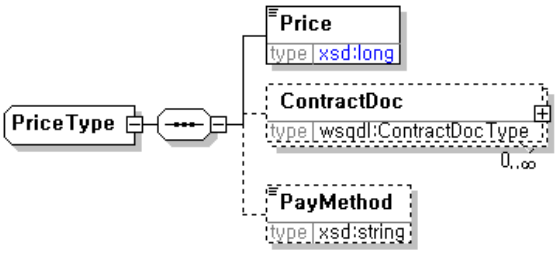
SAWSDL (Semantic Annotation for WSDL and XML Schema) este o recomandare W3C și reprezintă o extensie a WSDL care oferă adnotări semantice folosind ontologiile. SAWSDL nu specifică un limbaj pentru reprezentarea modelelor semantice (ontologiilor), dar oferă un mecanism prin care concepte ale modelelor semantice pot fi referite din cadrul documentului WSDL folosind adnotări [**24**].

O altă extensie WSDL este realizată folosind tehnici de generare bazate pe modele (model driven arhitecture). În [**56**] se prezintă un metamodel WSDL care este transformat într-un metamodel Q-WSDL (QoS-enabled WSDL). Extensia Q-WSDL propusă poate fi folosită pentru a specifica atributele QoS şi permite maparea automată a documentelor WSDL la Q-WSDL şi de la UML la servicii Q-WSDL. În aceasta variantă extinsă a WSDL pot fi definite atribute de calitate ale întregului serviciu, precum şi atribute de calitate care se referă doar la anumite operații ale serviciului.

Bazat pe Q-WSDL, lucrarea [**56**] prezintă o metodă bazată pe modelare folosită pentru descrierea si predicția automată a QoS a unei compuneri de servicii specificate folosind BPEL. Metoda propusă adresează atributul QoS “fiabilitate”, folosind posibilitatea oferită de Q-WSDL de a face adnotări legate de fiabilitate în modelul UML al compoziţiei bazat pe BPEL. Modelul UML este folosit apoi pentru a prezice şi descrie fiabilitatea compunerii specificate. În lucrarea mai sus menţionată, metoda propusă este prezentată şi din punct de vedere practic prin intermediul unei aplicații oferite ca exemplu.

WSQDL (Web Service Quality Description Language) [**57**] este o extensie a WSDL oferită de grupul OASIS. Această specificație oferă posibilitatea descrierii formale a atributelor de calitate ale serviciilor. Atributele de calitate sunt definite folosind o metodă de evaluare şi un factor de evaluare şi conţin valoarea cantitativă sau calitativă a atributului evaluat. Această descriere se face într-un format standardizat, folosind atribute XML definite de W3C şi care sunt validate folosind XML Schema.

Pentru a exemplifica folosirea WSQDL considerăm definirea costului unui serviciu web. Pentru a exprima atributul de calitate reprezentat de costul serviciului trebuie definite elemente cum ar fi preţul serviciului, modalitatea de plată, contractul de folosire al serviciului dintre distribuitorul de servicii web şi clientul serviciului.



Exprimat sub forma unei XML Schema aceste elemente au forma:

```
<xsd:complexType name="PriceType">
                <xsd:sequence>
                        <xsd:element name="Price">
                                <xsd:complexType>
                                        <xsd:simpleContent>
                                                <xsd:extension base="xsd:long">
                                                        <xsd:attribute name="unit" type="xsd:string"
use="required"/>
                                                </xsd:extension>
                                        </xsd:simpleContent>
                                </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="ContractDoc" type="wsqdl:ContractDocType" minOccurs="0"
maxOccurs="unbounded"/>
                        <xsd:element name="PayMethod" type="xsd:string" minOccurs="0"/>
                </xsd:sequence>
</xsd:complexType>
```

O descriere conform schemei XML de mai sus a costului serviciului are forma:

```
<BizValueFactor>
                <ServiceCost>
                        <ServicePrice>
                                <price unit="won">100000</price>
                                <ContractDoc>
                                        <ContractDocNumber>CN20034324</ContractDocNumber>
                                </ContractDoc>
                                <PayMethod>quaterly</PayMethod>
                        </ServicePrice>
                </ServiceCost>
        </BizValueFactor>
```

WSQDL are la bază modelul de calitate WSQM (Web Service Quality Model) dezvoltat de OASIS, care este folosit pentru configurarea calității serviciului web [**57**]. WSQM defineşte un model conceptual alcătuit din trei componente: factori de calitate, responsabili cu calitatea şi activități pentru asigurarea calității, precum şi interacţiunile dintre aceste componente.

Factorul de calitate specifică o clasificare / criteriu de evaluare a calității serviciului web. Fiecare factor de calitate reprezintă o componentă fundamentală. Responsabilii de calitate definesc constrângerile și responsabilităţile unei persoane / organizații care foloseşte serviciul respectiv. Activităţile oferă un cadru pentru specificarea acțiunilor efectuate de responsabilii de calitate în scopul menținerii calității serviciului. Responsabilii de calitate interacţionează cu serviciul la orice nivel, spre exemplu: contractare, monitorizare, notificare.

WSQM are meritul de a fi prima abordare care oferă standardizarea caracteristicilor de calitate a serviciilor. Modelul este însă la nivel de draft, la fel ca și specificația WSQDL, viitorul şi abordarea practică fiind incerte.

SQF (Web Service Quality Factor) definește un grup de caracteristici folosite pentru reprezentarea şi evaluarea calității serviciilor. Calitatea serviciilor este definită atât din punctul de vedere al serviciului, cât şi din punctul de vedere software (funcționalitatea implementată). WSQF împarte atributele de calitate în trei grupuri, în funcție de modul în care sunt folosite din punctul de vedere al utilizatorului serviciului: valoare de afaceri a serviciului, atribute ale nivelului serviciului (performanța, stabilitate), informaţii legate de sistemul informatic oferit. Valoarea de afaceri a serviciului este folosită în momentul în care un serviciu este selectat de un utilizator.

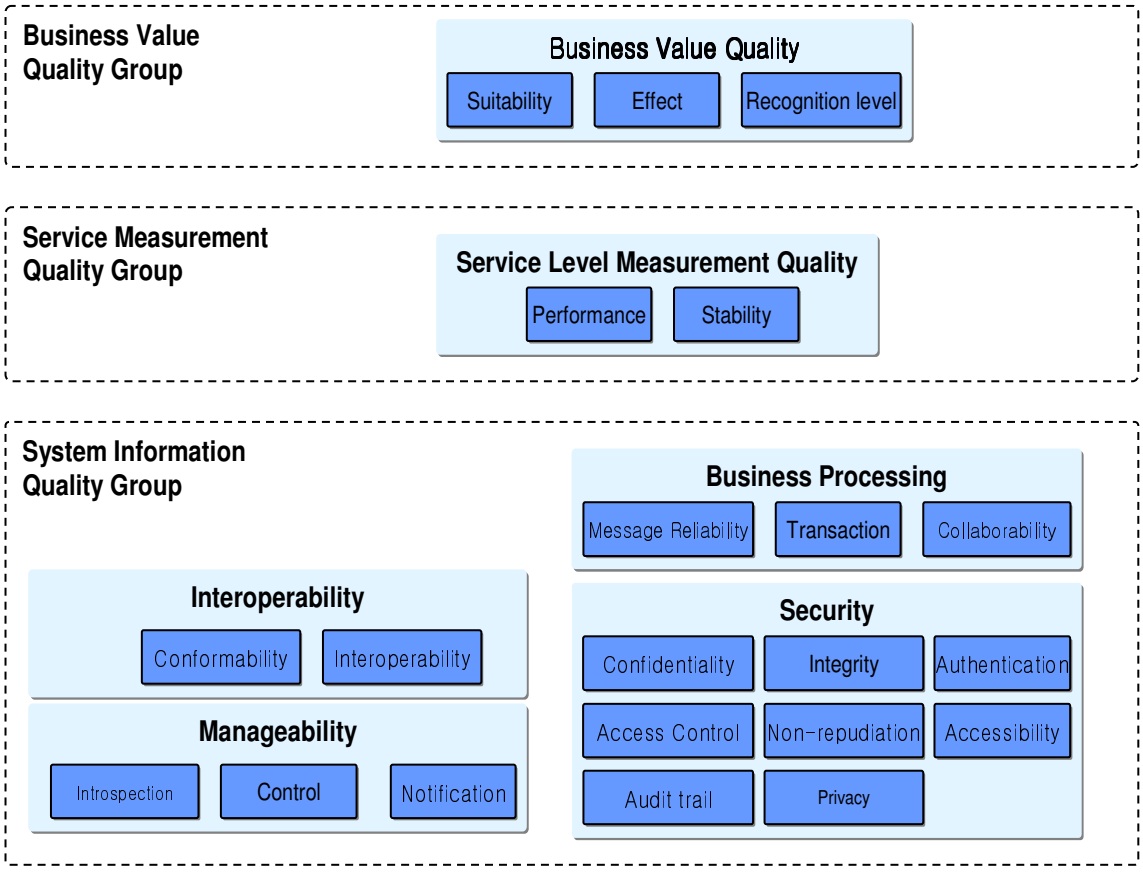


Fig. 6. Structura atributelor de calitate in WSQF

WSQDL organizează atributele de calitate în diverse categorii în funcție de natura lor şi de modul în care pot fi definite. În Fig. 7 este prezentată această clasificare.

WSQDL defineşte modul în care un serviciu web poate fi reprezentat în XML ca un tip complex ce conține factorii de calitate structuraţi conform clasificării din Fig. 7.

Exemplul de mai jos prezintă o posibilă descriere în WSQDL a costului unui serviciu:

```
<QualityFactor>
        <MeasureFactor>
                <ResponseTime>
                <EnvVariables>
                        <Variable>
                                <VarName>number of CPU</VarName>
                                <VarValue>4</VarValue>
                        </Variable>
                </EnvVariables>
                <MetricValue>
                        <Range>0.12-0.17</Range>
                        <Type>float</Type>
                        <Unit>second</Unit>
                        </MetricValue>
                </ResponseTime>
        </MeasureFactor>
        <BizValueFactor>
                <ServiceCost>
                        <ServicePrice>
                                <Price unit="won">100000</Price>
                        <ContractDoc>
                                <ContractDocNumber>CN20034324</ContractDocNumber>
                        </ContractDoc>
                        <PayMethod>quaterly</PayMethod>
                </ServicePrice>
                </ServiceCost>
        </BizValueFactor>

</ QualityFactor>
```

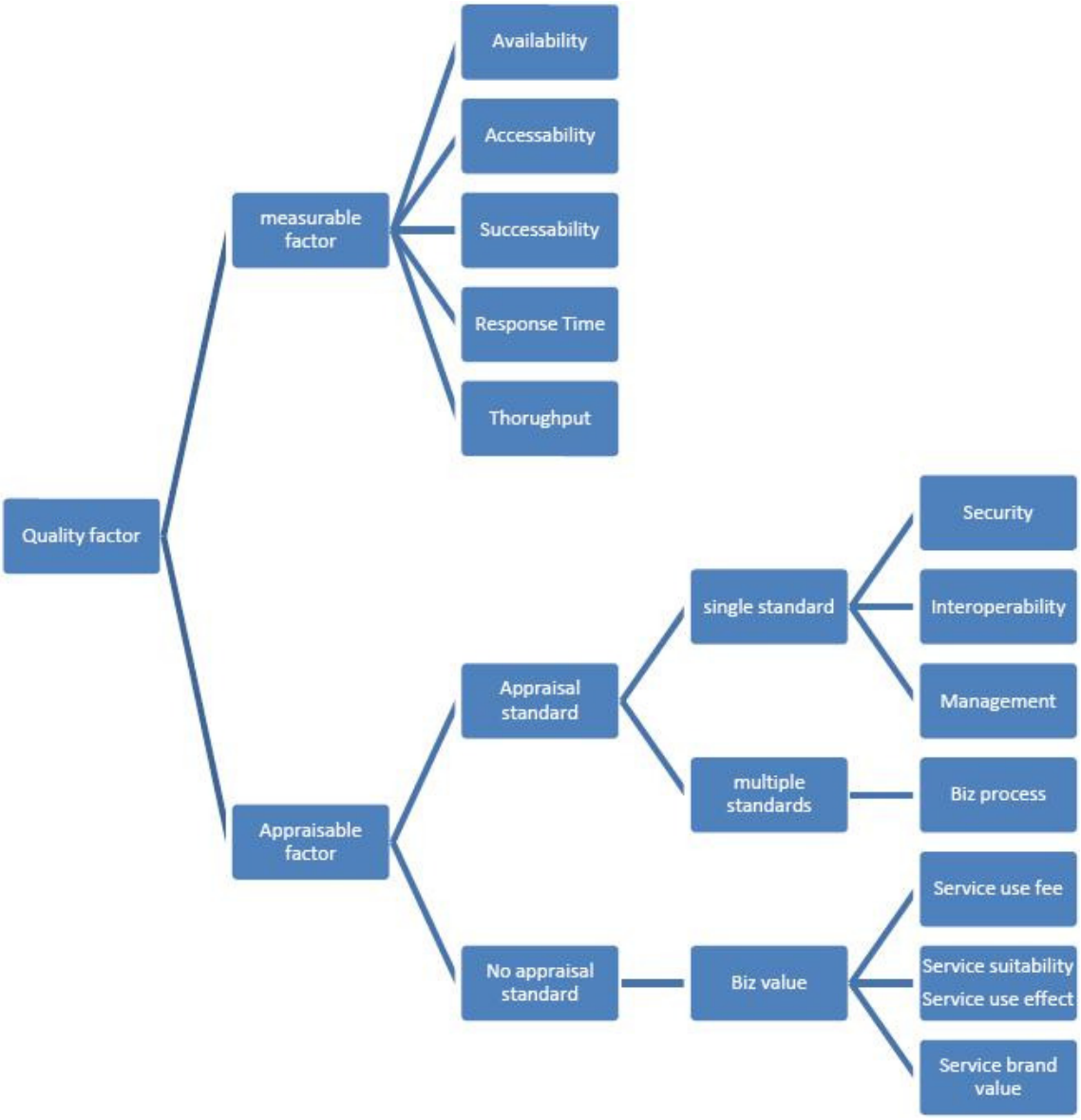


Fig. 7. Clasificarea WSQDL a atributelor de calitate.

4.1.2 Extensii UDDI pentru QoS

Registrul UDDI este prezentat detaliat în capitolul 2.5.2. În vederea administrării QoS a serviciilor folosite in BPMS (Business Process Management Systems), lucrarea [**58**] prezintă posibilitatea extinderii registrului UDDI pentru a include atributele QoS. Această abordare este bazată pe standardul WSPF (Web Services Policy Framework) şi pe UDDI tModels şi oferă o arhitectură pentru a integra această metoda în BPMS.

WSPF este o recomandare W3C [**59**] care oferă un model și sintaxa corespunzătoare pentru descrierea caracteristicilor/contractelor (policies) diferitelor entități dintr-un sistem bazat pe servicii web. Modelul definește o serie de termeni pentru descrierea unei palete largi de cerințe si capacități ale serviciului. Acești termeni pot fi folosiţi pentru a descrie atributele QoS ale serviciului. Specificarea caracteristicilor QoS oferă clienților posibilitatea de a diferenţia servicii similare ca funcționalitate, dar având caracteristici QoS diferite. Pentru aceasta, UDDI trebuie să fie extins în așa fel încât să permită integrarea caracteristicilor specificate folosind WSPF în UDDI. În cadrul acestei abordări, informația QoS stocată folosind caracteristicile WS-Policy ale WSPF este organizată în UDDI sub forma unei structuri separate, numite QoS Policy.

Pentru a folosi această extensie UDDI, furnizorii de servicii trebuie să specifice atributele de calitate QoS folosind WS-Policy. Un exemplu [**59**] de poliţă de securitate descris folosind specificaţia WS-Policy este următoarul:

```
(01) <wsp:Policy
     xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
     xmlns:wsp="http://www.w3.org/ns/ws-policy" >
(02)  <wsp:ExactlyOne>
(03)   <wsp:All>
(04)    <sp:SignedParts>
(05)     <sp:Body/>
(06)    </sp:SignedParts>
(07)   </wsp:All>
(08)   <wsp:All>
(09)    <sp:EncryptedParts>
(10)     <sp:Body/>
(11)    </sp:EncryptedParts>
(12)   </wsp:All>
(13)  </wsp:ExactlyOne>
(14) </wsp:Policy>
```

Liniile 3-7 definesc o alternativă a poliţei de securitate pentru semnarea digitală a mesajelor. Liniile 08-12 definesc o altă alternativă a poliţei de securitate referitoare la criptarea mesajului trimis. Operatorul <wsp:ExactlyOne> grupează poliţele exprimate în alternative şi defineşte faptul că exact una dintre alternativele exprimate trebuie sa fie folosite. În exemplul prezentat un mesaj trebuie să fie transmis fie criptat fie folosind o semnătură digitală.

În cadrul UDDI există structuri de date numite tModels. Un tModel oferă o reprezentare a unei anumite categorii de servicii în cadrul registrului UDDI. Fiecare serviciu înregistrat în UDDI face parte dintr-o anumită categorie, iar clienții caută în registru serviciile folosind aceste categorii. tModel oferă o abstracție a documentației tehnice a serviciului gestionând informația despre categoria / tipul serviciului şi făcând-o disponibilă bazei de date a registrului. O altă structură UDDI *bindingTemplate* gestionează informația diferitelor servicii din cadrul unei categorii [**53**].

Modelul informațional UDDI este alcătuit din descrieri XML ale structurilor de date. Acestea sunt stocate persistent în registrele UDDI. Modelul informațional extins UDDI include şi structura *qosPolicy*, care cuprinde descrierea QoS a serviciului. qosPolicy este bazată pe tModel.

UDDI defineşte un set de APIs pentru a standardiza comunicația din cadrul UDDI, iar varianta UDDI extinsă dezvoltă aceste APIs pentru a facilita manipularea datelor QoS. De obicei interacțiunea cu UDDI se face folosind un broker care implementează funcționalități pentru gestionarea datelor QoS.

Această abordare nu este standardizată, având in acest moment o contribuție mai mult teoretică în acest domeniu.

4.1.3 Extensii ale limbajului WS-Agreement

O altă metodă de definire a atributelor QoS ale serviciilor web se bazează pe specificaţia WS-Agreement, care defineşte un limbaj de descriere a capacităţilor serviciului oferit şi a cerinţelor clientului, într-un mod formal ce poate fi interpretat automat. Astfel, într-un mediu SOA, existenţa unui contract SLA (Service Level Agreement) între providerul serviciului web şi consumatorul acestuia oferă garanţia respectării unor convenţii referitoare la aspectele de calitate cerute. Un contract stabileşte unul sau mai multe obiective de calitate SLO (Service Level Objective) pentru ambele părţi. WS-Agreement este un standard mai avansat comparat cu WS-Policy sau WSLA, oferind posibilitatea definirii unor condiţii necesare pentru îndeplinirea obiectivelor SLO specificate, precum şi a penalităţilor pentru nerespectarea lor. Contractele stabilite între părţi au un format simetric, astfel încât fiecare dintre părţile implicate poate stabili condiţii legate de garanţia calităţii stabilite şi poate oferi un set de alternative de calitate. Autorii [**60**] oferă un framework de selecţie a serviciilor web SWAPS (Semantic WS-Agreement Partner Selection), bazat pe compatibilitatea QoS semantică dintre calitatea oferită şi cea cerută. Limbajul WS-Agreement este extins prin introducerea unei noi ontologii, bazate pe OWL-S, ce cuprinde atribute de calitate definite de schema XML a specificaţiei WS-Agreement. Deasemena abordarea integrează şi alte ontologii de calitate ce cuprind parametrii independenţi de domeniu cum ar fi OWL Time, care defineţe attribute temporare. Pentru a putea exprima obiective de calitate complexe şi a permite transformarea unor specificaţii la va corespundenţa semantică echivalentă sunt folosite reguli SWRL. Această abordarea nu tratează problema metricilor în care sunt

exprimate aspectele de calitate. Un alt dezavantaj este faptul ca fiecare obiectiv de calitate SLO se bazează pe un singur atribut de calitate.

4.1.4 Extensii ale limbajului WS-Policy

WS-Policy [**59**] este o specificaţie care permite serviciilor web definirea, într-un format XML, a caracteristicilor QoS oferite, a cerinţelor QoS. Caracteristicile serviciilor sunt definite sub forma unei singure, sau a unei colecţii de *policy assertion*. Limbajul permite combinarea unor *policy assertions* pentru a exprima caracteristici sau cerinţe complexe. Limbajul oferit de WS-Policy permite specificarea în acelasi mod a cerinţelor şi caracteristicilor oferite de un service, astfel încât compararea acestor parametrii se poate realize uşor. Recomandarea WS-Policy nu defineşte cum policies definite sunt asociate unui serviciului web. Specificaţia WS-PolicyAttachement oferă posibilitatea asocierii unei policy serviciului web corespunzător precum şi asocierea de descrieri WSDL şi UDDI unei policy.

Acest exemplu defineşte o WS-Policy care specifică regulile de securitate:

```
<wsp:Policy xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
        <sp:TransportBinding>
                <wsp:Policy>
                        <sp:AlgorithmSuite>
                <wsp:Policy>
                        <wsp:ExactlyOne>
                                <sp:Basic256Rsa15 />
                                <sp:TripleDesRsa15 />
                        </wsp:ExactlyOne>
                </wsp:Policy>
        </sp:AlgorithmSuite>
        <sp:TransportToken>
                <wsp:Policy>
                        <sp:HttpsToken RequireClientCertificate="false" />
                </wsp:Policy>
    </sp:TransportToken>
  </sp:TransportBinding>
</wsp:Policy>
```

4.1.5 Broker – sistem intermediar pentru gestionarea informaţiilor QoS

Pentru realizarea unei arhitecturi care oferă anumite constrângeri QoS în cadrul întregii infrastructuri, de-a lungul tuturor nivelelor implicate, în mai multe lucrări [**61**][**62**]este întâlnită următoarea abordare: realizarea unui *broker*, un sistem intermediar între client și serviciu. Acest sistem gestionează informațiile QoS ale diferitelor servicii, ia decizii în privința alegerii serviciilor folosite și poate negocia cu servere ce oferă servicii pentru obținerea anumitor cerințe de QoS. Un *QoS broker* poate fi implementat ca parte a clientului, a serverului, sau ca un modul independent. Brokerele sunt folosite de obicei pentru

descoperirea serviciilor ce îndeplinesc anumite condiții. Un broker oferă un registru conținând servicii web și parametrii lor QoS și deseori oferă un cadru pentru monitorizarea dinamică a atributelor de calitate.

Un broker specializat pe descoperirea serviciilor oferă un modul care implementează un algoritm pentru evaluarea şi priorizarea serviciilor în funcție de proprietățile QoS ale acestora.

4.1.6 Ontologii de preferinţe

În prezent nu există un standard stabilit pentru descrierea proprietăţilor QoS. Deoarece descrierea proprietăţilor nefuncţionale este o direcţie de cercetare actuală există o serie de abordări bazate pe ontologii QoS, atât teoretice cât şi practice şi le vom analiza pe cele mai importante dintre ele care au la bază standardul semantic OWL-S.

Ontologia OWL-S permite descrierea proprietăţilor QoS ca perechi nume-valoare. Această abordare nu permite însă specificarea de cerinţe QoS complexe din partea clientului şi nici prezentarea proprietăţilor QoS complexe în cazul unui serviciu oferit.

Abordările bazate pe ontologii QoS încearcă să ofere un vocabular comun pentru conceptele QoS pentru a facilita intercomunicarea cu privire la proprietăţile QoS între medii heterogene [**63**]. Nici una dintre propunerile independente existente nu este destul de matură pentru a putea deveni un standard în această direcţie.

DAML-QoS [**63**] este o ontologie care extinde DAML-S, predecesorul lui OWL-S şi este compatibilă cu OWL-S. Autorii oferă ontologii separate pentru metrici şi unităţi de măsură şi oferă un profil de bază QoS care conţine toate metricile de bază şi poate fi extins. Dezavantajul aceste abordări este că atributele QoS sunt descrise folosind constrângeri de cardinalitate care restrâng de fapt numărul de valori pe care un atribut în poate avea şi nu valoarea atributului în sine. Această abordare nu permite specificarea cerinţelor QoS şi a regulilor de compromis.

WSMO-QoS [**64**] este o ontologie de nivel înalt care extinde modelul de specificaţie semantic WSMO pentru a cuprinde preferinţele QoS. Această ontologie de preferinţe oferă un vocabular de atribute QoS generale, independente de domeniile specifice ale problemelor şi permite definirea unor metrici complexe ce permit calculul dinamic, ataşarea de unitaţi de masură şi coversii. Această ontologie oferă şi un framework de selecţie a serviciilor bazat pe atributele QoS. Dezavantajele acestei ontologii sunt asocierea unu-la-unu a atributelor QoS şi a metricilor acestora şi acceptarea numai a relaţiilor de egalitate în specificarea constrângerilor. De asemena această ontologie nu este disponibilă public pentru a putea fi extinsă, modificată sau utilizată.

onQoS-QL[**65**] este un limbaj de interogare pentru atributele QoS, bazat pe ontologia onQoS. Limbajul de interogare se bazează pe limbajul de interogare semantică SPARQL [**66**], care realizează selecţia serviciilor pe baza cerinţelor QoS şi realizează o ordonare a serviciilor obţinute în urma selecţiei realizate de SPARQL. Ontologia onQoS este o ontologie

extensibilă, similară ontologiei OWL-Q care va fi prezentată in detaliu tot în această secţiune. OWL-Q este însă disponibilă public şi prezentată mai pe larg . Limbajul onQoS-QL nu permite definirea unor cerințe complexe care să cuprindă mai multi parametri.

Ontologia QoSOnt [**67**] este o ontologie modulară care oferă o serie de attribute QoS de bază ce pot fi măsurate folosind diferite metrici şi oferă functionalităţi pentru conversia unităţilor de măsură. QosOnt oferă posibilitatea specificării unor cerinţe simple QoS oferind posibilitatea specificării de reguli bazate doar pe operatorii AND / OR. Ontologia nu oferă posibilitatea specificării unor reguli de compromis.

