UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

GORAZD HRIBAR RAJTERIČ

UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

GRANULARNOST SPLETNIH STORITEV V ARHITEKTURI SOA ZA MINIMIZIRANJE STROŠKOV OBRATOVANJA

Ljubljana, september 2016 GORAZD HRIBAR RAJTERIČ

IZJAVA O AVTORSTVU

Podpisani Gorazd Hribar Rajterič, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Granularnost spletnih storitev v arhitekturi SOA za minimiziranje stroškov obratovanja, pripravljenega v sodelovanju s svetovalcem prof. dr. Tomažem Turkom

IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;
2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;
3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;
4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;
5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;
6. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;
7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;
8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;
10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne 14.09.2016 Podpis študenta: _____

KAZALO

KAZALO SLIK

UVOD

Uspešno poslovanje podjetja je vse tesneje povezano z učinkovitim informacijskim sistemom. Z rastjo podjetja raste tudi informacijski sistem, ki tako postaja vse dražji in predstavlja vse pomembnejši delež v strukturi stroškov. Dodajanje gradnikov informacijskega sistema zahteva povezavo teh gradnikov. Z večanjem števila pa postanejo te povezave vse težje obvladljive in posledično dražje.

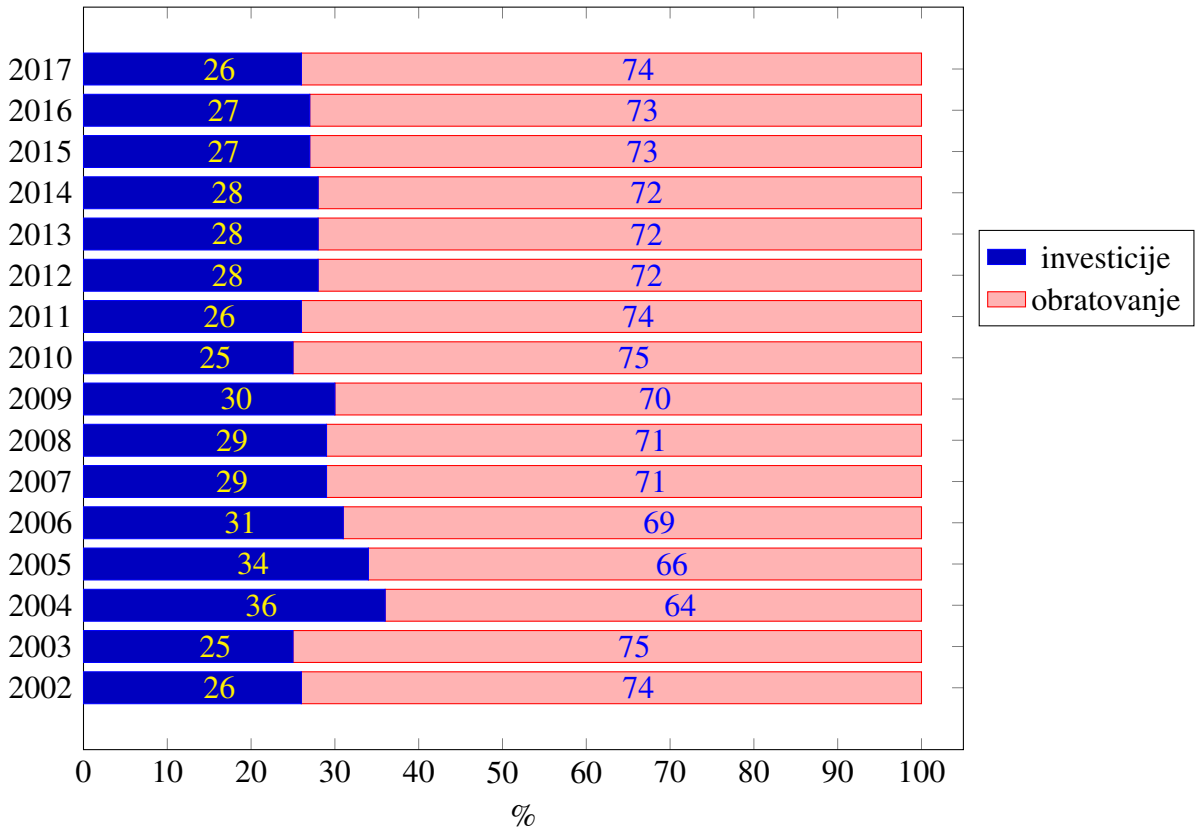
V drugi polovici devetdesetih let se je za rešitev težav, ki jih prinaša povezovanje gradnikov informacijskega sistema, začelo uveljavljati novo načelo v informatiki z nazivom EAI, ki predstavlja „uporabo načel in arhitekture programske opreme in računalniških sistemov za združevanje množice računalniških programov v podjetju" (EAI, 2015). Gre za povsem tehnično gledanje na problem povezovanja, ki je praviloma zelo učinkovit po tehnični plati, vendar neprilagodljiv in drag.

V iskanju prilagodljive in cenovno učinkovitejše rešitve je nastala **storitveno usmerjena arhitektura** (v nadaljevanju SOA). SOA na arhitekturnem nivoju združuje najboljše prakse in rešuje številne pomanjkljivosti starejših načel: odpravlja povezave točka-točka, omogoča odpravo tesne sklopljenosti sistemov, omogoča prehod med sinhrono in asinhrono komunikacijo, usklajuje tehnični in poslovni vidik storitev, promovira ponovno uporabo storitev, temelji na neodvisnosti od tehnologije posameznih gradnikov, omogoča preprostejše modeliranje poslovnih procesov, omogoča hitro spreminjanje in prilagajanje poslovnemu procesu ter temelji na uporabi uveljavljenih standardov. Dejanske prednosti pa so močno odvisne od načina uvedbe arhitekture v poslovno okolje.

SOA pa ima tudi negativne plati, kot so: visoka začetna cena uvedbe, praviloma dolg čas uvedbe, zahteva spremembo načina dela in razmišljanja ter ima visok delež neuspelih projektov (Trotta, 2003).

Poleg potrebnih začetnih vlaganj ob uvajanju SOA arhitekture, je zelo pomembno obvladovanje stroškov obratovanja. Raziskave podjetja Gartner Group (Tracy, Guevara & Stegman, 2008, str. 31), (Guevara, Hall & Stegman, 2012, str. 41) in (Guevara, 2016, str. 10) ter podjetja Computer Economics (Longwell, Ratta & Senia, 2014) in (Longwell & Ratta, 2015) kažejo, da se delež sredstev namenjenih za obratovanje informacijskih rešitev, z redkimi izjemami, giblje med 25 in 30% vseh sredstev vloženih v informacijske rešitve in se skozi zadnjih 14 let ni znatno spremenilo. Gibanje prikazuje Slika 1, pri čemer sta številki za leti 2016 in 2017 oceni podjetja Gartner Group.

Slika 1: Delež investicij proti obratovalnim stroškom IT



Vir: L. Tracy, J. K. Guevara & E. Stegman, IT Key Metrics Data 2009: Key Industry Measures: Multi Industry Spending Overview, 2008, str. 31; J. K. Guevara, L. Hall & E. Stegman, IT Key Metrics Data 2013: Executive Summary, 2012, str. 41; J. Longwell, A. Ratta & A. Senia, IT Spending and Staffing Benchmarks 2014/2015: Executive Summary, 2014; J. Longwell & A. Ratta, IT Spending and Staffing Benchmarks 2015/2016: Executive Summary, 2015; J. K. Guevara, 2016 Sample IT Budget Output Report, 2016, str. 10.

Pravkar zapisana ugotovitev in obljube o znižanju stroškov obratovanja ob uvedbi SOA arhitekture pa me je navedlo, da se v tem delu podrobneje posvetim prav temu področju.

Ob vzpostavitvi SOA arhitekture je potrebno določiti gradnike arhitekture, njihovo medsebojno povezanost, morebitne dodatne gradnike ter način in obseg povezovanja gradnikov med seboj. Vsaka od naštetih odločitev vpliva tako na velikost ter kompleksnost programskih rešitev, kot na potrebno količino strojne opreme in človeških virov za njihovo upravljanje. Nezanemarljiv pa je tudi vpliv zahtevanega znanja (usposobljenosti) ljudi potrebno za upravljanje sistema. Našteti faktorji pa odločilno vplivajo na stroške obratovanja (Turk, 2005, str. 161).

Prav zmanjševanje stroškov je eden od glavnih ciljev tudi v Telekomu Slovenije, d.d. (v nadaljevanju Telekom). Ob nastanku podjetja leta 1994 z ločitvijo tedanjega PTT podjetja na Telekom Slovenije, d.d. in Pošto Slovenije, d.o.o. (v nadaljevanju Pošta Slovenije), je po delitvenem dogovoru obstoječ informacijski sistem, temelječ na IBM Mainframe tehnologiji, prevzela Po-

šta Slovenije, Telekom pa je ostal brez lastnega informacijskega sistema. Vse informacijske storitve je najel pri Pošti Slovenije. V Telekomu so manjše skupine zaposlenih začele razvijati lastne rešitve. Pomanjkanje strategije informatike in podedovana razdrobljenost tako kadrov, kot skromne obstoječe informacijske infrastrukture ter različni poslovni procesi v vsaki od devetih poslovnih enot, je pripeljalo do večjega števila delnih, nepovezanih informacijskih rešitev, ki so se v nekaterih področjih poslovanja med seboj prekrivale, spet druga področja pa niso bila podprta. Celovite rešitve ni bilo.

Neobvladljivo stanje je pripeljalo vodstvo do odločitve najprej o nakupu dveh večjih sistemov: poslovni informacijski sistem SAP R/3 in sistem za obračun telekomunikacijskih storitev Alma Customer Care and Billing. Kasneje pa je bila izoblikovana strategija uporabe obstoječih komercialnih rešitev z integracijo v celovit informacijski sistem. Kmalu zatem je bila začeta prenova sistema za upravljanje odnosov s strankami (v nadaljevanju CRM), ki je poleg standardnih funkcij zajel tudi upravljanje naročil strank (angl. *Order Management*). Pridružili pa so se mu tudi novi sistemi za tehnično infrastrukturo, upravljanje napak, upravljanje incidentov in geografski informacijski sistem (v nadaljevanju GIS) (Pratt & Hribar Rajterič, 2010, str. 28).

Uvedba novih sistemov je odgovor na vse večjo konkurenco, zahteve regulatorja trga in predvsem želja po optimizaciji poslovanja in izboljšanja kakovosti storitev: skrajšanje časa dobave storitev, skrajšanje časa odprave napak, zmanjšanje števila napak in skrajšanje časa uvedbe novih storitev. Nenazadnje pa se je za nove sisteme vodstvo odločilo zaradi podpore obvezi k zamenjavi zastarelih tehnoloških rešitev s sodobnimi rešitvami temelječimi na IP protokolu.

Uvedba velikega števila novih sistemov v kratkem časovnem obdobju pa ni omogočila učinkovite povezave teh sistemov med seboj. Povezava novih sistemov predstavlja največji potencial in poslovno priložnost, hkrati pa predstavlja tudi past, v katero se ne smemo ujeti.

Razdrobljenost sistemov se je še povečala s pripojitvijo hčerinskega podjetja SiOL leta 2007 in ob združitvi s podjetjem Mobitel leta 2011. V obeh pridruženih podjetjih je bilo glavno vodilo interni razvoj informacijskih rešitev, posledično se je Telekom znašel s številnimi sistemi, ki opravljajo isto funkcijo za omejen obseg poslovanja: fiksne telefonske storitve, širokopasovne storitve, poslovne storitve, medoperaterske storitve in mobilne storitve.

V Telekomu Slovenije smo se odločili za uporabo SOA arhitekture, kot temelja za integracijsko platformo in za razvoj novih rešitev. Izbrali smo tudi komercialnega dobavitelja za glavne komponente arhitekture temelječe na J2EE specifikacijah. V letu 2007 smo pripravili tudi strateške dokumente (Heričko & Terbuc, 2007a; Heričko & Terbuc, 2007b; Heričko & Terbuc, 2007c; Heričko & Terbuc, 2007d; Heričko & Terbuc, 2008). Poleg same implementacije pa v

Telekomu manjka še odločitev o najučinkovitejši granularnosti storitev za minimiziranje stroškov obratovanja. To problematiko obravnavam v tem delu.

Opredelitev namena in ciljev. Namen magistrskega dela je proučiti vpliv granularnosti spletnih storitev v SOA arhitekturi na stroške obratovanja ter priporočiti najprimernejšo granularnost za najnižje stroške obratovanja.

S tem namenom je zbrana in proučena domača in tuja strokovna literatura s področja arhitektur IS, SOA arhitekture, granularnosti spletnih storitev in stroškovnih modelov v informatiki.

Cilji, ki jih želim doseči in izhajajo iz namena naloge, so naslednji:

- izdelati priporočilo za najučinkovitejšo granularnost spletnih storitev v SOA arhitekturi za dosego minimalnih stroškov obratovanja;
- izdelati priporočila za granularnost spletnih storitev v okolju Telekoma za področje OSS;
- izdelati splošni zvezni model granularnosti spletnih storitev za doseganje minimalnih stroškov obratovanja;
- predstaviti dodatne prednosti, ki jih prinaša SOA arhitektura v poslovanje podjetja;
- opozoriti na pogoste napake in pasti pri uvedbi SOA arhitekture.

Opredelitev metode dela. Magistrsko delo vsebuje podroben pregled in analizo strokovne literature, znanstvenih razprav, raziskav in člankov tako domačih, kot tujih strokovnjakov s področja obravnavane teme. Sinteza spoznanj je testirana na primeru Telekoma.

V prvem poglavju obravnavam integracijo obstoječih informacijskih sistemov s poudarkom na EAI pristopu, njegovih prednostih in slabostih.

V drugem poglavju predstavljam SOA arhitekturo, podporne tehnologije in standarde, metode vzpostavljanja SOA arhitekture ter metode določanja granularnosti.

V tretjem poglavju obravnavam vzorce, vzroke za njihovo uporabo, namen vzorcev, opredelim pojem vzorca, pokažem na obširno zbirko vzorcev ter opozorim na pasti pri uporabi vzorcev.

V četrtem poglavju se posvečam opredelitvi granularnosti spletnih storitev in opravim pregled metod določanja granularnosti v SOA arhitekturi s tehničnega vidika.

V petem poglavju predstavim metodologije upravljanja informacijskih sistemov ter njihovo pomembnost za to delo.

V šestem poglavju obravnavam področje obvladovanja SOA.

V sedmem poglavju se posvečam obravnavi metodologij za ugotavljanje stroškov, njihovi

medsebojni primerjavi ter prednostim in slabostim posamezne metodologije.

V osmem poglavju izdelam model optimalne granularnosti spletnih storitev za minimiziranje stroškov obratovanja na podlagi izbranih metodologij in danih omejitev. Prikažem tudi način uporabe modela.

V devetem poglavju model uporabim na konkretnem okolju Telekoma za področje OSS sistemov. Rezultat je priporočilo granularnosti glede na dane sisteme, njihove značilnosti in zahtevane poslovne cilje.

Omejitve: Integracija informacijskih sistemov je drag proces, kar še posebej velja ob uporabi SOA arhitekture. Zato je integracijo smiselno izvajati v velikih okoljih, kjer je zamenjava sistemov bodisi predraga, bodisi ni potrebna, ker zadovoljivo pokrivajo poslovni proces in želimo z integracijo dodati vrednost takemu sistemu. V vsakem primeru pa moramo dobro pretehtati koristi in stroške, preden se odločimo za integracijo. Posledično je integracija zanimiva predvsem za velika podjetja in pogojno za srednje velika podjetja. V malih podjetjih, kjer lahko uporabljajo cenejše informacijske sisteme, je pogosto bolj racionalno zamenjati celoten informacijski sistem.

Področje SOA arhitekture je povezano s številnimi standardi in organizacijami označenimi s kraticami. Zaradi preglednosti dela so kratice zbrane z njihovimi izvirnimi pomeni, prevodi in morebitnimi pojasnili v dodatku. V elektronski obliki tega dela so vse kratice povezane z razlago preko hiperpovezav.

1 INTEGRACIJA IS

Lehmann (2004, str. 3) potrebo po integraciji utemeljuje na podlagi potrebe po celovitem pogledu na podjetje: procesi znotraj podjetja potekajo preko različnih organizacijskih enot. Poleg tega pa je vsako podjetje vpeto tudi v okolje. Odločevalci, upravljavci in analitiki potrebujejo celovit vpogled v učinkovitost celotnega podjetja, kot tudi posameznih organizacijskih enot. Te informacije so pridobivali že pred uvedbo informacijskih tehnologij, vendar na počasen in drag način s številnimi napakami. Informacijski sistemi ponujajo možnost znatnega skrajšanja časa za pridobitev informacij, znižanja potrebnega vložka dela in povečanja natančnosti informacij, potrebnih za odločanje. Te možnosti pa je mogoče izkoristiti pod pogojem, da so informacijski sistemi integrirani med seboj in zagotavljajo celovit vpogled v delovanje podjetja.

S pojavitvijo prvih centraliziranih informacijskih sistemov (Mainframe), zgoraj navedeni pogoji niso bili izpolnjeni zaradi tehničnih omejitev in nezrelosti tehnologije. V takšnih okoljih

praktično ni bilo integracij, saj so bili snovalci osredotočeni na optimizacijo kode in deloma na optimizacijo posameznih poslovnih funkcij. Edina vrsta integracije, ki se je pojavljala, je bila med posameznimi aplikacijami znotraj istega sistema. Prav tako je bila arhitektura sistema preprosta: centralna procesna enota s povezanimi terminali in vhodno-izhodnimi enotami. Zato se strokovna javnost v tem obdobju z integracijami ni ukvarjala.

S pojavitvijo cenovno ugodnih računalniških sistemov, sprva mini računalnikov, predvsem pa z mikroračunalniki in še posebej s sistemoma Windows in Unix, ter z razmahom najprej lokalnih računalniških omrežij in še izraziteje kasneje z internetom, so v organiziranih okoljih ob pomanjkanju strokovne literature, hitrem razvoju in pod vplivom pojava, katerega so kasneje poimenovali **upravljanje na podlagi revij** (angl. *Management by Magazine*) (Braithwaite, 2002, str. 61), začeli uvajati parcialne informacijske rešitve na distribuiranih sistemih. Priljubljenost gre pripisati predvsem hitrim rezultatom, znatno nižjim začetnim stroškom investicij in cenejši izdelavi programske opreme, ki je bila posledica orodij s hitro krivuljo učenja, kar je v relativno kratkem času privedlo do velikega števila programerskega kadra. S takšnim pristopom so se začele uvajati parcialne, nestandardizirane in izolirane informacijske rešitve z uporabo različnih orodij in s tem povezanim potrebnim naborom znanj, katere je bilo težko in drago vzdrževati. Po drugi strani pa so podjetja želela zavarovati svoje pretekle investicije v IS in preizkušenih sistemov niso želele takoj zavreči in zamenjati z novejšimi, v katere bi bilo potrebno vložiti znatna sredstva z nepredvidljivimi učinki. Poleg tehnoloških razlik v platformah in razvojnih orodjih, je bila med množico informacijskih rešitev pogosto tudi semantična razlika obravnavanja podatkov in neusklajeni procesi.

Primer: prodajni oddelek je izdelal (ali naročil izdelavo) informacijske rešitve za svoje potrebe, pri tem pa ni upošteval proizvodnih kapacitet in ne računovodstva s financami. Posledično je prišlo do anomalij, ko podjetje ni moglo pravočasno dostaviti naročenih izdelkov zaradi zasedenosti proizvodnje, pomanjkanja potrebnega repromateriala ali nezadostnih sredstev za nakup repromateriala. Drug primer je primer prodajnega oddelka, ki je prodajal strankam, ki so bile neplačniki ali pa je bilo podjetje celo v tožbi s stranko.

Z medsebojnim povezovanjem informacijskih rešitev naletimo na prvo težavo, saj je v splošnem potrebno povezati vsako rešitev z vsako. Tako v najslabšem primeru potrebujemo $n(n-1)$ integracij. Pri tem število n predstavlja število parcialnih informacijskih rešitev v podjetju. Praksa je pokazala, da je potrebnih povezav znatno manj. Povprečna informacijska rešitev ima od 5 do 30 integracij z drugimi rešitvami in gradniki IS (Worden, 2005, str. 3). Pri tem imajo nove informacijske rešitve bliže 5 integracij, skozi življenjsko dobo pa število integracij raste do približno 30.

Integracija ni primerna za vsa poslovna okolja zaradi dodatnih posrednih in neposrednih stroškov, ki jih integracija prinaša (Goodhue, Wybo & Kirsch, 1992, str. 294). Zato je bil razvit model presoje, kot del analize stroškov in koristi (angl. *Cost-Benefit Analysis*), v nadaljevanju ASK). Pri tem avtorji trdijo, da potreba po integraciji izvira iz: interakcij med organizacijskimi enotami, kompleksnih opravil posameznih organizacijskih enot in iz nestabilnega okolja specifičnega za vsako organizacijsko enoto. Sam dodajam še nestabilno okolje znotraj organizacijske enote in potrebo po neodvisnosti od posameznikov v podjetju. Slednja dva vira izhajata iz osebnih izkušenj: Telekom Slovenije je v zadnjih enajstih letih prestal šest reorganizacij, izločitve posameznih dejavnosti, pripojitve hčerinskih podjetij, nakup podjetij v regiji, združitev s hčerinskim podjetjem in odprodajo hčerinskega podjetja. Po vsaki spremembi je poslovodstvo zahtevalo tudi konsolidacijo IS, pri tem pa se je strategija razvoja spreminjala od povsem internega razvoja do vključevanja lokalnih partnerjev in nakupov uveljavljenih komercialnih rešitev. Dodatno zahtevnost pa je predstavljalo specifično tehnološko znanje, ki je bilo v začetni fazi predvsem v glavah ključnih nosilcev področij. Ob organizacijskih spremembah pa je bilo to znanje le delno preneseno na nove ljudi ali pa sploh ne. Prav zato je bila ena pomembnih nalog zajem znanja v formalizirani obliki in prenos v IS.

Zgoraj omenjena opozorila in pozivi k razumni presoji pa so bila v veliki meri prezrta. Posledično pa je bilo vse več sredstev namenjeno prav integracijam in povezovanju. IBM je leta 1999 ocenjeval, da je 70% vse programske kode namenjeno vmesnikom, protokolom in drugim načinom povezovanja različnih sistemov („ERP RIP?”, 1999, str. 32).

Tako se je konec devetdesetih let prejšnjega stoletja in v začetku novega tisočletja pojavilo novo področje v informatiki imenovano Integracija aplikacij v velikih podjetjih (v nadaljevanju EAI). EAI gradi na razpoložljivih tehnikah dostopnih v času nastanka. Nekatere izmed teh tehnik so tedaj veljale za sporne, saj so jih uporabljali za nelegalne vdore v informacijske sisteme.

V času pojavitve EAI je bil končni cilj integracij **poslovanje brez časovnega zamika** (v nadaljevanju ZLE) (Liles, 2000, str. 674). ZLE je organizacijsko stanje, v katerem so vse informacije takoj po nastanku na voljo celotnemu podjetju (skozi aplikacije) ne glede na tehnološke in organizacijske meje z namenom doseganja poslovnih koristi. To je tudi idealni cilj naše arhitekture. Kot pa bomo videli kasneje, takšnega stanja v celoti ni mogoče doseči oziroma bi bil vložek prevelik za doseganje idealnega stanja. Zato se bomo temu stanju poskušali čimbolj približati ob upoštevanju ASK analize.

Linthicum (1999, str. 8) opredeljuje EAI kot: „EAI je neomejena izmenjava podatkov in poslovnih procesov med poljubnimi medsebojno povezanimi aplikacijami in podatkovnimi viri v podjetju.”

1.1 Stopnje integracije

Integracije lahko po zrelosti klasificiramo po stopnjah (Worden, 2005, str. 3):

- ročno kodirani podatkovni vmesniki,
- orodja za preslikavo med ponorom in izvorom,
- integracijska infrastruktura z razdelilniki in posredniki (angl. *Hubs and Brokers*) in
- integracija na podlagi polnega modela.

Vsaka od naštetih stopenj zahteva določeno zrelost podjetja in njegovega IS ter količino virov za izvedbo. Hkrati pa z vsako višjo stopnjo dosegamo IS, ki je bližje ZLE konceptu in ga je lažje obvladovati.

1.1.1 Ročno kodirani podatkovni vmesniki

Ročno kodirani podatkovni vmesniki pogosto premamijo neizkušene arhitekte in predvsem programerje. Pri tej stopnji vsako integracijo med praviloma dvema rešitvama načrtujemo in izvajamo znova glede na postavljene funkcionalne zahteve. Glede na prej omenjeno tipično število integracij, to pomeni med 5 in 30 integracijskih projektov, vsak s svojimi pravili za zajem in pretvorbo podatkov. Ne glede na navidezno preprostost, pa je potrebno zelo podrobno poznavanje podatkovnih struktur tako na izvoru, kot na ponoru.

Tudi rutinski deli kode za zajem in pretvorbo podatkov, so vsakič napisani znova. Takšen nivo nujno vodi v razdrobljenost poslovnih pravil, redundanco kode in velike količine namensko napisane kode. Ponovne uporabe praktično ni.

Takšna stopnja integracij je tudi glavni vzrok za katastrofalne ugotovitve o potrebnih vložkih v integracije.

1.1.2 Orodja za preslikavo med ponorom in izvorom

Orodja za preslikavo med izvorom in ponorom so grafična in prikazujejo podatkovni model enega sistema na eni in podatkovni model drugega sistema na drugi strani. Naloga izvajalca integracije je, da poveže sorodna polja med seboj in po potrebi vstavi funkcijo za pretvorbo, če orodje to omogoča. Orodja se med seboj razlikujejo prav po možnostih izvajanja preslikav in načinu izvedbe. Rezultat je praviloma generiran sistem za preslikavo med izvorom in ponorom ali konfiguracija za strežnik, ki nato izvaja preslikave. Orodja se ločijo tudi po možnostih proženja: nekatera zaznavajo spremembe takoj in prožijo kopiranje, druga se prožijo periodično, spet tretja pa zahtevajo ročno proženje. Takšna orodja danes poznamo pod kratico ETL.

Primer takšnih orodij so: Microsoft BizTalk Mapper, Data Mirror Transformation Server, Ab Initio, IBM InfoSphere DataStage, Informatica, Oracle Data Integrator, SAP Data Integrator, Apatar, CloverETL, Pentaho in Talend. Slednja štiri so na voljo brezplačno kot odprtokodna orodja.

Žal pa tudi takšna orodja ne odpravljajo vseh slabosti ročno kodiranih vmesnikov: še naprej je potrebno preslikovati med posameznimi podatkovnimi viri oziroma rešitvami, katerih število strmo raste. Dodatna težava pa so podatki, ki po naravi niso relacijski, ampak imajo nestrukturirano obliko ali vsebujejo večnivojsko gnezdenje, kot je primer pri XML obliki podatkov.

1.1.3 Integracijska infrastruktura z razdelilniki in posredniki

Infrastruktura z razdelilniki in posredniki vsebuje orodja za preslikavo podatkov, poleg tega pa prinaša še integracijo procesov in orkestracijo. Tipično integracija v tem primeru ne poteka neposredno med dvema rešitvama, temveč med rešitvijo in prilagojeno centralizirano predstavitvijo podatkov na razdelilniku. Slednje rešuje težavo potrebnih n^2 integracij na $2n$ integracij.

Pri tej stopnji je pred preslikavo potrebno ustvariti centralizirano celovito predstavitev podatkov oziroma centraliziran model. Izvajalec preslikuje podatkovni model vsake rešitve na prilagojen centraliziran model podatkov. Pri tem izgradnja centraliziranega modela zahteva znaten trud in poznavanje tako poslovnih, kot tehnoloških zahtev in omejitev. Običajno centraliziran model nastaja postopoma z dodajanjem posameznih informacijskih rešitev v centraliziran IS.

Žal so orodja v praksi omejena predvsem pri definiranju prilagojenega centraliziranega podatkovnega modela in pogosto ne omogočajo kompleksnejših sodobnih podatkovnih struktur. Poleg tega so takšna orodja praviloma draga, uvedba pa zahteva veliko dodatnih vložkov. Pogosto so potrebne spremembe centralnega modela podatkov ob dodajanju delnih informacijskih rešitev, kar integracijo z razdelilniki in posredniki še dodatno oteži, podaljša čas izvedbe projekta in ga podraži.

Primera orodij na tej stopnji sta: Informatica Data Integration Hub in Action Data Integration Hub.

1.1.4 Integracija na podlagi polnega modela

Pri integraciji na podlagi polnega modela nadgrajujemo prejšnji nivo na ta način, da v centralni model dodamo polno izrazno moč UML objektnega modela in vsa pravila preslikav izvornih podatkovnih virov. Dejanske transformacije pa potem izvajamo avtomatizirano. V kolikor bi to bilo zares mogoče izvesti, bi potrebovali samo n preslikav.

Slabost te stopnje je prav v pomanjkanju zmogljivih orodij in dejstvu, da bi bilo potrebno za integracijo vseh podatkovnih virov potrebno pripraviti obsežen UML model, ki bi postal neobvladljivo kompleksen. Podobni poskusi izdelave vseobsegajočega podatkovnega modela podjetja so v preteklosti že večkrat propadli. Glavna razloga sta:

- Spremenljivo okolje ne dopušča statičnega modela realnosti. Dejansko okolje se neprestano spreminja.
- Poskusi vzpostavitve celovitega podatkovnega modela so bili izvajani še pred uveljavitvijo objektnega načina razmišljanja in niso upoštevala dedovanja. Posledično so bili modeli prekompleksni za izvedbo in vzdrževanje.

Poleg tega je potrebno opraviti še veliko dela (integracij) znotraj posameznih vsebinsko zaključenih področij, kot so: CRM, nabava, logistika, proizvodnja ipd. Zato vseobsegajoč model, vsaj na začetku, ni potreben.

1.2 Nivoji integracije

Integracijo je mogoče izvajati na različnih nivojih informacijskih rešitev. Nivoji so v tem primeru opredeljeni kot arhitekturni gradniki posamezne aplikacije z vidika arhitekture posamezne informacijske rešitve in ne kot elementi arhitekture informacijskega sistema, ki zajema več informacijskih rešitev.

Različni avtorji v okviru obravnave EAI opredeljujejo tri (Puschmann & Alt, 2004, str. 107) ali štiri nivoje (Linthicum, 1999, str. 21; Emmerich, Ellmer & Fieglein, 2001, str. 567; Arsanjani, 2002, str. 32; Lee, Siau & Hong, 2003, str. 58; Themistocleous & Irani, 2004, str. 394; Bahli & Ji, 2007), na katerih je mogoče izvajati integracijo IS. Trinivojska arhitektura je redkeje obravnavana. Pri avtorjih, ki opredeljujejo tri nivoje, so ti nivoji naslednji:

- predstavitveni nivo,
- podatkovni nivo in
- funkcijski nivo.

Pri avtorjih, ki opredeljujejo štiri nivoje integracije so nivoji različno poimenovani, v tem delu pa uporabljam naslednja imena:

- podatkovni nivo,
- nivo uporabniškega vmesnika,
- nivo aplikacijskih vmesnikov in
- nivo metod.

Vsebinsko gledano, je štirinivojska opredelitev natančnejša in podrobneje opredeljuje funkcijski nivo trinivojske opredelitve iz dveh različnih zornih kotov. Zato v tem delu uporabljam štirinivojsko opredelitev.

Pri tem je potrebno opozoriti, da izbira najprimernejšega nivoja ni preprosta. Pravilna izbira je odvisna od možnosti, ki jih ponujajo informacijske rešitve, katere želimo povezati, razpoložljivih virov, potrebnih vložkov in pričakovanih koristi. Izbira je tudi močno pogojena s ciljem, ki ga želimo doseči. V kolikor gre za preprosto izmenjavo šifranta kupcev, bo izbira drugačna, kot v primeru želene centralizacije poslovne logike in funkcij. Kot bomo videli v nadaljevanju, nivo metod prinaša najboljši približek idealnemu končnemu stanju, zahteva pa tudi največji vložek.

Kombinacija nivojev in tehnik integracije ni prepovedana. Še več: pri integraciji kompleksnejših aplikacij v celovit IS je v praksi kombinacija nivojev in tehnik najpogostejši pristop k integraciji. Pri tem pa je potrebno upoštevati vse prednosti in slabosti, ki jih uporabljene tehnike prinašajo in njihov vpliv na končno rešitev.

Vsak nivo integracije je kratko predstavljen v nadaljevanju.

1.2.1 Podatkovni nivo

Podatkovni nivo je najenostavnejša oblika povezovanja aplikacij. Ta način ne zahteva spremembe aplikacij. Podatkovni nivo je zelo priljubljen tudi zaradi širokega nabora tehnologij, ki omogočajo povezovanje na podatkovnem nivoju. K temu so v devetdesetih letih zelo pripomogle relacijske zbirke podatkov. Nikakor pa to ne pomeni, da je uporaba podatkovnega nivoja preprosta. Tako od arhitekta, kot od izvajalca zahteva poglobljeno poznavanje strukture in vsebine podatkov tako v povezovani, kot v povezani aplikaciji. Dodatno stopnjo zahtevnosti pa običajno prinaša dejstvo, da podatki že na izvoru niso normalizirani, ampak pogosto redundantni in vsebujejo podatke potrebne za delovanje posamezne aplikacije.

1.2.1.1 Neposreden dostop do zbirke podatkov

Še dandanes je neposreden dostop do zbirke podatkov (v nadaljevanju: neposreden dostop) pogosta tehnika integracije dveh ali več različnih sistemov. Vzrok za takšno priljubljenost je v preprostosti vzpostavitve povezave (na voljo so gonilniki za dostop do skoraj vseh vrst zbirk podatkov na svetu) in v standardizaciji jezika za poizvedbe: SQL. Praviloma je neposreden dostop tudi zelo hiter in ponuja možnost cikličnega izboljševanja tako s hitrostnega, kot iz vsebinskega vidika. Prednost je tudi možnost dvosmerne komunikacije, saj lahko v zbirko

podatkov tudi neposredno vpisujemo podatke pod pogojem, da poznamo tehnična in poslovna pravila.

Hkrati pa so prej naštete prednosti tudi slabosti: v kolikor imamo opravka s starejšim sistemom, ki ne uporablja standardne zbirke podatkov, npr. mrežno zbirko podatkov ali datotečni sistem z lastno strukturo zapisa podatkov, potem obstoječega gonilnika praviloma ni. Nestandardne zbirke praviloma ne podpirajo standardnega jezika za poizvedbe. Velika slabost takega pristopa je tudi dejstvo, da z njim zaobidemo obstoječo poslovno logiko in jo moramo zgraditi ponovno, sicer tvegamo napačno interpretacijo podatkov ali, v primeru dvosmerne komunikacije, celo okvaro podatkov in povzročitev zastoja v delovanju IS. Neposreden dostop do zbirke podatkov delno odpravlja tesno sklopljenost sistemov, saj sistema neposredno nista med seboj odvisna, je pa nov sistem odvisen od delovanja zbirke podatkov.

1.2.1.2 Distribuirane zbirke podatkov

Distribuirane zbirke podatkov (angl. *Federated Database*) je pristop, kjer je nad različnimi zbirkami podatkov narejen skupni logični model, sistem za distribucijo pa poskrbi za preslikavo posameznih delov logičnega modela v in iz posamezne zbirke podatkov. Z uporabniškega stališča je delo z distribuirano zbirko podatkov enako, kot delo z eno centralno zbirko podatkov in skrije kompleksnost dostopa do več zbirk hkrati.

Prednost te tehnike so podobne, kot pri tehniki neposrednega dostopa do zbirke podatkov, s pomembno razliko: ta način ni uporaben za integracijo dveh ali več obstoječih rešitev brez spremembe vsaj ene izmed njih. Novim rešitvam pa so na voljo vsi podatki iz obstoječih zalednih rešitev. Še več, ob uporabi distribuirane zbirke je mogoče eno ali več rešitev ukiniti in jo nadomestiti z novejšo, ne da bi odjemalci distribuirane zbirke to opazili, oziroma, da bi jih bilo potrebno spreminjati.

Velika slabost te tehnike je velik trud, ki ga je potrebno vložiti v izgradnjo navideznega modela distribuirane zbirke podatkov. Pogosto so starejše zbirke med seboj vsebinsko težko združljive ali povsem nezdružljive.

1.2.1.3 Replikacija podatkov

Zelo priljubljena tehnika pri integraciji IS je tudi replikacija celotne ali dela zbirke podatkov (v nadaljevanju: replikacija). Ob uporabi te tehnike je potreben dostop do zbirke podatkov starejše aplikacije. Obstaja tudi možnost replikacije preko zajemanja zaslona, vendar je običajno takšna replikacija prepočasna za resno uporabo pri integraciji. Naslednja težava je zaznavanje sprememb v podatkih. Neposreden dostop namreč ne omogoča proženja dogodkov iz izvorne

aplikacije, zato je potrebno ob uporabi te tehnike vgraditi mehanizem detekcije spremembe podatkov. Replikacijo je že več let mogoče izvesti tudi na nivoju diskovnega polja. Primer takšne rešitve sta BCV in SRDF podjetja EMC[2].

Prednost te tehnike je neodvisnost sistemov, saj ima novi sistem lokalno dosegljive podatke starega sistema neodvisno od njegovega delovanja. Posledično je tudi dostop do podatkov hiter in zanesljiv.

Slabosti so naslednje: dodatni diskovni prostor potreben za eno ali več kopij podatkov, potrebno je časovno okno za osveževanje kopije podatkov, običajno samo replikacija ne zadostuje in je potrebna tudi pretvorba podatkov v obliko, ki jo lahko ciljni sistem uporabi, zavržena je obstoječa poslovna logika pri uporabi podatkov in jo je potrebno ponovno implementirati.

Zadnjih 10 let je ta tehnika najpogosteje uporabljena za podatkovna skladišča (angl. *Data Warehouse*), poslovno inteligenco (angl. *Business Intelligence*) in analizo velikih količin podatkov (angl. *Big Data Analysis*). Orodja za replikacijo spadajo danes v skupino ETL orodij.

1.2.1.4 Izmenjava datotek

Obstoječi IS ima pogosto možnost izmenjave podatkov preko zunanjih datotek (uvoz in izvoz podatkov). Običajno je bila takšna izmenjava namenjena poročilnemu sistemu oziroma podatkovnim skladiščem. Ta mehanizem pa lahko uporabimo tudi za integracijo sistemov.

Prednost te tehnike je preprostost uporabe, šibka sklopljenost in posledično medsebojna neodvisnost tako povezanih sistemov.

Slabosti izmenjave podatkov preko datotek pa so: časovni zamik, pomanjkanje podpore transakcijam in omejitve obsega podatkov vključenih v izmenjavo z datotekami. Slednje pomeni, da je mogoče na ta način izmenjevati samo tiste podatke, za katere ima obstoječ IS predviden uvoz in/ali izvoz v datoteke. S to tehniko se oddaljujemo od koncepta ZLE.

1.2.2 Nivo uporabniškega vmesnika

1.2.2.1 Zajemanje zaslona

Zajemanje zaslona (angl. *Screen Scraping*) je najbolj primitivna tehnika, uporabna predvsem v primeru integracije s starejšim sistemom, ki uporablja tekstovni terminal za uporabniški vmesnik in nima drugih možnosti zunanjega dostopa. Ideja pri zajemanju zaslona je preprosta: nadomestiti uporabnika za tekstovnim terminalom z računalniškim modulom, ki bo znal vnesti vhodne podatke in prebrati rezultate iz navideznega ekrana, praviloma skritega končnemu

uporabniku. Tako pridobljene podatke pa potem ponuditi novejši aplikaciji na poljubni platformi v nadaljnjo obdelavo.

Prednost takšnega pristopa je uporaba nespremenjenega obstoječega sistema, pri čemer ni potrebno podrobno poznavanje poslovne logike. Potrebno je samo natančno poznavanje vnosnih mask, predvidene vhodne podatke in obliko pričakovanih izhodnih podatkov. Dobra uporabniška dokumentacija takšen pristop močno olajša. Nov in star sistem sta ob uporabi te tehnike lahko na popolnoma različnih platformah.

Ima pa ta pristop tudi nekatere slabosti. Težava nastane ob nepredvidenih dogodkih in izjemah. Primer takšnih dogodkov so prekinitev na omrežju ali nepredvidena oblika ali vsebina podatka bodisi na vnosni maski ali v odgovoru. Druga velika slabost je potreba po sočasnem delovanju starega in novega sistema. Zajemanje zaslona zahteva, da sta tako nov, kot star sistem hkrati delujoča. Izpad enega ali obeh povzroči izpad celotnega povezanega sistema. To je posledica tesne sklopljenosti, ki jo zahteva zajemanje zaslona. Pogosto je pri tej tehniki težava tudi z odzivnostjo, saj je programsko težko ugotoviti, kdaj je ekranska maska v celoti napolnjena z odgovorom oziroma, kdaj lahko začnemo z branjem odgovora iz navideznega ekrana.

1.2.2.2 Spletni portali

Spletne IR je mogoče združiti tudi s pomočjo portalskih strežnikov. Gre za spletne strežnike, ki združujejo spletne uporabniške vmesnike skupine aplikacij na enem mestu. Portalski strežniki prestrežejo spletni vmesnik integriranih IR, jih združijo in prikažejo kot enoten uporabniški vmesnik. Praviloma so v ta namen uporabljeni HTML okviri (angl. *Frames*) in razdelki (angl. *Divisions*). V bolj zapletenih primerih pa lahko portalski strežnik obdela podatke pridobljene iz posameznih aplikacij in jih združene prikaže končnemu uporabniku.

1.2.3 Nivo aplikacijskih vmesnikov

Številne tudi starejše programske rešitve ponujajo programski vmesnik (API) za povezovanje, prilagajanje in razširitve osnovnega produkta. Sprva so ponudniki komercialnih rešitev zapirali svoje produkte, vendar je potreba po povezovanju IR med seboj prevladala in dobavitelji so začeli vse večji del funkcionalnosti svojih produktov izpostavljati v obliki programskih vmesnikov. Sprva rezervirano, danes pa si komercialnega izdelka z dobrim programskim vmesnikom ne moremo več niti zamisliti. Z uveljavitvijo standardnih načinov dostopa do objektov, kot sta CORBA in Microsoftov COM, je uporaba API vmesnikov dobila še dodaten zagon. Razširitev teh vmesnikov z možnostjo oddaljene uporabe objektov preko omrežja z uporabo standardnih protokolov: Java RMI, IIOP ter DCOM, pa so API vmesniki omogočili tudi distribuirane integracijske rešitve v heterogenih okoljih.

Prednosti te tehnike so: uradno podprt način integracije in s tem podpora izdelovalca komercialne rešitve. Izdelovalec zagotavlja konsistenco podatkov in procesov, s čimer je preprečeno nekonsistentno spreminjanje podatkov, kar se lahko zgodi pri neposrednem dostopu do zbirke podatkov. Praviloma je razpoložljiva dokumentacija in pogosto tudi šolanja za uporabo.

Nekatere slabosti: praviloma tesna sklopljenost odjemalca s ciljnim sistemom, zahtevano poznavanje produkta in tehnologije, kar običajno zahteva dodatno usposabljanje ekipe za implementacijo ter dodatna odvisnost integracije od podpore in izdaj proizvajalca ciljnega produkta. Zahteva veliko programiranja v običajno vnaprej določenih programskih jezikih in okolju. Potrebna je tudi relativno dolga krivulja učenja, saj je vsak API specifičen in uporablja koncepte osnovnega komercialnega produkta.

1.2.4 Nivo metod

Ideja pri povezovanju na nivoju metod je v izločitvi poslovne logike integracij v ločen modul. Izhaja iz podobne ideje, kot objektno usmerjeno programiranje in združuje podatkovne strukture in poslovno logiko. S tem dosežemo, da je poslovna logika skoncentrirana na enem mestu in jo je tako lažje upravljati in vzdrževati. Takšen pristop omogočajo prav tehnologije omenjene v prejšnjem razdelku: Java RMI, CORBA, IIOP, J2EE in DCOM. V okviru EAI se pogosto omenjajo sestavljene aplikacije (angl. *Composite Applications*). To so aplikacije povezane med seboj preko poslovne logike in metod. Navzven delujejo kot celovit IS, kar je tudi zaželeno končno stanje arhitekture EAI.

Prednosti integracije na nivoju metod so: enotna poslovna logika združena na enem mestu, lažje upravljanje, lažje vzdrževanje in predvsem enoten zunanji pogled na IS. Tak način omogoča tudi izvedbo visoko razpoložljivega sistema za integracijo skozi uporabo gruč (angl. *Clusters*), strežniških farm in porazdeljevanja bremena (angl. *Load Balancing*). S slednjim pristopom lahko IS raste na razmeroma preprost način brez spreminjanja načina integracije. Potrebno je samo dodajati ustrezno strojno opremo v ozadju.

Največja slabost tega pristopa je v potrebi po podvajanju že obstoječe poslovne logike posameznih IR ter vzdrževanje podvojene logike tako v osnovnem sistemu, kot v integracijskem modulu. Pogosto je potreben tudi vzvratni inženiring obstoječega sistema, saj poslovna logika, ki jo želimo izpostaviti drugim IR, ni dokumentirana ali pa se je dokumentacija izgubila. Poseben problem pa predstavljajo komercialni produkti, kjer posamezni deli poslovne logike niso dokumentirani, vzvratni inženiring pa je izrecno prepovedan.

Povezovanje na nivoju metod omogoča podjetju ponovno uporabo poslovne logike skozi metode, ki so lahko implementirane v integracijskem modulu ali pa integracijski modul služi le kot

posrednik za dostop do enega ali več povezanih IR.

1.3 Vmesna programska oprema

Vmesna programska oprema (angl. *Middleware*) je še eden od različno razumljenih in definiranih izrazov, ki so pogosto tudi zlorabljeni s strani proizvajalcev komercialne programske opreme. Kratek izbor definicij je naslednji:

„Vmesna programska oprema je tista, ki sestavlja sloj programske opreme za konverzijo in preslikavo. Vmesna programska oprema prav tako konsolidira in integrira." (Middleware, 2016a)

„Vmesna programska oprema omogoča izmenjavo podatkov med dvema uporabniškima rešitvama v istem okolju ali preko različne strojne opreme in omrežnega okolja." (Middleware, 2016b)

„Vmesna programska oprema je sloj programske opreme nad operacijskim sistemom, vendar nižje od uporabniške programske opreme, ki zagotavlja skupno programsko abstrakcijo preko celotnega distribuiranega sistema" (Bakken, 2003, str. 4).

„Preprosta definicija vmesne programske opreme je ‚programska oprema, ki je v praksi potrebna za izdelavo distribuiranih sistemov'" (Britton & Bye, 2004, str. 7)

„Vmesna programska oprema je sloj programske opreme med operacijskim sistemom in uporabniško programsko opremo, ki zagotavlja višji nivo abstrakcije v distribuiranih sistemih." (Bruneo, Puliafito & Scarpa, 2007, str. 148)

Predvsem definiciji avtorjev Bakken (2003) in Bruneo, Puliafito & Scarpa (2007) uspeta zajeti tudi sodobnejše sisteme, ki jih uvrščamo med vmesno programsko opremo, zanemarjata pa izvajalno funkcijo, poleg tega sta za praktično uporabo presplošni.

Za potrebe tega dela bom med vmesno programsko opremo štel programsko opremo, ki omogoča razvoj in izvajanje uporabniške programske opreme, ponuja višji nivo abstrakcije, zagotavlja varnost in integriteto podatkov, implementira standarde za izmenjavo in pretvorbo podatkov ter ponuja standarde za povezovanje uporabniške programske opreme v heterogenih distribuiranih okoljih. Z izjemo računalništva v oblaku, vmesna programska oprema ne izvaja poslovnih funkcij, ponuja pa orodja za konfiguracijo, upravljanje in spremljanje tako vmesne programske opreme, kot uporabniške programske opreme, ki se izvaja v okolju, katerega zagotavlja vmesna programska oprema. S stališča uporabniške programske opreme, je vmesna programska oprema **infrastruktura**, ki omogoča izvajanje in povezovanje z drugo uporabniško programsko opremo

ter zagotavlja nižjenivojske funkcije. Vsa vmesna programska oprema pa zahteva dodatno programsko opremo za izvedbo poslovnih funkcij. Za razliko od programskih knjižnic, ima vmesna programska oprema svoje izvajalno okolje in ni del uporabniške programske opreme.