OWL-Q este o ontologie complexă, fiind compusă din mai multe faţete diferite care pot fi extinse sau îmbunătăţite separat. Ontologia oferă posibilitatea specificării simetrice a proprietăţilor QoS atât pentru cererea de servicii cât şi pentru descrierea calităţii serviciilor oferite. Algoritmul de comparare a cererii cu oferta transformă problema într-o problemă de satisfacere a constrângerilor (CSP) şi extinde abordările CSP actuale. OWL-Q adresează problema specificării priorităţilor asociate atributelor QoS din punctul de vedere al clientului oferind posibilitatea specificării unor ponderi atributelor cerute. Ponderea asociată unui atribut specifică importanţa acestuia şi permite ordonarea ofertelor în funcţie de cerinţele clientului. Noi susţinem că asocierea de ponderi cerinţelor QoS nu este o metodă suficientă pentru a exprima acurat cerinţele QoS.

În urma analizei ontologiilor QoS existente, considerăm ontologia OWL-Q ca fiind cea mai completă şi potrivită abordare existentă pentru a fi extinsă astfel încât să ofere posibilitatea specificării unor constrângeri şi a unor reguli de compromis QoS complexe care sa reflecte cerinţele clientului. În cadrul extensiei oferite de noi voi detalia dezavantajele metodei folosite de OWL-Q de a asocia ponderi atributelor QoS şi voi demonstra că această abordare nu este suficientă pentru a exprima în mod exact cerinţele de calitate ale clientului.

5. Compunerea serviciilor web în funcţie de parametrii QoS

5.1 Optimizarea compunerii serviciilor web in funcție de QoS

În cadrul arhitecturilor SOA, în cazul în care cerinţele unei aplicaţii nu pot fi îndeplinite de către un singur serviciu web, mai multe servicii şi procese existente pe Internet sunt interconectate pentru a îndeplini cerinţele clientului. Astfel, fiecare scenariu al unei aplicaţii este împărţit în sarcini ce pot fi realizate de către servicii atomice deja existente. Cu toate că în prezent există numeroase abordări atât practice cât şi teoretice în această direcţie, compunerea serviciilor web rămâne o problemă de actualitate complexă deoarece nici una dintre abordarile existente nu s-a impus ca standard. Complexitatea este cauzată în principal de numărul mare de servicii existente ce trebuie analizate şi a eterogenităţii acestora. De asemenea, aplicaţia finală trebuie să respecte criteriile de calitate cerute de client.

În cadrul compunerii de servicii se remarcă sistemele de tip workflow (Workflow Management Systems). În prezent există sisteme de tip workflow dinamice şi flexibile ce oferă posibilitatea selecţiei automate a serviciilor concrete ce pot implementa serviciile abstracte ce definesc un proces, din punct de vedere al funcţionalităţii oferite. Aceste sisteme trebuie extinse pentru a ţine cont şi de atributele de calitate ale serviciilor alese. Criteriile QoS au constituit dintodeauna o preocupare majoră în cadrul aplicaţiilor de timp real şi a reţelelor. Abia mai târziu însă a început să fie abordată şi problema integrării QoS în cadrul workflow-urilor [**68**].

În general, pentru fiecare serviciu abstract din cadrul unui proces există o serie de servicii concrete care implementează funcţionalitatea dorită. Un sistem de compunere automată a serviciilor trebuie să selecteze câte un serviciu concret din lista de servicii candidat asociată fiecărui serviciu abstract, astfel încât serviciul compus obţinut să aibă calitatea optimă. În cazul selecției serviciilor pe baza a mai mult de doi parametri de calitate diferiţi, avem de a face cu o problemă de optimizare multicriterială. Astfel de probleme sunt NP-complexe, iar pentru rezolvarea lor sunt folosiți cel mai adesea algoritmi euristici. În secţiunea 6.5 vom trece în revistă cele mai întâlnite astfel de abordări.

Pentru a putea optimiza selecţia serviciilor concrete care împreună oferă o compunere de servicii conformă criteriilor clientului, trebuie găsită o metodă de calcul a valorii QoS a serviciului compus. Pentru rezolvarea acestei probleme există două abordări principale. Cea dintâi constă în execuţia unor simulări pentru determinarea valorii QoS a serviciului compus. O astfel de abordare este descrisă în [**69**]. Cea de a doua abordare, pe care o vom prezenta pe larg în secţiunea următoare, constă în calcularea valorilor QoS agregate pe baza unor formule matematice.

5.2 Agregarea atributelor QoS

Agregarea atributelor QoS este necesară pentru a calcula QoS global al unei compuneri de servicii plecând de la atributele QoS ale serviciilor participante la compunere. Estimarea valorii globale QoS este foarte importantă pentru clienții unei compuneri de servicii. Pentru a determina valoarea globală QoS este nevoie ca toate atributele QoS ale serviciilor implicate să fie descrise folosind un limbaj comun. De obicei însă, descrierile QoS ale diferitelor servicii sunt eterogene, fapt ce îngreunează estimarea unei valori globale. Atributele QoS au un format diferit, în funcție de natura mărimii considerate (spre ex. numeric sau procent în cazul unei mărimi statistice, sau text).

Un număr mare de contribuții ştiințifice adresează această problemă și încearcă stabilirea unui set de funcții matematice pentru a putea calcula valoarea globală QoS a unei compuneri [**53**].

O serie de lucrări [**68**][**70**] abordează calcularea valorii QoS agregate pe baza unui scenariu, pentru un set de parametri predefiniți cum ar fi timpul de răspuns sau costurile. În aceste cazuri, modelul de compoziție nu definește structuri generale, iar scenariile sunt limitate pentru exemplele descrise. De obicei există o serie de atributele de calitate dependente de domeniul aplicaţiei, de aceea este necesară posibilitatea generalizarii calculului valorii QoS.

În [**71**] autorii realizează o ontologie pentru agregarea atributelor QoS, ontologie ce poate fi folosită pentru a determina agregarea corectă a parametrilor QoS adnotați. Ontologia definește o clasificare a parametrilor, un set de posibile forme de agregare și maparea acestor tipuri de agregare pe modele din cadrul gestionării workflow-urilor.

Pentru a găsi un formalism care să descrie agregarea QoS, trebuie analizate elementele componente: atributele şi variantele de compunere posibile, formalismul determinat fiind o funcție de agregare a fiecărei combinații de parametri și variante de compunere.

Lucrarea [**72**] definește conceptele de bază ce stau la baza unei compuneri și analizează şabloanele de compunere existente în workflow-uri. Sunt stabilite ca fiind de bază șapte şabloane de compunere: secvență, iterație, XOR-XOR, AND-AND, AND-DISC, OR-OR și OR-DISC. Plecând de la ideea de bază că orice compunere de servicii web poate fi prezentată ca fiind o combinație a acestor şabloane și folosind funcțiile care calculează valoarea QoS pentru fiecare dintre ele, se poate calcula valoarea QoS agregată. Fluxul activităților din cadrul compunerii de servicii este modelat ca un graf, iar acest graf este transformat într-un graf de şabloane de compunere definite anterior.

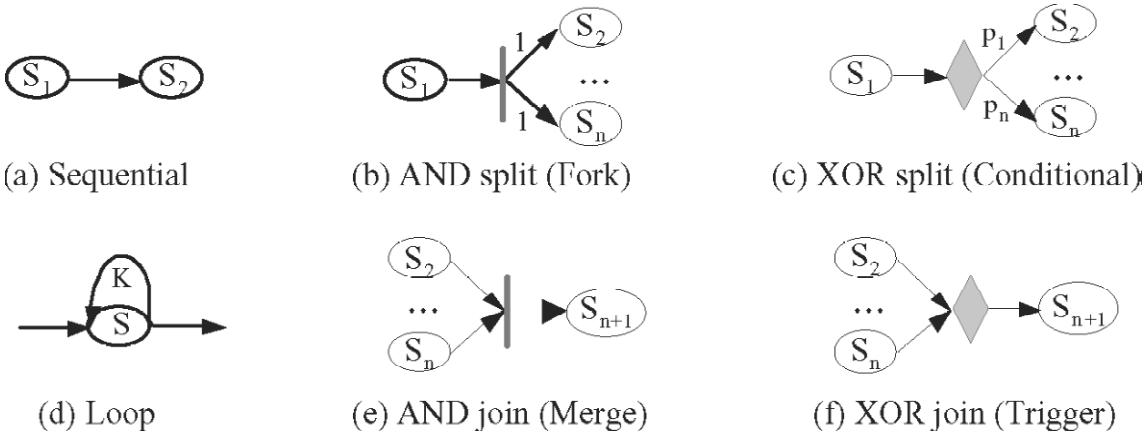


Fig. 8 Şabloane de compunere a serviciilor web

Pentru un şablon de tipul XOR, fiecărei posibilităţi de execuţie îi este asociată o probabilitate specificată de către client, iar valoarea finală a atributului pentru această secvenţă este calculată folosind această probabilitate.

Pentru o secvenţă de tip Loop este necesară definirea unui probabilităţi k de execuţie a operaţiei astfel încât calculul final QoS este produsul dintre probabilitatea k şi valoarea atributului pentru o operaţie.

În funcţie de natura unui atribut QoS, calculul valorii QoS totale se face folosind diferite calcule matematice. Spre exemplu, în cazul unei secvenţe, un atribut de tipul cost va fi calculat adunând costurile tuturor serviciilor implicate. De asemenea timpul total de răspuns va fi calculat ca fiind suma timpilor individuali de răspuns. Disponibilitatea unui sistem va fi calculată pentru o secvenţă ca fiind produsul disponibilităţiilor sistemelor implicate, la fel ca şi coeficientul de siguranţă al unui sistem. Modul în care se calculează valoarea totală pentru un anumit atribut QoS trebuie sa fie specificat de client, acesta fiind cel care cunoaşte cel mai bine funcţionarea practică a aplicaţiei sale. Pentru anumite atribute de bază, cum ar fi cele specificate mai sus, o aplicaţie ar putea sugera o metodă de calcul, dar pentru atributele specifice domeniului clientului funcţia de calcul trebuie să fie în mod obligatoriu specificată de client.

Lucrarea [**73**] defineşte următoarea tabelă de calcul a valorilor agregate QoS pentru atributele QoS frecvent întâlnite. Formulele de agregare au la bază metoda de calcul a valorilor QoS agregate specificate de autorii [**74**]. Aceasta este cea mai des întilnită abordare de calcul a valorilor agregate, fiind la baza multor abordări din cadrul direcţiei compunerii serviciilor web bazată pe QoS.

QoS Attr.	Sequence	Switch	Flow	Loop
Time (T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$Max\{T(t_i)_{i\in\{1\dots p\}}\}$	$k * T(t)$
Cost (C)	$\sum_{i=1}^{m} C(t_i)$	$\sum_{i=1}^{n} p_{ai} * C(t_i)$	$\sum_{i=1}^{p} C(t_i)$	$k * C(t)$
Availability (A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$
Reliability (R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$
Custom Attr. (F)	$f_S(F(t_i))$ $i \in \{1 \dots m\}$	$f_B((p_{ai}, F(t_i)))$ $i \in \{1 \dots n\}$	$f_F(F(t_i))$ $i \in \{1 \dots p\}$	$f_L(k, F(t))$

Tab. 1 Funcţii de agregare ale atributelor QoS [73]

O altă categorie de atribute QoS sunt atributele care au o valoare booleană şi pentru care valoarea agregată poate fi calculată folosind un operator XOR. O altă metodă de calcul este transformarea valorii booleane într-o valoare de tip întreg, unde valoarea booleană *false* este transformată în valoarea întreagă 0, iar valoarea booleană *true* este transformată în valoarea întreagă 1, astfel valoarea QoS totală putând fi calculată ca fiind produsul valorilor existente. Din această categorie fac parte atributele de securitate, cum ar fi autentificarea, criptarea, confidenţialitatea. În cartea [**13**] autorii împart atributele QoS în trei categorii: atribute run-time, de business şi de securitate. Autorii folosesc termenul de „calitate a operaţiei“ pentru calitatea serviciului, ţinând cont de faptul ca un serviciu web oferă în general mai multe operaţii iar atributul de calitate se referă la o anumită operaţie definită.

Valoarea QoS agregată este calculată recursiv, prin agregarea atributelor QoS calculate la fiecare nivel de şabloanele de compunere. Această metodă ar putea fi folosită pentru a integra calculul valorilor QoS agregate în cadrul unei aplicații ce modelează o compunere de servicii web [**72**].

5.3 Calculul valorii QoS agregate în cazul serviciilor compuse

Abordările existente pentru estimarea valorii QoS agregate în cazul serviciilor compuse propun metode care diferă prin restricţiile impuse tipologiei compunerii.

Autorii lucrării [**75**] analizează diferite soluţii comerciale de management a workflowurilor şi diferenţele dintre limbajele şi restricţiile impuse de acestea. Lucrarea analizează modul în care restricţiile sintactice impuse de un limbaj afectează expresivitatea acestuia.

Unele dintre soluţiile comerciale de management al workflowurilor cum ar fi SAP R/3, Filenet Visual Workflow permit doar definirea de workflowuri structurate. Deasemenea diagramele de activitate UML permit doar specificarea de workflowuri structurate. Un workflow structurat este considerat ca fiind cel mai restrictiv tip de workflow. Designerul procesului business este pus în situaţia de a specifica workflowul ţinând cont de restricţiile oferite de produsul folosit sau de a specifica în mod liber şi a converti apoi workflowul rezultat la o formă structurată.

Majoritatea metodelor de calcul a valorii QoS agregate se limitează la modele de orchestrare care sunt reprezentate de workflow-uri structurate. Autorii [**75**] arată ca transformarea unui workflow la o formă structurată creşte complexitatea acestuia prin necesitatea introducerii de variabile auxiliare şi de dependinţe între deciziile existente. Anumite workflowuri nu pot fi transformate, nici prin transformări complexe, la o forma structurată. În practică worklowurile structurate sunt mai puţin expresive şi mai puţin potrivite pentru a descrie procese business reale. Astfel pentru a descrierea acurată a proceselor business reale sunt folosite workflowuri arbitrare, fapt care îngreunează calculul valorii QoS agregate.

Yang et al. [**6**] au introdus o metodă care depăşeşte aceste restricţii, oferind posibilitatea de calculare a valorii QoS agregate pentru workflowuri nestructurate.

Această metodă primeşte ca parametri de intrare un model de orchestrare şi pentru fiecare activitate din model un serviciu concret correspondent. Modelul de orchestrare este un graf orientat care ataşează probabilităţi de execuţie nodurilor componente. Suma probabilităţilor de pe muchiile ce pornesc dintr-o poartă XOR este 1. Toate celelalte muchii au asociată probabilitatea 1.

Modelele de orchestrare sunt decompuse în *componente de orchestrare* care sunt reprezentate de subgrafuri cu un singur punct de intrare şi un singur punct de ieşire. Valoarea QoS agregată este calculată folosind o metodă bottom-up pentru fiecare componentă de orchestrare.

Modelele de orchestrare structurate, în care fiecare poartă SPLIT are o poartă JOIN corespondentă, pot fi uşor analizate. În funcţie de tipul parametrului QoS există diferite formule de agregare,care pot fi împărţite în trei categorii:

1. **Cale critică** - Valoarea QoS agregată este determinată de calea critică a modelului de orchestrare. Exemple: *timpul de execuţie*, pentru care se foloseşte calea critică cea mai lungă, sau *toleranţa la defecte*, pentru care se foloseşte calea critică cea mai scurtă.
2. **Aditiv** - Valoarea QoS agregată este suma ponderată a valorilor QoS ale serviciilor componente, folosind ponderi proporţionale cu frecvenţa cu care fiecare serviciu este invocat. Exemplu: *costul*.
3. **Multiplicativ** - Valoarea QoS agregată este produsul valorilor QoS ale serviciilor componente, aplicând fiecărei valori QoS un exponent proporţional cu frecvenţa cu care acesta este invocat. Exemplu: *fiabilitatea*.

Un pas premergător agregării QoS este folosirea tehnicii de structurare introdusă în [**76**] pentru a transforma un model de orchestrare nestructurat într-un model de orchestrare maxim-structurat.

Componentele care în urma folosirii acestei metode ramân ireductibile se numesc *componente rigide* şi sunt de două tipuri: *grafuri orientate aciclice* (DAG - Acyclic Graphs) şi

bucle ireductibile de tipul *intrări-multiple - ieşiri-multiple* (MEME - multiple-entry, multiple-exit).

Autorii lucrării [**6**] oferă un algoritm care transformă componentele DAG ireductibile în componente de decizie echivalente. Acesta este bazat pe noţiunea de *traseu* (*run)*, care reprezintă un subgraf al componentei DAG care conţine mulţimea tuturor muchiilor traversate într-o posibilă execuţie a componentei. Algoritmul găseşte toate *traseele* posibile şi calculează pentru fiecare dintre ele probabilitatea ca o execuţie a componentei să urmeze acel traseu. O componentă echivalentă, dar structurată este construită introducând o poartă SPLIT-XOR care are câte o ramură pentru fiecare traseu, cu probabilitatea corespunzătoare traseului respectiv.

Buclele MEME ireductibile pot fi transformate în componente rigide echivalente, unde concurenţa este complet încapsulată în subcomponente. Pentru aceste componente echivalente se va calcula, folosind metode standard, de câte ori un nod din cadrul buclei MEME va fi vizitat. Astfel se poate calcula valoarea QoS a componentei ireductibile folosind formulele de agregare caracteristice categoriei atributului QoS.

Această abordare este foarte utilă pentru că oferă flexibilitatea definirii de workflowuri arbitrare şi permite în acelasi timp calcularea calitaţii agregate a serviciilor.

Importanţa calculării automate a valorii globale QoS a unui sistem este dată atât de dorinţa de a oferi o soluţie de compunere iniţială optimă, cât şi de a putea recalcula o noua compunere de servicii în cazul în care unul dintre serviciile componente devine indisponibil, ţinând cont de caracterul imprevizibil al sistemelor reale.

6. Evaluarea bazată pe criterii multiple

În prezent, există o ofertă foarte mare de servicii web care îndeplinesc funcționalități identice sau similare. Designerul unei aplicații web se confruntă cu problema complicată a alegerii serviciului web care corespunde cel mai bine criteriilor unui utilizator având la dispoziție o multitudine de alternative. Pentru rezolvarea acestei probleme uzuale un rol esențial îl au analiza și compararea caracteristicilor nefuncționale QoS (Quality of Service) ale serviciilor web. Pentru selecţia de servicii web bazată pe proprietăţile nefuncţionale sunt interesante tehnicile de optimizare bazate pe criterii multiple şi evaluarea multi-criterială. Astfel, pentru fiecare atribut QoS putem asocia o funcţie obiectiv.

În practică există în mod curent situații în care trebuie urmărite mai multe obiective, de obicei conflictuale. Un exemplu ar fi cazul des întâlnit în care se doreşte un serviciu web care să ofere un timp de răspuns mic, indisponibilitate a sistemului redusă şi în acelaşi timp un preţ mic.

În această situaţie este posibil să nu existe o soluţie care să optimizeze toate criteriile aşa că în acest caz soluţia cea mai bună este soluţia care conţine cel mai bun compromis. Considerăm următorul exemplu, care va fi folosit şi în secţiunile teoretice următoare, pentru a ilustra diverse concepte legate de evaluarea multicriterială.

Atribut QoS	WS1	WS2	WS3	WS4
Preţ	30	32	30	31
Timp de răspuns	7	8	6	3
Indisponibilitate	1%	2%	2%	1%

Tab. 2 Exemplu servicii web candidat

Aşa cum se observă la o primă analiză a exemplului din tabela de mai sus, toate atributele web ale serviciului WS2 au valori inferioare celorlalte servicii, în timp ce între celelalte servicii nu există o relaţie de superioritate clară. Clientul trebuie să aleagă cea mai bună soluţie de compromis, dintre serviciile WS1, WS3 şi WS4, pe bază preferinţelor sale. O altă problemă a evaluărilor multicriteriale este dată de spaţiul de căutare care poate fi foarte larg şi complex. Astfel, pentru rezolvarea unei probleme multicriteriale trebuie ţinut cont de ambele provocări.

În cadrul mecanismelor de luare a deciziilor bazate pe criterii multiple există numeroase modele matematice care pot fi folosite pentru a realiza o astfel de comparație. Astfel, așa cum este precizat în [**77**] alegerea unui model pentru reprezentarea unei probleme de decizie bazată pe criterii multiple este, în sine, o decizie bazată pe criterii multiple.

Pentru a alege soluția optimă din cadrul unei mulțimi de alternative trebuie să definim noțiunea de „optim” în cadrul domeniului de evaluare bazată pe criterii multiple. Noțiunea de optim general acceptată în prezent a fost propusă încă din anul 1881 de către Francis Ysidro Edgeworth și apoi generalizată de către Vilfredo Pareto în anul 1896. Termenul sub care este cunoscută şi folosită această noțiune este *Pareto optimum* [**78**].

6.1 Problema generală de optimizare multi-obiectiv

O problemă de optimizare multi-obiectiv poate fi descrisă în termeni matematici în felul următor:

Se dau:

- funcţiile obiectiv $f_1, f_{2,\dots} f_k: \mathbb{R}^n \to \mathbb{R}$, unde $k > 1$ reprezintă numărul de obiective;
- funcţiile de constrângere de inegalitate $g_1, g_2 \dots g_p: \mathbb{R}^n \to \mathbb{R}$;
- funcţiile de constrângere de egalitate $h_1, h_2 \dots h_q: \mathbb{R}^n \to \mathbb{R}$;

Se cere să se găsească $\overrightarrow{x^*} \in \mathbb{R}^n$, care satisface constrângerile:

- $g_i(\overrightarrow{x^*}) \leq 0$, $1 \leq i \leq p$
- $h_i(\overrightarrow{x^*}) = 0$, $1 \leq i \leq q$

şi care minimizează vectorul de funcţii $F(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \dots f_k(\vec{x}))$.

Argumentul $\vec{x} \in \mathbb{R}^n$ al funcţiilor obiectiv este denumit *vectorul de decizie* şi conţine valorile parametrilor de optimizare ai problemei date. O funcţie f_i reprezintă funcţia obiectiv pentru criteriul i. Astfel $f_i(\vec{x})$ reprezintă valoarea obiectivului *i* în punctul $\vec{x}$.

Mulţimea $S \subseteq \mathbb{R}^n$ care conţine toţi vectorii de decizie care satisfac constrângerile impuse de funcţiile g_i şi h_i poartă numele de *regiune fezabilă a problemei*. Ea poate fi referită şi ca fiind *setul fezabil*, *spaţiul de căutare* sau *spaţiul de soluţii* al unei probleme. Un vector $\overrightarrow{x^*} \in S$ poartă numele de *soluţie candidat*.

Mulţimea $Z \subseteq \mathbb{R}^k$, care reprezintă imaginea regiunii fezabile S este denumită *regiune obiectivă fezabilă*: $Z = \{\vec{z} \in \mathbb{R}^k \mid \exists\, \vec{x} \in S$, a.î. $F(\vec{x}) = \vec{z}\}$.

Soluţia unei probleme multi-obiectiv este considerată soluţia care optimizează simultan toate funcţiile obiectiv. Această soluţie ideală este numită şi *soluţie utopică*, deoarece de obicei o astfel de soluţie nu există sau nu aparţine regiunii fezabile S. Deoarece această soluţie ideală nu poate fi atinsă in general, se încearcă gasirea unei soluţii de compromis, care să fie cât mai aproape de vectorul ideal.

6.2 Pareto Optimum

6.2.1 Definiţie

Spunem că un vector de variabile de decizie $\overrightarrow{x^*} \in S$ este Pareto optimal dacă nu există un alt vector $\vec{x} \in S$ astfel încât $f_i(\vec{x}) \leq f_i(\overrightarrow{x^*})$ pentru orice $i = 1,\dots,k$ şi $f_j(\vec{x}) < f_j(\overrightarrow{x^*})$ pentru cel puţin un $j = 1,\dots,k$.

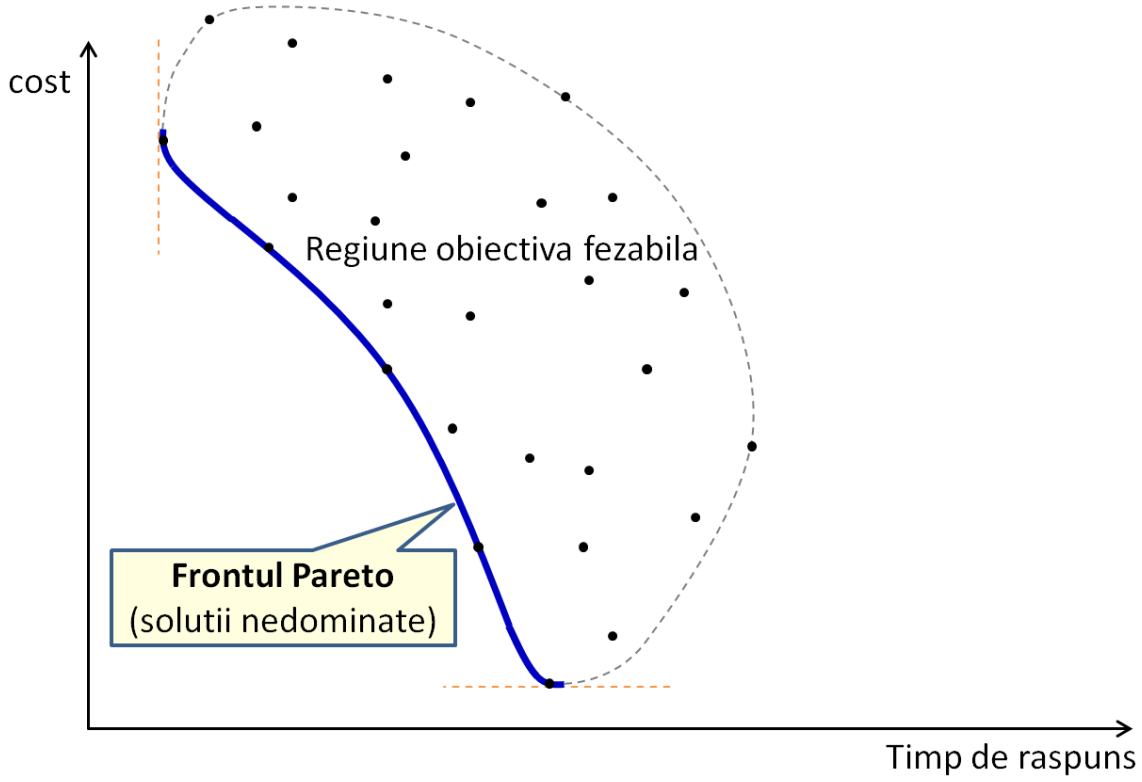
Conform definiţiei, un vector $\overrightarrow{x^*}$ din regiunea fezabilă este Pareto optimal dacă nu există un alt vector de variabile de decizie $\vec{x}$ care ar putea produce o îmbunătăţire a unui anumit criteriu fară a cauza simultan o înrăutăţire a unui alt criteriu. De obicei această abordare nu

oferă doar o singură soluţie eficientă, ci mai multe soluţii care împreună reprezintă setul de soluţii *Pareto optimal*, denumit şi *noninferior*. Vectorii $\overrightarrow{x^*}$ corespunzând soluţiilor din setul Pareto optimal se numesc *nedominați*.