Ključni kriteriji kakovosti vmesne programske opreme so:

- zanesljivost,
- zmogljivost,
- skalabilnost,
- transparentnost in
- skladnost s standardi.

Vmesna programska oprema omogoča izgradnjo distribuiranih sistemov. Distribuirani gradniki sistema, s stališča zanesljivosti, zmanjšujejo zanesljivost celotnega sistema zaradi medsebojne odvisnosti (Shooman, 2003, str. 334–335). Zanesljivost pa je tudi funkcija sklopljenosti gradnikov (Shooman, 2003, str. 172). Tesneje, kot so gradniki sklopljeni, bolj so odvisni med seboj. Posledično je zanesljivost končnega sistema nižja od zanesljivosti posameznih gradnikov. Zato je zanesljivost vmesne programske opreme ključnega pomena in neposredno vpliva na stroške celotnega IS.

Dodatni sloji programske opreme med uporabniško programsko opremo in strojno opremo neizogibno porabijo nekaj strojnih virov tudi zase, s čimer se del zmogljivosti razpoložljive strojne opreme uporabi za izvajanje vmesne programske opreme. Ta vpliv naj bo za vmesno programsko opremo čim manjši. Z uporabo distribuiranega predpomnilnika in hevrističnih algoritmov je mogoče znižati vpliv dodatne porabe virov vendar le v omejenem obsegu. Poleg tega potrebujejo tako predpomnilniki, kot tudi hevristični algoritmi dodaten čas po zagonu za pridobitev potrebnih podatkov za optimalno delovanje. V tem začetnem času pa je poraba virov še znatno višja, zmogljivost pa nižja.

Uvedba vmesne programske opreme je za podjetje strošek, ki sam po sebi ne prinaša poslovnih koristi. Zato je zelo zaželeno, da lahko vmesno programsko opremo in z njo podrejene informacijske vire uvajamo v najmanjšem potrebnem obsegu, ki lahko z rastjo podjetja in poslovanjem postopoma raste. To pa je mogoče le, če je vmesna programska oprema skalabilna.

Za vsakodnevno delovanje vmesne programske opreme so potrebna orodja in človeški nadzor. Še tako zanesljiv sistem ima napake, ki se lahko pokažejo ob določeni kombinaciji procesov in podatkov. Poleg tega so lahko napake tudi v uporabniški programski opremi, ki teče v okolju vmesne programske opreme. Za učinkovito in hitro odpravo napak, pa je potrebno čim hitreje identificirati vzroke in napake odpraviti. Transparentnost delovanja, ki omogoča vpogled v izva-

janje in delovanje vmesne in uporabniške programske opreme, je ključna za identifikacijo napak in njihovih vzrokov. Pomanjkanje transparentnosti je delno mogoče nadomestiti z dodatnimi orodji za nadzor, kar pa poviša vložek v uvedbo vmesne programske opreme.

Za vmesno programsko opremo je zelo pomembno, da implementira standarde. S tem si zagotovimo, da je mogoče zamenjati gradnik vmesne programske opreme brez ponovnega programiranja ali zamenjave uporabniške programske opreme. S tem povečamo neodvisnost od proizvajalca, kar omogoča prehod na cenejšo rešitev v prihodnosti. Po drugi strani pa standardi omogočajo tudi večjo izbiro pri uporabniški programski opremi in večjo izbiro programerskega kadra in kadra za vzdrževanje sistema.

Ob prehodu tisočletja, so bili nekateri tipi vmesne programske opreme že uveljavljeni, zato jih imenujemo tudi **tradicionalna vmesna programska oprema** (angl. *Traditional Middleware*). Zaradi hitrega razvoja tega področja pa je beseda **tradicionalni** zelo relativna in se obseg produktov, ki naj bi sodili v to skupino, hitro spreminja. Ob prej naštetem razumevanju vmesne programske opreme, so najpomembnejši predstavniki naslednji:

- sistemi za upravljanje zbirk podatkov,
- centralizirani sistemi za varnostno kopiranje podatkov,
- sistemi za gručenje (angl. *Clustering*),
- porazdeljevalniki bremena (angl. *Load Balancers*),
- transakcijski nadzorniki,
- sistemi za sporočilne vrste (angl. *Message Queueing*),
- aplikacijski strežniki in
- programska ogrodja (angl. *Frameworks*).

Podrobnejši opis nekaterih od zgoraj naštetih predstavnikov vmesne programske opreme presega obseg tega dela. Zato sledi samo njihov kratek opis.

Sistemi za upravljanje zbirk podatkov so lahko hierarhični, mrežni, relacijski ali katera vrsta od distribuiranih, kot je Hadoop. Njihove glavne značilnosti so: standardizirana abstrakcija (npr. SQL), konsistenca, varnost in integriteta podatkov ter hiter dostop.

Centralizirani sistemi za varnostno kopiranje podatkov skrbijo za varnostno kopijo podatkov in omogočajo povrnitev podatkov skupine sistemov ali uporabniške programske opreme na stanje v točno določenem trenutku. Prav slednje je ključni kriterij, kdaj lahko sistem za varnostno kopiranje štejemo med vmesno programsko opremo.

Sistemi za gručenje združujejo dva ali več sorodnih informacijskih virov v en navidezni vir.

Delujejo lahko na več nivojih: na nivoju strojne opreme, na nivoju operacijskega sistema ali na nivoju vmesne programske opreme. Ne glede na nivo lahko delujejo v načinu aktivno-aktivno ali aktivno-pasivno. V prvem primeru sta dva ali več virov združeni v en navidezni večji vir, podobno kot pri izenačevalnikih bremena, v drugem načinu pa je del virov postavljen v pripravljenost in se aktivira v primeru napake na aktivnih virih. S tem načinom zagotavlja višjo razpoložljivost vira.

Porazdeljevalniki bremena služijo kot vstopna točka za zahteve po obdelavi opravil, ki jih dodeljujejo virom v zaledju na tak način, da so obremenitve virov porazdeljene glede na zahtevane parametre opravil in trenutno obremenjenost posameznega vira. Za učinkovito dinamično razporejanje mora imeti vsaka zahteva opredeljen vektor obremenitve (npr. pričakovana obremenitev procesorja, poraba pomnilnika, ipd.), mejne omejitve (npr. zakasnitev, najdaljši čas izvajanja, ipd.), razporejevalnik pa mora imeti informacijo o zmogljivosti posameznega vira in njegovi obremenitvi. Od uporabljenega algoritma porazdeljevanja je odvisno, kateremu viru bo posamezna zahteva dejansko dodeljena. Porazdeljevalniki opravil so v informacijski tehnologiji uporabljeni na različnih nivojih: od večprocesorskih sistemov, diskovnih sistemov, spletnih strežnikov, povezav v omrežju in drugih.

Medtem, ko so sistemi za upravljanje z zbirkami podatkov že dolgo poznali transakcije, so transakcije ostale v domeni zbirk podatkov omejenih na en strežnik oziroma instanco zbirke podatkov. Z integracijo različnih IR je rasla in dozorela potreba po koordinatorju transakcij med različnimi sistemi. Potreba je združila različna podjetja in posameznike v neprofitno združenje **X/Open Company**, ki se je leta 1996 združilo z **Open Software Foundation** in nastalo je združenje **The Open Group**, ki deluje še danes na področju sprejemanja odprtih standardov iz področja informacijskih tehnologij. Združenje je v letih od 1991 do 1996 sprejelo in objavilo serijo odprtih standardov iz področja distribuirane obdelave transakcij (*Distributed Transaction Processing: Reference Model* 1996; *Distributed TP: The XA Specification* 1991; *Distributed TP: The XA+ Specification* 1994; *Distributed Transaction Processing: The TX (Transaction Demarcation) Specification* 1995). Najpomembnejši rezultat je **XA vmesnik**. XA vmesnik je nabor funkcij in klicev med upravljavcem transakcij in upravljavcem vira, ki omogoča večnivojsko obravnavo transakcij ter uveljavljanje in razveljavljanje sprememb. Transakcijski nadzorniki preko XA vmesnika koordinirajo in nadzirajo transakcije med neodvisnimi viri. Poleg tega omogočajo dodajanje poslovne logike v obliki programov ali vstavkov.

Z razvojem tako transakcijskih nadzornikov, kot aplikacijskih strežnikov, je razlika postajala vse manjša. Danes transakcijske nadzornike le redko uporabljamo samostojno. Postali so pomemben del večjih gradnikov, predvsem aplikacijskih strežnikov.

1.4 Arhitektura EAI

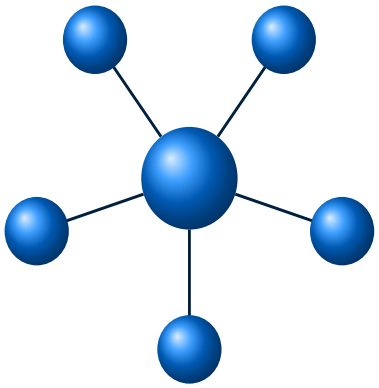
Same tehnike pa še ne prinesejo uporabne rešitve. Za učinkovito povezavo potrebujemo tudi ustrezno arhitekturo in gradnike. EAI prinaša tudi arhitekturni vidik povezovanja IS. Uvaja dva možna načina integracij:

- razdelilnik - udeleženec (angl. *Hub – Spoke*) in
- vodilo (angl. *Bus*).

1.4.1 Posrednik - udeleženec

Posrednik - udeleženec arhitektura izhaja iz računalniških omrežij, še pred tem pa iz logistike prometa in povezav. Predstavlja arhitekturo, v kateri so sistemi povezani med seboj preko osrednjega posrednika v topologiji zvezde, kot prikazuje Slika 2.

Slika 2: Topologija zvezde



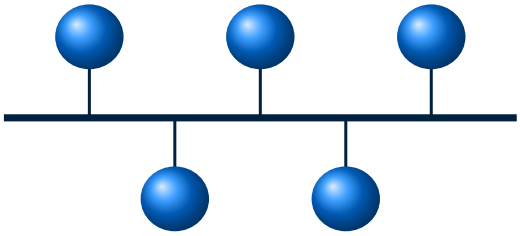
Vir: D. S. Linthicum, Enterprise Application Integration, 1999, str. 263.

V arhitekturi posrednik - udeleženec je potreben centralni posrednik za usmerjanje sporočil. Prilagojevalniki (angl. *Adapters*) za vse sodelujoče udeležence so lahko nameščeni bodisi pri posredniku, pri udeležencu ali kot ločena komponenta. Predvsem pristop z namestitvijo vseh prilagojevalnikov na posebej za to namenjen strežnik je priporočen zaradi lažjega vzdrževanja in upravljanja. Na ta način so vsi prilagojevalniki združeni na enem mestu. V vsakem primeru pa je posrednikova glavna funkcija izvedba usmerjanja sporočil. Centralni posrednik bi lahko bil šibka točka arhitekture v primeru okvare, kot tudi ozko grlo v komunikaciji zaradi vseh sporočil, ki morajo potovati preko posrednika. Obe pomanjkljivosti lahko odpravimo z dodajanjem fizičnih posrednikov, ki morajo logično delovati kot en gradnik. Najpogosteje to doseženo z gručenjem.

1.4.2 Vodilo

Arhitekturo vodila prikazuje Slika 3. Pri vodilu ni osrednjega posrednika. Vsak udeleženec mora imeti prilagojevalnik, ki prilagodi vhodna in izhodna sporočila, poleg tega pa mora imeti še informacijo potrebno za usmerjanje sporočil k prejemniku. Vodilo deluje kot prenosni medij med pošiljateljem in prejemnikom. V primeru vodila gre za skupni medij, v katerem lahko pride do kolizij, če več udeležencev želi hkrati oddajati sporočila. Zato je potrebno implementirati protokol uporabe vodila, preko katerega razrešimo kolizije. Eden od pristopov k reševanju problema kolizij je pristop z žetonom (Tanenbaum & Wetherall, 2011, str. 44), najbolj znana implementacija pa je Token Ring protokol podjetja IBM, ki je kasneje postal mednarodni standard (IEEE, 1998).

Slika 3: Topologija vodila



Vir: D. S. Linthicum, Enterprise Application Integration, 1999, str. 264.

Vodilo v arhitekturnem pogledu je logični gradnik arhitekture in je lahko izvedeno kot programska komponenta (npr. Enterprise Service Bus), kot omrežna povezava ali kot fizično vodilo znotraj strežnika. Praviloma pa gre za kombinacijo vseh naštetih izvedb. Kot logični gradnik tudi vodilo predstavlja šibko točko arhitekture v primeru okvare in kot ozko grlo v komunikaciji. Podobno kot pri posredniku, lahko tudi v primeru vodila uvedemo več paralelnih vodil, ki morajo delovati kot en logični gradnik.

1.5 Opredelitev arhitekture

Za opredelitev arhitekture IS in njene vloge v podjetju, moramo imeti pravilno predstavo o tem, kaj je namen obstoja in delovanja podjetja. Eno najbolj znanih opredelitev namena podjetja v svobodni ekonomiji je podal Friedman (2009, str. 133): „Obstaja samo ena socialna odgovornost podjetja - da uporabi vse svoje vire in izvaja aktivnosti, ki povečujejo dobiček v okviru danih pravil igre, kar pomeni, da deluje v odprti in svobodni konkurenci brez zavajanja in prevar.”

Zgornja neosebna opredelitev ni privlačna, izraža pa kruto realnost kapitalistične ureditve. Namen podjetja opredeljuje samo iz vidika lastnikov, ne pa tudi iz vidika širšega družbenega okolja. Družbi preko lokalnih in državnih institucij prepušča določanje pravil igre. Po drugi

strani pa lahko tudi na družbo gledamo kot na posebno vrsto podjetja, ki svoje prihodke ustvarja preko podjetij, za katera ustvarja okolje v katerem delujejo.

Drucker (2007) namen podjetja opredeljuje kot: „Namen podjetja mora obstajati izven podjetja samega. Dejansko mora obstajati v družbi, saj je podjetje organ družbe. Zato obstaja samo ena veljavna opredelitev namena podjetja: da ustvari in obdrži stranko." (Drucker, 2007, str. 31)

Na prvi pogled povsem drugačna opredelitev se ne razlikuje tako močno od Friedmanove, saj poudarja stranko, ki v končni fazi prinaša dobiček podjetju. Pomemben dodatek je zunanji pogled na podjetje iz vidika družbe.

Friedmanova opredelitev ne pojasnjuje obnašanja nekaterih strank, ki so pripravljene za isto blago ali storitev plačati več, če prihaja iz lokalnega okolja. Njihovo obnašanje postane racionalno, če obravnavamo posameznika in gospodinjstvo kot podjetje. Za zaposlene v podjetjih je vir prihodka njihovo plačilo, ki je v normalnih okoliščinah povezano z uspešnostjo podjetja. Več prihodkov, ki jih ustvarijo lokalna podjetja, pomeni več sredstev za lokalno skupnost in tako posredno ugodnejše okolje za lokalna podjetja in posameznike. Za nezaupanje do podjetij in skupnosti, ki prihajajo iz drugih okolij pa obstaja še en razlog: vse več podjetij se ne drži pravil igre. To privede do nezaupanja do podjetij, ki jih potencialne stranke ne poznajo in ustvarja negotovost. Nižja cena izdelkov in storitev je premija za premagovanje tveganja.

Pravila igre se prav tako razlikujejo od regije do regije in v različnih družbenih okoljih ter državah. To je še posebej pomembno v današnjem globaliziranem svetu, kjer je vse več nakupov opravljenih oddaljeno (Brewster et al., 2016).

To nas privede do širše opredelitve namena podjetja: „Namen podjetja ‚ni ustvarjanje dobička, pika.' Pravi namen je ustvarjanje dobička tako, da lahko ljudje v podjetju počnejo, kar želijo ter z njimi njihove družine in drugi, ki so jim blizu in daleč. Handy trdi, da bi morala biti podjetja upravljana kot skupnosti, kar tudi so: skupnosti zaposlenih organiziranih, da strežejo skupnostim strank. Ko je to doseženo z upoštevanjem moralnih vrednot, sistem zadrži notranjo in zunanjo integriteto." (Handy, 2002, str. 52)

Zgornja definicija zajema tako podjetje in njihove lastnike, kot posameznike v podjetju in njegovi okolici ter jih združuje v skupnosti. Manjka pa tesnejša povezava med skupnostmi znotraj podjetja in skupnostmi strank. V lokalnem okolju se ti dve skupnosti pogosto zelo prepletata.

Najbližja mi je dopolnitev Friedmanove opredelitve, saj podjetja praviloma ustanovijo posame-

zniki ali druga že obstoječa podjetja. Ustanovitev je povezana s poslovno idejo in pričakovanji. Skupni imenovalec pričakovanj je povečanje vrednosti za vlagatelje *na dolgi rok*.

Vse naštete opredelitve predpostavljajo razpršenost kapitala, s čimer interesi podjetja posredno predstavljajo interes večine ljudi. Ob globalizaciji in koncentraciji kapitala Marx (1967), pa interesi podjetij predstavljajo interes manjšine. Začeto razmišljanje pa že presega obseg tega dela.

Za potrebe tega dela in za poenostavitev bom v tem delu predpostavil, da je namen podjetja povečevanje kapitala za vlagatelje na dolgi rok.

Britton & Bye (2004) opredelita arhitekturo IS kot: „IT arhitektura je rešitev problema ‚Kako naj strukturiram IT aplikacije, da bodo najbolje zadovoljile potrebe mojega podjetja?'"

V tej definiciji je izpostavljena povezava med arhitekturo IS in potrebami podjetja. Če povečevanja vrednosti kapitala povežem z dobičkom in dobiček opredelim kot razliko med ustvarjeno vrednostjo in vložki, potem mora arhitektura IS povečevati ustvarjeno vrednost in/ali znižati potreben vložek. Vložek v arhitekturo IS mora biti nižji od razlike med povečano vrednostjo in znižanimi vložki. To pa ne velja samo za arhitekturo IS, temveč za vse vložke v IT rešitve.

Perry & Wolf (1992, str. 41–43) obravnavata arhitekturo programskih rešitev, ki je veljavna še danes tudi za kompleksnejše informacijske sisteme in rešitve. Vzporednice vlečeta iz gradbene stroke, od koder sta povzela različne vidike in nivoje opisa končnega izdelka. Arhitekturo sta postavila v okvir procesa nastanka programske rešitve:

- **zahteve** opredeljujejo informacije, obdelave ter lastnosti informacij in obdelav, kot jih potrebuje uporabnik oziroma naročnik;
- **arhitektura** opredeljuje izbiro arhitekturnih elementov, njihovo interakcijo, omejitve elementov in interakcij potrebnih za zagotovitev ogrodja, znotraj katerega bodo izpolnjeni pogoji za izpolnitev zahtev in bodo služili kot podlaga za zasnovo;
- **zasnova** opredeljuje razdelitev na module, podrobnosti vmesnikov elementov, njihovih algoritmov, procedur in podatkovnih struktur za podporo arhitekturi in za izpolnitev zahtev;
- **izvedba** je konkretizacija algoritmov in podatkovnih struktur, ki izpolnjujejo skladnost z zasnovo, arhitekturo in zahtevami.

Zgornja klasifikacija je groba poenostavitev strukture IR, uporabljena predvsem za predstavitev, kakšna je relacija med arhitekturo na eni strani ter zahtevami, zasnovo in izvedbo na drugi strani. Obstajajo tudi pristopi k razvoju IR, ki niso skladni z uporabljeno klasifikacijo. Primer takšnega pristopa je **raziskovalno programiranje** (angl. *Exploratory Programming*) (Hölzle,

1994), vendar za uporabo v tem delu ni relevantna.

Kruchten (1995, str. 43) povzema opredelitev arhitekture po Perry & Wolf (1992) in (Gacek, Abd-Allah, Clark & Boehm, 1995) v obliki enačbe:

$$Arhitektura\ programske\ opreme = \{Gradniki, Interakcije, Zahteve/Omejitve\} \quad (1)$$

Kruchten (1995, str. 42–43) naprej za opis arhitekture uporablja 5 pogledov:

- logični pogled,
- procesni pogled,
- razvojni pogled,
- fizični pogled in
- scenariji.

Enačba je dovolj generična tudi za arhitekturo IS, čeprav je izvirno zastavljena za razvoj programske opreme. Seznam in opisi pogledov pa zahtevajo nekaj prilagoditve. Prilagojeni opisi pogledov

Logični pogled predstavlja gradnike arhitekture IS in logične povezave med njimi, preko katerih poteka interakcija. V času snovanja arhitekture najprej postavimo osnovne gradnike, ki niso predmet razvoja IS. Primer takšnih gradnikov so zbirke podatkov, mehanizmi za asinhrono komunikacijo, spletni strežniki, aplikacijski strežniki, servisno vodilo in drugi. Praviloma gre za elemente srednje programske opreme. V okviru logične arhitekture opredelimo tudi povezave med njimi, pri čemer je pomemben vidik, preferenca povezav: obvezne, zaželene, nezaželene in prepovedane. Opredelimo dodatne lastnosti in s tem omejitve, ki izhajajo iz zahtev za IS. Pomemben del logičnega pogleda je umestitev načrtovanih uporabniških gradnikov v arhitekturo in njihovo interakcijo med seboj ter z drugimi gradniki IS.

Procesni pogled obravnava nekatere nefunkcionalne zahteve, kot sta zmogljivost in razpoložljivost. V tem pogledu rešujemo vprašanje paralelizma in distribucije, integriteto sistema in toleranco v primeru okvar. Procesni pogled prikazuje tudi grupiranje glavnih abstrakcij iz logičnega pogleda v elemente procesnega pogleda: katerih gradniki bodo sodelovali v posameznih procesih. Procesni pogled je lahko sestavljen iz več pogledov z različnimi nivoji abstrakcije. Vsak nivo obravnava različne zahteve in vidike sistema. Na najvišjem nivoju lahko procesni pogled razumemo kot nabor izvajajočih neodvisnih logičnih skupin gradnikov, ki opravljajo zaključeno funkcijo (poimenujmo jih „procesi") in se izvajajo distribuirano na množici osnovnih gradnikov povezanih med seboj. Ob uporabi pojma **distribuirani gradniki**, imam v mislih neodvisne programske gradnike. Gradnik lahko obravnavamo kot objekt, ki ni omejen na samo

en program, programski jezik ali implementacijo (Jurič, Heričko & Rozman, 2000, str. 3).

Razvojni pogled se osredotoča na dekompozicijo IS na programske komponente, katere lahko razdelimo manjšim skupinam v razvoj ali kombiniramo obstoječe komponente z novimi. Komponente naj bodo organizirane v več nivojev, ki imajo med seboj dobro definirane vmesnike. Pri dekompoziciji na komponente moramo paziti predvsem na možnost ponovne uporabe in uporabo že razvitih komponent ali obstoječih IR. Natančno je potrebno opredeliti pravila za dekompozicijo na komponente in združevanje v večje enote ter predvideti bodoče razširitve sistema. Poleg funkcionalnih zahtev moramo v razvojnem pogledu vzpostaviti in upoštevati pravila za lažji razvoj, upravljanje sistema ter se prilagoditi razpoložljivim omejitvam orodij, ki jih imamo na voljo.

Fizični pogled primarno prikazuje nefunkcionalne zahteve za IS, kot so: razpoložljivost, zanesljivost (toleranca v primeru napak), zmogljivost (prepustnost) in skalabilnost. Gradniki se izvajajo na mreži strežnikov ali procesnih vozlišč. V prejšnjih pogledih identificirani gradniki: procesi, aplikacijski strežniki, servisno vodilo, zaledni sistemi in drugi so v fizičnem pogledu postavljeni na različne fizične strežnike. Pri tem določimo, kateri gradniki so na podlagi zahtev kritični in jih je potrebno ščititi s podvajanjem, kje pričakujemo s časom povečano obremenitev in pripravimo način za postopno povečevanje zmogljivosti, kot so strežniške farme s posredniki (angl. *Proxy*) ali z distribucijo bremen.