Pentru exemplul din Tab. 2 soluţiile candidat WS1, WS3 şi WS4 se află pe frontul Pareto, în timp ce WS2 este o soluţie dominată.

În cadrul unei probleme de optimizare multi-obiectiv, setul Pareto al problemei este dat de setul tuturor soluțiilor optime Pareto – numit și *frontul Pareto.* Soluţiile cele mai bune pentru o problemă de optimizare multi-obiectiv sunt cele care formează setul optim Pareto, iar frontul Pareto conţine compromisul cel mai bun dintre obiective.

Considerăm cazul în care trebuie să selectăm cea mai bună soluţie ţinând cont de criteriile cost şi timp de răspuns şi definim următorul grafic, pe care marcăm regiunea obiectivă fezabilă (Z) şi frontul Pareto.



Frontul Pareto poate avea o formă concavă, convexă, non-convexă sau discontinuă asa cum se poate vedea în imaginea următoare. În cazul frontului concav, frontul este curbat în jos pentru soluţii mai bune, pentru frontul convex, frontul este curbat în sus pentru soluţii mai bune. În cadrul frontului non-convex există zone convexe ce alternează cu zone concave. Frontul Pareto cu forma discontinuă conţine porţiuni în care nu există soluţii optime, deoarece ar fi dominate de soluţii aflate în regiunea validă a frontului.

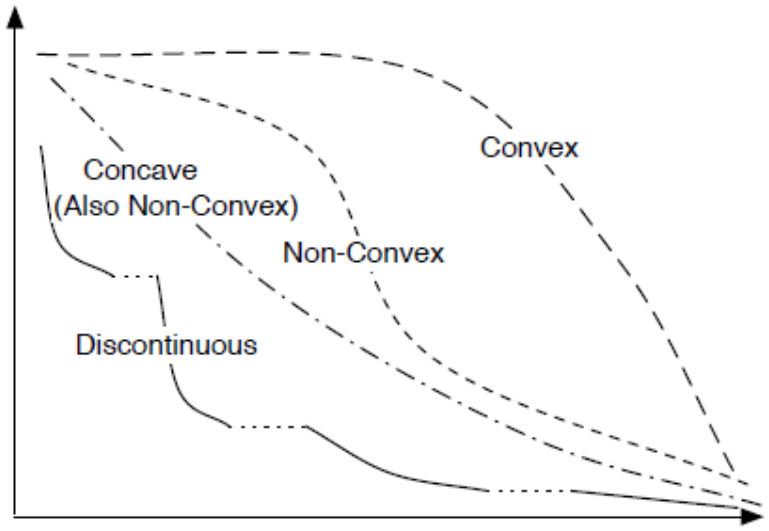


Fig. 9 Tipuri de front Pareto [79]

Dominare Pareto

Pentru oricare doi vectori de decizie a şi b, considerăm că

$a \succ b$ (a domină b) *dacă şi numai dacă* $f(a) > f(b)$
$a \geq b$ (*a domină* slab b) *dacă şi numai dacă* $f(a) \geq f(b)$
$a \sim b$ (a este indiferent faţă de b) *dacă şi numai dacă* $f(a) \ngeq f(b) \wedge f(b) \ngeq f(a)$
$a \leq b$ (a este slab dominat de b) *dacă şi numai dacă* $f(a) \leq f(b)$
$a \prec b$ (a este dominat de b) *dacă şi numai dacă* $f(a) < f(b)$

6.3 Luarea deciziei

Tehnicile de optimizare bazate pe criterii multiple se pot clasifica în funcţie de momentul în care se iau în considerare preferinţele în cadrul căutarii şi luării deciziei privind soluţia optimă aleasă, astfel:

- Tehnici în care preferinţele sunt enunţate *a-priori* presupun ca înainte de a efectua optimizarea preferinţele să fie analizate şi agregate într-o singură funcţie / un singur obiectiv de către factorul de decizie. Această abordare transformă o problemă multi-obiectiv într-o problemă cu un singur obiectiv.
- Tehnici în cadrul cărora preferinţele sunt enunţate în mod progresiv şi sunt integrate în procesul de căutare. Aceste tehnici sunt considerate a fi interactive iar preferinţele sunt aplicate pe măsura ce se efectuează căutarea. Factorul de decizie interacţionează cu programul de optimizare în timpul procesului de optimizare. În cadul acestor tehnici, pe parcurs sunt oferite seturi de soluţii noi, iar factorul de decizie hotărăşte cum sunt schimbate criteriile de căutare.
- Tehnici în cadrul cărora preferinţele sunt enunţate a-posteriori. Căutarea se face înainte de luarea unei decizii, algoritmul de optimizare oferă un set de soluţii

potrivite, din spaţiul Pareto optim, dintre care factorul de decizie alege soluţia optimă.

6.4 Metode de rezolvare

6.4.1 Metode tradiţionale

Din cadrul tehnicilor a-priori fac parte ordonarea lexicografică, ordonarea min-max, abordarea neliniară, abordarea bazată pe sumă ponderată, folosind agregarea obiectivelor. Aceste metode transformă problema multicriterială într-o problemă liniară obișnuită.

6.4.1.1 Metoda sumei ponderate

Abordarea bazată pe suma ponderată este des întâlnită în practică, fiind o metodă uşor de implementat. Fiecărei funcţii obiectiv îi este asociată o pondere w_i unde $\sum w_i = 1$.

$$y = f(x) = w_1 \cdot f_1(x) + w_2 \cdot f_2(x) + \ldots + w_k \cdot f_k(x))$$

Spre exemplu în cazul în care un client stabileşte că preţul serviciului are cea mai mare prioritate şi celelalte atribute, cum ar fi timp de raspuns si disponibilitate sunt la fel de importante punem alege următoarele valori ca şi ponderi: 0,5 pentru preţul serviciului, 0,25 pentru timp de răspuns şi 0,25 pentru disponibilitate.

Pentru exemplul prezentat în Tab. 2 obţinem următoarele valori pentru funcţia y de mai sus.

	WS1	WS2	WS3	WS4
Y	17	18,5	17	16,5

Astfel, conform calculului de mai sus serviciul WS4 este serviciul optim conform ponderilor alese de client.

Metoda are însă ca dezavantaj faptul că alegerea unei ponderi care sa îi fie asociată fiecărui criteriu în parte, mai ales în cazul evaluărilor cu un număr mare de parametri, este o sarcină dificilă. Metoda este a-priori deoarece clientul trebuie să ataşeze ponderi obiectivelor. De asemenea, majoritatea parametrilor QoS sunt parametri calitativi care pot fi cuantificaţi la parametri cantitativi prin asocierea unei ponderi. Ponderile asociate obiectivelor trebuie să fie relative la importanţa acestora şi să exprime în mod corect preferinţele utilizatorului. O analiză amplă a acestei abordări este prezentată în [**80**], unde autorii concluzionează că metoda sumei ponderate este în mod fundamental incapabilă să cuprindă preferinţe complexe. Un alt dezavantaj este faptul că nu oferă soluţii din zona concavă a frontului Pareto, aşa cum este ilustrat în figura următoare:

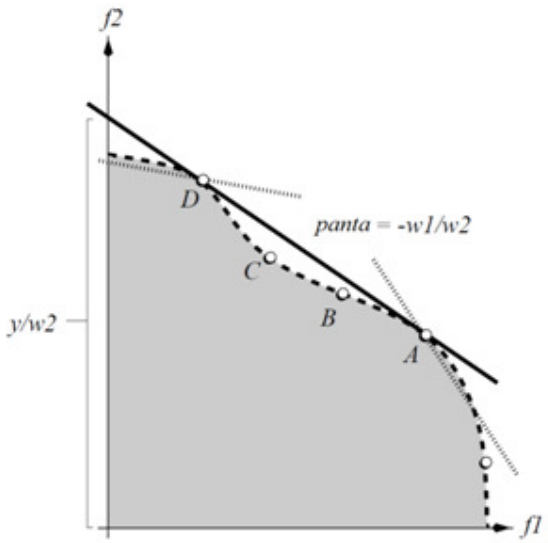


Fig. 10 Reprezentare grafică metoda sumei ponderate [81**]**

Pentru ponderile fixe $w1$ şi $w2$ soluţia *x* trebuie să obţină valoarea maximă pentru funcţia

$y = w1 \cdot f1(x) + w2 \cdot f2(x)$ Ecuaţia poate fi reformulată în felul următor

$$f2(x) = -\frac{w1}{w2} f1(x) + \frac{y}{w2}$$

Această funcţie defineşte o linie cu panta $-\frac{w1}{w2}$ ce oferă valori din spaţiul obiectiv $\frac{y}{w2}$.

Punctele A, B, C, şi D reprezintă soluţii aflate pe frontul Pareto. Aşa cum se poate vedea în reprezentarea grafică punctele B şi C aflate în regiunea convexă nu vor putea minimiza niciodată funcţia f, deci nu vor fi alese niciodată ca soluţii deşi aparţin frontului Pareto.

6.4.1.2 Metoda ε–constraints

În cadrul acestei metode este păstrat numai unul dintre obiective în timp ce celelalte sunt considerate având valori constante alese de client. Obiectivul variabil poate fi ales arbitrar. Formal metoda poate fi descrisă astfel:

$y = f(x) = fh(x)$ unde $ei(x) = fi(x) \geq \varepsilon i$, $(1 \leq i \leq k, \quad i \neq h)$ şi $x \in X_f$

Aşa cum se observă în Fig. 11, soluţiile alese pot fi şi din spaţiul concav Pareto (porţiunea dintre punctele D şi B).

Această metodă permite teoretic găsirea tuturor punctelor de pe frontul Pareto, dar nu specifică o metoda de a decide care dintre aceste puncte reprezintă soluţia optimă.

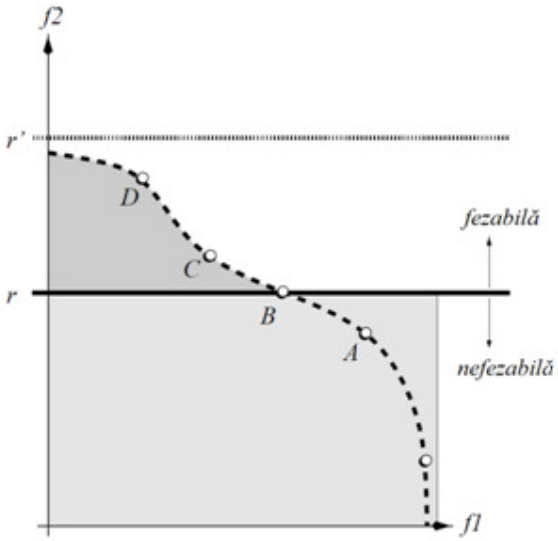


Fig. 11. Reprezentare grafică a metodei ε−constraints [**81**]

6.4.1.3 Ordonarea lexicografică

Ordonarea lexicografică este o altă metodă simplă pentru a modela un mecanism de decizie rațional. Euristicile lexicografice fac parte din mecanismele de decizie din cadrul unor domenii diferite de cercetare cum ar fi psihologia, economia sau marketingul, deoarece oferă o metodă simplă de a stabili compromisuri complexe între criteriile de alegere. Acest model de decizie este des întâlnit în viaţa de zi cu zi, deoarece este similar modului în care oamenii iau decizii în mod real. În cazul în care este folosit un altfel de model, de obicei acel model este translatat la un model lexicografic care poate fi apoi uşor comunicat verbal [**77**]. În cadrul modelului de decizie lexicografic se consideră un set de *n* criterii şi se aplică o ordine liniară asupra acestui set. Se consideră că o alternativă precede lexicografic o altă alternativă dacă are o valoare preferată pentru primul criteriu a cărui valoare diferă. În cadrul acestei metode preferințele sunt definite folosind o ordonare lexicală care conduce la o ordonare strictă. Dezavantajul major al acestei metode este faptul că este necompensatoare.

Pentru exemplul prezentat în Tab. 2 , considerând preţul ca fiind cel mai important criteriu alegem serviciile WS1 şi WS3 ca fiind cele mai bune şi pentru a le departaja, le comparăm conform următorului atribut, timpul de răspuns. Astfel serviciul WS3 este serviciul căştigător, având timpul de răspuns mai mic.

6.4.1.4 Ordonarea semilexicografică

O extensie a ordonării lexicografice este ordonarea semilexicografică. Aceasta permite stabilirea unor compromisuri în cazul în care o îmbunatățire majoră a unui criteriu poate compensa o depreciere minoră a criteriului principal. Această abordare este bazată pe teoria lui Tversky pentru ordonarea semilexicografică, în care se consideră că o alternativă *x* este mai bună decât o alternativă *y* în cazul în care criteriul cel mai important care stabileste această ordonare este mai mare pentru alternativa *x* față de alternativa *y* cu o valoare care depășeste un prag fix stabilit. Această metodă previne alegerea unei alternative ca fiind cea

mai bună în cazul în care aceasta este doar nesemnificativ mai bună pentru criteriul principal, dar considerabil mai proastă pentru celelalte criterii.

Pentru a ilustra această abordare cu un exemplu, putem considera cazul unei achiziţii imobiliare pentru care preţul este criteriul principal, iar dimensiunea locuinţei criteriul secundar. Totuşi, în cazul unei diferenţe mici de preţ, criteriul secundar se poate transforma în criteriu decisiv. Astfel, această regulă schimbă ordonarea strictă în care preţul era criteriul principal. De aceea, se poate considera că dimensiunea locuinţei este un factor compensator pentru diferenţa de preţ.

Tversky însuşi prezintă însă abordarea ca având dezavantajul de a fi restrictivă, fiind bazată pe un principiu necompensator, ce poate fi prea restrictiv în multe situaţii. În cazul unor preferinţe intranzitive, semiordonarea lexicografică nu poate trata caracterul intranzitiv obţinând o ordine totală, ci se obţine un rezultat ciclic.

Pentru exemplul prezentat în Tab. 2 , considerăm preţul ca fiind cel mai important criteriu, iar valoarea de prag fix este 3. Astfel o valoare mai mică de 3 a preţului nu este relevantă atât timp cât permite obţinerea unei valori considerabil mai bune pentru unul dintre celelalte două atribute. WS1 şi WS3 au cel mai bun preţ (30) dar WS4 are un preţ doar puţin mai mare (31) în schimb un timp de răspuns considerabil mai bun. De aceea, folosind metoda semilexicografică el va fi desemnat cel mai bun dintre serviciile web disponibile.

6.4.2 Algoritmi evolutivi de optimizare

O alternativă recentă la metodele clasice pentru rezolvarea problemelor multicriteriale este folosirea de algoritmi evolutivi. Algoritmii evolutivi folosesc populaţii ce oferă mai multe soluţii de compromis potenţiale, într-o singură rulare. De asemenea, algoritmii evolutivi pot lucra cu spaţii de căutare mari şi complexe.

Algoritmii evolutivi reprezintă o clasă de metode de optimizare stohastică ce simulează procesul natural de evoluţie. Există numeroase metodologii evolutive, de bază fiind algoritmii genetici, programarea şi strategiile evoluţioniste. Algoritmii au la bază o mulţime de soluţii candidat ce sunt modificate succesiv folosind două principii de bază ale evoluţiei: selecţia si variaţia.

Mecanismul de selecţie alege soluţiie cele mai bune, soluţiile care au probabilitatea cea mai mare să se reproducă. În cadrul algoritmilor evolutivi, pentru a alege soluţia cea mai bună este simulat un proces de selecţie stohastic în care fiecare soluţie se poate reproduce de un număr de ori, în funcţie de calitatea acesteia. Prin evaluarea indivizilor populaţiei şi atribuirea de valori de fitness scalare acestora se poate compara calitatea lor.

Variaţia, cel de-al doilea principiu de bază, simulează fenomenul natural de creare de noi indivizi folosind recombinări şi mutaţii.

Cu toate că principiile ce stau la baza algoritmilor evolutivi sunt simple, algoritmii oferă o metodă de căutare puternică [**81**]. Algoritmii evoluţionişti sunt în mod deosebit potriviţi

pentru rezolvarea problemelor de optimizare multicriterială datorită capacităţii lor de a genera multiple soluţii Pareto optimale într-o singură rulare si a posibilităţii de recombinare a acestor soluţii pentru a obţine soluţii superioare.

Există numeroşi algoritmi genetici multi-obiectiv, şi numeroase lucrări care compară algoritmii existenţi [**82**]. Printre cei mai cunoscuţi şi performanţi algoritmi genetici multi-obiectiv, folosiţi în numeroase aplicaţii practice amintim: Vector Evaluated Genetic Algorithm (VEGA) [**83**] urmat de Multi-Objective Genetic Algorithm (MOGA) [**84**], Weight-based Genetic Algorithm (WBGA) [**85**], Strength Pareto Evolutionary Algorithm (SPEA) [**86**]

Multi-objective Multi-State Genetic Algorithm (MOMS-GA) [**87**] reprezintă un algoritm genetic modificat pentru optimizarea generală a unui sistem, ţinând cont de disponibilitatea, fiabilitatea şi nivelele de performanţă ale componentelor sistemului. Algoritmul foloseşte transformata universală UMGF (universal moment generating function) din cadrul abordărilor probabilistice din [**88**] pentru a calcula fiabilitatea si disponibilitatea unui sistem. Algoritmul oferă soluţii Pareto dintre care clientul îşi poate alege soluţia preferată. Autorii folosesc scenarii de optimizare multicriterială pentru a demonstraza eficienţa şi performanţa algoritmului.

Bazat pe algoritmul Strength Pareto Evolutionary Algorithm (SPEA) [**86**], autorii lucrării [**89**] oferă un algoritm modificat SPEA2 şi o abordare multi-obiectiv practică testată pe un scenariu cu patru servicii abstracte, respectiv având serviciile concrete ataşate serviciilor abstracte. Metoda oferă rapid soluţii Pareto, bine distribuite pe frontul Pareto.

În practică metodele bazate pe algoritmi genetici trebuie adaptate la funcţiile obiectiv, la constrângerile şi codificarea problemei date, iar setul de soluţii Pareto găsite de algoritmii genetici trebuie minimizat pentru ca designerul aplicaţiei să poată alege cea mai bună variantă dintr-un set mic de soluţii optime. O astfel de variantă nu este eficientă pentru compunerea dinamică a serviciilor web, deoarece are nevoie de inverventie umană.

6.5 Abordări existente pentru probleme de optimizare multiobiectiv

În cadrul lucrării [**90**] autorii folosesc un model bazat pe semiordonarea lexicografică pentru rezolvarea unei probleme de decizie. Această lucrare adresează problema preferinţelor şi a compromisului dintre preferinţe folosind euristici lexicografice pentru stabilirea unei ordonări a preferinţelor.

Autorii lucrării [**91**] propun o metodă de rezolvare a problemei alegerii celui mai potrivit compromis de preferinţe oferind un model pentru descrierea preferinţelor QoS (atât criteriile obligatorii, cât şi cele opţionale) care este folosit atât de către client, cât şi de către furnizorul serviciului. Pentru compararea modelelor este folosit un mecanism de decizie fuzzy bazat de criterii multiple. Modelul folosit pentru descrierea proprietăţilor QoS este bazat pe modelul OMG UML QoS Framework astfel încât este nevoie de cunoştinţe complexe de UML pentru a descrie structura complexă a proprietăţilor QoS. Metoda de comparare folosită permite asocierea de ponderi preferinţelor specificate.

Autorii lucrării [**92**] propun o extensie a arhitecturii web service, prin adăugarea la registrul UDDI a unei componente numite Multi-Criteria Evaluation Component (MEC), ce realizează evaluarea multi criterială. Evaluarea se bazează pe o versiune web a sistemului de decizie IRIS (Interactive Robustness Analysis and Parameters Inference for multicriteria Sorting Problems) [**93**] care la rândul lui se bazează pe metoda de sortare ELECTRE TRI.

O metodă de rezolvare a problemei folosind algoritmi genetici este propusă de autorii lucrării [**73**] care codifică compunerea de servicii folosind un genom. Genomul este reprezentat de un şir de valori întregi, având o lungime egală cu numărul de servicii abstracte participante la compunere. Fiecare valoare din acest şir reprezintă indexul serviciului concret din cadrul listei de servicii care implementează funcţionalitatea serviciului abstract din cadrul şirului de servicii abstracte. Mutaţia pentru alegerea serviciului concret care implementează un serviu abstract este facută aleator. Problema de selecţie este reprezentată de o funcţie cu fitness dinamic având ca scop minimizarea unor criterii cum ar fi costul şi în acelasi timp alegerea valorii maxime pentru alte criterii cum ar fi disponibilitatea sistemului. Funcţia de fitness trebuie şa penalizeze serviciile care nu îndeplinesc cerinţele QoS. Calculul valorii agregate QoS este realizată folosind o metodă de agregare existentă a autorilor [**74**], prezentate şi de noi, mai pe larg în cadrul secţiunii 5.2.

O altă tehnică de optimizare se bazează pe metoda programării întregi (Integer Programming). Autorii [**94**] oferă o notaţie bazată pe diagrame de stare pe care o folosesc pentru a specifica n model de compunere împărţindu-l în mai multe căi de execuţie. Fiecare cale de execuţie este considerată a fi un graf aciclic direcţionat. Metoda realizează o optimizare locală prin asocierea fiecărui serviciu concret care implementează un serviciu abstract, a unui vector de calitate. Pe baza valorilor asociate atributelor QoS, alese de client şi folosind tehnica MCDM (Multiple Criteria Decision Making) este ales serviciul care îndeplineşte criteriile QoS cerute şi are scorul maxim. Pentru optimizarea globală, autorii

folosesc tehnici de programare liniară cum ar fi metoda programării întregi. Autorii susţin că selecţia serviciilor trebuie sa fie realizată în momentul execuţiei şi nu în momentul proiectării unei compuneri de servicii, având în vedere caracterul fluctuant al atributelor QoS. Autorii propun o replanificare şi reoptimizare a compunerii care să aibă loc în timpul execuţiei.

Autorii [**95**] aleg sa reprezinte problema compunerii serviciilor ca un program stohastic multi-obiectiv care optimizează simultan un număr de parametri. Lucrarea alege ca parametri fixi durata execuţiei, costul total, disponibilitatea şi fiabiliatatea. Modelul ţine cont de instabilitatea parametrilor QoS şi foloseşte tehnica stohastică average value–at-risk (AvaR) pentru a minimiza parametrii de cost şi durată a execuţiei, în timp ce disponibilitatea şi fiabilitatea sunt definite ca fiind constrângeri. Abordarea nu oferă clienţilor posibilitatea exprimării unor cerinţe QoS.

Autorii [**96**] specifică o arhitectură pentru selecţia serviciilor web dintr-un model de compunere, ţinând cont de valorile globale ale parametrilor QoS ale acestuia. Abordarea modelează problema de compunere sub forma unui model combinatorial şi a unui model bazat pe grafuri. Modelul bazat pe grafuri este folosit pentru modelele de compunere care au o structură secvenţială şi modelul combinatorial este folosit pentru a reprezenta un model de compunere care conţine structuri de tip joncţiune (join), bucla (loop), bifurcaţii (split) etc. Modelul combinatorial este rezolvat folosind tehnica de optimizare combilatorială multi-choice knapsack problem (MMKP). Modelul bazat pe grafuri defineşte problemă sub forma unei probleme multi-constrained optimal path (MCOP). Pentru a optimiza algoritmnii MMKP şi MCOP care sunt NP-complete autorii folosesc diferite euristici respectiv programare întreagă pentru a rezolva problema în timp polinomial. Metoda propusă permite specificarea unor funcţii definite client pentru optimizarea unor atribute QoS particulare, specifice unei aplicaţii şi deasemenea specificarea unor cerinţe QoS globale.

În cadrul abordărilor bazate pe metode combinatoriale se află şi metoda [**97**] care analizează asemănările dintre problema optimizării multicriteriale a compunerii serviciilor web în funcţie de mai mulţi parametri QoS şi probleme multicriteriale combinatoriale cum ar fi Knapsack Problem şi Resource Constraint Project Scheduling Problem (RCPSP). Lucrarea analizeaza eficacitatea unor posibile euristici pentru rezolvarea acestor probleme şi eficacitatea lor în cazul problemelei selecţiei serviciilor web.

O abordare care combină mai multe tehnici de optimizare este prezentată în [**98**] [**99**]. Lucrările, este spre deosebire de celelalte lucrări prezentate anterior care tratează în principal problema optimizării, oferă o abordare completă a problemei compunerii serviciilor web. Autorii oferă o arhitectură CaaS (Compostion as a Service) care îşi propune să permită specifiarea, optimizarea şi execuţia unei compuneri, astfel încât clienţii unei compuneri de servicii să nu aibă nevoie de o infrastructură proprie pentru a definii şi rula o compunere de servicii. Autorii definesc un limbaj specific, VCL (Vienna Composition Language), pentru specificarea cerinţelor funcţionale şi nefuncţionale. VCL permite clienţilor definirea cerinţelor de calitate, putând fi specificate cerinţe şi preferinţe cărora li se poat asocia ponderi reprezentând importanţa acestora pentru client. Optimizarea compunerii este

realizată transformând problema QoS într-o problemă de optimizare a constrângerilor COP (Constraint Optimization Problem) şi într-o problemă de programare întreagă IP (Integer Programming). Pentru executarea compoziţiei este oferită infrastructura VRESCO (Vienna Runtime Environment for Service-Oriented Computing) [**100**]. Abordarea are însă şi o serie de dezavantaje cum ar fi faptul ca pentru specificarea importanţei preferinţelor QoS se pot asocia doar ponderi şi nu este posibilă specificarea unor reguli complexe de compromis. Modelul de compoziţie nu permite existenţa unor structuri ciclice, graful folosit pentru descrierea procesului fiind aciclic.

În prezent există un număr foarte mare de metaeuristici pentru rezolvarea problemelor multiobiectiv, iar compararea lor este o problemă de actualitate. Această temă este adresată de frameworkul jMetal [**101**], un framework Java ce îşi propune realizarea comparării diferitelor metaeuristici oferind o serie de module de test pentru aplicarea unor metrici de performanţă şi a analizării rezultatului. jMetal oferă un cadru extensibil, ce poate fi folosit pentru a implementa diferiţi algoritmi şi oferă pentru exemplificarea funcţionalităţii o implementare Java a algoritmilor evoluţionişti NSGA-II SPEA2, PAES, precum şi a unei metaeuristici scatter search şi particle swarm, oferind de asemenea posiblitatea comparării rezultatelor.

7.Metoda condiţional lexicografică QoSPref

În cadrul unei probleme de selecţie având obiective multiple, distingem între obiective care trebuie obligatoriu îndeplinite şi obiective opţionale. Astfel, serviciul ales trebuie să îndeplinească cerinţele obligatorii, iar în cazul în care niciun serviciu nu îndeplineşte toate preferinţele opţionale ale clientului, trebuie ales cel mai bun compromis de preferinţe opţionale conform cerinţelor acestuia. În acest scop, avem nevoie de un limbaj în care clientul să îşi poată descrie preferinţele şi prioritatea lor şi de un mecanism de decizie care să poata interpreta cerinţele clientului şi să aleagă soluţia cea mai potrivită conform cerinţelor specificate, în mod automat, dinamic.

Metoda pe care o propunem în acest capitol se bazează pe modul în care raţionează oamenii aflaţi în situaţia de a alege o variantă dintre o mulţime de posibilităţi. În mod uzual oamenii îşi crează un set de reguli care să îi ajute să analizeze şi să compare variantele existente. Prioritizarea preferinţelor este similară cu impunerea unei ordini lexicografice asupra setului de preferinţe, dar în practică o ordonare strictă este insuficientă pentru a descrie complet cerinţele existente. Astfel, oamenii stabilesc un set de reguli care schimbă prioritatea preferinţelor atunci când anumite condiţii sunt îndeplinite.

Pentru descrierea preferinţelor, metoda propusă pune la dispoziţie o notaţie simplă, care oferă posibilitatea ataşării de reguli la preferinţele lexicografice. Notaţia oferă un limbaj intuitiv, care nu necesită cunoştinte tehnice, astfel putând fi folosit de un număr mare de clienţi. Metoda foloseşte un algoritm care analizează atât priorităţile preferinţelor, cât şi regulile ataşate lor şi stabileşte o ordine totală a alternativelor.

7.1 Notaţie pentru descrierea preferinţelor

Notaţia propusă oferă posibilitatea descrierii preferinţelor clientului. Preferinţele QoS se definesc în două blocuri separate: un bloc care conţine constrângerile obligatorii şi un bloc în care sunt definite preferinţele opţionale şi eventual condiţiile asociate lor.

Pentru a exemplifica, presupunem cazul unei companii care oferă servicii de vizualizare a datelor, cum ar fi generarea de grafice pe baza unui set de date. Compania deleagă generearea graficelor altor firme externe, care oferă servicii web pentru generarea de grafice. Un client îşi specifică preferinţele folosind această notaţie iar pe baza acestei specificaţii şi a algoritmului descris în secţiunea 7.2 un *web service broker* alege serviciul care oferă cea mai potrivită funcţionalitate pentru cererile clientului. Serviciile web care oferă funcţionalitatea de generare de grafice au atât proprietăţi QoS independente de domeniul de activitate, cum ar fi timpul de răspuns sau disponibilitatea, cât şi proprietăţi din cadrul domeniului de activitate, cum ar fi tipul de grafic generat, preţul solicitat pentru generarea unui grafic sau rezoluţia graficelor generate.

Considerăm drept client al acestei companii un sistem de achiziţii de date care afişează în mod regulat (spre exemplu, la intervale de 5 secunde) un grafic care prezintă situaţia unui

proces de producţie. Graficul este vizualizat pe un monitor având o rezoluţie de 1280x780 pixeli, care suportă maxim 65536 culori. Graficul este actualizat la fiecare 5 secunde, iar în cazul în care acesta nu este disponibil, este omisă o actualizare iar următoarea afişare are loc după încă un interval, adică abia după 10 secunde.

Constrângerile sunt definite în cadrul unui bloc "*constraints*", sub forma unei liste, în care fiecare intrare constituie o condiţie booleană care trebuie să fie îndeplinită în mod obligatoriu de către serviciu. Ordinea în care sunt definite constrângerile în cadrul listei nu este importantă, deoarece toate constrângerile trebuie îndeplinite.

Pentru exemplul ales de noi extragem urmatoarele constrângeri: clientul are nevoie de o vizualizare a datelor folosind grafice de tipul "time series", costul serviciului trebuie să nu depăşească 10 unităţi (de ex. Euro), serviciul trebuie să fie disponibil în proporţie de 95%, rezoluţia necesară este de 1280x720 pixeli, iar timpul de răspuns trebuie să fie mai mic decât 10 unităţi (de ex. secunde). Considerăm următoarea definiţie de constrângeri, pentru exemplul ales:

```
constraints {
        chartType = "time series",
        cost < 10,
        availability > 0.95,
        imageResolution = "1280x720",
        responseTime < 10
}
```

Clientul îşi defineşte preferinţele opţionale asupra serviciului sub forma unei liste, în cadrul blocului de preferinţe "*preferences*". În timp ce ordinea cerinţelor nu este relevantă pentru constrângerile obligatorii, pentru preferinţe ordinea este importantă. Preferinţele trebuie specificate în ordinea importanţei, pe primul loc fiind definită cea mai importantă preferinţă considerată.

```
preferences {
        cost,
        availability : high,
        responseTime,
        colors : high
}
```

Pentru fiecare preferinţă definită clientul trebuie să indice direcţia în care se găsesc valorile optimale, spre exemplu, un preţ este considerat cu atât mai bun, cu cât valoarea lui este mai mică, în timp ce disponibilitatea serviciului este cu atât mai bună, cu cât valoarea sa este mai mare. Valorile posibile sunt "*low*" si "*high*", în funcţie de direcţia care indică valorile mai

bune. Operatorul default, care este considerat în cazul în care nu este specificată nici o direcţie, este operatorul "*low*", adică valorile mai mici sunt preferate. Astfel, în definiţia de mai sus, preferinţele "cost" şi "timp de răspuns" (*responseTime*) folosesc opreratorul default, având direcţia "*low*". Datorită ordinii specificate, preferinţa "preţ" este considerată ca fiind cea mai importantă preferinţă.

Preferinţele specificate în forma actuală nu reflectă în mod corect şi complet cerinţele clientului. Metoda propusă stabileşte în mod automat o ordine totală a alternativelor şi are nevoie de definirea într-un mod complet a preferinţelor clienţilor în notaţia aleasă pentru a putea stabili serviciul web câştigător.

O primă problemă pe care o putem considera este situaţia în care timpul de răspuns al serviciului depăşeşte 5 secunde. În acest caz, o actualizare a graficului ar fi omisă, situaţie care trebuie evitată chiar dacă trebuie considerat un preţ mai mare. Constrângerea definită de client legată de preţ trebuie respectată, dar o diferenţă de preţ între două servicii considerate care oferă timpi diferiţi de răspuns trebuie să poată fi specificată. O diferenţă a timpului de răspuns este relevantă pentru client numai în condiţiile în care un serviciu web oferă un timp de răspuns mai mare decât cele 5 secunde, iar celălalt un timp de răspuns mai mic. În cazul in care serviciile comparate au timpi de răspuns diferiţi, dar mai mici decât 5 secunde, valoarea exactă a timpului de răspuns nu contează, pentru că actualizarea graficului se face doar după 5 secunde, astfel încât un timp de răspuns mai mic nu aduce niciun avantaj practic.

O altă situaţie care trebuie considerată este diferenţa de calitate a graficului rezultat, datorată numărului de culori. În cazul în care un serviciu oferă grafice având un număr mai mic de culori decât 65536, calitatea graficului nu este optimală. Pe de altă parte, o diferenţă a numărului de culori pentru valori mai mari decât numărul maxim de culori suportat de monitorul clientului, nu este sesizabilă pentru acesta, deci nu trebuie sa fie considerată relevantă. Pentru valori nesatisfacătoare ale acestui parametru, o diferenţă mică a preţului trebuie acceptată în cazul în care duce la o creştere a calităţii imaginii graficului generat.

Pentru a putea specifica aceste condiţii din cadrul exemplului descris, notaţia propusă oferă patru operatori unari care sunt definiţi în tabela următoare:

Operator	Semnificaţie
AT_LEAST_ONE* (condition)	condition($service_1$) OR condition($service_2$)
EXACTLY_ONE (condition)	condition($service_1$) XOR condition($service_2$)
ALL (condition)	condition($service_1$) AND condition($service_2$)
DIFF (attribute)	|$service_1$.attribute - $service_2$.attribute|

Fig. 12. Operatori oferiţi de notaţie

Primii trei opreratori necesită ca parametru o expresie booleană care este evaluată de două ori, pentru ambele servicii web care sunt comparate. Expresia booleană poate avea ca parametri unul sau mai multe atribute QoS. Operatorii booleeni OR, XOR, AND evaluează rezultatele expresiilor booleene corespunzătoare serviciilor comparate şi oferă rezultatul condiţiei exprimate.

Ultimul operator, DIFF, necesită ca parametru un atribut QoS şi oferă ca rezultat modulul diferenţei dintre valorile acestui atribut pentru serviciile comparate.

Operatorul AT_LEAST_ONE este definit ca operator default şi nu trebuie specificat explicit.

Folosind aceşti operatori, cerinţele clientului descrise în exemplul de mai sus pot fi formulate folosind notaţia noastră în felul următor:

```
preferences {
        [EXACTLY_ONE(responseTime >5)] responseTime,
        [DIFF(cost) > 2] cost,
        [colors < 65536] colors : high,
        cost,
        availability : high,
        responseTime,
        colors : high
}
```

Fig. 13. Definiţia preferinţelor clientului

Condiţia $[\text{colors} < 65536]\ \text{colors} : \text{high}$ nu are definit în mod explicit un operator aşa că este folosit operatorul default AT_LEAST_ONE.

7.2 Algoritm pentru ordonarea alternativelor

Pentru a afişa rezultatul comparării a două servicii web s1 şi s2 introducem următoarea notaţie:

$s_1 \succ s_2$ serviciul web s_1 este preferat serviciului s_2

$s_1 \prec s_2$ serviciul web s_2 este preferat serviciului s_1

$s_1 \sim s_2$ între serviciile web s_1 şi s_2 există o relaţie de indiferenţă
(nu putem spune că unul dintre servicii este preferat celuilalt)

$s_2 \succ_k s_1$ serviciul web s_1 este preferat serviciului web s_2
(regula *k* din blocul de preferinţe a fost cea care a stabilit această ierarhie)

$s_1 \prec_k s_2$ serviciul web s_2 este preferat serviciului web s_1
(regula *k* din blocul de preferinţe a fost cea care a stabilit această ierarhie)

Algoritmul folosit pentru a compara serviciile web pe baza notaţiei definite în 7.1 este enunţat în pseudocod în continuare:

```
1.  function compareServices(service1, service2, preferences)
2.    for i ← 1 .. length(preferences) do
3.      cond ← preferences[i].condition
4.      attr ← preferences[i].attribute
5.      dir ← preferences[i].direction
6.      if cond = null OR cond(service1, service2) = true then
7.        result ← compare(service1.attr, service2.attr, dir)
8.        if result ≠ 0 then
9.          return {result, i}
10.       end if
11.     end if
12.   end for
13.   return null
14. end function
```

Algoritmul analizează toate condiţiile din cadrul blocului de preferinţe, în ordinea în care au fost enunţate, adică în ordinea importanţei lor. În cazul în care condiţia analizată nu are ataşat nici un operator, sau operatorul este evaluat pozitiv (linia 6), valorile atributului QoS specificat în cadrul acestei condiţii sunt evaluate pentru serviciile comparate folosind următoarea funcţie:

```
function compare(attr1, attr2, direction)
        if attr1 = attr2 then
                result ← 0
        else if attr1 < attr2 then
                result ← 1
        else
                result ← -1
        end if
        if direction = high then
                result ← -result
        end if
                return result
end function
```

Funcţia compară valorile atributelor celor două servicii web şi returnează o valoare numerică pozitivă în cazul în care primul argument este mai bun, negativă dacă cel de-al doilea argument este mai bun sau 0 dacă argumentele funcţiei sunt egale.

În cazul în care valorile atributului comparat sunt egale, algoritmul evaluează condiţia următoare din cadrul blocului de preferinţe. În momentul în care se constată o diferenţă, se returnează rezultatul comparaţiei şi numărul condiţiei care a dus la stabilirea acestei ierarhii. Stabilirea regulii care a dus la realizarea unei clasificări a serviciilor este necesară pentru a elimina eventualele intranzivităţi ale preferinţelor şi a stabili o ordine totală a serviciilor, ţinând cont de prioritaea preferinţelor. Compararea perechilor de servicii nu este suficientă pentru a stabili o ordine totală, deoarece metoda acceptă preferinţe intranzitive. Matematicianul psiholog Amos Tversky a demonstrat într-o serie de experimente faptul că oamenii au preferinţe intranzitive, de aceea este importantă acceptarea acestor preferinţe de către metoda propusă.

Pentru a exemplifica situaţia intranzivităţii preferinţelor, considerăm următoarele cinci servicii web din Tab. 3, care au specificate trei atribute de calitate QoS.

	WS_1	WS_2	WS_3	WS_4	WS_5
responseTime	7.0	7.0	5.5	4.5	7.5
Cost	4.0	5.0	6.5	8.0	7.5
Colors	256	256	256	65536	65536

Tab. 3. Exemplu de servicii web având specificate atributele QoS

Dintre serviciile web trebuie aleasă cea mai potrivită alternativă conform următoarei versiuni simplificate a preferinţelor enunţate de client în Fig. 13 în cadrul definiţiei notaţiei.

```
preferences {
        [EXACTLY_ONE(responseTime >5)] responseTime,
        [DIFF(cost) > 2] cost,
        [colors < 65536] colors : high,
        responseTime
}
```

Toate perechile de servicii web sunt comparate, folosind algorimul enunţat anterior, rezultatul fiind afişat în tabela Tab. 4.

Pentru a ordona serviciile, comparăm parametrii de calitate în ordinea importanţei, conform regulilor de preferinţe exprimate în blocul *preferences*: începem cu timpul de răspuns, continuăm cu costul serviciilor, iar apoi cu numărul de culori. În cazul serviciilor WS1 şi WS2, timpul de răspuns fiind egal, analizăm diferenţa de cost. Diferenţa de cost este 1, iar conform regulii `[DIFF(cost) > 2] cost`, diferenţa fiind mai mică decât 2, vom analiza următorul criteriu: numărul de culori. Numărul de culori este egal, astfel încât relaţia dintre serviciile WS1 şi WS2 este una de indiferenţă.

Pentru a compara serviciile WS1 şi WS3, analizăm timpul de răspuns. Ambele servicii au un timp de răspuns mai mare decât 5, astfel încât analizăm următoarea regulă din blocul de preferinţe: costul serviciilor. Diferenţa de cost este de 2,5 şi conform regulii de preferinţe `[DIFF(cost) > 2] cost`, WS1 este preferat serviciului WS3, datorită regulii numărul 2.

În cazul serviciilor WS_1 şi WS_4, timpul de răspuns este criteriul care determină ordonarea, serviciul WS_4 având valoarea 4,5 care este mai mică decât 5 conform regulii de preferinţe numărul 1: `[EXACTLY_ONE(responseTime >5)] responseTime`. Serviciul WS_1 este deci preferat serviciului WS_4 conform regulii de preferinţe numărul 1.

Pentru perechea de servicii WS_1 şi WS_5, timpul de răspuns este în ambele cazuri mai mare decât 5, aşa încât analizăm costul. Diferenţa de cost fiind mai mare decât 2, serviciul mai ieftin WS_1 este considerat ca fiind mai bun datorită regulii de preferinţe 2.

Comparând WS_2 şi WS_3, nu putem stabili o ordonare a serviciilor conform primelor 3 reguli din blocul de preferinţe deoarece timpul de răspuns este în ambele cazuri mai mare decât 5, diferenţa de cost fiind mai mică decât 2 nu trebuie luată in considerare, iar numărul de culori este egal. Astfel regula de preferinţe numărul 4, timpul de răspuns este cea care stabileşte că serviciul WS_2 este mai bun decât serviciul WS_3, având un timp de răspuns mai mic.

Pentru serviciile WS_2 şi WS_4, prima regulă de preferinţe determină relaţia de superioritate, serviciul WS_4, având un timp de răspuns considerabil mai bun.

Comparând WS_2 şi WS_5, timpul de răspuns este în ambele cazuri mai mare decât 5, dar diferenţa de cost a serviciilor este de 2,5, astfel încât, conform relaţiei de preferinţe 2, serviciul WS_2 este superior serviciului WS_5.

Pentru serviciile WS_3 şi WS_4, datorită timpului foarte bun de răspuns al serviciului WS_4 acesta este superior serviciului WS_3 conform primii reguli de preferinţe.

Serviciul WS_3 este, conform regulii de preferinţe 3 considerat superior serciviului WS_5 oferind grafice de calitate superioară, cu un număr mai mare de culori.

Pentru compararea serviciilor WS_4 şi WS_5 timpului foarte bun de răspuns al serviciului WS_4 stabileşte relaţia de superioritate, astfel încât serviciul WS_4 este superior serviciului WS_5 conform primii reguli de preferinţe.

Rezulatul comparării unei perechi de servicii web WS_i şi WS_j este afişat în tabelă în dreptul simbolului corespunzător i/j.

1/2	1/3	1/4	1/5	2/3	2/4	2/5	3/4	3/5	4/5
$\sim$	$\succ_2$	$\prec_1$	$\succ_2$	$\prec_4$	$\prec_1$	$\succ_2$	$\prec_1$	$\prec_3$	$\succ_1$

Tab. 4. Rezultatele comparaţiei perechilor de servicii web

Analizând rezultatul comparaţiei se pot observa următoarele cazuri de intranzitivitate a preferinţelor:

$WS_1 \succ WS_3 \succ WS_2$
$WS_2 \sim WS_1$

Astfel, deşi conform comparaţiei perechilor de servicii WS_1 cu WS_3 şi WS_3 cu WS_2 reiese că serviciul WS_1 este superior serviciului WS_2, în momentul comparaţiei directe, WS_2 este echivalent ca preferinţe cu WS_1.

Un alt exemplu de intrazitivitate este următorul:

$WS_2 \prec WS_3 \prec WS_5$
$WS_5 \prec WS_2$

Pentru a stabili o ordine totală a alternativelor, definim un algoritm care ataşează fiecărui serviciu web *i,* un vector de valori întregi $V_i \in \mathbb{N}^{r+1}$, unde r reprezintă numărul total de preferinţe. Următorul algoritm calculează vectorii de scor ai fiecărui serviciu web comparat. Cu *n* este notat numărul de servicii web care sunt comparate.

procedure createScoreVectors ()

for i ← 1 .. n **do**

for k ← 1 .. r **do**

$V_i^k \leftarrow$ *numărul de cazuri în care serviciul* WS_i *este preferat unui alt serviciu web datorită regulii k (i.e., datorită unei relaţii* $\succ_k$*).*

end for

$V_i^{r+1} \leftarrow$ *numărul de cazuri în care serviciul* WS_i *este într-o relaţie de indiferenţă cu un alt serviciu web.*

end for

end procedure

Folosind acest algoritm calculăm pentru cele 5 servicii web din exemplul anterior următorii vectori de scor:

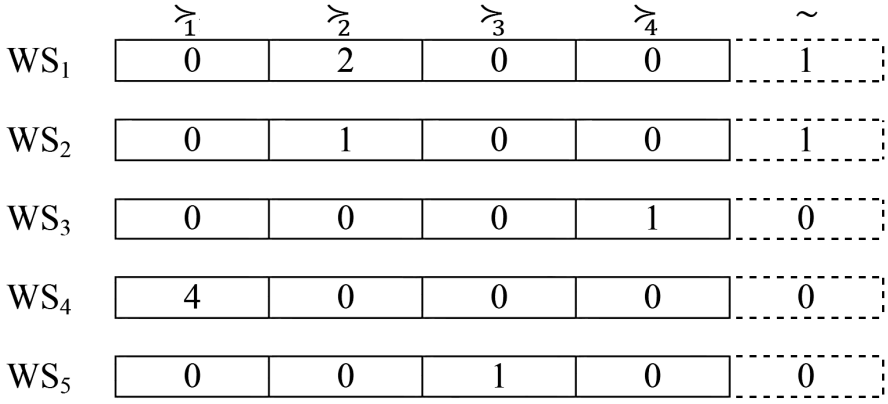


Fig. 14. Vectori de scor

Pentru a compara vectorii de rezultate obţinuţi folosim următorul algoritm, care stabileşte o ordine totală peste mulţimea alternativelor existente. Pentru fiecare pereche de servicii web se stabileşte si se compară numărul total de cazuri în care un serviciu web este preferat altor servicii web. Serviciul care este mai des preferat este ales ca fiind câştigător. În cazul în care acest prim pas nu este suficient pentru a determina cel mai bun dintre cele două servicii ale unei perechi, se compară de câte ori un serviciu este preferat altor servicii datorită celei mai importante preferinţe (prima intrare din vectorul de scor) şi astfel se stabileşte serviciul câştigător. În cazul în care nici acest pas nu este suficient, se trece la următoarea intrare din vectorul de scor şi procedeul se repetă până când se poate desemna un serviciu web câştigător. În cazul în care vectorii de scor sunt identici, algoritmul returnează valoarea 0, serviciile web fiind echivalente ca prioritate.

Algoritmul descris în pseudocod este următorul:

function compareScores(V_1, V_2)
  $\text{count}_1 \leftarrow \sum_i^r V_1^i$
  $\text{count}_2 \leftarrow \sum_i^r V_2^i$
    if $\text{count}_1 \neq \text{count}_2$ **then**
      return count_1 - count_2
    end if
    for i ← 1 .. r + 1 **do**
      if $V_1^i \neq V_2^i$ **then**
        return $V_1^i - V_2^i$
      end if
    end for
    return 0
end function

Folosind acest algoritm pentru exemplul de servicii web din Tab. 3 ajungem la următoarea clasificare: (WS_4, WS_1, WS_2, WS_5, WS_3), unde serviciul WS_4 este considerat a fi cea mai potrivită alternativă pentru cerinţele date. Astfel algoritmul înlătură clasificarea ciclică, stabilind o ordine totală chiar şi în situaţia existenţei preferinţelor intranzitive.

Metoda introdusă în acest capitol oferă posibilitatea de a selecta cel mai bun serviciu web dintr-o mulţime de alternative posibile, ţinând cont de preferinţele QoS exprimate de client. Avantajul principal al metodei noastre este că permite descrierea preferinţelor într-o manieră simplă şi intuitivă, care nu impune cunoaşterea unor tehnici de analiză şi decizie multicriterială. Faptul că preferinţele sunt exprimate a-priori, fără a necesita intervenţia ulterioară a clientului în procesul de decizie, face posibilă selecţia dinamică şi automată a celui mai bun servicu web.

În plus, metoda propusă poate fi adaptată pentru a realiza compunerea dinamică a serviciilor web ţinând cont de preferinţele QoS ale clienţilor. Acest aspect va fi abordat în capitolul 7.3.

7.3 Aplicaţie practică

Pentru a demonstra practic viabilitatea metodei noastre, oferim o implementare a motorului de selecţie dinamică a celui mai bun serviciu web dintr-o serie de alternative, pe baza abordării condiţional lexicografice propuse în capitolul 7.2. Motorul de selecţie este oferit sub forma unei biblioteci Java, pe care o punem la dispoziţie sub forma unui proiect open-source denumit QoSPref, accesibil la adresa http://qospref.sourceforge.net. Un scenariu tipic de ultilizare a acestei biblioteci este ca parte integrantă a unui framework SOA, care accesează motorul de selecţie prin intermediul interfeţei de programare (API) puse la dispoziţie de biblioteca noastră. Framework-ul SOA, care joacă rol de client al motorului de selecţie, trebuie să ofere ca date de intrare informaţii legate de serviciile web candidat, de atributele QoS ale acestora, precum şi de preferinţele QoS care trebuie luate în considerare.

În plus, pentru a ilustra modul de utilizare a bibliotecii şi a uşura înţelegerea funcţionării motorului de selecţie, QoSPref oferă şi o interfaţă grafică prin intermediul căreia pot fi accesate funcţiile oferite de bibliotecă.

7.3.1 Interfaţa de programare (API) a motorului de selecţie

În continuare, prezentăm o descriere mai detaliată a interfeţei de programare a motorului de selecţie implementat în QoSPref. Această API pune la dispoziţie o singură operaţie, care realizează ordonarea serviciilor web candidat pe baza preferinţelor exprimate de client:

$$ranking \leftarrow \textbf{\textit{rankServices}}\ (qosAttributes, qosPreferences, servicesInfo)$$

7.3.2 Parametrii de intrare

Operaţia *rankServices* necesită 3 parametri de intrare, pe care îi descriem pe larg în cele ce urmează.

7.3.2.1 qosAttributes

Motorul de selecţie trebuie să cunoască numele şi tipul atributelor QoS care caracterizează serviciile web candidat. Această informaţie este transmisă sub forma unei notaţii textuale prin intermediul parametrului *qosAttributes*. În exemplul de mai jos, această notaţie este folosită pentru a specifica existenţa a 3 atribute QoS (*responseTime*, *cost* şi *colors*):

```
quantities {
  responseTime: real,
  cost : real,
  colors : integer
}
```

Atributele QoS sunt enumerate în cadrul blocului *quantities*, iar tipul unui atribut apare după numele acestuia, separat prin simbolul '**:**'. În momentul de faţă, tipurile acceptate pentru atributele QoS sunt: *integer*, *real* şi *boolean*. Implementarea noastră asociază tipul Java *long* atributelor de tip *integer* şi tipul Java *double* atributelor de tip *real*. Ne propunem ca într-o versiune viitoare să oferim suport şi pentru atribute care reflectă apartenenţa la o anumită categorie şi ale căror valori posibile pot fi definite sub forma unei enumerări. De exemplu, în cazul în care unul din atributele QoS considerate este metoda de criptare utilizată de serviciul web, valorile posibile ar putea fi: (noEncryption, AES, DES, 3DES).

Descrierea formală a sintaxei folosite pentru specificarea parametrului *qosAttributes* (precum şi a celorlalţi parametri de intrare) este prezentată în anexa 11.