Scenariji povezujejo vse poglede v celoto in omogočajo preveritev skupnega delovanja gradnikov iz prejšnjih štirih pogledov skozi manjši nabor pomembnih scenarijev, ki so natančno opredeljeni primeri podmnožice širšega nabora primerov uporabe (angl. *Use Cases*). Zanje pripravimo ustrezne avtomatizirane preveritve v obliki skript ali drugih orodij za testiranje tako, da simuliramo najpomembnejša zaporedja interakcij med gradniki in procesi.

Z uporabo pravkar navedenih pogledov v fazi snovanja določimo lastnosti arhitekture do takšnega nivoja podrobnosti, ki omogoča začetek implementacije sistema. V odvisnosti od uporabljene metodologije, bo zasnova arhitekture lahko narejena v celoti pred začetkom implementacije ali pa bo pripravljena postopoma v iterativnem procesu z vse več podrobnostmi. Ne glede na izbrano metodologijo pa se mora snovalec zavedati, da vsaka opredelitev lastnosti arhitekture pomeni po drugi strani omejitve, ki lahko prizadenejo uporabnost arhitekture in z njo končnega sistema oziroma mu lahko močno skrajšajo uporabno življenjsko dobo. V tem primeru mora snovalec slediti Einsteinovemu reku „Modeli naj bodo enostavni kolikor je le mogoče, vendar ne več kot toliko.” (May, 2004, str. 793).

Pogosto pri snovanju arhitekture izhajamo iz obstoječega stanja. V takšnem primeru moramo

popisati arhitekturo obstoječega sistema. Pri tem si pomagamo z obstoječo dokumentacijo in povratnim inženiringom. Zgoraj opisani pogledi arhitekture so primerni tudi za takšen opis. Pri povratnem inženiringu obstaja več nevarnosti:

- prvotne zahteve praviloma niso znane,
- identifikacija gradnikov ni vselej očitna,
- vmesniki med gradniki so praviloma skriti in jih je težko podrobno opisati,
- identifikacija procesov zahteva sodelovanje izkušenih uporabnikov in
- pogosto je težko ločiti med fizičnim in logičnim pogledom.

Za dobro izvedbo povratnega inženiringa na nivoju arhitekture je najboljša možnost sodelovanje najizkušenejših snovalcev in arhitektov.

Za opis arhitekture IS je primeren **Jezik za opis arhitekture** (ADL) (Medvidovic & Taylor, 2000).

Morris (2013, str. 16–17) opredeli dobre lastnosti arhitekture: „Skrivnost dobre arhitekture je v skrivanju visoke kompleksnosti in predstavitvi na enostaven način, da jo lahko razume vsak, ki potrebuje razumevanje, in lahko hitro ponudi pravo rešitev za trenutne in/ali bodoče potrebe poslovanja."

Ne glede na pot, ki je pripeljala do arhitekture IS, je skupna zaželena lastnost rezultata robustnost in prilagodljivost na nove zahteve ter s tem na spremembe. Prav možnost spreminjanja v prihodnosti pa se s časoma zmanjšuje, sistem pa postaja vse bolj krhek in občutljiv (Brooks, Jr., 1975, str. 120–123).

Perry & Wolf (1992, str. 43) navaja dva razloga, zaradi katerih prihaja do zmanjševanja možnosti sprememb ter povečane krhkosti in občutljivosti IS:

- arhitekturna erozija in
- arhitekturni odmik.

Do **arhitekturne erozije** prihaja s časom zaradi kršenja pravil arhitekture. Kršitve pogosto vodijo do vse večjih težav in krhkosti sistema. V končni fazi pa lahko povzročijo katastrofalne posledice in daljše izpade. Primeri kršitev, ki povzročajo arhitekturno erozijo so neupoštevanje zahtev po visoki zanesljivosti - postavljanje gradnikov na samostojne strežnike namesto v gruče, premajhno dimenzioniranje virov na strežnikih brez možnosti enostavnega dodajanja virov v obliki strežniških farm, opustitev posrednikov, ki onemogoča prenos gradnikov na nove, močnejše strežnike, opustitev mehanizmov, ki omogočajo spremljanje delovanja sistema in

podobno.

Arhitekturni odmik je posledica neupoštevanja arhitekturnih pravil in navodil. Neupoštevanje vodi bolj v nefleksibilnost in praviloma ne povzroča operativnih težav. Posledici pa sta nepreglednost in nekonsistentnost. Ti naprej vodita v pogostejše kršitve pravil arhitekture, ki tako postaja vse manj pregledna. Primeri so izločanja in podvajanje poslovne logike v več gradnikih, kjer je zahtevano centralno upravljanje poslovne logike, dostopi do zalednih sistemov mimo transakcijskih koordinatorjev ali storitvenega vodila, najpogostejši primer pa je dostop do različnih zbirk podatkov iz uporabniškega vmesnika mimo vseh vmesnih slojev. Slednje onemogoča ali vsaj otežuje zamenjavo posamezne zbirke podatkov, gradnika in/ali celotnega zalednega sistema z novim, ne da bi analizirali celoten sistem in poiskali vse reference na del sistema, katerega želimo nadomestiti.

1.6 Pomanjkljivosti EAI

EAI je pripeljala arhitekturni vidik v kompleksne IS in s tem postavila osnovo za sistematičen pristop k uvedbi nadzorovanega okolja za integracijo obstoječih IR z novimi rešitvami v celovit IS. Kljub temu pa ni dosegla zaželenega poslovnega cilja: znižanja stroškov IS. Vzroki pa so v številnih neuspelih projektih uvedbe EAI.

Leta 2003 so poročali o 70% neuspelih EAI projektih (Trotta, 2003). Razlogi za tako velik delež neuspešnih projektov so bili naslednji:

- stalne spremembe - zahteve so se med izvedbo projektov nenehno spreminjale,
- pomanjkanje znanja in strokovnjakov iz področja EAI,
- nestandardizirane rešitve,
- obravnava EAI kot orodja namesto kot sistema,
- obravnava vmesnikov kot znanost in ne kot umetnost,
- nezadostno upoštevanje podrobnosti in
- nejasna odgovornost.

Obvladovanje sprememb med izvajanjem EAI projektov je ključnega pomena za uspešen zaključek. Najpogostejša napaka pri izvedbi EAI projektov je bil fiksni projektni načrt in proračun, s katerim so predvsem tehnično usmerjeni navdušenci in zagovorniki poskušali prepričati odločevalce v investicije v EAI projekte. Spreminjajoča narava poslovanja in okolja pa takšnega pristopa ne dovoljuje. Pri uvedbi daljših projektov je potreben nivojski pristop, ki dovoljuje prilagajanje skozi čas trajanja projekta, pri ocenjevanju stroškov pa je potrebno upoštevati tako spreminjajoče zahteve, kot tudi vse koristi in stroške ne samo v času projekta, tudi v času življenjske dobe IS. Najprimernejša metoda je iterativna ASK analiza.

Predvsem pomanjkanje znanja in strokovnjakov ter pomanjkanje standardov oziroma nestandardizirane rešitve sta med seboj tesno povezana. Vsak EAI projekt je uvajal svoja pravila za oblikovanje sporočil med sistemi in za skupni podatkovni model, kar je vsakič znova zahtevalo razvoj prilagojenih rešitev v odvisnosti od problemske domene in sistemov, ki so bili integrirani v posameznem projektu. Strokovnjaki s tako širokimi znanji pa so na voljo le v omejenem obsegu. Dodatno je k pomanjkanju pripomogla tudi kompleksnost EAI rešitev. Posledično so projekti zamujali in bili celo opuščeni. Z uporabo standardnih mehanizmov, kot sta XML in SOAP, se je kasneje uveljavila SOA arhitektura, ki pa je prinesla še druge nove rešitve in pristope.

Prepogost pristop k EAI projektom je bil zaradi EAI samega, pri čemer so pobudo prevzeli IT strokovnjaki, pri tem pa pozabili na osnovni cilj vsakega IS, ki je izpolnitev poslovnih zahtev. Projekt, ki ne izpolnjuje poslovnih zahtev pa je že v začetku obsojen na neuspeh.

Osnovni cilj EAI je povezovanje poslovnih gradnikov ob ohranjanju ali povečevanju vrednosti poslovnih podatkov. Izdelava abstraktnih vmesnikov za posamezno organizacijsko enoto zamegljuje cilj EAI. Za uspešno izvedbo je potrebno vzpostaviti pogajalski proces med vsemi vpletenimi organizacijskimi enotami in skozi ta proces uskladiti potrebe in definirati vmesnike za vse vpletene organizacijske enote. V nasprotnem primeru bo soglasje težko doseči, projekt bo trajal dlje in stal več od predvidenega, končni rezultat pa najverjetneje ne bo izpolnjeval zahtev in pričakovanj naročnika.

EAI povezuje več organizacijskih enot, kot tudi različnih podjetij med seboj. Zaradi povečevanja obsega sistema, analitiki pogosto prezrejo podrobnosti v zahtevah naročnikov posameznih organizacijskih enot, naročniki pa zaradi časovnega pritiska in trenutnih potreb, ne razmišljajo dovolj podrobno o prihodnjih potrebah. Prav te manjkajoče podrobnosti pa lahko močno vplivajo na uporabnost sistema kasneje, ko je sistem že v produkcijski uporabi.

V IS, ki presega meje organizacijskih enot in včasih tudi meje podjetij, bo prej ko slej prišlo do težav v produkcijskem obratovanju, h katerim je potrebno pristopiti sistematično. To pa zahteva kombinacijo informacijskih in poslovnih veščin. Pri reševanju pogosto naletimo na notranje politične ovire, kar lahko rešitev preprostega vprašanja „Kam bo organizacijsko umeščena skupina za podporo uporabnikom?” zavleče za mesece.

2 SOA

Ob številnih odprtih vprašanjih, ki jih pušča EAI, pomanjkanju standardiziranih rešitev in drugih pomanjkljivostih, se je odprl prostor za iskanje rešitev, ki bi dale odgovor na zastavljene izzive.

SOA je eden od odgovorov, ki pa predstavlja veliko več, kot samo arhitekturo za integracijo (Erl, 2005, str. 2).

Za potrebe tega dela uporabljam besedno zvezo **SOA arhitektura** za ožji arhitekturni pomen, medtem ko pojma **SOA** in **storitvena usmerjenost** uporabljam za širši pomen.

Pomembna prednost SOA arhitekture pred EAI je tudi v tem, da ponuja sistematičen pristop tako k integraciji obstoječih sistemov, kot k izgradnji novih rešitev. To omogoča podjetju, da se loti uvedbe SOA arhitekture na več načinov, kakor najbolj ustreza trenutnim in bodočim potrebam podjetja. Lahko se loti najprej integracije obstoječih sistemov z oblikovanjem poslovnih storitev, lahko se loti prenove IS z razvojem novih komponent kot poslovnimi storitvami ali poljubno kombinacijo obojega. Zaradi velike kompleksnosti in s tem tudi obsega, odsvetujem pristop z zamenjavo ali predelavo vseh obstoječih rešitev naenkrat. Postopen pristop ponuja največjo verjetnost uspeha (Hribar Rajterič, 2013) in bo omogočila tudi spremembo v načinu razmišljanja in ustrezno kadrovsko dopolnitev oziroma zamenjave.

SOA sodi med številne zlorabljene in pogosto različno ali celo napačno razumljene pojme v informacijski tehnologiji. Najpogostejše zlorabe so uporaba SOA v marketinške namene kot del imena komercialnega produkta (IBM) in navajanje SOA kot zmožnosti posameznih produktov. Pogosto napačno prepričanje je tudi, da že uporaba in konzumiranje spletnih storitev predstavlja SOA arhitekturo (Jurič, 2005). K tem zmotam znatno pripomore tudi nerodno izbrano ime, ki že vsebuje besedo „arhitektura" in s tem zavaja veliko ljudi. Razlikovati bi morali SOA kot arhitekturo IS in **objektno usmerjenost**, ki predstavlja pristop k izgradnji in obratovanju IS. Tega pa ne razlikuje večina strokovnjakov vključno z najbolj priznanimi.

2.1 Definicija SOA

Pregled zelo različnih definicij pojma SOA potrjuje tezo o različnem razumevanju:

„Storitveno usmerjena arhitektura (SOA) je način zasnove, razvoja, postavitve in upravljanja sistemov, katerih glavna značilnost je groba granularnost storitev in njihovih odjemalcev. Storitve predstavljajo poslovne funkcije, katere je mogoče večkrat ponovno uporabiti. Odjemalci storitev sestavljajo aplikacije ali sisteme z uporabo standardnih vmesnikov." (Lewis, Smith & Kontogiannis, 2010, str. 1)

„SOA je arhitekturni stil, čigar cilj je doseči šibko sklopljenost med programskimi komponentami, ki komunicirajo med seboj. Storitev je enota dela, ki jo izvede ponudnik storitve za dosego želenega končnega rezultata odjemalca storitve. Tako ponudnik, kot odjemalec storitve sta vlogi, ki ju prevzamejo programske komponente v imenu njihovih lastnikov." (He, 2003, str. 1)

„Storitveno usmerjena arhitektura (SOA) je arhitektura programske opreme temelječa na ključnih konceptih: uporabniški vmesnik aplikacije, storitev, repozitorij storitev in storitveno vodilo. Storitev je sestavljena iz pogodbe, enega ali več vmesnikov in implementacije." (Krafzig, Banke & Slama, 2004, str. 56)

enquoteSOA je arhitekturni koncept v katerem so vse funkcije ali storitve definirane z opisnim jezikom in kjer je njihove vmesnike mogoče odkriti v omrežju. Vmesnik je definiran nevtralno neodvisno od strojne platforme, operacijskega sistema in programskega jezika v katerem je storitev implementirana. (Maréchaux, 2006, str. 2)

Thomas Erl, kot priznan strokovnjak in najpopularnejši avtor iz področja storitvene usmerjenosti, storitveno usmerjenost opredeljuje širše. Opozarja na napačno predstavo o tem, kaj SOA je: „**Lažna SOA:** Mnogi so zavedeni v prepričanje, da tehnični vidik storitveno usmerjene arhitekture sestavljajo preprosto spletne storitve. To je pogosta a zelo nevarna predpostavka, ki vodi v številne napake podjetij, ki nameravajo uvesti SOA. Pričakovanje, da bodo obljubljene koristi trenutno sprejetega razumevanja SOA, dosežene izključno z dodatnimi investicijami v platforme za spletne storitve." (Erl, 2005, str. 2–3)

Erl opredeljuje tudi idealno storitveno usmerjenost, za katero pa priznava, da v praksi ni dosegljiva: „**Idealna SOA:** Storitvena usmerjenost predstavlja ideal vizije sveta, v katerem so viri strogo razmejeni in konsistentno predstavljeni. Ob uporabi te vizije na IT arhitekturi, storitvena usmerjenost vzpostavi univerzalni model, v katerem sta tehnična logika, kot tudi poslovna logika skladni s to vizijo. Model je enako veljaven na različnih nivojih: opravilo, rešitev, podjetje, skupnost in širše. S sprejetjem vizije so pretekle tehnične in načelne razlike prekrite s sloji abstrakcij, ki prinašajo globalno sprejet standard za predstavitev logike in informacij. Ta nivo standardizacije omogoča ogromno potencialno korist podjetjem, saj mnoge tradicionalne izzive, ki jih prinaša stalno spreminjajoče IT okolje, lahko rešimo z uporabo standardiziranih slojev." (Erl, 2005, str. 3)

„Realna SOA zahteva spremembo načina razmišljanja: na poslovno logiko moramo sedaj gledati v kontekstu storitvene usmerjenosti. Uveljavljanje tega konteksta pa zahteva spremembe v tehnični logiki, saj moramo sedaj rešitve graditi na tak način, da podpirajo storitveno usmerjenost. Poleg tega potrebujemo novo infrastrukturo, ki je zmožna gostiti storitveno usmerjeno tehnično logiko, ta pa prinaša nove tehnologije in infrastrukturne zahteve.

Realna SOA zahteva dejanske spremembe, pravilno predvidevanje in resnično predanost. Predvsem pa potrebuje nadzor in usmeritve." (Erl, 2005, str. 4)

„Sodobna SOA predstavlja odprto, agilno, razširljivo, federirano, sestavljivo arhitekturo, ki je

zgrajena iz avtonomnih, QoS zmožnih, od dobaviteljev neodvisnih, interoperabilnih, odkriljivih in potencialno ponovno uporabljivih storitev implementiranih kot spletne storitve." (Erl, 2005, str. 54)

Nobena od naštetih definicij ne zajema pravega bistva SOA kot storitvene usmerjenosti. Izjeme so Erlove definicije idealne, realne in sodobne SOA, ki pa so preveč abstraktne, da bi jih lahko uporabili brez dodatne razlage. Erl sicer ponuja te razlage v seriji svojih knjig, kar pa od bralca zahteva znaten vložek časa in razumevanja.

Za potrebe pričujočega dela uporabljam naslednjo definicijo SOA kot storitvene usmerjenosti: „SOA kot storitvena usmerjenost je nabor standardov, priporočil, usmeritev in predvsem vzorcev, ki omogočajo vzpostavitev nadzorovanega, distribuiranega okolja in izgradnjo storitev, v katerem lahko na zanesljiv, preprost in učinkovit način prilagajamo storitve spreminjajočim potrebam podjetja na standardiziran in tehnološko neodvisen način."

Groba granularnost, šibka sklopljenost, ponovna uporaba storitev, kompozicija storitev, upravljanje storitev, orkestracija storitev in drugi pogosto uporabljeni pojmi v literaturi, so samo orodja za dosego cilja in posledice narave SOA arhitekture.

Cilj storitvene usmerjenosti je omogočiti hitro prilagajanje IS poslovnim potrebam in spremembam. Za ta cilj je potrebno skrajšati čas od zaznave potrebe po spremembi do izvedbe v IS. Čas od zaznave do izvedbe je dolg zaradi vseh procesov, ki jih je potrebno izvesti za izvedbo spremembe v IS: analiza, specifikacija, načrtovanje, snovanje, implementacija, testiranje in namestitev. Ti procesi pa so posledica semantičnega prepada med poslovanjem in informacijsko tehnologijo. Potrebno je zmanjšati semantični prepad med poslovnim delom podjetja in informacijsko tehnologijo. To lahko dosežemo tako, da spremembe IS lahko izvedejo zaposleni v poslovnem delu brez sodelovanja IT oddelka oziroma z minimalnim sodelovanjem IT oddelka. Storitvena usmerjenost to omogoča skozi **kompozicijo storitev** in **orkestracijo storitev** kot ključni orodji oziroma aktivnosti, ki ju želimo podpreti s SOA arhitekturo. Kompozicija storitev je statično, vnaprej programirano sestavljanje tehničnih storitev, drobno granuliranih poslovnih storitev in funkcij v poslovno storitev. Kompozicija je nadgradnja koncepta dedovanja iz objektno usmerjenega programiranja. Kompozicija še vedno zahteva IT razvoj, ki pa je v primeru kompozicije storitev enostavnejši in hitrejši od klasičnega razvoja. Orkestracija je po drugi strani namenjena poslovnim uporabnikom, ki lahko s pomočjo orodij znotraj SOA arhitekture, sami povežejo obstoječe storitve v kompleksnejše poslovne storitve in procese. Prednost orkestracije storitev je v znatno krajšem času od zaznane poslovne potrebe do izvedbe v IS. V obeh primerih pa je rezultat nova storitev, ki združuje nižjenivojske storitve in dodatno logiko, zato je obsežnejša in kompleksnejša od prej obstoječih storitev. To pa pomeni, da so takšne storitve **grobo granulirane**. Groba

granularnost torej ni cilj storitvene usmerjenosti in SOA arhitekture, temveč njena posledica. Pri tem absolutnih vrednosti za drobno granularnost in grobo granularnost ne moremo opredeliti vnaprej, saj je vsaka storitev kandidat za ponovno uporabo v novi, bolj grobo granulirani storitvi. Da pa orkestracija storitev lahko deluje, mora uporabniku predstaviti nabor razpoložljivih storitev in deklarativnega dela vmesnikov, ki jih pogosto imenujemo tudi pogodba (angl. *Contract*) ali dogovor (angl. *Agreement*). V ta namen sta potrebna gradnika SOA arhitekture imenovana **imenik storitev** (angl. *Service Registry*) in **repozitorij storitev** (angl. *Service Repository*). V imeniku storitev morajo biti zavedeni osnovni podatki vseh storitev, ki so na voljo orkestraciji, dobre prakse pa priporočajo vključitev vseh storitev, ne glede na to, ali nastopajo pri orkestraciji storitev, ali ne. Pogodbe storitev so shranjene ločeno v repozitoriju storitev. Rezultat orkestracije je nova storitev, ki mora biti prav tako shranjena tako v imenik storitev, njena pogodba pa v repozitorij storitev. Tak pristop zagotovi centralno shrambo in evidenco tako lokacij storitev (implementacije), kot njihovih pogodb in zmanjšuje možnost redundance in nekonsistentnosti. Repozitorij storitev pa ni namenjen izvajanju storitev. V ta namen potrebujemo še tretji gradnik, najpomembnejši gradnik SOA arhitekture: **storitveno vodilo**.

Kljub številnim standardom in gradnikom arhitekture ter s tem kompleksnostjo, je arhitektom in izvajalcem prepuščeno veliko svobode, saj vsi vidiki niso standardizirani. S kompleksnostjo in posledičnim napačnim razumevanjem pa so povezani neuspešni projekti, ki so botrovali številnim kritikam SOA-e. Leta 2009 se pojavi večje število objav in člankov nenaklonjenih SOA pristopom (Thomas Manes, 2009) in celo direktno nasprotujočih (Král & Žemlička, 2009, str. 271). Pri tem pa je le malokdo nasprotoval konceptom storitvene usmerjenosti. Največjih kritik je bila deležna draga programska oprema, ki služi kot osnova za SOA arhitekturo, dolgi in pogosto neuspešni projekti ter pomanjkanje kadra in znanja iz področja uvajanja SOA arhitekture. Temu je pripomogel tudi razmah računalništva v oblaku, ki pa ni alternativa ali konkurenca SOA storitveni usmerjenosti. Interesi proizvajalcev programske in strojne opreme so se preselili v promocijo storitev v oblaku. Storitve v oblaku pa niso prinesle dodatnih prihrankov, kot kaže tudi preglednica vložkov v uvodu tega dela.

Razlogi za kritike so utemeljeni, saj se SOA je in se še uvaja ter uporablja na načine, ki so vnaprej obsojeni na neuspeh. Slabe prakse so tako razširjene, da so se izoblikovali vzorci oziroma SOA protivzorci (Král & Žemlička, 2009, str. 271) in (Galster, Lapre & Avgeriou, 2014, str. 79–80). Za uspešno uvedbo in uporabo storitvene usmerjenosti in SOA arhitekture je ključna sprememba v načinu razmišljanja.

SOA kot storitvena usmerjenost je opredeljena na konceptualnem nivoju. Je množica standardov, usmeritev, vzorcev in nasvetov, kako doseči cilj, ki ni samo SOA arhitektura. SOA arhitektura je

zgolj tehnični vidik, ki omogoča dosego končnega cilja: prilagodljiv IS. Pri tem je pomemben tudi ekonomski vidik ob upoštevanju koristi in potrebnih vložkov.

SOA v širšem pomenu predstavlja:

- storitveno usmerjeno snovanje,
- storitveno usmerjeno arhitekturo,
- storitveno usmerjeno analizo,
- storitveno usmerjen razvoj in
- upravljanje sistema.

Papazoglou & Heuvel (2006) predlagata storitveno usmerjeno metodologijo za snovanje in razvoj, ki sledi konceptom storitvene usmerjenosti v vseh fazah življenjskega cikla. Obravnava metodologije presega obseg tega dela.

2.1.1 Protivzorci SOA

Sledi opis najpogostejših primerov napačnega razumevanja in slabega upravljanja uvedbe SOA, ki si je „zaslužil" svoje ime: protivzorci. **Protivzorec** je napačna praksa, ki je pogosto v uporabi (Král & Žemlička, 2009, str. 271).

2.1.1.1 Pretirana standardizacija SOA

Standardizacija je potrebna in koristna. Pretirana standardizacija vseh vidikov SOA pa vodi v veliko število SOA standardov, katerih obseg in kompleksnost rasteta čez vse meje. Veliko število standardov povezanih s SOA je v stalnem razvoju in spreminjanju. Sledenje takšnemu tempu sprememb je za podjetja lahko velik izziv. Praviloma ga obvladujejo samo veliki dobavitelji s svojimi komercialnimi produkti. Isti dobavitelji so hkrati tudi ključni člani standardizacijskih združenj in tako narekujejo hitrost sprememb. S tem pa se ustvarja odvisnost od velikih dobaviteljev. Poleg tega veliko število standardov podrobno opredeljuje lastnosti vmesnikov storitev, ki morajo biti zato drobno granulirane, kar pa je v nasprotju s storitveno usmerjenostjo.

2.1.1.2 Nič novega protivzorec

Številni razvijalci programske opreme z izkušnjami iz objektne usmerjenosti, ne vidijo v storitveni usmerjenosti nič pomembno novega. Zato uporabljajo standardne principe objektne usmerjenosti v storitveno usmerjenem okolju, kar povzroča številne ovire v SOA arhitekturi in je vzrok tudi drugim protivzorcem SOA. Najpogostejši primer je uporaba SOAP ovojnice kot tanek ovoj okoli RPC klicev in s tem izdelavo drobno granuliranih storitev. Tak pristop sam po sebi še

ni napačen, kadar je uporabljen za nizkonivojske storitve, ko pa je uporabljen za konstrukcijo poslovnih storitev, krši storitveno usmerjene principe in pogosto vodi do neuspešne vpeljave SOA.

2.1.1.3 Drobno granulirani vmesniki in storitve

SOA je pogosto zasnovana kot ogromen nabor drobno granuliranih storitev. Vmesniki teh storitev temeljijo na izmenjavi velikega števila preprostih sporočil z enostavno sintakso in preprosto vsebino. To predstavlja koncept klicev oddaljenih procedur (RPC). Pogosto so preproste funkcije ovite z uporabo WSDL in SOAP ter izpostavljene kot spletne storitve v SOA arhitekturi. V nekaterih primerih so drobno granulirane storitve res potrebne, vendar je v tem primeru potrebno uveljaviti upravljanje storitev. Običajno je to posledica nerazumevanja storitvene usmerjenosti. Tedaj je najlažje implementirati storitve z obstoječim znanjem in preprosto oviti metode objektov in jih izpostaviti kot vmesnike storitev. Drobno granulirani vmesniki niso primerni za poslovne storitve in avtomatizacijo procesov, v katere so vključeni ljudje. Rezultat drobno granuliranih vmesnikov je lahko tudi posledica pretirane standardizacije. Rešitev je v eliminaciji odvečnih standardov, ozaveščanju IT osebja o nevarnostih drobno granuliranih vmesnikov in v razvoju uporabniško usmerjenih sporočilnih formatov temelječih na človeku razumljivih jezikih.

2.1.1.4 Veliki pok

Pogosto želi naročnik celoten sistem temelječ na SOA arhitekturi zgraditi naenkrat. Praviloma je ta protivzorec povezan s protivzorcem „Vse moramo prenoviti" oziroma z opustitvijo obstoječih sistemov. V prenovo sodijo tudi sistemi, katere je potrebno ponovno nastaviti in prilagoditi. Gre za strategijo, ki je koristna za velike dobavitelje in predstavlja močno odvisnost od dobaviteljev. Drobno granulirani vmesniki povzročijo, da storitve niso dovolj neodvisne. Skrita odvisnost storitev pa znižuje možnost postopnega razvoja. Drobno granulirani vmesniki ob zamenjavi tehnologije zahtevajo tudi vsebinske spremembe. Veliki pok pa močno obremenjuje tudi uporabnike sistemov, ki se morajo svojega dela ponovno učiti. Predelava s pristopom velikega poka mora vsebovati tudi uporabo grobo granuliranih uporabniških vmesnikov.

2.1.1.5 Pogojne odvisnosti

V določenih primerih predelava velikosti ali granularnosti storitve in/ali vmesnika vpliva na velikost in granularnost druge storitve in/ali njenih vmesnikov. Odvisnost je lahko linearna, eksponentna, logaritmična ali obratno sorazmerna. Do težave pride, ko vmesnik odvisne storitve postane drobno granuliran do tolikšne mere, da se pojavijo prej našteti protivzorci. Rešitev te težave je v zagotavljanju neodvisnosti storitev in njihovih vmesnikov od drugih storitev, kjer pa

to ni povsem mogoče, pa morajo snovalci imeti identificirane odvisnosti dokumentirane in stalno pod nadzorom.

2.2 Pregled podpornih tehnologij SOA

Za razvoj in uveljavitev SOA so pomembne tri neprofitne organizacije, ki skrbijo za razvoj, uveljavitev in promocijo standardov in priporočil, na katerih temelji SOA.

World Wide Web Consortium (W3C) je leta 1994 ustanovil Tim Berners-Lee, izumitelj svetovnega spleta (World Wide Web Consortium, 2016). Konzorcij je najzaslužnejši za razvoj in popularizacijo svetovnega spleta. Njegova misija je popularizacija svetovnega spleta kot globalnega medija za izmenjavo informacij temelječ na odprtih standardih. Najpomembnejši standardi konzorcija so HTML, ki je postal najpopularnejši tehnični jezik v IT industriji, ter XML Schema in XSLT. Danes ima združenje 416 institucionalnih članov. Med člani so vsa najpomembnejša tehnološka podjetja, telekomunikacijski operaterji in drugi.

Organization for the Advancement of Structured Information Standards (OASIS) je bila leta 1993 ustanovljena pod imenom SGML Open. Današnje ime je dobila pet let kasneje, z željo poudariti preusmeritev iz SGML k XML usmerjenim standardom. Z več kot 600 institucionalnimi člani, je OASIS priznano standardizacijsko telo. Med najvidnejše standarde organizacije sodijo WS-BPEL (znan tudi pod imenom BPEL4WS), ebXML in dopolnitve UDDI specifikacije. V povezavi s spletnimi storitvami se je OASIS osredotočila na varnostne razširitve spletnih storitev. Standarda SAML in XACML sta pomembna za podporo izvedbe enkratne prijave (angl. *Single Sign-On*) in avtorizacije. Najpomembnejši prispevek organizacije OASIS pa je ogrodje WS-Security.

The Web Services Interoperability Organization (WS-I) je bila ustanovljena leta 2002 z namenom zagotoviti odprto interoperabilnost obstoječih standardov. Njihov najbolj znan prispevek je **Osnovni profil** (angl. *Basic Profile*), priporočilo s seznamom standardov za dosego interoperabilne arhitekture. S formalno umestitvijo točno določenih verzij standardov za WSDL, SOAP, UDDI, XML in XML Schema, je Osnovni profil postal pomemben dokument v IT skupnosti. Drug pomemben prispevek organizacije je **Osnovni varnostni profil**, ki je narejen po enakem principu, kot Osnovni profil, le da obravnava najpomembnejši nabor standardov varnosti za spletne storitve in XML. Organizacija je od leta 2010 naprej del organizacije OASIS in je prenehala obstajati kot samostojen subjekt.

SOA temelji na številnih standardih in tehnologijah. Sledi kratek pregled ključnih tehnologij in standardov ter njihove zgodovine.

2.2.1 XML

Prvo verzijo XML je objavil World Wide Web konzorcij (v nadaljevanju W3C) kot priporočilo leta 1998 (World Wide Web Consortium, 1998a). V takratni različici je bil to „sistem za definicijo, preveritev in izmenjavo formatov dokumentov preko svetovnega spleta" (World Wide Web Consortium, 1998b). Glavna prednost XML specifikacije sta bili uporaba takrat že razširjenega svetovnega spleta in njegovih protokolov HTTP in HTTPS ter nezahtevnost orodij za izdelavo XML dokumentov. Potreben je samo navaden urejevalnik besedil. V prvotnem predlogu XML ni zajemal sistema za sheme in ne podpore za podatkovne tipe. XML se je s tem šele začel razvijati. Pomembno vlogo pri razvoju so v začetku imele tudi multinacionalke, kot sta IBM in Microsoft, ki sta intenzivno vlagala in sodelovala kot aktivni članici W3C. S tem so se hotela velika podjetja pridružiti trendom svetovnega spleta in si priboriti del trga, ki so ga v preteklosti zanemarjala. Proces razvoja XML poteka še danes.

XML standardizira komunikacijo tako med različnimi gradniki IS, kot med različnimi IS v okolju. Namenjen je strojni izmenjavi sporočil. Številni modernejši programski paketi imajo implementiran programski vmesnik (API) z uporabo XML sporočil.

2.2.2 SOAP

Paralelno z delom na specifikaciji XML je potekalo delo tudi na specifikaciji SOAP. Vodilno podjetje v tem obdobju je bilo Microsoft. SOAP je nastajal zaradi potrebe po standardnem načinu proženja oddaljene logike in kot nadomestilo takrat obstoječim nestandardiziranim rešitvam temelječ na RPC protokolu (Erl, 2005, str. 73). Čeprav je bilo delo že leta 1998 zrelo za prvo objavo, se Microsoft iz poslovnih in političnih razlogov ter zaradi kratkovidnosti odgovornih oseb v podjetju, za objavo še ni odločil.

Tako je Winer leta 1998 objavil standard XML-RPC (Winer, 1999), ki je bil izsek iz takrat delovne različice specifikacije SOAP (Box, 2001). XML-RPC je predpisoval oddaljeni klic preko svetovnega spleta z uporabo HTTP-POST protokola, kjer je poslano sporočilo zakodirano kot XML dokument. Kot odziv na takšno sporočilo, se izvede logika na strežniku in vrne odgovor, ki je ponovno oblikovano, kot XML dokument. XML-RPC opredeljuje le osnovne gradnike dokumenta z zahtevo in dokumenta z odgovorom: ime metode in parametre metode. Pri tem opredeli tudi podatkovne tipe parametrov, ki so lahko enostavni podatkovni tipi, seznami in sestavljene strukture. Pri odgovoru XML-RPC dokument omejuje na eno samo vrednost, ki je lahko vrnjena, ali strukturo s kodo in opisom napake v primeru neuspešne izvedbe klica. Tako pri klicu, kot pri odgovoru so pomembne tudi glave protokola HTTP. S tem so bili doseženi

cilji: transparentnost preko požarnih pregrad, možnost odkrivanja (angl. *Discoverability*) in preprostost implementacije.

Prvo priporočilo SOAP je bilo objavljeno leta 2000, vendar je bil Microsoft ob objavi deležen številnih kritik. V veliki meri so bile kritike posledica konkurenčne vojne, ki se je odvijala ob prelomu tisočletja. Po pridružitvi IBM k delu na protokolu, je bila leta 2001 objavljena široko sprejeta verzija SOAP 1.1 (World Wide Web Consortium, 2000) pod okriljem W3C, kot priporočilo. Čeprav nova verzija ni imela bistvenih sprememb glede na prejšnjo, je bila sprejeta širše, saj je bila objavljena pod okriljem W3C, k sodelovanju pa so bili povabljena tudi druga podjetja. Ta verzija je ostala veljavna vse do leta 2007, ko je bila objavljena verzija 1.2 (World Wide Web Consortium, 2007a), ki je veljavna še danes.

SOAP specifikacija opredeljuje ogrodje za izmenjavo sporočil, ki je sestavljeno iz:

- procesni model SOAP definira pravila za obdelavo SOAP sporočil;
- razširitveni model SOAP definira koncepte SOAP zmožnosti in modulov;
- ogrodje za povezovanje SOAP s podložnim protokolom opisuje pravila za definiranje in povezovanje s podložnim protokolom, ki je lahko uporabljen za prenos SOAP sporočil med SOAP vozlišči;
- struktura sporočil SOAP.

Nevtralnost SOAP protokola je primerna za uporabo s poljubnim transportnim protokolom, pri čemer je HTTP (in HTTPS) najpogosteje uporabljen transportni protokol. Drugi pogosto uporabljeni transportni protokoli so še: SMTP, JMS in sporočilne vrste. SOAP ob uporabi protokola HTTP ne potrebuje nadgradnje obstoječe infrastrukture svetovnega spleta in ga lahko uporabljamo tako v lokalnih omrežjih, kot v svetovnem spletu.

SOAP ima tudi slabosti, med katere štejemo predvsem kompleksnost, obsežne metapodatke, slaba podpora za pošiljanje binarnih podatkov in fiksno vlogo udeležencev v seji (odjemalec, strežnik). Predvsem kompleksnost, obsežnost metapodatkov in dodatne zahteve po kodiranju binarnih podatkov, ki še dodatno povečajo količino prenesenih podatkov, so v praksi vzpodbudile iskanje alternativ. Vse večjo popularnost sedaj pridobiva pristop REST, opisan v nadaljevanju.

2.2.3 WSDL

SOAP predpisuje protokol in format struktur za klic in vračanje rezultatov storitve preko omrežja ali svetovnega spleta, kar ne zadostuje za odkrivanje storitev. Za zagotovitev ločitve opisa vmesnika od implementacije je zato paralelno s pripravo SOAP tekla tudi priprava WSDL. WSDL omogoča opis pogodbe med odjemalcem in storitvijo, ki jo storitev implementira, odjemalec pa

uporabi na treh ločenih nivojih: abstraktnem, storitvenem in implementacijskem. To omogoča paralelen razvoj odjemalca in storitve. IBM je s svojim pristopom k razvoju WSDL uskladil specifikacijo z obstoječim UDDI protokolom. Sedaj veljavna verzija WSDL specifikacije je 2.0 in je bila pod okriljem W3C sprejeta leta 2007 (World Wide Web Consortium, 2007c).

WSDL na abstraktnem nivoju opisuje storitev skozi sporočila, ki jih storitev sprejema in pošilja. Sporočila so opisana neodvisno od transportnega sistema s sistemom opisa tipov, ki je tipično XML shema. **Operacija** asociira eno ali več sporočil z vzorcem izmenjave sporočil. **Vzorec izmenjave sporočil** opredeljuje zaporednost in kardinalnost sporočil pri sprejemu in oddaji, kot tudi logične pošiljatelje in prejemnike sporočil. **Vmesnik** logično združuje v skupine posamezne operacije neodvisno od transportnega sistema in fizičnega formata. Vmesnik opisuje potencialni način interakcije s spletno storitvijo. Storitev mora implementirati vse opisane interakcije, odjemalec pa se lahko sam odloči, katere bo uporabil in katere ne. Na implementacijskem ali konkretnem nivoju, **povezava** (angl. *Binding*) podrobno opredeljuje format na transportnem in fizičnem nivoju enega ali več vmesnikov. **Končna točka** (angl. *Endpoint*) asociira omrežni naslov s povezavo. V končni fazi **storitev** združuje končne točke, ki predstavljajo implementacijo skupnega vmesnika.

Za doseganje večje prilagodljivosti, je opredelitev povezave na implementacijskem nivoju lahko odložena na čas objave storitve iz stališča storitve oziroma na čas klica storitve iz stališča odjemalca. S tem dosežemo pozno povezovanje (angl. *Late Binding*), kar poveča neodvisnost odjemalcev in storitve med seboj oziroma šibko sklopljenost odjemalcev in storitve. Cena poznega povezovanja pa je dodaten čas, potreben za vzpostavljanje povezave pri vsakem klicu storitve in znižana preglednost izvedbe.

WSDL je največji povod za kritike spletnih storitev. WSDL je najškodljivejša tehnologija v okviru spletnih storitev (Webber, Parastatidis & Robinson, 2010, str. 385). Deloma je razlog v kompleksnosti in okornosti, ki jo prinaša WSDL, bolj sporna pa je podpora neustreznemu modelu za spletne sisteme, obstoječa orodja pa zavajajo arhitekte in izvajalce v neustrezno uporabo in tesno sklopljenost pogodbe in implementacije storitev. WSDL zagotavlja neodvisnost od transportnega nivoja, kar v specifikacijo vnaša večjo kompleksnost, ki je nepotrebna, saj se WSDL v več kot 95% primerov uporablja preko HTTP (in HTTPS) protokola.

2.2.4 UDDI

Opis spletnih storitev, kot ga ponuja WSDL, pa ni uporaben, če ga odjemalec ne more najti. Po vzoru posrednikov v CORBA arhitekturi in kasneje v EAI, je leta 2000 nastala specifikacija UDDI, ki je leta 2002 postala standard pod okriljem OASIS. UDDI je protokol v obliki nabora

spletnih storitev in metapodatkov. Ideja UDDI je bila tudi v vzpostavitvi globalnega posrednika, ki skrbi za imenik spletnih storitev in omogoča vpis storitev v imenik, opis z metapodatki ter iskanje oziroma odkrivanje ustreznih storitev za prijavljene odjemalce. Storitve so lahko na voljo vsem odjemalcem ali pa le omejenemu krogu odjemalcev.

UDDI ni bil široko sprejet v industriji. Glavni komercialni ponudniki UDDI posrednikov: IBM, Microsoft in SAP so tako januarja 2006 ustavili svoje javno dostopne registre in zaprtje predstavili, kot uspeh (Hately, 2015). Že konec leta 2007 pa je OASIS razpustil tehnični odbor (McRae, 2008) in prenehal z razvojem UDDI specifikacije. Leta 2010 je sledilo še obvestilo Microsofta, da je umaknil podporo UDDI iz svojega operacijskega sistema (Microsoft, 2010).

Nekateri proizvajalci še naprej ponujajo izdelke s podporo UDDI standardu (IBM in Oracle), predlagane pa so bile tudi razširitve standarda za podporo različicam (Jurič, Saša, Brumen & Rozman, 2009). Praviloma se UDDI uporablja znotraj enega podjetja za interno uporabo, obstaja pa tudi možnost povezave med podjetji s pregledovanjem več UDDI registrov (Al-Masri & Mahmoud, 2007, str. 1255). Zaradi prevelike kompleksnosti UDDI za interni namen, so ga v veliko okoljih nadomestile enostavnejše, nestandardne rešitve.

2.2.5 WADL

Podobno, kot SOAP, tudi REST predpisuje protokol in način klica REST storitev, sestavljanja URI, pošiljanja parametrov in vračanja rezultatov. Za odkrivanje REST storitve je lahko uporabljen tudi UDDI protokol, v praksi pa so uporabljene preprostejše imeniške storitve, kot sta DNS in LDAP. Za opredelitev pogodbe pa je pri REST storitvah ena od možnosti uporaba specifikacije WADL.

Strokovna javnost je razdeljena glede (ne)uporabe opisa REST storitev. Predvsem tehnično usmerjeni zagovorniki trdijo, da opis REST storitev ni potreben, kar je skladno z „lahkostjo" REST storitev v primerjavi s SOAP storitvami. Zagovarjajo stališče, da je pogodba storitve razvidna iz storitve in ne potrebuje dokumentacije. To v veliki večini primerov drži, dokler so REST storitve uporabljene v ozkem krogu odjemalcev in implementirajo relativno preproste funkcije.

SOA pa je namenjena kompleksnim poslovnim okoljem in temelji na orkestraciji, zato je strojno berljiv opis pogodbe potreben in zahtevan.

WADL omogoča strojno obdelavo opisa spletne aplikacije temelječe na protokolu HTTP. Pri tem zajema vse možne izpeljanke REST storitev: z notacijo v XML obliki ali z notacijo v JSON

obliki.

Razvoj WADL specifikacije je v letih 2005–2009 izvajalo podjetje Sun Microsystems (danes Oracle Corporation). Avgusta 2009 je bil predlog standarda posredovan W3C konzorciju (World Wide Web Consortium, 2009). Specifikacija ima še danes status predloga.

Vsebinsko ima WADL podobne pomanjkljivosti, kot sorodna WSDL specifikacija in vodi v RPC način izvedba storitev, kar je odmik od storitvene usmerjenosti. Razlika v primerjavi z WSDL je predvsem v tem, da je WADL znatno manj v uporabi in zanj obstaja le peščica orodij, ki zavajajo izvajalce v napačno smer.

2.2.6 Swagger

Swagger je popularnejša alternativa specifikaciji WADL. Swagger je nabor pravil (specifikacija) za opis REST programskih vmesnikov (SmartBear, 2014). Swagger temelji na JSON notaciji, omogoča pa opis REST storitev z notacijo v XML obliki ali z notacijo v JSON obliki. Ne pokriva vseh možnosti REST storitev, je pa zato preprostejši od WADL specifikacije.

2.2.7 Spletne storitve

Spletne storitve so temeljni kamen SOA arhitekture. Tako kot osnova in glavni razlog za popularnost storitvene usmerjenosti in SOA arhitekture, pa so tudi tarča mnogih kritik. Kritike so posledica napačnega razumevanja in uporabe spletnih storitev ter SOA arhitekture v povezavi s storitvami. Naslednji dve definiciji kažeta razliko v razumevanju: „Spletna storitev je programska oprema namenjena podpori za interoperabilno interakcijo med stroji preko omrežja. Ima vmesnik opisan v obliki primerni za strojno obdelavo (WSDL). Interakcija drugih sistemov s spletno storitvijo poteka v obliki predpisani v opisu vmesnika z uporabo SOAP sporočil, tipično s prenosom preko HTTP protokola in z uporabo XML serializacije v povezavi z drugimi spletnimi standardi.” World Wide Web Consortium (2004)

„V splošnem je **storitev** računalniški program, ki svojo funkcionalnost izpostavlja skozi objavljen tehnični vmesnik, imenovan **storitvena pogodba**.” (Erl, Carlyle, Pautasso & Balasubramanian, 2013)

Razlika med navedenima definicijama je predvsem v času objave in v dejstvu, da je prva definicija nastala pod okriljem organizacije W3C, ki promovira spletne storitve temelječe na SOAP in WSDL protokolih. Zaustavitev razvoja UDDI, kompleksnost in slabo razumevanje ter posledično napačna uporaba spletnih storitev temelječih na SOAP, pa so povzročili vse večjo popularnost REST storitev. Druga definicija je prav iz knjige **SOA with REST**. Danes se v praksi pojavljajo tri vrste storitev:

- SOAP spletne storitve,
- REST storitve z XML obliko sporočil in
- REST storitve z JSON obliko sporočil.

2.2.7.1 SOAP spletne storitve

SOAP spletne storitve temeljijo na standardih XML, SOAP, WSDL in UDDI. Razvoj in promocija tečeta v okviru W3C konzorcija. Vsak program ali skripta, katere vmesnik je opisan z WSDL in ki zna sprejeti, obdelati in odgovoriti s sporočili v SOAP obliki pa še ni spletna storitev. Da je storitev lahko spletna storitev in del SOA arhitekture, mora zadoščati naslednjim kriterijem:

- avtonomnost - storitev opravlja svojo funkcijo neodvisno od drugih storitev,
- šibka sklopljenost - storitve in odjemalci so čimbolj neodvisni drug od drugega in med seboj,
- standardizirana storitvena pogodba - storitev ima opisan vmesnik v standardni obliki in na standarden način,
- ponovna uporabnost - storitev implementira generično in ponovno uporabno logiko in pogodbo,
- abstrakcija - minimizacija potrebe po opisu storitve z metapodatki,
- neodvisnost od stanja - storitev naj implementira prilagodljivo logiko brez upravljanja stanja in
- zmožnost odkrivanja - za storitev naj bodo implementirane meta informacije za komunikacijo.

Na tem mestu naj opozorim na pogosto napačno razumevanje šibke sklopljenosti: šibka sklopljenost v izvajalnem smislu omogoča ponudniku in odjemalcu storitve, da delujeta neodvisno drug od drugega. Takšno razumevanje šibke sklopljenosti ima poleg izrazite prednosti povečevanja zanesljivosti IS tudi izrazito slabost v podaljšanem odzivnem času v primerjavi s tesno sklopljenima ponudnikom in odjemalcem storitve in zahteva dodatne mehanizme preverjanja uspešnosti izvedbe storitve. Šibka sklopljenost je praviloma izvedena z asinhronimi vrstami, kjer odjemalec izvede pošiljanje sporočila po principu sproži in pozabi (angl. *Fire and Forget*) ter nadaljuje z delom. Odjemalec nima na voljo neposrednega mehanizma, s katerim bi lahko preveril kdaj in kako uspešno se je izvedla storitev. Zahteva gre v vrsto kjer čaka na izvedbo. V odvisnosti od narave storitve, zasnove sistema in zasedenosti virov, lahko zahteva čaka na izvedbo nekaj sekund, minut, ur, dni ali mesecev. Šibka sklopljenost v storitveno usmerjenem smislu pa se nanaša na skrivanje izvedbe storitve pred odjemalcem, ki pozna le pogodbo storitve, v času izvajanja pa je odvisen od SOA arhitekture (predvsem imenika storitev, repozitorija storitev in storitvenega vodila). Storitveno vodilo poskrbi za dejansko izvedbo klica storitve, posredovanja parametrov, morebitno pretvorbo sporočila in vračanje rezultata. Implementacija se na ta način lahko poljubno spreminja pod pogojem, da zadošča pogodbi storitve.