7.3.2.2 qosPreferences

Pentru a specifica preferinţele QoS, se foloseşte notaţia introdusă în secţiunea 7.1. Este permisă numai utilizarea atributelor QoS care au fost descrise prin intermediul parametrului *qosAttributes*. De exemplu, în concordanţă cu atributele QoS descrise în exemplul din secţiunea anterioară, parametrul *qosPreferences* ar putea conţine următoarea specificaţie:

```
constraints {
  cost < 10,
  responseTime < 10
}

preferences {
  [EXACTLY_ONE(responseTime > 5)] responseTime,
  [DIFF(cost) > 2] cost,
  [colors < 65536] colors: high,
  responseTime
}
```

7.3.2.3 servicesInfo

Pentru fiecare dintre serviciile web candidat, motorul de selecţie trebuie să cunoască valorile atributelor QoS asociate. Acestea sunt transmise prin intermediul parametrului *servicesInfo*, care este reprezentat ca o listă de elemente de tipul QoSContext, fiecare element

corespunzând unui serviciu web candidat. QoSContext este o structură de date care permite citirea valorilor atributelor QoS ale unui serviciu web. Deoarece atributele QoS ce trebuie luate în considerare nu sunt prestabilite, ci sunt oferite ca date de intrare (prin intermediul parametrului *qosAttributes*), această structură de date trebuie să fie generică. De aceea, ea este specificată în API sub forma următoarei interfeţe Java:

```
public interface QoSContext {
        long getInteger(String qosAttribute);
        double getReal(String qosAttribute);
        boolean getBoolean(String qosAttribute);
}
```

Fiecare element al listei *servicesInfo* trebuie să constituie o implementare a acestei interfeţe. Pentru fiecare atribut QoS declarat prin intermediul parametrului *qosAttributes*, motorul de selecţie poate afla valoarea sa apelând metoda corespunzătoare tipului acestui atribut, folosind drept argument numele atributului. De exemplu, pentru atributele QoS considerate în exemplul din secţiunea 7.1, motorul de selecţie va executa pentru fiecare element al listei *servicesInfo* următoarele apeluri de funcţii:

- getReal("responseTime")
- getReal("cost")
- getInteger("colors")

Interfaţa de programare (API) a fost proiectată astfel încât să impună doar transmiterea informaţiilor strict necesare motorului de selecţie. De aceea, parametrii de intrare nu conţin informaţii suplimentare, cum ar fi numele serviciilor web sau modul în care acestea pot fi apelate. Astfel de date sunt de obicei esenţiale pentru un client al bibliotecii noastre, care cel mai adesea este un framework SOA, dar sunt irelevante în procesul de evaluare a serviciilor web candidat. Motorul de selecţie identifică un serviciu web prin indicele corespunzător structurii sale QoSContext în cadrul listei *servicesInfo*.

7.3.3 Parametrii de ieşire

Operaţia *rankServices* evaluează serviciile web candidat specificate prin intermediul parametrului de intrare *servicesInfo* şi le ordonează pe baza preferinţelor exprimate prin intermediul parametrului de intrare *qosPreferences*. Această operaţie are un singur parametru de ieşire, denumit *ranking*, care este reprezentat sub forma unui tablou de n numere întregi, unde n este numărul de servicii web candidat. Valoarea *ranking*[i] reprezintă rangul serviciului candidat cu indicele i din lista *servicesInfo*, adică poziţia pe care acesta se află în clasamentul efectuat de motorul de selecţie. Cel mai bun dintre seviciile web candidat este cel cu rangul 0.

7.3.4 Detalii de implementare

Selecţia dinamică a celui mai bun serviciu web dintr-o serie de alternative este realizată pe baza metodei condiţional lexicografice descrise în secţiunea 7.1. Motorul de selecţie parsează parametrii de intrare *qosAttributes* şi *qosPreferences* şi generează dinamic clasele

Java necesare comparării serviciilor web. Compararea se realizează folosind algoritmii enunţaţi anterior, pe baza cărora se stabileşte o ordine totală a variantelor existente şi se poate alege cea mai bună soluţie.

Pentru a compara serviciile web trebuie ca limbajul folosit de client pentru descrierea atributelor QoS şi a preferinţelor să fie parsat. Parsarea este realizată folosind frameworkul ANTLR [**102**] (ANother Tool for Language Recognition).

ANTLR este un generator de parsere ce oferă un cadru pentru construirea de interpretoare de limbaj, compilatoare şi convertoare de gramatici în diferite limbaje cum ar fi Java, C, C++, C#, Javascript, Perl, PHP etc. Preferinţele clientului sunt parsate folosind gramatica descrisă in Anexa 11.1, pe baza căreia ANTLR îşi generează un arbore sintactic abstract care este transformat în paşi succesivi în byte-code-ul Java care realizează comparaţia.

Pentru generarea byte-codului Java este folosit framework-ul ASM [**103**]. ASM este un framework ce oferă posibilitatea analizării şi manipulării de byte-code Java. Alegerea de a genera byte-code a fost datorată aspectelor de performanţă, pentru a realiza o bibliotecă performantă şi uşor de folosit într-un mediu dinamic. Codul generat este bazat pe preferinţele şi regulile clientului şi poate fi folosit apoi pentru a compara orice alte alternative de servicii web fară a necesita vreo modificare.

O altă variantă de implementare a comparaţiei este generarea codului sursă al claselor ce realizează compararea serviciilor. Această variantă este mai simplă din punctul de vedere al implementării, dar necesită compilarea ulterioară a codului sursă generat, lucru posibil prin intermediul interfeţei JavaCompiler, care este disponibilă în Java începând cu versiunea 6. Această metodă este însă mai restrictivă, deoarece nu toate runtime-urile Java pun la dispoziţie o implementare a interfeţei JavaCompiler. De exemplu, Oracle oferă o astfel de implementare în JDK, dar nu in JRE.

Pentru a exemplifica totuşi cum arată codul sursă Java care realizează compararea serviciilor web, interfaţa grafică descrisă în secţiunea 7.3.5 oferă posibilitatea generării şi afişării acestui cod, din motive de transparenţă a implementării. Codul Java este generat folosind StringTemplate [**104**], o bibliotecă pentru generarea de text pe baza unor şabloane. Această bibliotecă este folosită de ANTLR pentru generarea codului sursă.

7.3.5 Interfaţa grafică

Pentru a ilustra funcţionarea bibliotecii, QoSPref oferă o interfaţă grafică (Fig. 15) cu ajutorul căreia utilizatorul poate introduce datele de intrare şi poate vizualiza modul în care motorul de selecţie a ordonat serviciile web candidat, precum şi codul Java corespunzător bytecode-ului generat dinamic de biblioteca noastră.

Interfaţa grafică poate fi accesată atât ca aplicaţie Java de sine stătătoare, cât şi prin intermediul unui applet.

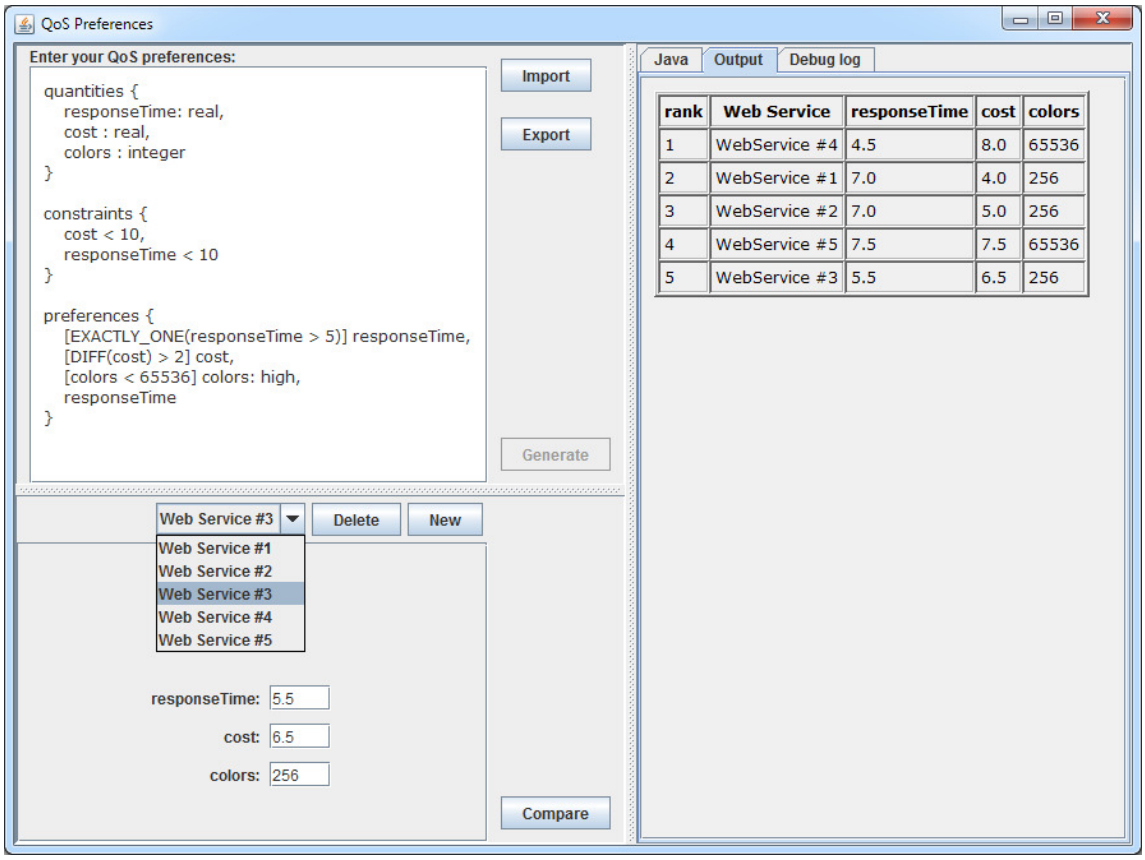


Fig. 15 Interfaţa grafică a aplicaţiei

Fereastra aplicaţiei este împărţită vertical în două regiuni. Partea stângă este folosită pentru a introduce datele de intrare, iar partea dreaptă pentru afişarea rezultatelor.

Aşa cum am menţionat în secţiunea 7.3.2, interfaţa de programare defineşte 3 parametri de intrare. Primi doi, *qosAttributes* şi *qosPreferences*, sunt exprimaţi cu ajutorul unei notaţii textuale. Utilizatorul poate introduce aceşti parametri în fereastra de editare din colţul din stânga-sus al aplicaţiei. Conţinutul acestei ferestre de editare poate fi exportat într-un fişier de preferinţe, având extensia "*.pref*", pentru o folosire ulterioară.

Partea din stânga-jos a ferestrei aplicaţiei este folosită pentru a introduce informaţiile legate de serviciile web candidat, adică valorile parametrului de intrare servicesInfo. Întrucât atributele QoS ce trebuie luate în considerare sunt făcute cunoscute prin intermediul parametrului de intrare qosAttributes, câmpurile de editare corespunzătoare acestora vor fi create dinamic de către interfaţa grafică în momentul apăsării butonului "Generate", care iniţiază procesul de parsare a specificaţiilor din fereastra de editare din colţul din stânga-sus. Un serviciu web candidat se introduce acţionând butonul "New" şi editând valorile atributelor sale de calitate.

Regiunea din partea dreaptă a interfeţei grafice conţine o fereatră multidocument pentru prezentarea rezultatelor. Dacă este selectat registrul "Output" (Fig. 15), interfaţa grafică prezintă sub formă tabelară rezultatul ordonării serviciilor web candidat pe baza

preferinţelor exprimate de utilizator. Selectând registrul "Java" (Fig. 16), poate fi vizualizat codul sursă Java care realizează compararea serviciilor. Aşa cum am menţionat în secţiunea 7.3.4, QoSPref nu foloseşte acest cod, ci generează direct bytecode Java. Codul sursă Java poate fi însă foarte util pentru a-i ajuta pe utilizatori să înţeleagă modul în care funcţionează motorul de selecţie. Registrul "Debug log", care permite afişarea mesajelor de depanare, se poate dovedi util pentru programatorii care doresc să extindă funcţionalitatea bibliotecii QoSPref.

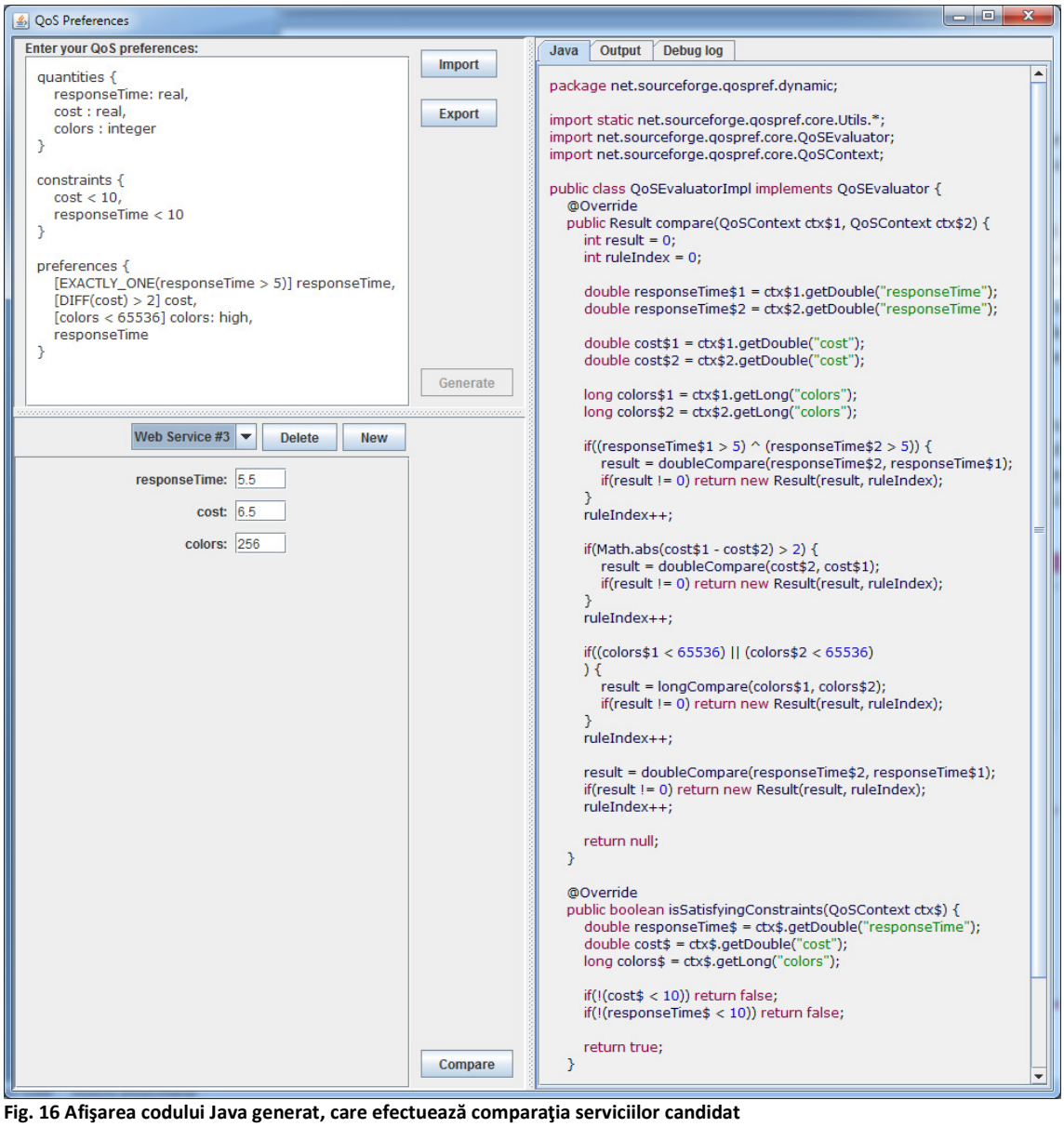


Fig. 16 Afişarea codului Java generat, care efectuează comparaţia serviciilor candidat

7.4 Concluzii

Metoda condiţional lexicografică propusă de noi permite ordonarea unui set de servicii web pe baza preferinţelor specificate de client. Clienţii îşi pot descrie preferinţele folosind un limbaj simplu şi intuitiv, care oferă posibilitatea ataşării de condiţii regulilor lexicografice

specificate. Limbajul oferit pentru specificarea preferinţelor se ghidează după modul în care oamenii raţionează şi stabilesc compromisuri în momentul în care se află în faţa unei decizii de a alege cea mai bună soluţie dintr-o mulţime de alternative.

În capitolul următor, ne propunem să studiem modul în care abordarea condiţional lexicografică poate fi utilizată în procesul de compunere dinamică a serviciilor web.

8. Compunerea serviciilor folosind QoSPref

În capitolul anterior am introdus limbajul QoSPref, care permite specificarea unor preferinţe QoS complexe şi am propus un algoritm de ordonare a serviciilor web în funcţie de aceste preferinţe. În continuare, ne propunem să studiem modul în care QoSPref poate fi folosit pentru selecţia serviciilor concrete care produc serviciul compus cu cei mai buni parametri QoS.

Există la ora actuală o serie de framework-uri SOA capabile să realizeze compunerea dinamică a serviciilor, însă niciunul dintre acestea nu rezolvă în mod satisfăcător problema preferinţelor QoS. Cele mai multe dintre ele oferă doar posibilitatea specificării de constrângeri. Foarte puţine permit şi exprimarea preferinţelor QoS, iar acest lucru este posibil numai într-o formă rudimentară. Intenţia noastră este de a oferi o tehnică de soluţionare a acestei probleme, care să poată fi în principiu integrată în oricare dintre framework-urile de compunere dinamică existente. Acest lucru este important, deoarece în prezent niciunul din framework-urile existente nu s-a impus ca standard şi nu există un consens referitor la arhitectura acestora sau la modalitatea optimă de descriere semantică a caracteristicilor funcţionale şi nefuncţionale ale serviciilor web.

Procesul de compunere dinamică a serviciilor web implică trei paşi importanţi [**105**]:

1. specificarea serviciului compus
2. selecţia serviciilor web componente
3. execuţia serviciului compus

Metoda condiţional lexicografică QoSPref poate fi utilizată pentru a implementa cel de-al doilea pas al compunerii dinamice: selecţia serviciilor web componente. Fiind dat un serviciu compus descris sub forma unui workflow care specifică serviciile abstracte participante, pasul 2 are rolul de a asocia fiecărui serviciu abstract un serviciu concret care implementează funcţionalitatea cerută. Acest pas poartă şi numele de "legare a serviciilor" (service binding). În cazul ideal, serviciile concrete sunt selectate astfel încât valoarea QoS agregată a serviciului compus rezultat să fie optimă.

În cadrul lucrării [**98**] autorii introduc conceptul de “Composition as a Service” (CaaS) şi prezintă modul în care această abordare a fost folosită pentru a implementa modulul de compunere a serviciilor pe care ei l-au integrat in framework-ul VRESCo. Scopul nostru nu este de a oferi o implementare completă a funcţionalităţii "Composition as a Service", ci de a oferi sub forma unui serviciu operaţia de selecţie a serviciilor web concrete care intră în componenţa unui serviciu compus. Abordarea noastră, pe care o vom denumi "Binding as a Service", are o granularitate mai mică decât "Composition as a Service". De aceea, există un număr mai mare de scenarii în care ea va putea fi folosită, iar practic orice framework de compunere dinamică a serviciilor va putea fi adaptat rapid pentru a apela servicul de selecţie oferit de noi. În acest fel, framework-ul respectiv va deveni capabil să ia în considerare preferinţe QoS complexe.

Pentru a implementa selecţia sub forma unui serviciu, va trebui să definim o interfaţă prin intermediul căreia clienţii să poată trimite cereri şi să poată primi rezultate. Aşa cum am menţionat, clienţii serviciului nostru de selecţie sunt reprezentaţi de framework-uri de compunere dinamică a serviciilor. Deoarece "Binding as a Service" corespunde pasului 2 din procesul de compunere dinamică, framework-ul client are la dispoziţie în momentul apelului specificaţia serviciului compus (construită în pasul 1). Din această specificaţie pot fi extrase în general cu uşurinţă următoarele informaţii care intră în componenţa unei cereri adresate serviciului de selecţie:

- descrierea serviciului compus sub forma unui workflow care specifică o serie de sarcini abstracte;
- pentru fiecare sarcină abstractă, o mulţime de servicii web concrete, care pot efectua sarcina respectivă;
- pentru fiecare atribut QoS, formulele de agregare corespunzătoare fiecărui şablon de compunere utilizat în descrierea workflow-ului unui serviciu compus;

În plus, o cerere adresată serviciului de selecţie trebuie să conţină şi:

- descrierea preferinţelor QoS ale clientului folosind notaţia condiţional lexicografică.

Descrierea preferinţelor QoS este singurul element al cererii care impune modificări în framework-urile de compunere existente în momentul de faţă. Celelate informaţii sunt deja disponibile şi nu necesită decât conversia la formatul impus de serviciul de selecţie.

Întrucât pentru fiecare sarcină abstractă din specificaţia serviciului compus există o mulţime de servicii web candidat, operaţia de selecţie a celei mai bune combinaţii a acestora reprezintă o problemă de optimizare combinatorială multiobiectiv. Un aspect important în implementarea funcţionalităţii "selection as a service" este deci alegerea unui algoritm de optimizare performant. În secţiunea 8.3, vom analiza eficienţa unei soluţii bazate pe algoritmi genetici.

8.1 Abordarea Binding-as-a-Service (BaaS)

În cadrul compunerii serviciilor web, un punct important îl constituie selecţia serviciilor concrete asociate activităţilor din cadrul modelului de orchestrare care stă la baza modelului de compunere. În general, pentru fiecare activitate există mai multe servicii concrete care implementează funcţionalitatea cerută, dar care sunt diferite în ceea ce priveşte caracteristicile nefuncţionale, precum fiabilitatea, costul sau timpul de răspuns. De aceea, calitatea serviciului (QoS) este factorul decisiv în alegerea unui serviciu concret pentru o anumită activitate. În principiu, procesul de selecţie poate utiliza o strategie locală sau una globală. O strategie de optimizare locală presupune selectarea serviciului optim pentru fiecare activitate în parte. Complexitatea acestei strategii este polinomială, dar ea are o serie de neajunsuri, precum inabilitatea de a lua în considerare constrângeri globale între parametrii QoS. Un alt dezavantaj major este faptul că valoarea parametrilor QoS ai serviciului compus obţinut nu este în general optimală [**106**]. Strategia preferată, pe care o

vom adopta şi noi, se bazează pe abordarea planificării globale, în care valorile constrângerilor şi preferinţelor QoS sunt exprimate relativ la valorile QoS globale ale compunerii,

Astfel, primul pas necesar pentru alegerea serviciilor concrete îl constituie calcularea valorii QoS agregate a serviciului compus, pe baza atributelor QoS ale serviciilor componente.

Frameworkurile actuale de compunere a serviciilor web implementează selecţia serviciilor pe baza unor modele care se limitează la workflowuri structurate. Noutatea abordării noastre BaaS este folosirea în acest context a metodei de agregare propuse de Yang et al. [**6**] care depăşeşte limitările frameworkurilor actuale. Această abordare a fost descrisă în cadrul secţiunii 5.3.

Posibilitatea specificării unor reguli de compromis între preferinţele QoS are o importanţă majoră pentru a putea alege serviciile optime pentru un model de compunere. Abordările actuale oferă doar posibilitatea asocierii de priorităţi sau de ponderi preferinţelor clientului. Această metodă este capabilă să descrie doar preferinţe elementare. Prin integrarea metodei QoSPref prezentate anterior, abordarea noastră BaaS oferă flexibilitatea exprimării unor preferinţe complexe care ţin cont de constrângerile şi preferinţele globale.

În continuare, oferim un exemplu care ilustrează problemele ce apar în cadrul compunerii dinamice a serviciilor web bazată pe calitatea serviciilor. Considerăm un sistem de trading online care oferă servicii pentru comercializarea de diverse instrumente financiare. Unele dintre serviciile oferite permit clienţilor tranzacţionarea de acţiuni domestice sau străine. Modelul care descrie procesul business este ilustrat în Fig. 17.

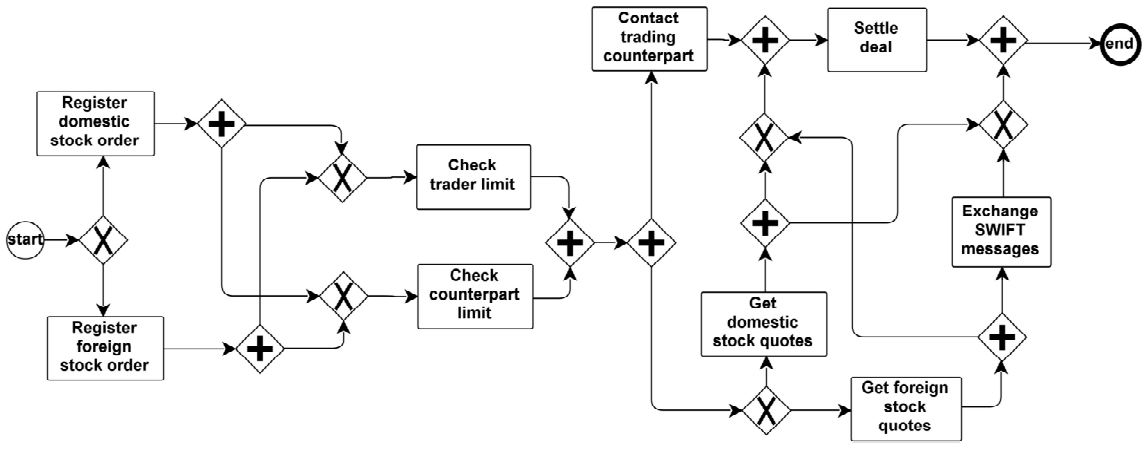


Fig. 17 Model BPMN al procesului de tranzacţionare de acţiuni bancare

Un prim pas îl constituie aducerea workflowului la o formă maxim-structurată, folosind metoda lui Yang et al. [**6**] descrisă in secţiunea 5.3. Modelul de orchestrare din Fig. 18 este din punct de vedere al funcţionalităţii echivalent cu modelul prezentat în Fig. 17, dar partea din stânga a modelului este bine structurată.

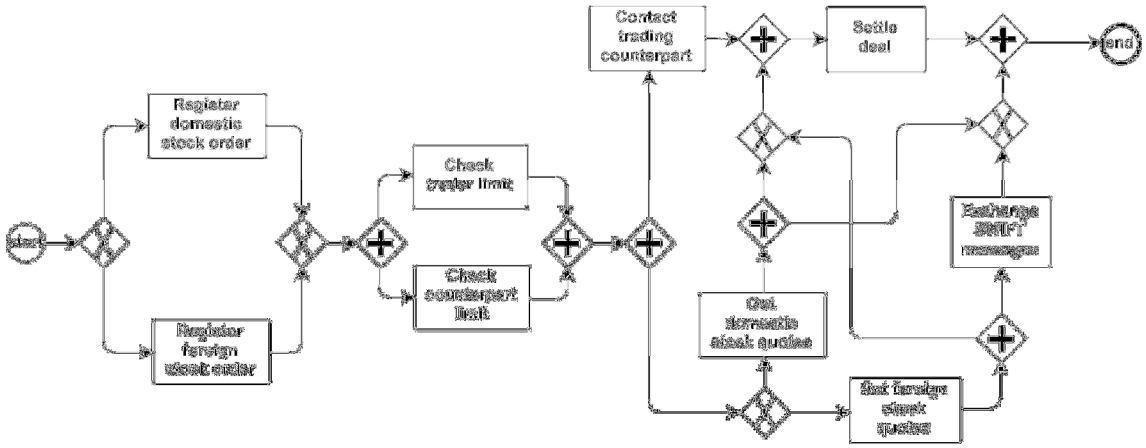


Fig. 18 Model BPMN maxim structurat al procesului de cumpărare de fonduri

Unele dintre procesele acestui acest model, cum ar fi cele pentru înregistrarea unei tranzacţii, reprezintă activităţi interne ale sistemului de tranzacţionare online. Alte procese, cum ar fi cele pentru căutarea cursurilor bancare, necesită interacţiune cu sisteme externe. Sistemul de tranzacţionare online realizează serviciul de cumpărare de fond-uri printr-un serviciu compus. În cazul proceselor care necesită interacţiune cu sisteme externe este necesar ca furnizorii acestor sisteme să ofere funcţionalitatea necesară sub formă de servicii web. De obicei, pentru fiecare activitate există mai multe alternative care implementează funcţionalitatea cerută. Spre exemplu, există multe servicii web care oferă cursul acţiunilor. Sistemul de tranzacţionare online trebuie să aleagă din oferta de servicii, serviciile web concrete care să implementeze fiecare funcţionalitate din cadrul modelului de compunere. Alegerea serviciilor web concrete este un process dinamic, deoarece serviciile web, fiind la rândul lor componente dinamice, pot în orice moment să dispară în timp ce noi servicii web oferind functionalitatea cerută apar.

În exemplul nostru, considerăm că singurele atribute QoS importante pentru sistemul de tranzacţionare sunt: timpul de execuţie, costul şi fiabilitatea. Valoarea QoS a serviciului compus de tranzacţionare este determinată de valorile QoS ale serviciilor componente. În timp ce costul serviciului compus poate fi calculat ca fiind suma costurilor serviciilor componente, nu există nici o metodă simplă pentru calculul timpului de execuţie şi a fiabilităţii. Pentru a putea aloca dinamic servicii concrete activităţilor serviciului compus de cumpărare de fonduri, sistemul de tranzacţionare online trebuie să poată compara în mod automat valorile QoS ale serviciilor componente.

Pentru a arăta de ce realizarea unei comparaţii în cazul serviciilor compuse care au numeroase atribute QoS este o sarcină complicată, detaliem exemplul propus. Proprietarii sistemul de trading online îşi doresc un profit maxim, de aceea consideră preţul ca fiind cel mai important parametru QoS. Cu toate acestea, sunt dispuşi să ignore diferenţe mici de preţ (sub 10 cenţi) dacă serviciul compus având un preţ mai mare are valori mai bune ale timpului de execuţie şi fiabilităţii. Pentru clienţii sistemului este foarte important ca tranzacţiile să fie executate cât mai rapid. De aceea, sistemul de tranzacţii garantează un timp de execuţie sub 30 de secunde pentru cumpărarea unui fond. Pentru fiecare încălcare a

acestei garanţii, proprietarii sistemului de trading trebuie sa plătească o taxă de penalitate a cărei valoare depinde de durata întârzierii. Astfel pentru a compara două servicii web compuse, timpul de execuţie devine un criteriu important în cazul în care unul dintre serviciile compuse are un timp de execuţie care depăşeşte 30 de secunde. Metodele tradiţionale de ordonare bazate pe sumă ponderată sau pe asocierea de priorităţi parametrilor nu sunt potrivite pentru un astfel de scenariu. În continuare vom adresa mai multe probleme care apar în exemplul prezentat.

	$\succ 1$	$\succ 2$	$\succ 3$	$\succ 4$	$\sim$
WS1	1	0	2	0	1
WS2	1	0	1	0	0
WS3	0	0	0	0	0
WS4	1	0	1	0	0
WS5	1	1	0	0	0

8.2 Implementarea Binding-as-a-Service

În prezent există un număr mare de frameworkuri pentru compunerea serviciilor web, bazate pe diferite arhitecturi şi metodologii. Toate aceste frameworkuri au nevoie de funcţionalitatea de binding pentru a aloca servicii concrete activităţilor din cadrul modelului de orchestrare. De aceea se impune oferirea funcţionalităţii de „binding” sub forma unui serviciu web. Clientul tipic al serviciului de binding este un modul al unui framework de compunere de servicii care trebuie să selecteze serviciile optime pentru activităţile din modelului de compoziţie.

Noi oferim o implementare a funcţionalităţii BaaS bazată pe metoda de agregare a lui Yang et al. [3] şi pe metoda noastră de ordonare bazată pe preferinţe QoSPref. Prototipul BaaS este o aplicaţie Java şi este oferit ca proiect open-source la adresa: http://baas.sourceforge.net/.

O cerere a unui serviciu web client către aplicaţia server BaaS trebuie să conţină următoarele informaţii:

1. modelul de orchestrare;
2. lista de atribute QoS;
3. pentru fiecare activitate din cadrul modelului de orchestrare, o listă de servicii concrete care oferă funcţionalitatea cerută;
4. constrângerile QoS
5. preferinţele QoS.

Modelul de orchestrare este reprezentat de un workflow care are ataşat nodurilor sale probabilităţile de execuţie. În cazul în care probabilităţile de execuţie lipsesc, implementarea

noastră va ataşa valori standard, după cum urmează: nodurilor care urmează după o poartă de tip XOR li se va ataşa probabilitatea 1/k, unde k este numărul de drumuri pornind din poarta XOR respectivă. Tuturor celorlalte noduri li se ataşează probabilitatea de execuţie 1.

Pentru specificarea workflourilor poate fi folosit direct formatul BPMN sau un format simplu, definit de noi pe care îl prezentăm în cadrul secţiunii 8.2.1.

Lista atributelor QoS trebuie să conţină informaţii despre categoria de agregare a fiecărui tip de atribut.

Lista serviciilor candidat pentru fiecare activitate trebuie să ofere şi informaţii despre valorile QoS ale fiecărui serviciu. Există situaţii în care nu toate serviciile web posedă toate atributele QoS specificate pentru modelul de compunere. Spre exemplu, un serviciu web compus care oferă funcţionalităţi grafice poate avea atribute QoS care precizează calitatea imaginilor oferite, spre exemplu numărul de culori, sau rezoluţia oferită. Serviciul compus poate avea un serviciu component care oferă funcţionalităţi de sortare, acesta neavând atribute QoS legate de numărul de culori şi rezoluţia imaginilor oferite de serviciul compus. În situaţia în care un atribut QoS lipseşte, implementarea noastră oferă valori standard, în funcţie de categoria de agregare a atributului QoS care nu este specificat.

8.2.1 Specificarea workflowurilor de intrare

Pentru specificarea workflowurilor de intrare definim un format XML, similar fomatului BPMN, dar simplificat pentru a cuprinde doar informaţia necesară pentru specificarea procesului compus fară alte elemente suplimentare, cum ar fi atributele care permit vizualizarea grafică a workflowului. Această abordare a apărut din dorinţa de a permite folosirea diverselor variante consacrate de specificare a workflowurilor, cum ar fi BPMN sau formatul folosit de IBM Websphere Business Modeller, prin conversia acestora la un format simplificat. Transformarea de la formatul proprietar IBM şi de la BPMN la formatul propus de noi se face folosind o transformare XSLT. Metoda poate fi bineînţeles extinsă şi pentru a accepta alte formate prin adăugarea unei noi transformări, adaptate la formatul nou acceptat. Pentru experimentele realizate folosim ca workflowuri de intrare specificaţii realizate în formatul BPMN şi în formatul proprietar IBM. Pentru exemplificare vom adăuga la Anexe câteva fişiere de intrare, în formatele amintite anterior.

Formatul XML definit are următoarea structură:

- În cadrul blocului <flowNodes> sunt definite toate activităţile şi porţile (XOR, AND) din cadrul modelului de orchestrare.
- Blocul <flows> defineşte tranziţiile existente între activităţile şi porţile modelului de compunere. Pentru fiecare tranziţie se poate specifica o probabilitate care poate avea o valore cuprinsă între 0 şi 1. În cazul în care valoarea lipseşte sau nu este cuprinsă în intervalul 0-1, se consideră că probabilitatea nu este specificată, iar aplicaţia va folosi valori standard, calculate după metoda descrisă în secţiunea anterioară.

```
<workflow>
  <flowNodes>
    <flowNode id="task1" type="TASK"/>
    <flowNode id="task2" type="TASK"/>
    <flowNode id="task3" type="TASK"/>
    <flowNode id="task4" type="TASK"/>
    <flowNode id="task5" type="TASK"/>
    <flowNode id="xor1" type="XOR"/>
    <flowNode id="xor2" type="XOR"/>
  </flowNodes>
  <flows>
    <flow id="flow1" source="task1" target="task2" probability="1"/>
    <flow id="flow2" source="task2" target="xor1"/>
    <flow id="flow3" source="xor1" target="xor2"/>
    <flow id="flow4" source="xor2" target="task3" probability="0.3"/>
    <flow id="flow5" source="task3" target="xor1" probability="-1.0"/>
    <flow id="flow6" source="xor2" target="task4" probability="0.7"/>
    <flow id="flow7" source="task4" target="task5" probability="-1.0"/>
  </flows>
</workflow>
```

8.3 Alegerea serviciilor web concrete folosind un algoritm genetic

Optimizarea valorii QoS agregate a unui serviciu compus este o problemă NP-hard. O căutare exhaustivă nu este posibilă decât pentru modele de compoziţie simple, cu un număr mic de task-uri şi un număr mic de servicii disponibile pentru fiecare task. Implementarea noastră BaaS foloseşte un algoritm genetic pentru a găsi combinaţia optimă de servicii concrete care implementează modelul de orchestrare abstract. Acest algoritm va fi descris în paragrafele următoare.

Un algoritm genetic menţine o populaţie de cromozomi, în care fiecare cromozom codifică o posibilă soluţie a problemei. În cazul nostru, un cromozom codifică o posibilă mapare a serviciilor web la task-uri din modelul de orchestrare, aşa cum se prezintă în Fig. 19.

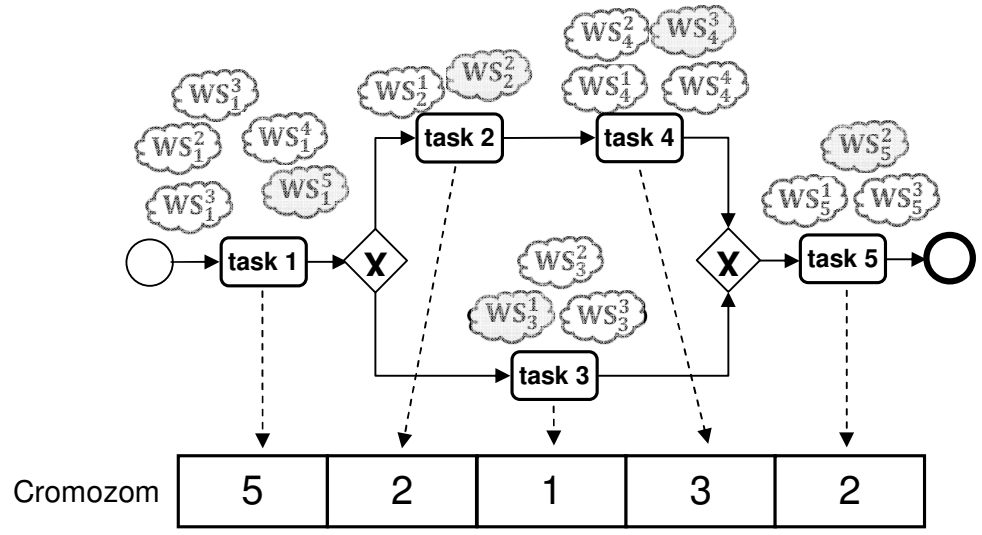


Fig. 19. Structura unui cromozom care codifică o posibilă mapare a serviciilor web la task-uri din modelul de orchestrare

Cromozomul este structurat ca un vector cu n elemente, unde n este numărul de task-uri din modelul de orchestrare. Valoarea elementului i din acest vector este un număr întreg care reprezintă indicele serviciului component asignat task-ului i. De exemplu, în Fig. 19 există 3 servicii care implementează funcţionalitatea task-ului cu numărul 5: WS_5^1, WS_5^2 şi WS_5^3. Soluţia codificată de cromozom mapează cel de-al doilea serviciu candidat (WS_5^2) la task-ul cu numărul 5. De aceea, valoarea celui de-al 5-lea element al cromozomului este 2.

Pentru recombinare, algoritmul nostru genetic foloseşte operatorul *two-point crossover*. Mutaţiile sunt realizate alegând în mod aleator un task şi schimbând indicele serviciului component asociat acestui task.

O particularitate a abordării noastre pentru specificarea preferinţelor este faptul că fitness-ul unei soluţii poate fi evaluat numai în contextul unei anumite populaţii, deoarece algoritmul de ranking (QoSPref) trebuie să execute comparaţii ale tuturor perechilor de soluţii candidat existente. Din acest motiv, nu este posibilă calcularea unei valori absolute a fitness-ului unei soluţii. Algoritmul nostru genetic calculează fitness-ul unei soluţii pe baza indicelui acesteia în lista de ranking obţinută de algoritmul QoSPref pentru populaţia curentă: soluţia din fruntea listei va avea fitness-ul maxim, iar cea de pe ultima poziţie fitness-ul minim. Pseudocodul procedurii de evaluare a fitness-ului este prezentat în Fig. 20.

```
procedure evaluateFitness()
        for i ← 1 .. n do
                qos[i] ← computeAggregateQoS(solution[i])
        end for
        for i ← 1 .. n-1 do
                for j ← i+1 .. n do
                        compResult[i][j] ← compareServices(solution[i], solution[j])
                end for
        end for
        createScoreVectors()
        rankList ← list of solutions sorted using compareScores()
        for i ← 1 .. n do
                fitness[i] ← populationSize - (index of solution[i] in rankList)
        end for
end procedure
```

Fig. 20. Procedura pentru evaluarea fitness-ului soluţiilor candidat

Funcţia *computeAggregateQoS* din pseudocodul de mai sus foloseşte metoda descrisă în secţiunea 5.3 pentru a estima costul serviciului compus pe baza valorilor QoS ale serviciilor componente. Funcţiile *compareSevices*, *createScoreVectors* şi *compareScores* au fost definite în secţiunea 7.2. Variabila *populationSize* este un parametru configurabil al algoritmului. În secţiunea 8.3.1 sunt prezentate rezultatele experimentale obţinute pentru diferite valori ale parametrului *populationSize*.

Există o serie de condiţii care pot fi combinate pentru a cauza terminarea algoritmului nostru genetic:

1. atingerea unui număr maxim de generaţii;
2. atingerea unei limite maxime a timpului de execuţie;
3. incapacitatea de a îmbunătăţi soluţia curentă în timpul ultimelor k generaţii.

Întrucât nu există o valoare absolută a fitness-ului unei soluţii, este dificil de verificat apariţia celei de-a 3-a condiţii. Pentru a rezolva această problema, algoritmul nostru genetic menţine o listă a celor mai bune soluţii găsite pâna la momentul curent. La sfârşitul fiecărei generaţii, cea mai bună soluţie din generaţia curentă este căutată în lista celor mai bune soluţii. Dacă nu este deja prezentă, ea este adăugată la această listă şi se calculează rangul ei folosind algoritmul QoSPref. Dacă noua soluţie este pe primul loc, înseamnă că am găsit o soluţie îmbunătăţită. Dimensiunea listei celor mai bune soluţii este limitată de o valoare configurată ca parametru al algoritmului. Dacă în urma adăugării noii soluţii dimensiunea listei depăşeşte această valoare limită, elementul cu cel mai slab rang va fi eliminat din listă.

8.3.1 Rezultate experimentale

În cadrul experimentelor noastre, am folosit 21 de modele de orchestrare din repositoriul public Oryx (disponibil la adresa https://code.google.com/p/oryx-editor/) şi 10 modele de orchestrare din biblioteca de procese business IBM BIT (disponibilă la adresa http://www.zurich.ibm.com/csc/bit/downloads.html). Modelele considerate au între 5 şi 20 de taskuri şi numărul de porţi variază între 0 şi 16. Am ales ca execuţia algoritmului să fie terminată atunci când după 20 de generaţii consecutive nu s-a obţinut nici o îmbunătăţire. Stabilim că dimensiunea maximă a listei cu cele mai bune soluţii este 10.

Am efectuat experimente folosind populaţii de dimensiunea 20, 50 şi 100.

Pentru fiecare dimensiune a populaţiei şi pentru fiecare model de orchestrare am rulat 100 de iteraţii folosind un set diferit de servicii web candidat pentru fiecare iteraţie. Fiecare set de servicii candidat a fost creeat generând între 2 şi 7 alternative pentru fiecare activitate din cadrul modelului de orchestrare curent. Pentru fiecare model de orchestrare şi pentru fiecare dimensiune a populaţiei prezentăm numărul mediu de generaţii în Tab. 1.

Din rezultatele experimentale putem sa observăm că şi o populaţie de 20 de invidivizi este suficientă pentru a atinge condiţia de terminare a rulării în mai puţin de 50 de generaţii. În timp ce, de obicei, folosirea unui populaţii mari conduce la un număr mic de generaţii, diferenţa nu este considerabilă faţă de folosirea de populaţii mici. În cazul modelului de orchestrare 16, o populaţie mare poate conduce chiar la rezultate mai proaste.

Trebuie de asemenea remarcat că, deşi numărul de generaţii necesare pentru atingerea condiţiei de terminare a rulării algoritmului este dependent de numărul de activităţi şi porţi, acest număr este influenţat şi de alţi factori, cum ar fi topologia modelului de orchestrare.

Aşa cum se observă din Fig. 21, timpul de răspuns este considerabil mai bun în cazul folosirii unei populaţii de 20 de indivizi. În această figură, numărul de noduri reprezinta suma numărului de activităţi şi a numărului de porţi. Timpul de răspuns a fost obţinut calculând media duratelor corespunztătoare intrărilor cu acelaşi număr de noduri din Tab. 5. (De exemplu, modelele 24, 27 şi 31 corespund unor modele de orchestrare cu 25 de noduri.)

Model	Număr de activităţi	Număr de porţi	Populaţie=20		Populaţie=50		Populaţie=100	
			Durata (ms.)	Număr de generaţii	Durata (ms.)	Număr de generaţii	Durata (ms.)	Număr de Generaţii
1	5	4	152	8.38	236	5.79	495	3.53
2	6	0	97	10.33	194	7.36	470	5.28
3	6	0	99	11.09	196	7.24	466	5.00
4	6	0	104	11.19	190	6.76	460	4.78
5	6	0	98	10.20	190	6.77	460	4.85
6	6	0	103	11.57	192	6.69	460	4.77
7	6	4	176	11.11	267	6.55	562	4.92
8	6	6	277	13.08	342	7.62	601	5.48
9	6	7	201	10.84	310	7.49	635	5.80
10	7	5	222	13.38	321	8.76	680	7.45
11	7	7	290	16.60	391	10.18	739	7.55
12	8	10	439	28.59	650	33.84	1185	46.76
13	8	3	212	16.21	338	11.49	716	9.38
14	8	3	230	20.38	349	13.08	722	10.03
15	9	4	294	24.58	575	40.19	931	38.32
16	9	4	262	22.39	402	15.79	795	12.16
17	9	8	352	21.50	469	13.88	896	11.11
18	9	8	430	26.12	824	45.97	1246	42.24
19	9	10	462	20.70	720	14.35	1403	11.90
20	10	6	321	21.94	467	15.83	937	12.92
21	10	6	336	21.89	482	15.03	938	11.93
22	10	9	367	22.54	533	17.71	982	13.10
23	11	6	352	24.06	519	17.82	1025	14.40
24	11	14	599	26.50	722	19.21	1302	16.14
25	12	5	416	29.04	613	22.90	1166	17.75
26	13	8	505	29.82	694	21.88	1477	19.03
27	13	12	663	35.12	900	26.75	1632	21.56
28	14	4	395	31.54	616	25.06	1249	20.64
29	15	16	828	30.67	1346	42.96	2174	44.82
30	18	12	928	39.96	1506	48.06	2206	40.26
31	20	5	650	38.90	1031	44.50	1400	32.12

Tab. 5 Rezultate experimentale

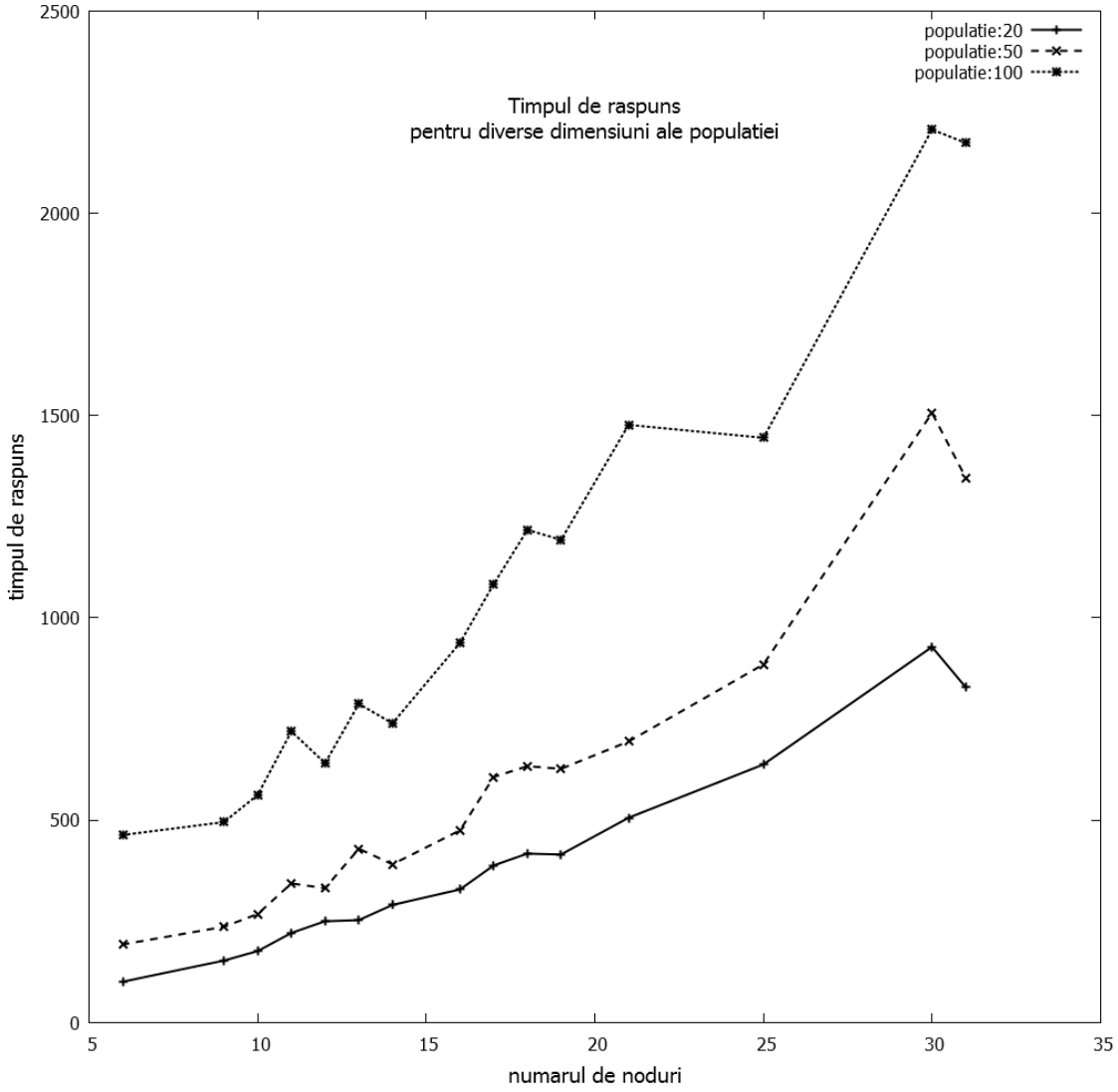


Fig. 21. Dependenţa timpului de răspuns al algoritmului de numărul de noduri al modelelor de orchestrare pentru diferite dimensiuni ale populaţiei.

9. Extensia ontologiei OWL-Q pentru specificarea preferinţelor

Pentru a putea folosi flexibilitatea metodei de exprimare a cerinţelor QoS prezentate în secţiunea anterioară în cadrul descoperii semantice, după ce am analizat ontologiile de preferinţe existente în cadrul secţiunii 4.1.6, am decis să extindem ontologia OWL-Q .

9.1 Specificarea cerinţelor QoS în cadrul ontologiei OWL-Q

În cadrul OWL-Q cererile şi oferta atributelor QoS sunt definite de faţeta QoSSpec. Clasa QoSSpec conţine descrierea QoS a serviciului web. Această faţetă conţine diferite atribute QoS de bază cum ar fi costul serviciului şi moneda în care este exprimat, atributele de securitate şi protocoalele acceptate, perioada de valabilitate a ofertei. QoSSpec cuprinde două clase separate QoSOffer şi QoSDemand care sunt folosite de furnizorii de servicii web, respectiv de clienţii serviciilor web pentru a specifica folosind aceeaşi metodă, într-un mod simetric, constrângerile QoS.