SOAP spletne storitve so neodvisne od transportnega sistema, praviloma se uporablja protokol HTTP, v redkejših primerih še protokola SMTP in FTP. Prav tako izmenjuje podatke, razen binarnih podatkov, v XML obliki znotraj SOAP ovojnice.

Kljub kritikam SOAP storitev in naraščajoči priljubljenosti REST storitev, so SOAP spletne storitve še vedno najbolj razširjene storitve v podjetjih. Za izdelavo in uporabo SOAP spletnih storitev je izdelanih veliko ogrodij in orodij. Najbolj razširjena so: Microsoft Visual Studio, Apache Axis, Apache Axis2, Apache CXF in JAX-WS.

2.2.7.2 REST storitve

Ob prelomu tisočletja prvič zasledimo združitev arhitekturnih konceptov načrtovanja omrežij in programskega inženiringa (Fielding, 2000). Rodil se je REST koncept.

REST storitve temeljijo na HTTP protokolu. Operacija je določena s HTTP ukazno glavo, objekt, nad katerim se bo operacija izvedla, pa je določen z URI, kot del URL-ja. Dovoljene so naslednje operacije:

- GET,
- POST,
- PUT in
- DELETE.

Parametri so lahko poslani bodisi v XML obliki ali JSON obliki. JSON oblika je krajša, XML oblika pa je skladna s standardi. Ker je popularnost REST storitev posledica enostavnosti in „lahkosti" v primerjavi s SOAP spletnimi storitvami, je JSON oblika vse popularnejša.

Pomembna zahteva REST pristopa je hranjenje seje pri odjemalcu. REST storitev ne sme pomniti stanja. Izvesti mora zahtevano operacijo in potem „pozabiti", kaj se je zgodilo. Tak način lahko povzroči povečanje količine prometa po omrežju, zato so pomembni gradniki v REST arhitekturi predpomnilniki z aktivno detekcijo sprememb podatkov.

REST storitve v SOA arhitekturi so lahko organizirane v večnivojskih, netransparentnih arhitekturnih slojih. To pomeni, da vsak sloj „vidi" samo sosednji podrejeni sloj (Erl, Carlyle, Pautasso & Balasubramanian, 2013, pol. 1532). Interakcija med storitvami, ki niso v istem ali sosednjem sloju, ni dovoljena.

2.2.8 WS-* in REST razširitve

Ob sodelovanju številnih distribuiranih gradnikov arhitekture se pojavi več izzivov:

- varnost,
- zanesljivost v smislu kakovosti storitev (v nadaljevanju QoS),
- upravljanje identitet,
- opis obnašanja storitev,
- nepreglednost,
- skalabilnost in
- transakcije med storitvami.

Osnovne tehnologije za podporo SOA arhitekture zgornjih izzivov ne obravnavajo. Za rešitev naštetih izzivov so nastale razširitve specifikacij in standardov. Uporaba razširitev v SOA arhitekturi je opcijska. Obseg uporabe razširitev kaže na zrelost stanja SOA arhitekture v danem okolju. Nekatere od naštetih razširitev imajo imena v drugačni obliki, zaradi prevladujoče oblike pa se je tako v literaturi, kot v praksi uveljavila okrajšava **WS-*** za razširitve spletnih storitev.

Podrobnejši opis razširitev je v Prilogi 2.

2.2.9 Alternative

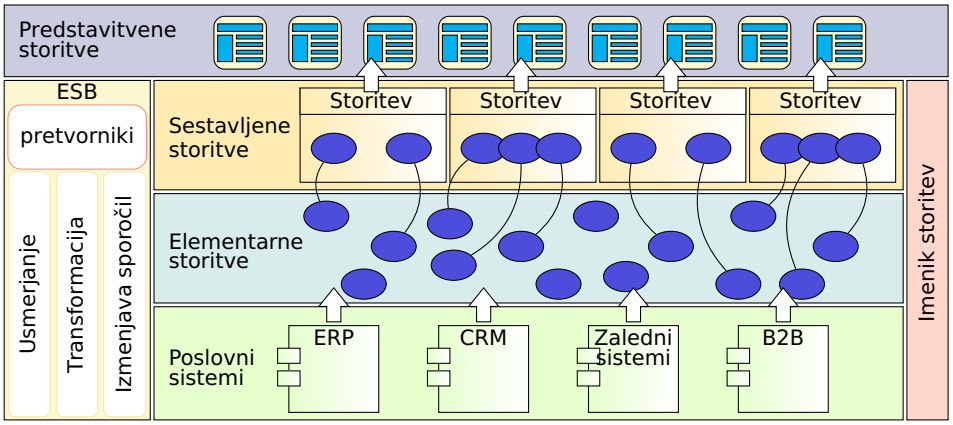
SOAP, WSDL in REST niso edine specifikacije spletnih storitev, ki obstajajo. Alternativa WSDL opisu spletnih storitev je OWL-S standard (DAML Services - Coalition, 2008). Razširjenost uporabe OWL-S in drugih alternativ v primerjavi s SOAP in REST storitvami pa je neprimerljivo nižja in jih zato v tem delu ne obravnavam.

2.3 Zgradba SOA arhitekture

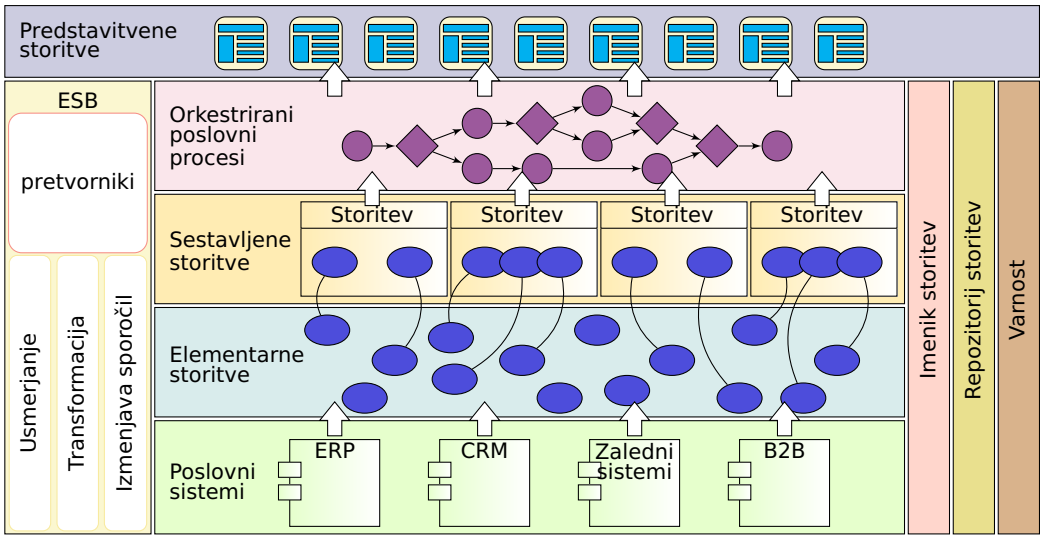
Osnovne logične gradnike SOA arhitekture sestavljajo storitveno vodilo, poslovni sistemi, spletne storitve, imenik storitev in uporabniški vmesnik. S tem naborom je implementirana **primitivna SOA** (Erl, 2005, str. 38) in je prikazana na sliki 4. Za implementacijo sodobne SOA arhitekture, potrebujemo še repozitorij storitev, sistem za upravljanje poslovnih procesov in sistem varnosti. Slika 5 prikazuje sodobno SOA arhitekturo.

Z razvojem in novimi spoznanji pa se SOA arhitektura in njeni gradniki dopolnjujejo. Seznam gradnikov, ki implementirajo različne sloje abstrakcij, ni dokončen. Trenutno so najpogostejši naslednji gradniki SOA arhitekture:

Slika 4: Primitivna SOA arhitektura



Slika 5: Sodobna SOA arhitektura



- storitveno vodilo,
- imenik storitev,
- repozitorij storitev in
- sistem za upravljanje poslovnih procesov.

Podrobnejši opis gradnikov je v Prilogi 3.

2.4 Štirje stebri SOA arhitekture

SOA temelji na preverjenih praksah, vzorcih, principih in tehnologijah. Ti podpirajo izvedbo in uporabo storitvene usmerjenosti. Zaradi izrazito strateške narave ciljnega stanja, katerega želimo doseči z uporabo storitvene usmerjenosti, obstajajo kritični faktorji uspeha, ki predstavljajo predpogoj za uspešno uvedbo storitvene usmerjenosti. Te pogoje imenujemo **stebri**, saj skupaj zagotavljajo zdravo osnovo za izgradnjo, postavitev in upravljanje storitev (Erl, 2011).

Ekipno delo – storitveno usmerjen pristop zahteva obsežnejše sodelovanje med projektnimi skupinami. Različne skupine si morajo medsebojno zaupati in se zanesti druga na drugo.

Izobraževanje – člani različnih skupin morajo med seboj učinkovito sodelovati. To lahko dosežemo skozi izobraževanje. Najpomembnejši del izobraževanja je poenotenje načina komuniciranja med vsemi člani skupin. Tako bodo imeli vsi člani enako predstavo o cilju, metodologijah in načinu izvedbe, s čimer je tveganje za neuspeh znatno zmanjšano.

Disciplina – za uspešno izvedbo morajo člani skupin disciplinirano uporabljati svoja znanja in izpolnjevati svoje naloge v okviru svojih vlog. Obvladovanje SOA arhitekture zagotavlja okvir za disciplino in njeno spremljanje.

Uravnotežen obseg – za realno izvedbo je nujna postavitev realno dosegljivih ciljev skozi realen obseg vpeljave, ki je uravnotežen in smiseln z vključitvijo relevantnih delov sistema in poslovnih funkcij.

2.5 Uvedba SOA

Postopna uvedba storitvene usmerjenosti in s tem sprememb je nujen pogoj za uspeh. Obsežne spremembe na številnih področjih narekujejo sistematičen pristop. Podobno, kot za druga področja IT, je tudi za SOA arhitekturo razvitih več modelov zrelosti: SOAMM, SIMM in CSOAMM (Söderström & Meier, 2007, str. 391-397). Modeli zrelosti opredeljujejo stopnje in pot do končnega cilja. Vsako podjetje je ob odločitvi za prehod na storitveno usmerjenost v določeni fazi zrelosti. Od trenutne faze pa so odvisni potrebni koraki za doseganje naslednje stopnje. Posameznih stopenj ni mogoče preskočiti.

Zrelostni modeli so lahko dobra pomoč pri postopni uvedbi SOA. Pomagajo identificirati trenutno in naslednjo stopnjo. S tem pa je opredeljeno tudi stanje, ki ga mora podjetje doseči ob koncu naslednje stopnje.

2.5.1 SOA zrelostni modeli

SOAMM so leta 2005 skupaj predlagala in objavila podjetja Sonic Software Corporation, AmberPoint Inc., BearingPoint Inc. in Systinet Corporation (Sonic Software Corporation, AmberPoint Inc., BearingPoint Inc. & Systinet Corporation, 2005). SOAMM opredeljuje pet nivojev zrelosti na podlagi CMMI modela:

1. začetne storitve (angl. *Initial Services*) ustrezajo CMMI zmogljivostnemu nivoju (angl. *Performed*)

2. storitve v arhitekturi (angl. *Architected Services*) ustrezajo CMMI upravljanemu nivoju (angl. *Managed*)
3. poslovne in skupne storitve (angl. *Business Services and Collaborative Services*) ustrezajo CMMI opredeljenemu nivoju (angl. *Defined*)
4. merljive poslovne storitve (angl. *Measured Business Services*) ustrezajo CMMI merljivo upravljanemu nivoju (angl. *Quantitatively Managed*)
5. optimizirane poslovne storitve (angl. *Optimized Business Services*) ustrezajo CMMI optimizirajočemu nivoju (angl. *Optimizing*)

Za kriterije, po katerih uvrsti podjetje na ustrezno stopnjo, so v SOAMM modelu izbrani:

- primarne poslovne koristi,
- področja,
- kritični tehnološki faktorji uspeha,
- kritični faktorji uspeha zaposlenih in organizacije in
- izbrani pomembni standardi.

SOAMM opredeli tudi tipične vrednosti kriterijev za vsak nivo. Preglednica tipičnih vrednosti in ustreznih stopenj je v Prilogi 4.

SIMM je leta 2005 objavil IBM (Arsanjani & Holley, 2006, str. 515). Model je namenjen opredelitvi zrelosti integracije storitev in ne SOA zrelosti. Opredeljuje sedem nivojev zrelosti podjetja:

1. silos
2. integriran
3. komponenten
4. storitve
5. sestavljene storitve
6. navidezne storitve
7. dinamično (re)konfigurabilne storitve

Za kriterije, po katerih uvrsti podjetje na ustrezno stopnjo, so v SIMM modelu izbrani naslednji kriteriji:

- poslovanje,
- organizacija,
- metode,
- aplikacije,

- arhitektura,
- informacije in
- infrastruktura.

Tipične vrednosti kriterijev in ustrezni nivoji so navedeni v preglednici v Prilogi 4.

CSOAMM naslavlja nezdružljivost SOAMM in SIMM modelov (Söderström & Meier, 2007, str. 394). CSOAMM ni nov model zrelosti, temveč kombinacija obeh prej opisanih modelov. V ta namen opredeli 10 nivojev zrelosti:

-2. silos
-1. integriran
 0. komponente
 1. tehnološki testi
 2. objavljene storitve
 3. institucionalizacija
 4. arhitekturne storitve
 5. notranje in zunanje storitve
 6. merjene storitve
 7. dinamična arhitektura

SIMM opisuje zrelost storitev, kar je širše področje, kot SOA. Zato sta prvi dve stopnji povzeti po SIMM modelu in nimata ustreznih stopenj v SOAMM modelu. CSOAMM prične s povezovanjem obeh prejšnjih modelov pri stopnji 0. Primerjalna tabela je v Prilogi 4. Na koncu CSOAMM ponudi še primerjavo s splošnim modelom zrelosti CMMI. Tudi ta primerjalna tabela je v Prilogi 4.

2.6 Prihodnost

Vsem skeptikom, ki se še sprašujejo ali je SOA koristna, odgovarja članek v **Harvard Business Review**, kjer avtorji predstavljajo, kako so podjetja, ki so sprejela storitveno usmerjenost in uvedla SOA arhitekturo, „odpravila ogromne količine nepotrebne programske opreme, dosegla velike prihranke s poenostavitvijo in avtomatizacijo prej ročnih procesov in močno dvignila produktivnost”. (Merrifield, Calhoun & Stevens, 2008, str. 74)

Podobno, kot je storitvena usmerjenost evolucija, ki izhaja iz objektno-usmerjenega programiranja, komponentnega modela in distribuiranih komponent, pričakujem, da bo storitvena usmerjenost v prihodnosti pridobila dodatno stopnjo abstrakcije, morda v smislu **vidične usmerjenosti** (angl. *Aspect-Oriented*), ki izhaja iz vidično-usmerjenega programiranja ter vidično

usmerjene analize in snovanja (angl. *Aspect-Oriented Analysis and Design*) (Clarke & Baniassad, 2005).

3 GRANULARNOST STORITEV

Granularnost storitev predstavlja obseg funkcionalnosti, ki jo ponuja storitev (Papazoglou & Heuvel, 2006, str. 416), oziroma velikost storitev (Haesen, Snoeck, Lemahieu & Poelman, 2008, str. 375). Groba granularnost storitev je pomemben princip storitvene usmerjenosti in pravilo k pristopu snovanja v SOA arhitekturi. Je logična posledica snovanja storitev na višjem nivoju abstrakcije. Naloga poslovnih uporabnikov je izvajanje poslovnih funkcij in praviloma nimajo časa, znanja in interesa za spoznavanje drobno granuliranih, tehnološko usmerjenih konceptov za avtomatizacijo svojega dela. V idealnih pogojih poslovni uporabniki lahko uporabijo avtomatizirane dele funkcionalnosti (storitve), ki ustrezajo njihovim enotam dela. Enote dela zajemajo širši obseg funkcionalnosti, kot je zajet v obdelavi programske kode. Storitve, ki podpirajo (del) poslovnega procesa zajemajo široko funkcionalnost, zato jih označujemo za grobo granulirane.

Za implementacijo enake funkcionalnosti potrebujemo torej več drobno granuliranih storitev ali manj grobo granuliranih storitev. Posledično je večji del poslovne logike pri uporabi drobno granuliranih storitev preseljen na višji nivo abstrakcije k storitvam, ki uporabljajo drobno granulirane storitve (jih orkestrirajo). V primeru grobo granuliranih storitev pa je poslovna logika vključena v same storitve in je le manjši delež poslovne logike potreben za njihovo orkestracijo. Izbira granularnosti ima vpliv tudi na ponovno uporabo. Drobno granulirane storitve so elementarne in bolj splošne. Uporabljene so v večjem številu višjenivojskih storitev ali procesov. Po drugi strani so grobo granulirane storitve namenjene specifični funkciji in je njihova ponovna uporaba redkejša. S tem pridem do pomembne ugotovitve: **Granularnost storitev je v obratnem sorazmerju s ponovno uporabo storitev**. Iz te ugotovitve sledi, da ni splošnega pravila za idealno granularnost. Optimalna granularnost je ravnovesje, ki zagotavlja optimalnost z vidika ponovne uporabe in čim nižjega števila slojev abstrakcije.

Obravnavanje granularnosti ločim na dva različna pogleda:

- obravnavanje granularnosti storitev obstoječega sistema in
- obravnavanje granularnosti storitev novega ali prenovljenega sistema.

Ob obravnavanju granularnosti storitev obstoječega sistema je mogoče izdelati podroben pregled in prešteti storitve, povezave, pogostost uporabe, odzivne čase, vrstice kode in druge lastnosti, ki podrobno opredelijo granularnost in nefunkcionalne lastnosti storitev. V nasprotju s tem pa

pri snovanju novega sistema ali snovanju prenove sistema, granularnosti storitev vnaprej ne poznamo. Lahko jih ocenimo na podlagi preteklih izkušenj in opredelimo splošna pravila ter ciljno stanje.

Analiza obstoječega sistema je pomemben sklop zagotavljanja optimalnosti in skladnosti sistema s poslovnimi potrebami podjetja. Služi lahko kot vir informacij, na podlagi katerih se kasneje odločimo za spremembo ali zamenjavo IS. Tudi pri obstoječem sistemu ni enotnega merila granularnosti komponent in storitev.

„Analiza granularnosti ni eksaktna znanost. Je proces ocenjevanja opredeljen na podlagi količine funkcionalnosti, ki jo storitev ponuja. Nanjo vpliva tudi pomembnost storitve iz poslovnega in tehnološkega vidika. Različni rezultati analize granularnosti ne bodo natančni." (Bell, 2008, str. 136)

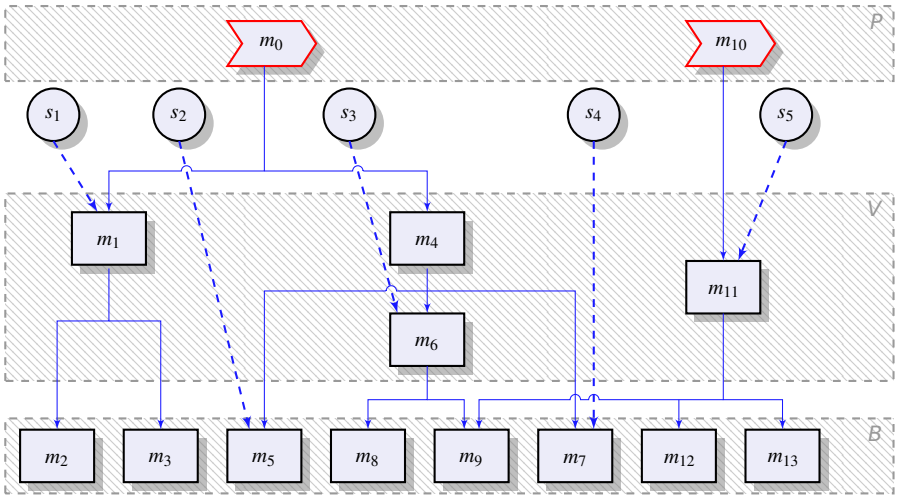
Obstoječe raziskave granularnosti storitev se skoraj izključno ukvarjajo z vzpostavitvijo IS (Steghuis, 2006; Reldin & Sundling, 2007; Haesen, Snoeck, Lemahieu & Poelman, 2008). Izjema je prispevek (Krammer, Heinrich, Henneberger & Lautenbacher, 2011, str. 345), ki poleg vzpostavitve, ponuja tudi obravnavo vzdrževanja IS iz stališča bodočih sprememb v življenjski dobi, ne pa tudi iz stališča obratovanja IS.

Kompozicija in izdelava storitev lahko poteka na tri različne načine: od zgoraj navzdol (angl. *Top-Down*), od spodaj navzgor (angl. *Bottom-Up*) ali od znotraj navzven (angl. *Middle-Out*). Kompozicija od zgoraj navzdol je uporabna samo enkrat ob postavljanju novega sistema v primeru, da nimamo namena integrirati obstoječih sistemov. Takrat poslovne storitve uskladimo s poslovnimi procesi in funkcijami. Poslovni procesi in funkcije tako že vnaprej določajo zahteve in s tem granularnost storitev. Ta pristop ne potrebuje analize granularnosti storitev, ki je že vnaprej dana. Pristop od spodaj navzgor uporabimo, kadar prevladuje integracija obstoječih sistemov in morebitnih že obstoječih komponent ter drobno granuliranih storitev, pogosto pa vodi v drobno zrnatost storitev in rešitev, ki ni optimalna. Večina v nadaljevanju opisanih modelov obravnava tak pristop. Kompozicija od znotraj navzven je največkrat uporabljen pristop, predvsem pri dodajanju novih funkcionalnosti in spreminjanju obstoječega IS temelječega na SOA (Rosen, Lublinsky, Smith & Balcer, 2008, str109). V tem primeru uporabimo že obstoječe komponente in drobno granulirane storitve ter obstoječim poslovnim procesom in funkcijam dodajamo nove. Izhajajoč iz obstoječega stanja in novih zahtev oblikujemo in spreminjamo storitve tako, da zapolnimo vrzel.

3.1 Krammerjev model

Krammer, Heinrich, Henneberger & Lautenbacher (2011, str. 349–353) predlagajo matematični model na podlagi funkcionalnosti v obliki usmerjenega grafa brez ciklov. Svojega modela sicer ne poimenujejo, za lažje sklicevanje pa ga v tem delu imenujem „Krammerjev model". V podporo modelu najprej opredelijo FSG graf, ki je prikazan na sliki 6.

Slika 6: FSG – Graf funkcionalnosti in storitev



Vir: A. Krammer, B. Heinrich, M. Henneberger & F. Lautenbacher, Granularity of Services, 2011, str. 350.

Na podlagi FSG grafa nato opredelijo 3 metrike za granularnost:

- metrika razdalje,
- metrika obsega in
- metrika velikosti.

Metrika razdalje temelji na dolžini poti od procesa do implementirane funkcionalnosti v razmerju z dolžino celotne poti:

$$z_w = \frac{d(m_p, m_j) - 1}{\max[1, d(m_p, m_n) - 1]} \tag{2}$$

Kjer je z_w metrika razdalje, m_j predstavlja implementirano funkcionalnost j in $d(mx, my)$ je dolžina poti od točke m_x do točke m_y v FSG grafu.

Obseg vrednosti z_w je normaliziran in leži v intervalu $[0, 1]$. Implementacija osnovne funkcionalnosti, ki predstavlja najbolj drobno granularnost, predstavlja vrednost 1, vrednost blizu 0 pa predstavlja zelo grobo granularnost.

Za normalizirano metriko moramo izračunati metriko vseh poti, ki vključujejo funkcionalnost m_j, upoštevamo matematično sredino in vrednost normaliziramo v interval $[0,1]$:

$$g_T(s_i) = \frac{\sum_{w \in W} z_w}{|W|} \tag{3}$$

Pri tem je $g_T(s_i)$ normalizirana metrika razdalje in W je množica vseh poti w, ki vsebujejo m_j.