Clienţii serviciilor web pot spefica constrângerile QoS folosind clasa QoSDemand şi pot asocia ponderi metricilor căutate folosind clasa QoSSelection. Clasa QoSSelection conţine o lista de intrări <metrică, pondere>. Ponderea asociată poate avea valorea 2.0 dacă este o constrângere obligatorie sau o valoare aflată în intervalul (0:0; 1:0) dacă este o preferinţă. O intrare <metrică, pondere> este definită de clasa QosSelectElem, iar clasa QoSSelectElemeList conţine o listă de elemente QosSelectElem. Pentru a ilustra mai bine clasele folosite pentru specificarea cerinţelor QoS şi a relaţiilor dintre ele Fig. 22 arată graful care reprezintă faţeta QosSpec. Acest graf a fost generat folosind editorul Protégé.

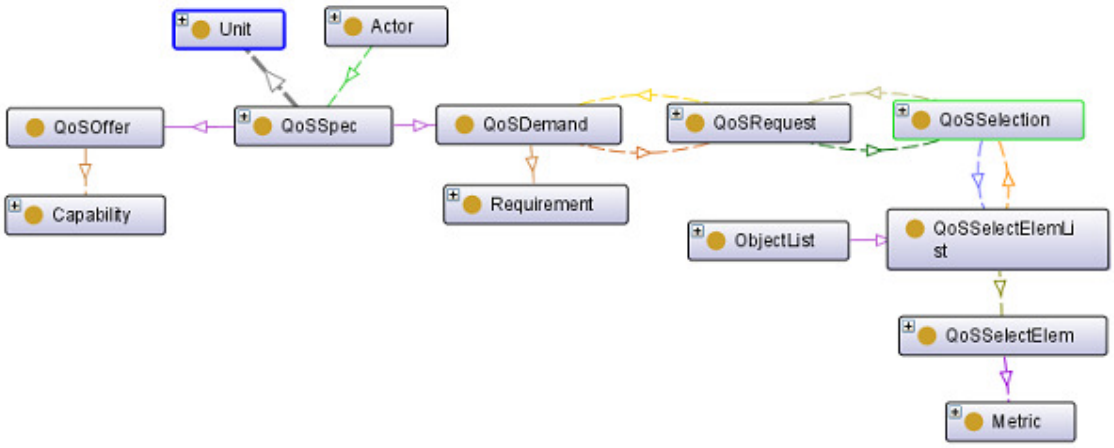


Fig. 22. Faţeta QoSSpec a ontologiei OWL-Q

OWL-Q conţine şi o faţetă care descrie tipurile de valori pe care un atribut QoS le poate avea. Clasa MetricValueType este împărţită în două subclase, care indică direcţia valorilor atributului QoS. Tipul PositivelyMonotonic indică faptul că valorile sunt ordonate de la cea mai scazută către cea mai mare valoare iar valoarea maximă este considerată ca fiind cea mai bună valoare. Tipul de valoare NegativelyMonotonic are o direcţie contrară, valoarea maximă de calitate fiind asociată valorii minime. Pentru a exemplifica, atributul QoS care măsoară disponibilitatea unui sistem este o valoare de tipul PositivelyMonotonic, în timp ce timpul de răspuns sau costul unui serviciu sunt valori de tipul NegativelyMonotonic deoarece valorile mici indică o calitate mai bună. Astfel atunci când specificăm o preferinţă folosind metoda propusă de noi QosPref, nu avem nevoie să specificăm explicit direcţia valorilor optimale pentru atributul respectiv acest lucru fiind deja specificat folosind această faţetă.

9.2 Extensia OWL-Q propusă

Pentru descrierea calităţii ofertei de servicii, realizată de către furnizorii acestora, vom folosi în continuare clasa QoSOffer din cadrul OWL-Q. Noi extindem faţeta QosSpec adăugând noi clase pentru selecţia preferinţelor.

Definim o noua clasă QoSSelectionWithTradeoffs, care va permite specificarea unei liste de preferinţe, analog notaţiei oferite de metoda QoSPref. Constângerile sunt specificate folosind clasa QoSDemand, la fel ca în OWL-Q.

O preferinţă este descrisă de clasa QoSPreference care defineşte formatul unei reguli din blocul de preferinţe. În cadrul metodei QosPref, o regulă din blocul de preferinţe conţine trei componente: o condiţie opţională, atributul QoS care stă la baza comparării şi direcţia valorilor optimale pentru atributul respectiv. Deoarece pentru specificarea direcţiei valorilor optimale vom folosi clasa MetricValueType, o regulă din cadrul blocului de preferinţe va conţine doar condiţia opţională şi atributul comparat. O intrare din cadrul blocului de preferinţe are următorul format: <(condiţie opţională) atribut QoS>.

Pentru specificarea condiţiei opţionale definim trei operatori logici: AT_LEAST_ONE (operator OR), EXACTLY_ONE (operator XOR), ALL (operator AND) şi operatorul aritmetic DIFF (-), aşa cum sunt definiţi în Fig. 12, pe care îi includem în cadrul ontologiei. Condiţia opţională este specificată sub forma <operator, atribut >.

În cadrul metodei QoSPref ordinea preferinţelor este importantă pentru algoritmul de sortare, prima preferinţă specificată fiind considerată ca fiind cea mai importantă. Astfel, ordinea în care sunt definite preferinţele din cadrul listei de preferinţe este importantă. Pentru ca notaţia semantică să fie explicită şi uşor de urmărit ataşămîn mod explicit prioritatea preferinţei fiecărei intrări din lista de preferinţe. Astfel clasa QoSPreference este definită de următoarea notaţie: <prioriatate, (condiţie) atribut>.

Extensia ontologiei este expusă în Fig. 23.

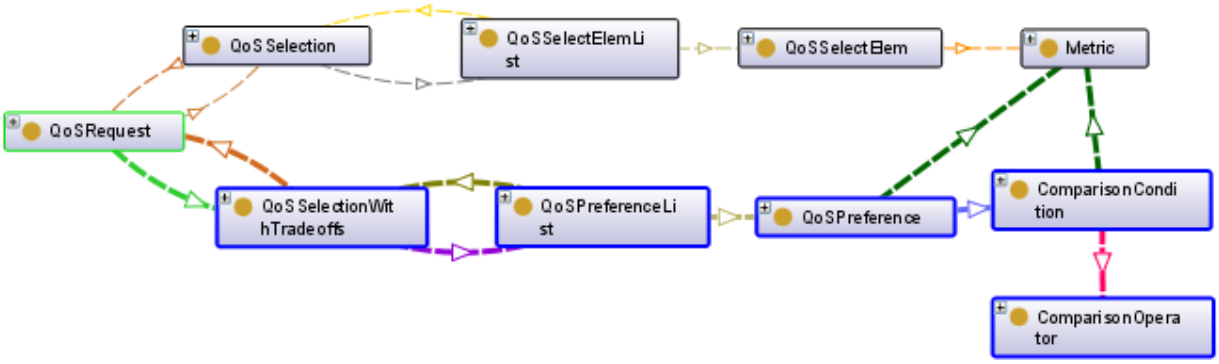


Fig. 23. Extensia OWL-Q compatibilă cu abordarea QoSPref

Clasa QoSRequest va putea folosi noua clasă QoSSelectionWithTradeoffs sau formatul vechi de specificare a preferinţelor QoSSelection. Clientul poate decide ce clasă pentru specificarea preferinţelor va folosi, în funcţie de complexitatea cerinţelor sale.

Extensia oferită de noi îi permite clientului să folosească o notaţie puternică pentru a specifica cerinţe QoS complexe, într-un cadru semantic.

10. Concluzii

În cadrul tezei de doctorat am introdus o nouă abordare, inovatoare, pentru compunerea serviciilor web, ţinând cont de calitatea serviciilor (QoS). Abordarea propusă oferă un cadru complet, care adresează toate etapele necesare realizării unei compuneri de servicii web, plecând de la specificarea cerinţelor clientului, descoperirea serviciilor web într-un mod semantic bazat pe cerinţele clientului, specificarea modelului de compunere şi găsirea serviciilor concrete optime care împreună realizează serviciul compus optim pentru cerinţele exprimate .

În prezent, arhitecturile bazate pe servicii beneficiază de un interes major. Oferta de servicii web este foarte mare, iar problema compunerii dinamice a serviciilor web ţinând cont de calitatea acestora este o direcţie de cercetare actuală de mare interes. Cu toate acestea, nu există în momentul de faţă un standard care să se fi impus.

În cadrul tezei de doctorat analizăm cele mai importante abordări existente şi identificăm problemele acestora. Soluţia propusă de noi se bazează pe observaţia că posibilitatea exprimării de către clienţi a unor cerinţe de calitate QoS complexe, care să permită specificarea atât a cerinţelor cât şi a preferinţelor de calitate şi a unor reguli de compromis, este de o importanţă crucială pentru găsirea soluţiei optime pentru fiecare client în parte.

Am oferit o metodă de specificare a preferinţelor, bazată pe un limbaj intuitiv, dar în acelaşi timp expresiv, numit QoSPref, care permite definirea unor cerinţe de calitate complexe. Am observat că descoperirea serviciilor web necesare, bazată pe o căutare sintactică are dezavantaje şi am ales să folosim o căutare semantică bazată atât pe criterii funcţionale cât şi de calitate. În acest scop am analizat ontologiile QoS existente şi am extins ontologia OWL-Q, care permite specificarea simetrică a parametrilor de calitate ai serviciilor şi a cerinţelor de calitate ale clientului. Extensia propusă permite specificarea preferinţelor şi a regulilor de compromis folosind limbajul QoSPref. Astfel, metoda de specificare a preferinţelor clienţilor este integrată şi în cadrul procesului de selecţie semantică.

Pentru a ordona serviciile în funcţie de preferinţele specificate de client, am propus un algoritm simplu, care nu se bazează pe tehnici multicriteriale complexe şi care realizează o ordonare totală a alternativelor. Algoritmul poate face faţă unor preferinţe intranzitive. În acest moment, algoritmul realizează o optimizare locală a etapei de căutare a serviciilor web.

Am oferit motorul de selecţie sub forma unei biblioteci Java, pe care am pus-o la dispoziţie sub forma unui proiect open-source denumit QoSPref, accesibil la adresa http://qospref.sourceforge.net. Un scenariu tipic de utilizare a acestei biblioteci este ca parte integrantă a unui framework SOA, care accesează motorul de selecţie prin intermediul interfeţei de programare (API) puse la dispoziţie de biblioteca noastră. Framework-ul SOA, care joacă rol de client al motorului de selecţie, trebuie să ofere ca date de intrare informaţii legate de serviciile web candidat, de atributele QoS ale acestora, precum şi de preferinţele QoS care trebuie luate în considerare.

În plus, pentru a ilustra modul de utilizare a bibliotecii şi a uşura înţelegerea funcţionării motorului de selecţie, QoSPref oferă şi o interfaţă grafică prin intermediul căreia pot fi accesate funcţiile oferite de bibliotecă.

Pentru a realiza o optimizare globală a compunerii de servicii, calculăm valorile agregate ale atributelor QoS, folosind funcţiile de agregare definite în cadul lucrării [**74**]. Algoritmul anterior de ordonare a alternativelor este folosit de această dată pentru a compara valorile de calitate agregate pentru diferitele posibile soluţii de compunere.

Pentru alegerea serviciilor concrete care implementează activităţile existente într-un model de compunere am propus, sub forma unui serviciu web, metoda BaaA (Binding-as-a-Service). Această metodă aduce o îmbunătăţile majoră faţa de metodele existente, deoarece permite ca workflowuri de intrare atât workflowuri structurate, cât şi nestructurate, fiind astfel foarte flexibilă şi depăşind constrângerile celorlalte abordări existente pentru această problemă. BaaS necesită ca parametri de intrare: specificaţia workflowului abstract, specificaţie ce poate fi realizată în BPMN, în formatul proprietar IBM Business Modeller sau într-un format simplu, propriu metodei noastre, lista de cerinţe QoS a clientului şi lista serviciilor web concrete care implementează fiecare activitate din cadrul workflowului abstract. Pentru calculul valorilor QoS agregate am folosit metoda eficientă de calcul introdusă în [**6**]. Această metodă calculează valorile QoS agregate pe baza unor modele de orchestrare care pot conţine şi componente nestructurate.

Implementarea funcţionalităţii BaaS bazată pe metoda de agregare a lui Yang et al. [3] şi pe metoda noastră de ordonare bazată pe preferinţe QoSPref este oferită ca proiect open-source la adresa: http://baas.sourceforge.net/. Proiectul este o aplicaţie Java care oferă funcţionalitatea implementată sub forma unui serviciu web. Clientul tipic al serviciului de binding este un modul al unui framework de compunere de servicii care trebuie să selecteze serviciile optime pentru activităţile din modelul de compoziţie.

Deoarece optimizarea valorii QoS agregate a unui serviciu compus este o problemă NP-hard, o căutare exhaustivă a serviciilor concrete optime nu este posibilă decât pentru modele de compoziţie simple cu un număr mic de task-uri şi un număr mic de servicii disponibile pentru fiecare task. Pentru a găsi combinaţia optimă de servicii concrete care implementează modelul de orchestrare abstract am folosit un algoritm genetic. Algoritmul genetic foloseşte ca funcţie de fitness metoda de ordonare propusă de noi.

Rezultatele experimentale, bazate pe workflowuri de intrare specificate în BPMN din cadrul repositorului academic public Oryx şi a unor workflowuri din cadrul repositorului public BIT, specificate în limbajul proprietar al IBM Business Modeller, demonstrează eficacicatea metodei propuse.

11. Anexe

În această anexele 11.1 şi 11.2 prezentăm gramatica folosită de analizorul sintactic şi cel lexical al bibliotecii QosPref. Notaţia folosită este cea caracteristică frameworkului ANTLR. Anexa 11.3 prezintă cele două formate acceptate de BaaS (BPMN şi formatul proprietar IBM Business Modeller) pentru specificarea workflowurilor de intrare.

11.1 Parser

```
specification : quantities definitions constraints? preferences;

quantities :  QUANTITIES '{' quantity (',' quantity )* '}';

quantity : ID (':' type)? ;

type : TYPE | enumeration;

enumeration : ENUM '(' ID (',' ID)* ')';

definitions : (definition)* ;

definition : DEFINITION ID '=' condExpr ;

constraints :  CONSTRAINTS '{' constraint (',' constraint )* '}' ;

constraint : condExpr;

preferences :  PREFERENCES '{' preference (',' preference )* '}' ;

preference :  condition addExpr direction;

condition : ('[' condExpr']')? ;

direction : (':' DIRECTION)? ;

condExpr : andExpr ('|' andExpr)* ;

andExpr : eqExpr ('&' eqExpr)* ;

eqExpr : relExpr (EQ_OP relExpr)* ;

relExpr : addExpr (REL_OP addExpr)* ;

addExpr : multExpr (ADD_OP multExpr)* ;

multExpr : unaryExpr (MUL_OP unaryExpr)* ;

unaryExpr : ADD_OP unaryExpr
        | PREF_OP unaryExpr
        | unaryExprPlain ;

unaryExprPlain : '!' unaryExpr | primary ;

primary : '(' condExpr ')' | func1 | func2 | ID | INTEGER | REAL | BOOLEAN | STRING  ;

func1 : MATH_ONE '(' addExpr ')' ;

func2 : MATH_TWO '(' addExpr ',' addExpr ')' ;
```

11.2 Lexer

```
QUANTITIES : 'quantities' ;

CONSTRAINTS : 'constraints' ;

DEFINITION : 'def' ;

PREFERENCES : 'preferences' ;

ENUM : 'enum' ;

TYPE : 'integer' | 'real' | 'boolean' ;

MATH_ONE : 'sin' | 'cos' | 'asin' | 'acos' | 'tan' | 'atan' | 'sinh' | 'cosh' | 'exp'
                | 'sqrt' | 'log' | 'log10' | 'abs' | 'ceil' | 'floor' | 'round' ;

MATH_TWO : 'pow' | 'atan2' | 'min' | 'max' ;

PREF_OP : 'DIFF' | 'ALL' | 'EXACTLY_ONE' | 'AT_LEAST_ONE' ;

ADD_OP : '+' | '-' ;

MUL_OP : '*' | '/' | '%' ;

EQ_OP : '=' | '<>' | '!=' ;

REL_OP : '<=' | '<' | '>=' | '>' ;

BOOLEAN : 'true' | 'false' ;

DIRECTION : 'high' | 'low' ;

ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')* ;

INTEGER : '0'..'9'+ ;

REAL
   :   ('0'..'9')+ '.' ('0'..'9')* EXPONENT?
   |   '.' ('0'..'9')+ EXPONENT?
   |   ('0'..'9')+ EXPONENT;

STRING :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"' ;

EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

ESC_SEQ
   :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
   |   UNICODE_ESC
   |   OCTAL_ESC;

OCTAL_ESC
   :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
   |   '\\' ('0'..'7') ('0'..'7')
   |   '\\' ('0'..'7');

UNICODE_ESC :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT ;
```

11.3 Specificaţii workflowuri de intrare

11.3.1 Format IBM Websphere Business Modeller - din cadrul bibliotecii publice BIT

```
<wbim:processModel>
  <wbim:processes>
   <wbim:process name="s00000021##s00000698">
    <wbim:description> ... </wbim:description>
    <wbim:inputs>
     <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="0"
name="s00000699"/>
     <wbim:inputCriterion name="s00000700">
      <wbim:input name="s00000699"/>
     </wbim:inputCriterion>
    </wbim:inputs>
    <wbim:outputs>
     <wbim:output associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="0"
name="s00000701"/>
     <wbim:output associatedData="s00000005##s00000447" isOrdered="false" isUnique="false" maximum="1" minimum="0"
name="s00000702"/>
     <wbim:output associatedData="s00000005##s00000435" isOrdered="false" isUnique="false" maximum="1" minimum="0"
name="s00000703"/>
     <wbim:outputCriterion name="s00000704">
      <wbim:output name="s00000701"/>
      <wbim:output name="s00000702"/>
      <wbim:output name="s00000703"/>
      <wbim:relatedInputCriteria>
       <wbim:inputCriterion name="s00000700"/>
      </wbim:relatedInputCriteria>
     </wbim:outputCriterion>
    </wbim:outputs>
    <wbim:flowContent>
     <wbim:startNode name="s00000705"/>
     <wbim:endNode name="s00000706"/>
     <wbim:decision isInclusive="false" name="s00000707">
      <wbim:inputBranch name="s00000708">
       <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="1"
name="s00000709"/>
       <wbim:input name="s00000710"/>
      </wbim:inputBranch>
      <wbim:outputBranch name="s00000711">
       <wbim:output name="s00000712"/>
       <wbim:output name="s00000713"/>
       <wbim:condition name="s00000714"/>
       <wbim:operationalData>
        <wbim:probability>
         <wbim:literalValue>60.0</wbim:literalValue>
        </wbim:probability>
       </wbim:operationalData>
      </wbim:outputBranch>
      <wbim:outputBranch name="s00000715">
       <wbim:output name="s00000716"/>
       <wbim:output name="s00000717"/>
       <wbim:condition name="s00000718"/>
       <wbim:operationalData>
        <wbim:probability>
         <wbim:literalValue>40.0</wbim:literalValue>
        </wbim:probability>
       </wbim:operationalData>
      </wbim:outputBranch>
     </wbim:decision>
    <wbim:merge name="s00000730">
      <wbim:inputBranch name="s00000731">
       <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="1"
name="s00000709"/>
       <wbim:input name="s00000732"/>
      </wbim:inputBranch>
     <wbim:inputBranch name="s00000735">
```

```
        <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="1"
name="s00000736"/>
        <wbim:input name="s00000737"/>
       </wbim:inputBranch>
  <wbim:merge name="s00000742">
       <wbim:inputBranch name="s00000731">
        <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="1"
name="s00000710"/>
        <wbim:input name="s00000709"/>
       </wbim:inputBranch>
       <wbim:inputBranch name="s00000733">
        <wbim:input associatedData="s00000005##s00000499" isOrdered="false" isUnique="false" maximum="1" minimum="1"
name="s00000737"/>
        <wbim:input name="s00000740"/>
       </wbim:inputBranch>
       <wbim:outputBranch name="s00000741">
        <wbim:output name="s00000717"/>
        <wbim:output name="s00000713"/>
       </wbim:outputBranch>
      </wbim:merge>
      <wbim:callToTask callSynchronously="true" name="s00000743" task="s00000023##s00000743">
       <wbim:additionalInput name="s00000709">
        <wbim:inputCriterion name="s00000700"/>
       </wbim:additionalInput>
       <wbim:additionalOutput name="s00000713">
        <wbim:outputCriterion name="s00000704"/>
       </wbim:additionalOutput>
      </wbim:callToTask>

  <wbim:callToTask callSynchronously="true" name="s00000746" task="s00000021##s00000746">
       <wbim:additionalInput name="s00000709">
        <wbim:inputCriterion name="s00000700"/>
       </wbim:additionalInput>
       <wbim:additionalOutput name="s00000713">
        <wbim:outputCriterion name="s00000704"/>
       </wbim:additionalOutput>
      </wbim:callToTask>
      <wbim:connection name="s00000749">
       <wbim:source node="s00000705"/>
       <wbim:target contactPoint="s00000709" node="s00000742"/>
      </wbim:connection>
    <wbim:annotation>
       <wbim:annotationText> ... </wbim:annotationText>
       <wbim:annotatedNode name="s00000705"/>
      </wbim:annotation>
      <wbim:annotation>
       <wbim:annotationText> ... </wbim:annotationText>
       <wbim:annotatedNode name="s00000744"/>
      </wbim:annotation>
      <wbim:annotation>
       <wbim:annotationText> ... </wbim:annotationText>
       <wbim:annotatedNode name="s00000743"/>
      </wbim:annotation>
     </wbim:flowContent>
   </wbim:process>
```

11.3.2 Format BPMN – din cadrul bibliotecii publice Oryx

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions id="oryx_80842717-ec89-4872-9f1f-9eb6f9db6f2b" typeLanguage="http://www.w3.org/2001/XMLSchema"
expressionLanguage="http://www.w3.org/1999/XPath" targetNamespace="http://www.omg.org/bpmn20"
xmlns="http://schema.omg.org/spec/BPMN/2.0" xmlns:bpmndi="http://bpmndi.org">
   <process id="oryx_bea118bb-4b0c-4b79-8318-17be7b8a80a2">
      <laneSet id="oryx_de3f5a78-ecad-448f-b4a8-b780de6b04fd">
         <lane name="DefaultLane" id="oryx_e50e3424-d2c9-4135-b4f8-5aa0376aa8c8">
            <flowElementRef>oryx_877974F0-BD10-426E-B379-CECC4E5D5A09</flowElementRef>
            <flowElementRef>oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260</flowElementRef>
            <flowElementRef>oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0</flowElementRef>
            <flowElementRef>oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7</flowElementRef>
            <flowElementRef>oryx_0878603E-954F-437B-B5B0-5D760A250043</flowElementRef>
            <flowElementRef>oryx_9B317250-16B6-4C71-AE1B-99B1449A9953</flowElementRef>
            <flowElementRef>oryx_148B6192-A05B-420F-8258-DDBBFDCCCC8F</flowElementRef>
```

```
        <flowElementRef>oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281</flowElementRef>
</lane>
    </laneSet>
    <startEvent name="start" id="oryx_877974F0-BD10-426E-B379-CECC4E5D5A09"/>
    <task completionQuantity="1" startQuantity="1" isForCompensation="false" name="Check Stock" id="oryx_44D5F02C-3B21-4CF1-
8B8C-6FCDAF03E260"/>
    <exclusiveGateway gatewayDirection="diverging" name="g1" id="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0"/>
    <task completionQuantity="1" startQuantity="1" isForCompensation="false" name="Reject Order" id="oryx_3513983D-27F0-4F3F-
BA6E-FFA6A98788A7"/>
    <exclusiveGateway gatewayDirection="converging" name="g4" id="oryx_0878603E-954F-437B-B5B0-5D760A250043"/>
    <parallelGateway gatewayDirection="converging" name="g3" id="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953"/>
    <task completionQuantity="1" startQuantity="1" isForCompensation="false" name="Ship Goods" id="oryx_148B6192-A05B-420F-
8258-DDBBFDCCCC8F"/>
    <parallelGateway gatewayDirection="diverging" name="g2" id="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281"/>
    <task completionQuantity="1" startQuantity="1" isForCompensation="false" name="Conform Order" id="oryx_C2D9B9C4-0C1E-484B-
A1BD-F721D49E64CE"/>
    <task completionQuantity="1" startQuantity="1" isForCompensation="false" name="Send Invoice" id="oryx_C8AF8438-5CCA-4FB0-
84B3-FDBF8CA62D0C"/>
    <endEvent name="end" id="oryx_DDEA5D3F-AB0D-49C3-969B-518115212378"/>
    <sequenceFlow targetRef="oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260" sourceRef="oryx_877974F0-BD10-426E-B379-
CECC4E5D5A09" name="" id="oryx_576C58CB-431F-4A12-AE70-F5E61F56A85E"/>
    <sequenceFlow targetRef="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0" sourceRef="oryx_44D5F02C-3B21-4CF1-8B8C-
6FCDAF03E260" name="" id="oryx_8AF6F2B1-E35A-4ED6-A80F-524A2CC052B0"/>
    <sequenceFlow targetRef="oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7" sourceRef="oryx_C944EAF3-085E-4D6D-8809-
FF0AA6F7EDC0" name="" id="oryx_E56E1E06-6D0F-4EF3-9059-C1D503E552E9"/>
   <sequenceFlow targetRef="oryx_0878603E-954F-437B-B5B0-5D760A250043" sourceRef="oryx_3513983D-27F0-4F3F-BA6E-
FFA6A98788A7" name="" id="oryx_5A410BBA-C099-406C-8386-E479E0764ED6"/>
    <sequenceFlow targetRef="oryx_DDEA5D3F-AB0D-49C3-969B-518115212378" sourceRef="oryx_0878603E-954F-437B-B5B0-
5D760A250043" name="" id="oryx_C59DD188-4EB6-4EB1-9789-988C2031138F"/>
  </process>
  <bpmndi:processDiagram processRef="oryx_bea118bb-4b0c-4b79-8318-17be7b8a80a2" id="oryx_bea118bb-4b0c-4b79-8318-
17be7b8a80a2_gui">
    <bpmndi:laneCompartment isVisible="false" height="0.0" width="0.0" y="0.0" x="0.0" name="DefaultLane" id="oryx_e50e3424-d2c9-
4135-b4f8-5aa0376aa8c8_gui">
      <bpmndi:eventShape eventRef="oryx_877974F0-BD10-426E-B379-CECC4E5D5A09" height="30.0" width="30.0" y="210.0" x="41.0"
name="start" id="oryx_877974F0-BD10-426E-B379-CECC4E5D5A09_gui"/>
      <bpmndi:activityShape activityRef="oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260" height="80.0" width="100.0" y="185.0"
x="116.0" name="Check Stock" id="oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260_gui"/>
      <bpmndi:gatewayShape gatewayRef="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0" height="40.0" width="40.0" y="205.0"
x="255.0" name="g1" id="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0_gui"/>
      <bpmndi:activityShape activityRef="oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7" height="80.0" width="100.0" y="75.0"
x="360.0" name="Reject Order" id="oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7_gui"/>
      <bpmndi:gatewayShape gatewayRef="oryx_0878603E-954F-437B-B5B0-5D760A250043" height="40.0" width="40.0" y="205.0"
x="840.0" name="g4" id="oryx_0878603E-954F-437B-B5B0-5D760A250043_gui"/>
      <bpmndi:gatewayShape gatewayRef="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953" height="40.0" width="40.0" y="290.0"
x="745.0" name="g3" id="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953_gui"/>
      <bpmndi:activityShape activityRef="oryx_148B6192-A05B-420F-8258-DDBBFDCCCC8F" height="80.0" width="100.0" y="330.0"
x="600.0" name="Ship Goods" id="oryx_148B6192-A05B-420F-8258-DDBBFDCCCC8F_gui"/>
      <bpmndi:gatewayShape gatewayRef="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281" height="40.0" width="40.0" y="290.0"
x="505.0" name="g2" id="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281_gui"/>
      <bpmndi:activityShape activityRef="oryx_C2D9B9C4-0C1E-484B-A1BD-F721D49E64CE" height="80.0" width="100.0" y="270.0"
x="360.0" name="Conform Order" id="oryx_C2D9B9C4-0C1E-484B-A1BD-F721D49E64CE_gui"/>
      <bpmndi:activityShape activityRef="oryx_C8AF8438-5CCA-4FB0-84B3-FDBF8CA62D0C" height="80.0" width="100.0" y="195.0"
x="600.0" name="Send Invoice" id="oryx_C8AF8438-5CCA-4FB0-84B3-FDBF8CA62D0C_gui"/>
      <bpmndi:eventShape eventRef="oryx_DDEA5D3F-AB0D-49C3-969B-518115212378" height="28.0" width="28.0" y="211.0"
x="925.0" name="end" id="oryx_DDEA5D3F-AB0D-49C3-969B-518115212378_gui"/>
    </bpmndi:laneCompartment>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_576C58CB-431F-4A12-AE70-F5E61F56A85E" label=""
targetRef="oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260_gui" sourceRef="oryx_877974F0-BD10-426E-B379-CECC4E5D5A09_gui"
id="oryx_576C58CB-431F-4A12-AE70-F5E61F56A85E_gui"/>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_8AF6F2B1-E35A-4ED6-A80F-524A2CC052B0" label=""
targetRef="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0_gui" sourceRef="oryx_44D5F02C-3B21-4CF1-8B8C-6FCDAF03E260_gui"
id="oryx_8AF6F2B1-E35A-4ED6-A80F-524A2CC052B0_gui"/>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_E56E1E06-6D0F-4EF3-9059-C1D503E552E9" label=""
targetRef="oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7_gui" sourceRef="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0_gui"
id="oryx_E56E1E06-6D0F-4EF3-9059-C1D503E552E9_gui">
      <bpmndi:bendpoint y="115.0" x="275.5"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_1E5CFC29-D955-4B99-8CD9-725FB8A04901" label=""
targetRef="oryx_C2D9B9C4-0C1E-484B-A1BD-F721D49E64CE_gui" sourceRef="oryx_C944EAF3-085E-4D6D-8809-FF0AA6F7EDC0_gui"
id="oryx_1E5CFC29-D955-4B99-8CD9-725FB8A04901_gui">
      <bpmndi:bendpoint y="310.0" x="275.5"/>
    </bpmndi:sequenceFlowConnector>
```

```
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_81A5FC2B-143B-4700-85FE-35949D122066" label=""
targetRef="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281_gui" sourceRef="oryx_C2D9B9C4-0C1E-484B-A1BD-F721D49E64CE_gui"
id="oryx_81A5FC2B-143B-4700-85FE-35949D122066_gui"/>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_ADB8C636-3793-4CA1-8D4E-58F59D6E505C" label=""
targetRef="oryx_C8AF8438-5CCA-4FB0-84B3-FDBF8CA62D0C_gui" sourceRef="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281_gui"
id="oryx_ADB8C636-3793-4CA1-8D4E-58F59D6E505C_gui">
      <bpmndi:bendpoint y="235.0" x="525.5"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_08401581-F22B-4CF8-A4FC-1573FFC9712B" label=""
targetRef="oryx_148B6192-A05B-420F-8258-DDBBFDCCCC8F_gui" sourceRef="oryx_E391C0AA-C3B5-4306-BE23-EBF5CFA9F281_gui"
id="oryx_08401581-F22B-4CF8-A4FC-1573FFC9712B_gui">
      <bpmndi:bendpoint y="370.0" x="525.5"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_99D3D08E-F93D-4FA8-A8BF-6E03E2C0D9CA" label=""
targetRef="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953_gui" sourceRef="oryx_C8AF8438-5CCA-4FB0-84B3-FDBF8CA62D0C_gui"
id="oryx_99D3D08E-F93D-4FA8-A8BF-6E03E2C0D9CA_gui">
      <bpmndi:bendpoint y="235.0" x="765.5"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_44130522-881B-40F2-BE03-F2CBF86C2C9B" label=""
targetRef="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953_gui" sourceRef="oryx_148B6192-A05B-420F-8258-DDBBFDCCCC8F_gui"
id="oryx_44130522-881B-40F2-BE03-F2CBF86C2C9B_gui">
      <bpmndi:bendpoint y="370.0" x="765.0"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_F9ECBC4A-BA0E-44AA-830C-EB53C064245A" label=""
targetRef="oryx_0878603E-954F-437B-B5B0-5D760A250043_gui" sourceRef="oryx_9B317250-16B6-4C71-AE1B-99B1449A9953_gui"
id="oryx_F9ECBC4A-BA0E-44AA-830C-EB53C064245A_gui">
      <bpmndi:bendpoint y="310.5" x="860.5"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_5A410BBA-C099-406C-8386-E479E0764ED6" label=""
targetRef="oryx_0878603E-954F-437B-B5B0-5D760A250043_gui" sourceRef="oryx_3513983D-27F0-4F3F-BA6E-FFA6A98788A7_gui"
id="oryx_5A410BBA-C099-406C-8386-E479E0764ED6_gui">
      <bpmndi:bendpoint y="115.0" x="860.0"/>
    </bpmndi:sequenceFlowConnector>
    <bpmndi:sequenceFlowConnector sequenceFlowRef="oryx_C59DD188-4EB6-4EB1-9789-988C2031138F" label=""
targetRef="oryx_DDEA5D3F-AB0D-49C3-969B-518115212378_gui" sourceRef="oryx_0878603E-954F-437B-B5B0-5D760A250043_gui"
id="oryx_C59DD188-4EB6-4EB1-9789-988C2031138F_gui"/>
  </bpmndi:processDiagram>
</definitions>
```

Abrevieri

B2B Business to Business

BaaS Binding as a Service

BDD Business Driven Development

BPML Business Process Modeling Language

BPEL4WS Business Process Execution Language for Web Services

BPMI Business Process Management Initiative

BPMN Business Process Management Notation

BPSS Business Process Schema Specification

CAT Composition Analysis Tool

CPA Collaborative Protocol Agreements

DAML DARPA Agent Markup Language

DAML-S DARPA Agent Markup Language for Services

DSL Domain Specific Language

ebXML Electronic Business Using Extensible Markup Language

EDI Electronic Data Interchange

HTTP Hypertext Transfer Protocol

HTN Hierarchical task network

MDA Model-Driven Architecture

MDD Model-Driven Design

NFP Non-functional Property(ies)

OASIS Organization for the Advancement of Structured Information Standards

OMG Object Management Group

OWL-DL Web Ontology Language Description Logic

OWL-S Web Ontology Language for Services

QoS Quality of Service

RDF Resource Description Framework

PASSAT Plan Authoring System based on Sketches, Advice, and Templates

SAWSDL Semantic Annotation for WSDL and XML Schema

SCA Service Component Architecture

SLA Service Level Agreement

SQF Web Service Quality Factor

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

SOC Service-Oriented Computing

SWRL Semantic Web Rule Language

SWS Semantic Web Service(s)

Q-WSDL QoS-enabled WSDL

UDDI Universal Description Discovery and Integration

UML Unified Modelling Language

URI Uniform Resource Identifiers

URL Uniform Resource Locators

WS Web Service

WS-BPEL Web Service Business Process Execution Language

WS-CDL Web Service Choreography Description Language

WS-CAF Web Service Composite Application Framework

WS-CF Web Service Coordination Framework

WS-CTX Web Service Context

WS-TXM Web Service Transaction Management

WSDL Web Service Definition Language

WSMF Web Service Modeling Framework

WSCI Web Service Choreography Interface

WSMO Web Service Modelling Ontology

WSQDL Web Service Quality Description Language

WSQM Web Service Quality Model

WSPF Web Services Policy Framework

WSRF Web Service Resource Framework

WSMF Web Service Modeling Framework

WSMO Web Service Modeling Ontology

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

Referințe bibliografice

[1] Raluca Iordache and Florica Moldoveanu, "An efficient approach for an end to end web service composition based on QoS preferences," *UPB Scientific Bulletin*, no. 3, 2014.

[2] Raluca Iordache and Florica Moldoveanu, "A conditional lexicographic approach for the elicitation of QoS Preferences," in *Lecture Notes in Computer Science, Volume 7565: Proceedings of the 20th International Conference on Cooperative Information Systems (CoopIS 2012)*. Rome: Springer, 2012, pp. 182-193.

[3] Kyriakos Kritikos and Dimitris Plexousakis, "OWL-Q for Semantic QoS-based Web Service Description and Discovery," Foundation of Research and Technology, Heraklion, Greece,.

[4] Raluca Iordache and Florica Moldoveanu, "QoS-aware web service semantic selection based on preferences," in *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, Zadar, Croatia, 2013.

[5] Raluca Iordache and Florica Moldoveanu, "A web service composition approach based on QoS preferences," in *Proceedings of the 6th IEEE International Conference on Service Oriented Computing & Applications (SOCA 2013)*, Kauai, Hawaii, 2013.

[6] Y. Yang, M. Dumas, L. García-Bañuelos, A. Polyvyanyy, and L. Zhang, "Generalized aggregate Quality of Service computation for composite services," *Journal of Systems and Software*, no. 85(8), pp. 1818-1830, 2012.

[7] Raluca Iordache and Florica Moldoveanu, "A genetic algorithm for automated service binding," in *24th DAAAM International Symposium on Intelligent Manufacturing and Automation*, Zadar, Croatia, 2013.

[8] Raluca Iordache, Serban Iordache, and Florica Moldoveanu, "A Framework for the Study of Preference Incorporation in Multiobjective Evolutionary Algorithms," in *Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation, GECCO 2014*. Vancouver, Canada: ACM, 2014, p. (to appear).

[9] Nicolai Jossuttis, *SOA in der Praxis*.: dpunkt.verlag, 2008.

[10] Adam Bien, "SOA Blueprints: Was bei einer service-orientierten Architektur alles zu beachten ist," *Java Magazin*, Nov. 2005.

[11] Benny Mathew and Poornachandra Sarang Matjaz B. Juric, *Business Process Execution Language for Web Services*, Second Edition ed., Packt Publishing, Ed., 2006.

[12] Fei Cao, "Model Driven Development and Dynamic Composition of Web Services," University of Alabama at Birmingham, PhD thesis 2005.

[13] Brahim Medjahed and Athman Bouguettaya, *Service Composition for the Semantic Web*.: Springer, 2011.

[14] S. A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," IEEE Intelligent Systems 2001.

[15] W3C, "RDF Vocabulary Description Language RDF Schema," W3C Recommendation 2008.

[16] W3C Member Submission - OWL-S. [Online]. http://www.w3.org/Submission/OWL-S/

[17] Vasile Alaiba. Programare logică şi web semantic.

[18] Axel Polleres, Dumitru Roman, Dieter Fensel James Scicluna. (2006, 3rd February) D14v0.2. Ontology-based Choreography and Orchestration of WSMO Services. http://www.wsmo.org/TR/d14/v0.2/.

[19] E. Cimpian, M. Moran, E. Oren, T. Vitvar, and M. Zaremba, "Overview and Scope of WSMX," WSMX Working Draft 2005.

[20] George Baryannis, Olha Danylevych, and Dimka Karastoyanova, "Service composition," in *Service research challenges and solutions for the future internet*. Berlin: Springer, 2010, pp. 55-84.

[21] Steffen Balzer, Thorsten Liebig, and Matthias Wagner, "Pitfalls of OWL-S: a practical semantic web use case," in *ICSOC04, Proceedings of the 2nd international conference on Service Oriented Computing*, 2004.

[22] Sheila A. McIlraith and Cao Son Tran, "Adapting Golog for Composition of Semantic Web Services," in *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, 2002, pp. 482-496.

[23] Maurice H. ter Bee, Antonio Bucchiarone, and Stefania Gnesi. (2009) A Survey on Service Composition Approaches:From Industrial Standards to Formal Methods.

[24] Oasis, "Web Services Quality Description Language v1.0," 2008,.

[25] W3C. (2005) Web Services Choreography Description Languagw. [Online]. http://www.w3.org/TR/ws-cdl-10/

[26] Oasis. (2009) WS-Coordination (Web Services Coordination) Version 1.2. [Online]. http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec.html

[27] Oasis. (2003) WS-CAF (Web Services Composite Application Framework. [Online]. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

[28] Oasis. (2009) Service Component Architecture Assembly Model Specification Version 1.1. [Online]. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.html

[29] D Skogan, Roy Gronmo, and Ida Solheim, "Web service composition in UML," in *EDOC '04 Proceedings of the Enterprise Distributed Object Computing Conference*.: IEEE Computer Society, 2004, pp. 47-57.

[30] Roy Gronmo and Michael C. Jaeger, "Model-Driven Semantic Web Service Composition," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Taipei, Taiwan, 2005.

[31] M. Tian, A. Gramm, H. Ritter, and J. Schiller, "Efficient Selection and Monitoring of QoS-Aware Web Services with the WS-QoS Framework," in *WI '04 Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*. Washington: IEEE Computer Society, 2004, pp. 152-158.

[32] Chun Ouyang, Marlon Dumas, Arthur H.M. ter Hofstede, and Wil M.P. van der Aalst, "Pattern-based translation of BPMN process models to BPEL web services.," in *International Journal of Web Services Research (JWSR*, 2007.

[33] Thomas Allweyer, *BPMN Business Process Modeling Notation*.: Books on Demand.

[34] Răzvan Petruşel. Afaceri Virtuale.

[35] Jörg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann, "BPEL for Semantic Web Services (BPEL4SWS)," in *Lecture Notes in Computer Science Volume 4805, OTM 2007 Workshops*.: Springer, 2007, pp. 179-188.

[36] Evren Sirin, James Hendler, and Bijan Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions," in *Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI-2003)*, Angers, France, 2003, pp. 17-24.

[37] Karen Myers, "Plan Authoring System based on Sketches, Advice, and Templates (PASSAT)," Artificial Intelligence Center, USA,.

[38] Yolanda Gil, and Marc Spraragen Jihie Kim, "A Knowledge-Based Approach to Interactive Workflow Composition," University of Southern California/Information Sciences Institute, USA,.

[39] Jinghai Rao and Xiaomeng Su, "A Survey of Automated Web Service Composition Methods," *Semantic Web Services and Web Process Composition*, pp. 43--54, 2005.

[40] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, and Ming-chien Shan, "Adaptive and Dynamic Service Composition in eFlow," in *Conference on Advanced Information Systems Engineering*, 2000, pp. 13-31.

[41] Hans Schuster, Dimitrios Georgakopoulos, Andrzej Cichocki, and Donald Baker, "Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes," in *Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, vol. 1789, Stockholm, Swedeb, dec 2000, pp. 247-263.

[42] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau, "HTN planning for Web Service composition using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 4, pp. 377-396, oct 2004.

[43] Dana S Nau et al., "SHOP2: An HTN Planning System," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379-404, 2003.

[44] Drew V Mcdermott, "Estimated-Regression Planning for Interactions with Web Services," in *International Conference on Automated Planning and Scheduling/Artificial Intelligence Planning Systems*, 2002, pp. 204-211.

[45] Michele Trainotti et al., "ASTRO: Supporting Composition and Execution of Web Services," in *International Conference on Service Oriented Computing*, 2005, pp. 495-501.

[46] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma, "Dynamic Workflow Composition: Using Markov Decision Processes," *International Journal of Web Services Research*, vol. 2, pp. 1-17, 2005.

[47] Aiqiang Gao, Dongqing Yang, Shiwei Tang, Ming Zhang, and Ming Zhang, "Web Service Composition Using Markov Decision Processes," in *Web-Age Information Management*, 2005, pp. 308-319.

[48] Judith Bishop, Luciano Baresi K. S. May Chan, "Survey and Comparison of Planning Techniques for Web Services Composition," University of Pretoria, Department of Computer Science, 0002 Pretoria, South Africa,.

[49] Jyotishman Pathak, Neeraj Koul, Doina Caragea, and Vasant G Honavar, "A Framework for Semantic Web Services Discovery," Artificial Intelligence Research Laboratory, Iowa State University,.

[50] Swapna Oundhakar, Amit Sheth, Kunal Verma Abhijit Patil, "METEOR-S Web service Annotation Framework," University of Georgia, Athens,.

[51] ISO, "ISO 8402:1994 - Quality management and quality assurance - Vocabulary," 2004.

[52] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*.: Springer, 2003.

[53] Marc Oriol Hilari, "Quality of Service (QoS) in SOA Systems A Systematic Review," Universitat Politècnica de Catalunya, 2009.

[54] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma, "A QoS-aware Selection Model for Semantic Web Services,".

[55] Y. H. Kang, "Extended Model Design for Quality Factor Based Web Service Management," Future Generation Communication and Networking, 2007.

[56] A. D'Ambrogio, "A Model-driven WSDL Extension for Describing the QoS ofWeb Services," in *ICWS '06 International Conference on Web Services*, 2006.

[57] Youngkon Lee, "Web Service Quality Description Language," Korea Polytechnic University, 2007.

[58] Diego Zuquim Guimarães Garcia and Maria Beatriz Felgar de Toledo, "A UDDI EXTENSION FOR BUSINESS PROCESS MANAGEMENT SYSTEMS," Institute of Computing – State University of Campinas, Brazil,.

[59] W3C. (2007, September) Web Services Policy 1.5 - Framework.

[60] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic WS-agreement partner selection," in *Proceedings of the 15th International conference on World Wide Web*, Edinburgh, Scotland, 2006.

[61] Klara Nahrstedt and Jonathan Smith, "The QoS Broker," Distributed Systems Laboratory, University of Pennsylvania, Philadelphia,.

[62] Kwei-Jay Lin Tao Yu, "A Broker-Based Framework for QoS-Aware Web Service Composition," IEEE International Conference on eTechnology eCommerce and eService , 2005.

[63] Chen Zhou and Bu-Sung Lee, "DAML-QoS ontology for Web services," in *Web Services, Proceedings. IEEE International Conference on* , Nanyang Technologie University, China, 2004.

[64] Shuyu Li and Juan Zhou, "The WSMO-QoS Semantic Web Service Discovery Framework," in *International Conference on Computational Intelligence and Software*

Engineering, China, 2009.

[65] Giuseppe Damiano, Ester Giallonardo, and Eugenio Zimeo, "onQoS-QL: A Query Language for QoS-Based Service Selection and Ranking," in *Lecture Notes in Computer Science Volume 4907*. ICSOC 2007, Vienna, Austria: Springer, 2009, pp. 115-127.

[66] W3C. (2008) SPARQL Query Language for RDF. [Online]. http://www.w3.org/TR/rdf-sparql-query/

[67] Glen Dobson, Russell Lock, and Ian Sommerville, "QoSOnt: an Ontology for QoS in Service-Centric Systems," Computing Department, Lancaster University,.

[68] Jorge Cardoso, *Quality of Service and Semantic Composition*. USA, University of Georgia, 2002.

[69] J.A. Miller, J.S. Cardoso, and G. Silver, "Using Simulation to Facilitate Effective Workflow Adaptation," in *Proceedings of the 35th Annual Simulation Symposium (ANSS'02)*, San Diego, 2002, pp. 177-181.

[70] Mike P-Papazoglou,Willem-Jan van den Heuvel Jian Yang, "Tackling the challenges of service composition in e-marketplaces," Proceedings of the 12th International Workshop on Research Issues in Data Engineering (RIDE '02) 2002.

[71] Joerg Leukel Paul Karaenke, "Towards Ontology-based QoS Aggregation for Composite Web Services," University of Hohenheim, 70599 Stuttgart, Germany,.

[72] Gero Mühl, Michael C. Jaeger Gregor Goldmann, "QoS Aggregation for Web Service Composition using Workflow Patterns," Berlin University of Technology, Germany,.

[73] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani, "An Approach for QoS-aware Service Composition based on Genetic Algorithms," RCOST - Research Centre on Software Technology University of Sannio , Italy,.

[74] J. Cardoso, A. Sheth, J. Miller, J Arnold, and K Kochut, "Quality of service for workflows and web service processes," in *Web Semantics: Science, Services and Agents on the World Wide Web 1*., 2004, pp. 281-308.

[75] B. Kiepuszewski, A.H.M. Hofstede, and C. Bussler, "On Structured Workflow Modelling,".

[76] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring Acyclic Process Models," in *Proceedings of the International Conference on Business Process Management*, New York, 2010, pp. 276-293.

[77] Jozsef Dombi, Csanad Imreh, and Nandor Vincze, "Learning Lexicographic Orders," Departament of Informatics Szeget, Hungary,.

[78] Carlos A. Coello Coello, "Evolutionary Multiobjective Optimization: Past, Present and Future," Depto. de Ingenieria Electrica, Seccion de Computacion, Av. Instituto Politecnico Nacional No. 2508, Mexico, D. F. 07300, MEXICO,.

[79] Sean Luke, *Essentials of Metaheuristics*. Department of Computer Science George Mason University: Online Version 1.0, 2010.

[80] R.T. Marler and J.S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and Multidisciplinary Optimization*, 2010.

[81] Eckart Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," Institut für Technische Informatik Eidgenössische Technische Hochschule Zürich, Zürich, PHD.

[82] Abdullah Konak, David W. Coit, and Alice E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," 2006, Reliability Engineering and System Safety 91 (2006) 992–1007.

[83] JD Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," Proceedings of the international conference on genetic algorithm and their applications, 1985.

[84] CM Fonseca and PJ Fleming, "Multiobjective genetic algorithms," London, IEE colloquium on 'Genetic Algorithms for Control Systems Engineering' 1993.

[85] P. Hajela and Cy Lin, "Genetic search strategies in multicriterion optimal design.," Struct Optimization 1992.

[86] E. Zitzler and E. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," IEEE Trans Evol Comput 1999.

[87] Heidi A. Taboada, Jose F. Espiritu, and David W. Coit, "MOMS-GA: A Multiobjective Multi-State Genetic Algorithm for System Reliability Optimization Design Problems," IEEE Transactions on Reliability 2007.

[88] Sheldon M. Ross, "Introduction to Probability Models," 2003.

[89] Li Li, Peng Cheng, Ling Ou, and Zili Zhang, "Applying Multi-objective Evolutionary Algorithms to QoS-Aware Web Service Composition," Advanced Data Mining and Applications 2010.

[90] Paola Manzini and Marco Mariotti, "Choice by lexicographic semiorders," University of St. Andrews, 2010.

[91] C. Herssens, I. J. Jureta, and S. Faulkner, "Capturing and Using QoS Relationships to Improve Service Selection".

[92] S. Chakhar, S. Haddad, L. Mokdad, and V. Mousseau, "Multicriteria Evaluation-Based Conceptual Framework for Composite Web Service Selection," Springer, Berlin, Evaluation and Decision Models: Real Case Studies 2011.

[93] Luis C. Dias and Vincent Mousseau, "IRIS: ADSS forMultiple Criteria Sorting Problems," Published online in Wiley InterScience.

[94] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng, "Quality driven web services composition," in *WWW '03 Proceedings of the 12th international conference on World Wide Web*. New York: ACM, 2003, pp. 411-421.

[95] W Wiesemann, "A Stochastic Programming Approach for QoS-Aware Service Composition," in *CCGRID '08. 8th IEEE International Symposium on Cluster Computing and the Grid*. Lyon: IEEE, 2008, pp. 226 - 233.

[96] Tao Yu, Yue Zhang, and Kwei-Jay Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints," in *Journal ACM Transactions on the Web (TWEB)*. New York: ACM, 2007.

[97] Michael C Jaeger, Gero Mühl, and Sebastian Golze, "QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms," in *On the Move to Meaningful Internet Systems 2005: CoopIS Lecture Notes in Computer Science Volume 3760*. Cyprus: Springer, 2005, pp. 646-661.

[98] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar, "Towards Composition as a Service – A Quality of Service Driven Approach," Distributed Systems Group, Technical University Vienna, Vienna,.

[99] F. Rosenberg, "Towards Composition as a Service - A Quality of Service Driven Approach," in *ICDE '09. IEEE 25th International Conference on Data Engineeering*. Shanghai: IEEE, 2009, pp. 1733 - 1740.

[100] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo," in *IEEE Transactions on Service Computing Issue 3 No. 3*.: IEEE, 2010, pp. 193-205.

[101] Juan J. Durillo, Antonio J. Nebro, Francisco Luna, Bernab´e Dorronsoro, and Enrique Alba, "jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics," Departamento de Lenguajes y Ciencias de la Computacion E.T.S.

Ingenieria Informatica, Malaga, 2006.

[102] Terence Parr. ANTLR. [Online]. http://www.antlr.org/

[103] Eric Bruneton, Pierre Cregut, Rémi Forax, Eugene Kuleshov, and Andrei Loskutov. ASM. [Online]. http://asm.ow2.org/

[104] Terence Parr. String Template. [Online]. http://www.stringtemplate.org/

[105] J. El Haddad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 73-85, 2010.

[106] L. Zeng et al., "QoS-Aware Middleware for Web Services Composition," in *IEEE Trans. on Software Engineering, 30, issue 5, pp. 311-327*, 2004.