Metrika obsega izhaja iz metrike razdalje, pri tem pa upošteva vse implementirane funkcionalnosti. V ta namen uvaja parameter n_{m_j}, ki predstavlja število implementiranih funkcionalnosti ne glede na razdaljo:

$$z_w = 1 - \frac{n_{m_j} - 1}{max(1, n_{m_a} - 1)} \tag{4}$$

n_{m_a} predstavlja funkcionalnost, ki neposredno sledi procesu.

Enako, kot pri metriki razdalje, je tudi pri metriki obsega nabor vrednosti normaliziran v interval $[0,1]$. Če je osnovna funkcionalnost implementirana kot storitev, potem velja, da je $n_{m_j} - 1 = 0$. V primeru, ko implementiramo funkcionalnost, ki neposredno sledi procesu, potem sta vrednosti n_{m_j} in n_{m_a} enaki.

Metrika velikosti temelji na obsegu funkcionalnosti, ki jo zajema storitev. V informacijskem inženiringu sta najbolj znani metodi merjenja obsega LOC (angl. *Lines of Code*) in funkcijske točke. Krammerjev model uporablja prilagojeno metodo LOC. Pogoj za uporabo te metode je možnost ocene ali meritve $size_{m_j}^{func}$ osnovne funkcionalnosti m_j, ki je:

$$size_{m_j} = \begin{cases} size_{m_j}^{func} & m_j \in B \\ size_{m_j}^{comp} + \sum_{i=1}^{|M|} I_{m_j,m_i} * size_{m_i} & m_j \in V \end{cases} \tag{5}$$

Pri tem $w(m_p, m_n)$ predstavlja celotno pot od m_p do m_n.

Metrika velikosti je torej:

$$z_w = 1 - \left[\frac{size_{m_j}}{size_{m_a}} \right] \tag{6}$$

V primeru, ko je $m_j = m_a$ je vrednost metrike enaka 0, kar predstavlja najbolj grobo granularnost.

Izbira metrike je odvisna od razpoložljivih informacij, narave sistema in namena uporabe. Čeprav imajo metrike vselej rezultat iz obsega $[0,1]$, kombinacija metrik v istem modelu ni smiselna, saj vmesne vrednosti ne predstavljajo enake granularnosti.

3.2 Xiao-Junov model

Xiao-Jun (2009) izhaja iz predpostavke, da morajo biti funkcije, ki se uporabljajo skupaj, tudi grupirane skupaj v isto storitev na tak način, da dobimo čimbolj grobo granularnost storitev. Na podlagi tega izhodišča opredeli metriko skladnosti dejanske granularnosti v primerjavi z idealno granularnostjo storitve.

Najprej opredeli informacijsko entropijo $H(S_i)$ kompleksnega elementa S_i:

$$H(S_i) = p \sum_{j=1}^{n_S} (-log(P_L(j))) = \frac{1}{n} \sum_{j=1}^{n_S} (-log(P_L(j))) \tag{7}$$

V enačbi (7) predstavlja n_S število atomarnih elementov kompleksnega elementa S_i, n predstavlja skupno število atomarnih elementov v celotnem grafu, p predstavlja verjetnost uporabe atomarnega elementa, pri čemer je $p = 1/n$, R_j predstavlja skupno število povezav atomarnega elementa j in $P_L(j)$ predstavlja verjetnost sprožitve dogodka na povezavah atomarnega elementa j, pri čemer je $P_L(j) = 1/(R_j + 1)$.

Naj bo A množica vseh operacij v storitveno usmerjenem sistemu in naj bo $P(A)$ nadmnožica, ki predstavlja vse možne pojavitve operacij. Z $o \in P(A)$ označimo množico operacij, ki jih je mogoče uporabljati skupaj, $\rho(o)$ verjetnost, da bo množica operacij o uporabljena skupaj. Naj bo O diskretna naključna spremenljivka z vrednostjo iz $P(A)$, $\rho(b)$ pa naj bo verjetnost izbire storitve iz množice storitev B. Definirajmo še diskretno naključno spremenljivko P, ki lahko zavzame poljubno vrednost iz množice storitev B. Skupno informacijsko vsebino med množico operacij uporabljenih skupaj nasproti storitvam lahko sedaj izrazimo kot:

$$I(\mathbf{P}, \mathbf{O}) = H(\mathbf{P}) - H(\mathbf{P}|\mathbf{O}) \tag{8}$$

Pri čemer $H(\mathbf{P})$ predstavlja entropijo povezano z vsemi storitvami sistema in $H(\mathbf{P}|\mathbf{O})$ predstavlja pogojno entropijo storitev nasproti različnim množicam operacij, ki so lahko uporabljene skupaj.

Entropijo vseh storitev sedaj lahko izrazimo na naslednji način:

$$H(\mathbf{P}) = -\sum_{b \in B} \rho(b)\ log(\rho(b)) \tag{9}$$

Pogojna entropija pa je:

$$H(\mathbf{P}|\mathbf{O}) = -\sum_{o \in P(A)} \rho(o) \sum_{b \in B} \rho(b|o)\ log\rho(b|o) \tag{10}$$

Doseči želimo čimbolj grobo granularnost. To pomeni, da mora biti v eni storitvi zajeta čim večja množica možnih operacij, ki bodo uporabljene skupaj. V ta namen Xiao-Jun opredeli metriko:

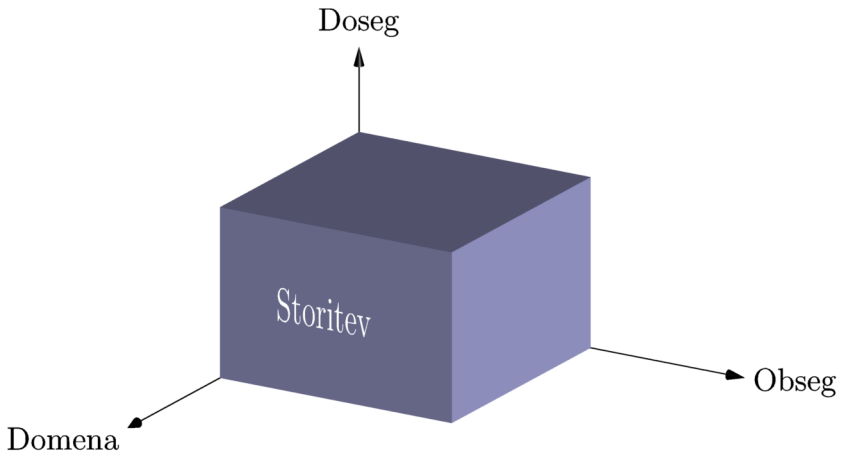
$$CUSO = \frac{I(\mathbf{P},\mathbf{O})}{H(\mathbf{P})} \tag{11}$$

Faktor $CUSO$ meri stopnjo verjetnosti, da bodo skupaj uporabljene operacije združene v storitev.

3.3 R^3 model

Reldin & Sundling (2007) opredelita granularnost na podlagi obsega funkcionalnosti, dosega in domene. Če vse metrike postavimo v tri-dimenzionalni koordinatni sistem, granularnost predstavlja volumen kvadra, ki je omejena z izhodiščem in vrednostjo vsake od metrik, kot je prikazano na sliki 7.

Slika 7: R^3 model granularnosti



Vir: P. Reldin & P. Sundling, Explaining SOA Service Granularity, 2007, str. 57.

Doseg predstavlja osebe (ali sisteme), katerim je storitev dosegljiva, **obseg** predstavlja količino funkcionalnosti, ki jo ponuja storitev in **domena** predstavlja vrste funkcionalnosti, ki jih ponuja storitev.

Dimenzije v R^3 modelu so med seboj enakovredne. Posledično je granularnost storitev z enakim volumnom v grafu, med seboj enakovredna. Ob neomejenih virih bi želeli imeti čim večji doseg, čim več funkcionalnosti in kar največ domen – čimbolj grobo granulirane storitve. Ker pa so viri omejeni, je potrebno poiskati najprimernejši kompromis med možnostmi in omejitvami. Razširitve storitve v eni od dimenzij lahko povzročijo neobvladljivo povečanje porabe virov, zato je včasih potrebno zmanjšati obseg v drugih dveh dimenzijah. Ob vsaki spremembi je nujno potrebno proučiti medsebojni vpliv vseh treh dimenzij.

3.4 Mehke opredelitve

Drugi avtorji, kot so Zhengyu, Baotian & Li (2009, str. 391–395) in Kulkarni & Dwivedi (2008, str. 423–430), metrik ne opredeljujejo tako natančno, kot prej opisani modeli. Storitve kategorizirajo zgolj na grobo in drobno granulirane storitve na podlagi obsega funkcionalnosti in količine podatkov, ki so izmenjani v eni interakciji. Kulkarni & Dwivedi (2008, str. 426) pri določanju nivoja granularnosti upoštevata še število sprememb stanj virov sistema.

Pomembna ugotovitev praktično vseh obravnavanih metod je, da:

- grobo granulirane storitve povečujejo kompleksnost vmesnika (pogodbe), povečujejo kompleksnost logike storitve, povzročajo izmenjavo večje količine podatkov, kot bi bilo včasih potrebno, zmanjšujejo možnost ponovne uporabe in povečujejo tveganje potrebnih sprememb po spremembi pogodbe storitve, kar so nezaželene lastnosti v SOA arhitekturi;
- drobno granulirane storitve zahtevajo več interakcij za izvedbo storitve, kar otežuje zagotavljanje atomarnosti transakcij in povzroča dodatne zakasnitve med klici in s tem zagotavljanje kakovosti storitve.

4 UPRAVLJANJE IS

Z rastjo IS, povezanostjo in posledično kompleksnostjo ter z vse večjo frekvenco sprememb znotraj IS, kot odziv na hitro spreminjajoče se okolje podjetja in s tem poslovnih zahtev, je postalo upravljanje IS velik izziv. Striktna delitev na infrastrukturo, razvoj aplikacij in operativno delo je postala neustrezna, saj so se procesi v življenjskem ciklu IS začeli prepletati med seboj in vplivati drug na drugega. Ob tesno povezanih gradnikih IS je praktično nemogoče spremeniti en gradnik, ne da bi s tem vplivali tudi vsaj na enega drugega ali proces, znotraj katerega je ta gradnik uporabljen.

Dodatno kompleksnost so vpeljala tudi orodja za poslovne procese ter druga orodja in okolja temelječa na konceptu delovnih tokov. V teh okoljih je praktično nemogoče zaustaviti celoten IS in zamenjati različico programske opreme ali cel gradnik, saj je veliko primerkov posameznega toka v danem trenutku v izvajanju, takšne tokove pa je težko in drago nadgraditi med izvajanjem. Nekatera okolja in orodja omogočajo paralelno izvajanje več različic istega delovnega toka. V tem primeru pa pride do potencialnih težav, če različici delovnega toka zahtevata različni verziji istega vmesnika. Z uporabo registra storitev ter repozitorija storitev v povezavi s federacijo storitev (angl. *Service Federation*), ki poskrbi za klic in izvedbo ustrezne različice storitve, je to težavo mogoče rešiti, zahteva pa znaten vložek v infrastrukturo in disciplino pri implementaciji tako delovnih tokov, kot samih storitev in postavitve v produkcijsko okolje.

V literaturi in v praksi so se zato začeli pojavljati članki o sistematičnih pristopih, metodah, standardih in najboljših praksah za upravljanje in učinkovito izrabo IS. Najbolj uveljavljen med njimi je IT Information Library (v nadaljevanju ITIL). ITIL je v osemdesetih letih vzpostavila Britanska vladna agencija Central Computer and Telecommunications Agency (v nadaljevanju CCTA) z namenom zagotavljanja bolj učinkovite izrabe IT storitev in virov (*In A Nutshell: A Short History of ITIL*, 2016). CCTA se je aprila 2001 združila v Office of Government Commerce (v nadaljevanju OGC). Druge metode in standardi s področja upravljanja IS so še:

- Standard ISO 20000 in BS 15000: leta 2006 je bil standard BS 15000 sprejet po hitrem ISO postopku v upanju, da bo postal pravi mednarodni standard ISO 20000. Standard se sedaj uveljavlja kot globalna gonilna sila za zagotavljanje strukturiranega in stroškovno učinkovitega upravljanja IT storitev (Greiner, 2007).
- Universal Service Management Body of Knowledge (v nadaljevanju: USMBOK) (*The USM-BOK Navigator* 2016)
- Standard for lightweight service management in federated IT infrastructures (FitSM);
- ISO/IEC 20000-1: Information technology - Service management;
- COBIT 5: A Business Framework for the Governance and Management of Enterprise IT - ISACA;
- MOF - Microsoft Operations Framework in
- TOGAF - The Open Group Architecture Framework.

4.1 ITIL

ITIL je zbirka najboljših praks. Prvič je bila objavljena v osemdesetih v več kot 40 knjigah. Svojo priljubljenost in veljavo pa je pridobila z različico v2, objavljeno v letih med 2000 in 2002, ki je vsebovala 7 knjig:

- podpora storitvam (angl. *Service Support*),
- dobava storitev (angl. *Service Delivery*),
- načrtovanje izvedbe upravljanja storitev (angl. *Planning to Implement Service Management*),
- upravljanje IKT infrastrukture (angl. *ICT Infrastructure Management*),
- upravljanje aplikacij (angl. *Applications Management*),
- upravljanje varnosti (angl. *Security Management*) in
- poslovni vidik (angl. *The Business Perspective*).

Leta 2007 je bila objavljena prenovljena različica ITIL v3, ki je bila zadnjič posodobljena leta 2011. Ta izdaja je veljavna še danes in je sestavljena iz 5 knjig, ki vsebujejo 84 postopkov in

predlogov dokumentov ter 33 procesov. Prenova je bila narejena predvsem zaradi poenostavitve in racionalizacije zbirke:

- strategija storitev (angl. *Service Strategy*),
- snovanje storitev (angl. *Service Design*),
- tranzicija storitev (angl. *Service Transition*),
- obratovanje storitev (angl. *Service Operation*) in
- stalna izboljšava storitev (angl. *Continual Service Improvement*).

Strategija storitev vključuje 2 procesa. Opredeljuje upravljanje portfelja storitev in upravljanje IT financ. Izhaja iz analiz trga in daje trgu velik poudarek. V okviru strategije storitev je potrebno izvajati: ocene trenutnih storitev na trgu, opredelitve splošnih ciljev storitev, posodobitve portfelja storitev v skladu z novimi strategijami, definirati, prožiti in nadzirati programe in projekte izvedbe strategije storitev, zbirati in časovno usklajevati zahteve trga, definirati ustrezne strukture za upravljanje finančnega planiranja in stroškov, določiti potrebne finančne vire za prihajajoča planska obdobja, analizirati dejanske stroške in dobičkonosnost storitev ter izvajati zaračunavanje storitev.

Snovanje storitev vključuje 10 procesov. Opredeljuje upravljanje kataloga storitev, upravljanje nivoja storitev, upravljanje tveganj, upravljanje kapacitet, upravljanje razpoložljivosti, upravljanje neprekinjenega delovanja IT storitev, upravljanje IT varnosti, upravljanje skladnosti, upravljanje IT arhitekture in upravljanje dobaviteljev. V okviru snovanja storitev je potrebno izvajati: zagotavljati obratovanje in vzdrževati katalog storitev z ažurnimi podatki, izpogajati nivo storitev s strankami, zasnovati storitve v skladu z izpogajanimi nivoji storitev, izpogajati, skleniti, nadzirati in poročati dosežen nivo storitev podizvajalcev, identificirati, oceniti in nadzirati tveganja v skladu z vrednostjo sredstva, grožnjami in ranljivostjo, zagotoviti zadostno kapaciteto IT storitev in IT infrastrukture za dobavo dogovorjenih nivojev storitev na stroškovno učinkovit način v razpoložljivem času, definirati, analizirati, meriti, planirati in izboljševati vse vidike razpoložljivosti IT storitev, analizirati, planirati in izvesti vse aktivnosti potrebne za zagotavljanje neprekinjene dobave storitev, zbirati, informirati in nadzirati skladnost storitev z zakonodajo in regulacijo, spremljati, meriti, planirati in zagotavljati zadostne vire IT infrastrukture za planirane bodoče potrebe ter identificirati, dogovoriti, spremljati in obračunati storitve dobaviteljev in podizvajalcev.

Tranzicija storitev vključuje 8 procesov. Opredeljuje upravljanje sprememb, vrednotenje sprememb, upravljanje projektov, razvoj aplikacij, upravljanje izdaj in nameščanje, validacijo in testiranje storitev, upravljanje s sredstvi in konfiguracijami storitev ter upravljanje znanja. V okviru tranzicije storitev je potrebno izvajati: nadzor življenjskega cikla vseh sprememb, oce-

njevanje večjih sprememb pred naslednjo fazo življenjskega cikla, planiranje in koordiniranje virov potrebnih za namestitev večjih izdaj v okviru predvidenih stroškov, časa in kakovosti, zagotavljanje aplikacij in sistemov za potrebne funkcionalnosti IT storitev, planiranje, razporejanje in nadzor premika izdaj na testno in živo okolje, izpolnjevanje pričakovanj strank glede nameščenih izdaj in storitev, preverjanje podpore IT oddelka izdanim različicam, vzdrževanje informacij nosilcev konfiguracij potrebnih za dobavo IT storitev ter zbiranje, analiza, shranjevanje in distribucija znanja znotraj organizacije za izboljšanje učinkovitosti.

Obratovanje storitev vsebuje 9 procesov. Opredeljuje upravljanje dogodkov, upravljanje incidentov, izpolnjevanje zahtev, upravljanje dostopov, upravljanje težav, nadzor IT operacij, upravljanje lokacij, upravljanje aplikacij in tehnično upravljanje. V okviru obratovanja storitev je potrebno izvajati: stalni nadzor nosilcev konfiguracij in storitev, kategorizacija in filtriranje dogodkov za pravilno odločanje, upravljanje življenjskega cikla vseh incidentov, izpolnjevanje manjših zahtev za storitve, dodelitve pravic upravičenim uporabnikom, preprečevanje dostopa nepooblaščenim uporabnikom, upravljanje življenjskega cikla vseh težav, proaktivno preprečevanje težav, spremljanje in nadzor IT storitev in podporne infrastrukture, upravljanje fizičnega okolja, kjer je nameščena IT infrastruktura, upravljanje aplikacij v njihovem življenjskem ciklu ter zagotavljanje strokovnega znanja in podpore pri upravljanju IT infrastrukture.

Stalna izboljšava storitev vključuje 4 procese. Opredeljuje revizijo storitev, vrednotenje procesov, definicijo iniciativ za izboljšave in spremljanje izvedbe iniciativ za izboljšave. V okviru stalne izboljšave storitev je potrebno izvajati: periodično revizijo poslovnih in infrastrukturnih storitev, redno vrednotenje procesov, definiranje konkretnih iniciativ za izboljšavo storitev in procesov na podlagi rezultatov revizije storitev in vrednotenja procesov ter spremljanje in preverjanje izvedbe iniciativ za izboljšave.

4.2 Frameworx

Za Telekom Slovenije so najpomembnejša vodila najboljše prakse in standardi telekomunikacijskih združenj. Med temi združenji pa izstopa TeleManagement Forum (v nadaljevanju TM Forum) (Hribar Rajterič, 2012). TM Forum je globalno, neprofitno združenje ponudnikov storitev, dobaviteljev, sistemskih integratorjev in globalnih korporacij. Najpomembnejši izdelek TM Foruma je ogrodje Frameworx, ki predstavlja skupino najboljših praks in standardov za učinkovito poslovanje. Najpomembnejša vodila iz nabora Frameworx so eTOM, SID in TAM (Hribar Rajterič & Terbuc, 2010).

eTOM je ogrodje za poslovne procese (angl. *Business Process Framework*). Natančno opredeljuje poslovne procese IKT ponudnika in jih matrično umešča po eni strani na področja:

- strategija, infrastruktura in produkti;
  - strategija in zaveze,
  - upravljanje življenjskega cikla infrastrukture,
  - upravljanje življenjskega cikla produktov;
- obratovanje,
  - podpora obratovanju in pripravljenost,
  - izpolnitev naročil,
  - zagotavljanje storitev,
  - obračun in upravljanje prihodkov;

. . . in po drugi strani na domene:

- domena marketinga in prodaje,
- domena produktov,
- domena strank,
- domena storitev,
- domena virov,
- domena partnerjev,
- domena celotnega podjetja in
- domena skupnih vzorcev procesov.

eTOM opredeljuje poslovne procese IKT ponudnika in z umeščanjem v področja in domene določa odgovornost za posamezne procese ter faze, v katerih se procesi izvajajo.

SID (angl. *Shared Information Model*) je podatkovni model prirejen IKT ponudniku. Odpravlja nejasnost pri uporabi pojmov, kot so npr. priključek, storitev, stranka, idr. Uvaja poenoteno nomenklaturo in poenoten podatkovni model za celotno podjetje. Poleg tega opredeljuje tudi obvezne in neobvezne atribute ter relacije med entitetami. Vse entitete umešča tudi v enake domene, kot pri eTOM-u in s tem določa odgovornost in lastništvo nad podatki.

TAM (angl. *Telecom Applications Map*) je ogrodje aplikacij (angl. *Application Framework*) opredeljuje aplikativna področja in jih umešča po enakih kriterijih, kot eTOM. Nevarnost TAM-a je v morebitnem napačnem razumevanju. Vsako aplikativno področje ni nujno tudi samostojna aplikacija. Žal pa je tako razumevanje pogosto in vodi v razdrobljenost IT sistemov.

4.3 ITIL in Frameworx

Ob tehnološkem zbliževanju telekomunikacij z IT tehnologijo in tudi ob dejstvu, da vsako podjetje uporablja IS vsaj v poslovne namene, sta tudi itSMF in TM Forum julija 2009 izdala

rezultate raziskave usklajenosti ITIL in eTOM. Raziskava je postala osnova za stalno usklajevanje med ITIL in eTOM (TM Forum & itSMF, 2009).

Priporočila raziskave dajo navodila, kako lahko uporabimo ITIL in eTOM skupaj ter preslikujejo eTOM procese na procese opredeljene v ITIL. Dokumente vzdržuje združenje TM Forum in so osveženi ob vsaki novi izdaje Frameworx ogrodja. Dokumenti o skupni uporabi nosijo oznako GB921 in so na voljo članom združenja TM Forum.

5 OBVLADOVANJE SOA

SOA izhaja iz distribuiranih sistemov in spreminja perspektivo pogleda na IS na bolj podroben in razpršen način. Pri tako razpršeni odgovornosti pa nujno potrebujemo dodatne mehanizme in orodja za obvladovanje snovanja, razvoja, sprememb in obratovanja, ki zaradi kompleksnosti niso več obvladljivi na ad hoc način.

Obvladovanje SOA (angl. *SOA Governance*) je pomemben vidik storitvene usmerjenosti in SOA arhitekture (Müller, 2009). Razlog je v ohlapni opredelitvi in svobodi, ki je dana arhitektom, snovalcem, analitikom in izvajalcem v SOA. To pa zahteva dodatna pravila, politike, navodila in usmeritve v konkretnem okolju pri izbiri gradnikov, njihovi povezanosti, določanju granularnosti storitev, varnosti, razpoložljivosti, zanesljivosti in izvedbi storitev. Proces vzpostavitve dodatnih pravil, politik, navodil in usmeritev, ter njihov nadzor in uveljavljanje imenujemo **obvladovanje SOA**. Uspešno (ali neuspešno) obvladovanje SOA ima neposreden vpliv na uspeh SOA projektov, kasneje pa preprečuje razvodenitev doseženih rezultatov skozi arhitekturno erozijo in arhitekturni odmik, obravnavana na strani 26.

„Obvladovanje SOA se osredotoča na odločitve v celotnem življenjskem ciklu storitev, da bi podjetju omogočilo doseganje pričakovanih koristi uvedbe SOA. Je pristop k uvedbi nadzora in obvladovanja tveganj z vzpostavitvijo organizacijskih struktur, procesov, politik in metrik za zagotovitev usklajenosti uvedbe, implementacije, obratovanja in razvoja SOA s strategijo in cilji podjetja ter skladnost z zakonodajo, regulacijo in najboljšimi praksami.” (Ott, Korthaus, Böhmann, Rosemann & Krcmar, 2011, str. 45)

5.1 Organizacijske vloge

Ott, Korthaus, Böhmann, Rosemann & Krcmar (2011) obravnavajo obvladovanje SOA iz organizacijskega stališča, identificirajo potrebne aktivnosti in vloge ter nato dodelijo odgovornost za posamezne aktivnosti vlogam. V nadaljevanju potegnejo vzporednico z že uveljavljenimi

metodologijami za obvladovanje IT okolij, kot sta Cobit in ITIL. Na podlagi raziskave trga dela identificirajo naslednje vloge:

- poslovni analitiki,
- krovni arhitekti (angl. *Enterprise Architect*),
- lastniki storitev,
- knjižničarji storitev in
- projektni vodje.

Tudi Erl (2008, str. 488–495) poudari potrebo po spremembi v organizacijski strukturi in opredeli nove vloge, potrebne za uspešno uvedbo in obvladovanje SOA:

- analitiki storitev,
- arhitekti storitev,
- skrbniki storitev,
- skrbniki shem,
- skrbniki politik,
- skrbniki registra storitev,
- tehnični strokovnjaki za komunikacije,
- krovni arhitekti (angl. *Enterprise Architect*) in
- skrbniki (in nadzorniki) standardov snovanja v podjetju.

Z izjemo lastnikov storitev in knjižničarjev storitev, so vloge opredeljene po Ott et al. splošne, slabše razdelane in niso specifične za SOA okolje, zato se v tem delu opiram na Erlovo opredelitev.

Analitiki storitev morajo znati pravilno izbirati kandidate za storitve, zmožnosti kandidatov za storitve in kandidate za kompozicije storitev. Analitiki storitev morajo biti dobro usposobljeni za vse faze procesa storitveno usmerjene analize, vključno z dobavo kandidatov za storitve skozi proces modeliranja storitev.

Arhitekti storitev so primarno usmerjeni v fizično zasnovo storitev. Svoje delo izvajajo v okviru procesa snovanja storitev in drugih procesih, ki vključujejo različne vidike snovanja povezanega z modelom storitev. Arhitekte storitev je potrebno izbrati v trenutku, ko podjetje začne s fazami snovanja in razvoja v okviru SOA uvedbe. Njihovo delo temelji na identificiranih kandidatih za storitve. Na njih uporabijo standarde in konvencije snovanja, rezultat pa je pogodba storitve in zasnova logike.

Skrbniki storitev so zadolženi za eno ali več storitev. Njihove zadolžitve pa ne obsegajo samo

razširitve in dopolnitve logike storitve, temveč morajo poskrbeti za celovitost in integriteto konteksta storitve ter njenih funkcionalnih mej. Svoje delo lahko pričnejo takoj, ko je definiran kontekst storitve v fazi analize storitve.

Skrbniki shem so potrebni v okoljih, kjer so v uporabi (SOAP) spletne storitve. Prilagodljivost ogrodja spletnih storitev, ki omogoča arhitekturo predstavitve podatkov sestavljenih iz različnih XML shem, kreiranih neodvisno od samih storitev, omogoča neodvisno definicijo in vzdrževanje shem. Skrbniki shem morajo imeti poglobljeno poznavanje informacijske arhitekture v podjetju, zelo dobro pa morajo poznati jezik XML shem. Vključeni so vsakič, ko je potrebno uvesti novo XML shemo. Njihov glavni izdelek so prav XML sheme, skrbijo pa tudi za različice shem.

Skrbniki politik so tesno povezani z XML shemami in skrbijo za politike povezane s pogodbami spletnih storitev. Skrbniki politik in skrbniki shem so pogosto iste osebe. Povezava med shemami, spletnimi storitvami in varnostno politiko je običajno identificirana v kasnejših fazah, ob snovanju dejanske logike storitve. Druge vrste politik, kot so različna poslovna pravila ali politike, pa so lahko identificirane v različnih fazah. Lastništvo nad temi politikami je skupno s tehničnimi in poslovnimi analitiki.

Skrbniki registra storitev so zadolženi za vsebino registra storitev. V primeru, da vsebina registra storitev kadarkoli zastari ali postane celo netočna, register storitev izgubi svojo vlogo, kot osrednja točka SOA infrastrukture. Skrbniki registra storitev so zadolženi za ažurnost vsebine registra, praviloma pa s svojimi specifičnimi znanji izvajajo tudi namestitve in vzdrževanje registra.

Tehnični strokovnjaki za komunikacije morajo na podlagi začetnega opisa profila storitve in pripadajočih meta podatkov izdelati podroben opis za potrebe odkrivanja storitve. Sposobni morajo biti opisati storitev v naravnem jeziku z uporabo standardnega besednjaka tako, da ga bo razumela večina članov projektne skupine, da bodo lahko sestavili poizvedbe, skozi katere bodo pogodbo storitve ter pripadajoče profile odkrili in razumeli.

Krovni arhitekt je obstoječa vloga v IT okoljih, ki postane z uvajanjem SOA še pomembnejša zaradi povezovanja številnih aplikacij in poslovnih področij v podjetju. Člani te vloge pripravljajo in sodelujejo pri vzpostavljanju standardov snovanja, sodelujejo pri dobavi storitev, da zagotovijo pravilno umeščanje neodvisnih storitev, spremljajo porabo virov med obratovanjem za načrtovanje infrastrukture, obravnavajo varnostni vidik posameznih storitev, pomagajo definirati načrt inventarja storitev in so pogosto lastniki repozitorija storitev. Z vpeljavo SOA se poveča potreba po centralnih virih celotnega podjetja, kar vodi v potrebo po povečavi skupine krovnih arhitektov.

Skrbniki (in nadzorniki) standardov snovanja v podjetju skrbijo za razvoj in nadzor nad standardi snovanja v celotnem podjetju. Z rastjo skupine krovnih arhitektov, standarde snovanja postavlja več različnih oseb. Pride do specializacije posameznikov po posameznih vidikih, kot so: varnost, zmogljivost, transakcijska celovitost, idr. Da so standardi medsebojno usklajeni in uporabljeni, je potrebno uvesti vlogo formalnih skrbnikov. Naloga skrbnikov je posodabljanje in razvoj standardov, kot tudi nadzor nad njihovo uporabo oziroma preprečevanje kršenja standardov. Ta vloga zato pogosto vključuje preglede predlaganih standardov in zasnov storitev.

5.2 Orodja za obvladovanje SOA

Ob številnih vlogah in aktivnostih povezanih z obvladovanjem SOA ter še številnejšimi dokumenti, je ročno vodenje zahtev, politik in specifikacij ter uveljavljanje istih praktično nemogoče. Za obvladovanje SOA so potrebna orodja.

Naloge in odgovornosti zgoraj opisanih vlog so šibko opredeljene in torej mehke. Celotnega obvladovanja SOA zato ni preprosto podpreti z univerzalnimi orodji. V vsakem okolju bi bilo praktično potrebno uvesti prilagojena orodja. Skozi pregled strokovne literature, pa vseeno na površje prideta dve orodji, ki ju je mogoče uporabiti praktično v vsakem storitveno usmerjenem okolju (Derler & Weinreich, 2006, str. 119):

- konzola repozitorija storitev in
- brskalnik storitev.

Obe našteti orodji temeljita na standardu MoWS združenja OASIS. Standard MoWS je bil sprejet v sedanji različici (v1.1) avgusta 2006. Standard temelji na standardu MUWS združenja OASIS namenjenem upravljanju spletnih storitev. MoWS opredeljuje upravljanje spletnih storitev v poslovnem okolju. Oba omenjena standarda sta člana družine standardov WSDM združenja OASIS, namenjena distribuiranemu upravljanju spletnih storitev.

Konzola repozitorija storitev je namenjena pripravi predlogov storitev, njihovih opisov, relacij z drugimi storitvami in za definicijo namestitve storitev. Orodje omogoča več različnih pogledov. Takšno orodje zbira informacije o vseh aktivnih in predlaganih storitvah na enem mestu na enoten način in preprečuje nepotrebno podvajanje funkcionalnosti ter preprečuje morebitne konflikte.

Brskalnik storitev omogoča iskanje in brskanje po repozitoriju ter seznanjanje s podrobnimi lastnosti storitev, njihovih relacij in stanja. Vključuje tudi programske module in komponente, ki še niso vključeni v obstoječe storitve, so pa potencialni gradniki katere od novih storitev. Iskanje

poteka na podlagi različnih kriterijev. Omogoča tudi vpogled v medsebojne odvisnosti med storitvami in storitev od drugih komponent in gradnikov IS.

6 STROŠEK IS

Integracija ima pomemben vpliv na strošek IS in druge pozitivne in negativne posledice. Med druge negativne posledice sodita predvsem izguba avtonomije in fleksibilnosti (Goodhue, Wybo & Kirsch, 1992, str. 293).

Informacijski sistem podjetja obravnavam kot celoto, ki je sestavljena iz več gradnikov. Podobno, kot mora podjetje delovati kot celota, mora tudi IS podpirati poslovanje kot celota. Posledično ne moremo obravnavati različnih IS znotraj podjetja, saj le-ti ne morejo obstajati neodvisno ločeni drug od drugega. V takem primeru govorimo o različnih informacijskih rešitvah. V kolikor bi imelo podjetje dva ali več nepovezanih IS, morajo zaposleni ročno prepisovati vsaj majhen del podatkov iz enega v drug IS, odločevalci pa bi morali pridobiti podatke za odločanje iz različnih virov. Tipičen primer so različna poročila. Računovodsko stroške zaposlenih sicer vodimo pri stroških dela, dejansko pa gre v takem primeru za prikrit strošek integracije IS.

Ob uvajanju novih IS je uveljavljen pristop izbire uporaba izračuna TCO (Cowan, 2005, str. 87). Ta izračun naj bi zajemal izračun vseh stroškov v življenjski dobi IS. To velja tako za strojno opremo, kjer je mogoč nakup s podaljšano garancijsko dobo za predvideno življenjsko dobo strojne opreme, kot pri programski opremi, kjer je mogoče skleniti vzdrževalno pogodbo za časovno omejeno ali neomejeno obdobje in tako fiksirati stroške vzdrževanja v tem obdobju. Tak pristop pa le redko zajema strošek integracij in še redkeje stroške obratovanja IS. Zato je lahko pristop z izračunom TCO pomanjkljiv in ne daje optimalnega rezultata gledano s stališča celotnega IS in iz stališča podjetja kot celote.

Matematično gledano odločitve o uvedbi novega IS, ki temeljijo na parcialnih izračunih TCO, optimizirajo parcialne stroške, ne pa stroške celotnega IS podjetja. Posledično torej ob zamenjavi posamezne komponente IS iščemo lokalni minimum namesto globalnega.

Za analitične in operativne potrebe je res pogosto potrebno IS razdeliti na manjše sklope, v nobenem trenutku pa ne smemo izgubiti celovitega pogleda na IS.

Primer pretiranega drobljenja IS v Telekomu Slovenije je obračunski sistem: konec devetdesetih let je bil vpeljan nov obračunski sistem sestavljen iz dveh delov:

- Skrb za uporabnika in obračun (angl. *Customer Care and Billing*) in
- Zbiralnik zapisov (angl. *Record Collector*).

Modul „Skrb za uporabnika in obračun” je vseboval naslednje funkcionalnosti:

- Skrbništvo strank (angl. *Customer Care*),
- Skrbništvo storitev in produktov,
- Obračun storitev,
- Izdaja računov,
- Saldakonti kupcev,
- Izterjava dolžnikov,
- Reklamacije,
- Poročilni sistem,
- Preprečevanje prevar in
- Vmesnik za glavno knjigo.

Modul „Zbiralnik zapisov” je vseboval naslednje funkcionalnosti:

- Zbiranje zapisov iz central in drugih virov ter
- vrednotenje zapisov.

Danes so zgoraj naštete funkcionalnosti razbite med številne ločene aplikacije:

- Upravljanje naročil (angl. *Order Management*),
- Skrb za stranke (CRM),
- Produktni katalog,
- Mediacija,
- Vrednotenje (angl. *Rating*),
- Obračun (angl. *Billing*),
- Izdaja računov (angl. *Invoicing*),
- Saldakonti kupcev,
- Zagotavljanje prihodkov (angl. *Revenue Assurance*) in
- Poslovni procesi (ti povezujejo zgornje aplikacije).

Obračunski sistem sestavljen iz dveh aplikacij enega dobavitelja, je v petnajstih letih zrasel v devet ločenih aplikacij šestih različnih dobaviteljev in lastnega razvoja. Ni potrebno posebej poudarjati, da so poleg procesov, naštetih zgoraj, ki že opravljajo funkcijo integracije, potrebne še številne integracije med posameznimi aplikacijami.

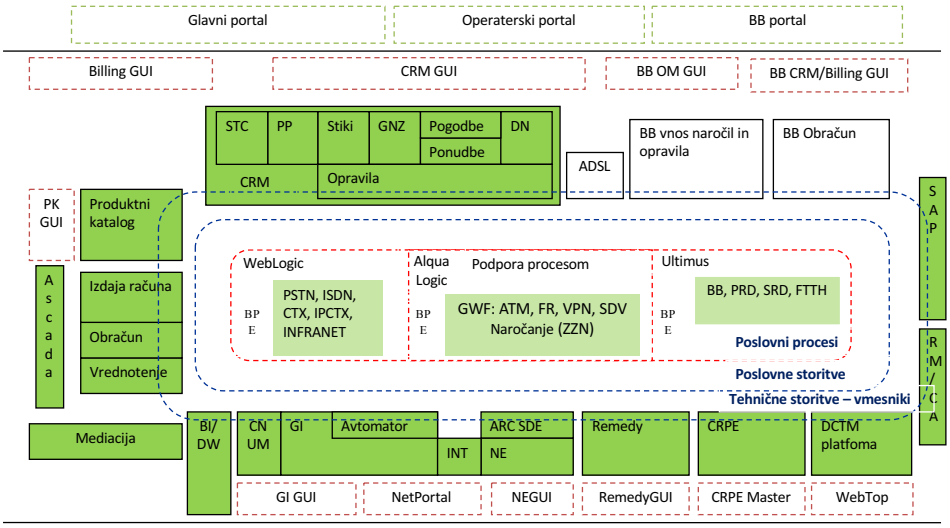
Takšno drobljenje je seveda interes dobaviteljev programske opreme in velikih integratorjev, nikakor pa ne podjetja, ki takšne rešitve uporablja.

Vloga združenja TM Forum je tukaj dvolična: dobavitelji programske opreme in integratorji

imajo v združenju velik vpliv, zato ni presenetljivo, da se drobljenje odraža tudi v standardih TM Foruma. Pozitivna stran pa je standardizacija, ki jo uvaja združenje, predvsem skozi enovit podatkovni model SID, kot tudi skozi preostale dele modela Frameworx, kot sta eTOM in TAM.

Sliki 8 in 9 predstavljata kompleksnost, ki je nastala skozi leta dograjevanja IS in organizacijskih sprememb v Telekomu Slovenije.

Slika 8: Arhitektura IS v Telekomu Slovenije



Vir: G. Hribar Rajterič & L. Terbuc, Vpeljava celovite rešitve upravljanja naročil telekomunikacijskih storitev, 2010.

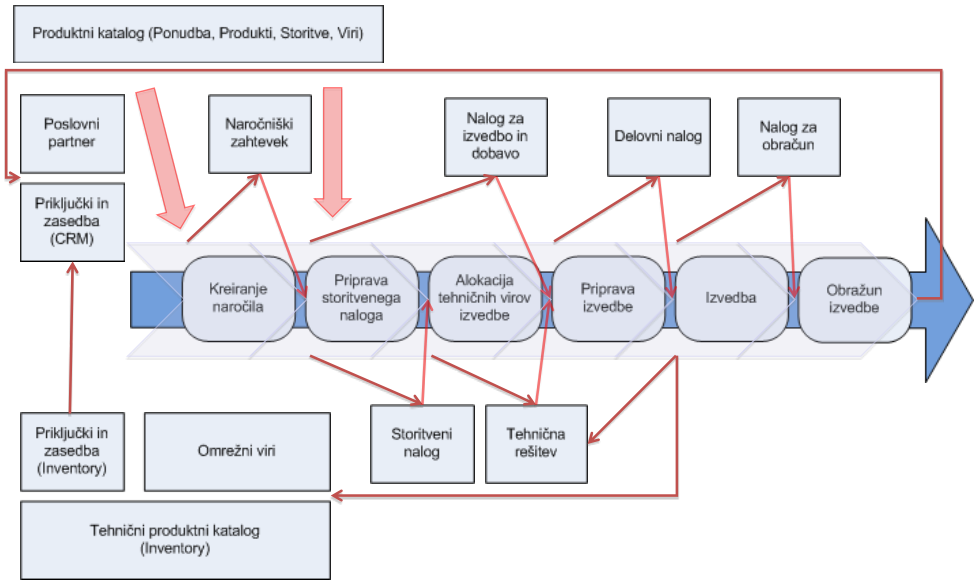
Nastala arhitektura v Telekomu Slovenije in pot, ki je pripeljala do nje, kažeta na tipične pomanjkljivosti v slovenskih podjetjih (Hribar Rajterič, 2013).

Peklaj (2003, str. 48) opozarja, da je dejanski strošek IS 3- do 4-krat višji od nabavne vrednosti samega sistema. Poleg tega so ključnega pomena za uspešno uvedbo novega IS interni IT kadri in uporabniki. To pomeni, da stroški dela predstavljajo velik delež v strukturi stroškov uvedbe novega IS. Pri obratovanju pa so stroški dela lahko še višji.

Raziskave so pokazale (Peklaj, 2003, str. 50), da je bila povprečna naložba v vpeljavo novega IS od 9- do 10-krat višja od koristi oziroma prihrankov, ki so jih podjetja z uvedbo novega IS dosegla.

Leta 2009 je bila informatika v slovenskih podjetjih zapostavljena (Groznik et al., 2010, str. 3). To pa se odraža tudi v sredstvih namenjenih za vlaganja in še bolj izrazito v zniževanju sredstev za obratovanje in vzdrževanje IS.

Slika 9: Proces naročila in dobave storitve v Telekomu Slovenije



Vir: G. Hribar Rajterič & L. Terbuc, Vpeljava celovite rešitve upravljanja naročil telekomunikacijskih storitev, 2010.

Stroške IT sistemov sestavljajo (Turk, 2005, str. 161):

- stroški strojne opreme,
- stroški infrastrukture,
- stroški licenčne programske opreme in
- stroški dela.

Stroški dela so lahko interni ali zunanji v odvisnosti od optimalne izbire načina vzdrževanja. Tipično stroške dela sestavljajo stroški podpore in stroški zagotavljanja obratovanja IT sistemov. Stroške podpore lahko razdelimo na tri nivoje:

- prvi nivo - podpora končnim uporabnikom, posredovanje vprašanj in prijava napak na drugi nivo;
- drugi nivo - podpora ključnim uporabnikom, odgovori na vprašanja prvega nivoja, reprodukcija napak na testnih sistemih, posredovanje vprašanj in prijava napak na tretji nivo;
- tretji nivo - odgovori na vprašanja drugega nivoja, analiza prijavljenih napak, odprava napak v kodi.

Pomemben delež stroškov so zaposleni, ki predstavljajo strošek dela. Pri stroških dela nastopa več dejavnikov:

- plačilo, ki je odvisno od izobrazbe, znanj in vloženega truda;

- delovni čas in dežurstva;
- izobraževanje.

Stroški infrastrukture zajemajo:

- stroški prostorov,
- stroški elektroenergetskih sistemov (napajanje in hlajenje),
- stroški elektronskih komunikacij,
- strošek električne energije in
- strošek varovanja.

6.1 Faktorji vpliva na stroške IS

Visoke cene informacijskih rešitev in omejene tehnične možnosti so botrovale uvajanju informatizacije sprva samo posameznih poslovnih funkcij. Prve so si lahko privoščili računalniške rešitve samo vlade velikih držav in velika podjetja. Nižanje cen pa je omogočilo vse širše pokrivanje poslovanja in vstop v informacijsko dobo tudi manjšim podjetjem ter kasneje zasebnim uporabnikom. Od uvedbe prvega poslovnega računalnika UNIVAC leta 1948 (Stern, 1979, str. 11) do širše uporabe v manjših podjetjih in v naših krajih, je minilo približno 25 do 30 let.

Danes je uporaba informacijske podpore nujna za uspešno poslovanje skoraj vsakega podjetja, obseg poslovanja podprt z IT pa še raste. **Obseg IS** je glavni faktor vpliva na strošek IS in je neposredno povezan s obsegom funkcionalnosti podprto v IS. Merjenje obsega funkcionalnosti IT rešitev lahko izvajamo z uporabo metode funkcijskih točk ali z metodo števila vrstic kode (LOC) (Krammer, Heinrich, Henneberger & Lautenbacher, 2011, str. 351). Slabost merjenja obsega funkcionalnosti s funkcijskimi točkami je v njeni subjektivnosti, slabost metode LOC pa v neprimerljivosti med različnimi programskimi jeziki oziroma njihovo izraznostjo in posledično potrebnim številom vrstic za implementacijo enake funkcionalnosti. Za konsistentne rezultate je pomembno, da uporabljamo enako metodo skozi celotno fazo analize in snovanja ter da se zavedamo pomanjkljivosti uporabljene metode in se jim v največji možni meri izognemo. Za potrebe tega dela je pomembna granularnost storitev, ki je relativna mera in zato celotnega obsega IS ne potrebujemo.

Zanesljivost IS povečamo s podvajanjem gradnikov arhitekture, geografskim ločevanjem infrastrukture, z zniževanjem sklopljenosti storitev in gradnikov med seboj ter z avtomatiziranim spremljanjem in ukrepanjem v primeru ugotovljenih napak in/ali predvidenih težav. Opisani ukrepi neposredno vplivajo na dodatno potrebno strojno opremo ter dodatno licenčno programsko opremo, niso pa povezani z granularnostjo storitev.

Hitrost delovanja IS lahko ob danih drugih zahtevah izboljšamo s hitrejšo strojno opremo, optimizacijo implementacije storitev in komponent ter z uvajanjem paralelnosti obdelav skozi podvajanje fizičnih komponent arhitekture IS. Hitrost delovanja je povezana tudi s kompleksnostjo IS in posledično z granularnostjo storitev. Zaradi obstoja drugih mehanizmov in pristopov k izboljšanju hitrosti in odzivnosti IS ter relativno majhnega vpliva granularnosti na hitrost delovanja, v tem delu zaradi poenostavitve, hitrosti ne upoštevam pri vplivih na stroške IS.

Kompleksnost storitev in celotnega IS je povezana z zrelostjo SOA arhitekture, številom gradnikov arhitekture (Hirzalla, Cleland-Huang & Arsanjani, 2009, str. 47), granularnostjo storitev, heterogenostjo okolja ter povezanostjo in sklopljenostjo okolja. Višja, kot je kompleksnost, težje in dražje je obratovanje takšnega okolja. Neposreden vpliv kompleksnosti okolja pa je na usposobljenost kadrov, ki izvajajo in zagotavljajo obratovanje IS. To pa je naprej povezano s stroški dela.

Heterogenost okolja vpliva na stroške različne licenčne programske opreme, kot so licenčnine za operacijske sisteme, SOA gradnike ter orodja za nadzor in upravljanje. Poleg vpliva na licenčnine ima heterogenost vpliv na potrebna znanja zaposlenih, ki izvajajo in zagotavljajo obratovanje IS ter s tem na stroške dela. Po drugi strani heterogenost okolja zmanjšuje odvisnost od enega proizvajalca in s tem omogoča podjetju boljše pogajalsko izhodišče pri sklepanju in podaljševanju pogodb za nakupe in vzdrževanje licenčne programske opreme.

Nadzor in spremljanje IS lahko izvajamo z nadzornimi orodji ali ročno z ustreznim osebjem. V praksi sta nadzor in spremljanje kombinacija tako avtomatiziranega zajema podatkov o delovanju sistema, delno avtomatiziranega spremljanja delovanja in ročnega spremljanja ter ukrepanja v primeru odstopanj oziroma ugotovljenih napak in nezaželenih trendov. SOA okolje zahteva visok nivo nadzora in spremljanja (Müller, 2009, str. 340) prav zaradi kompleksnosti, ki je v SOA okolju neizbežno. Za potrebe tega dela uvajam faktor avtomatizacije F_A, ki predstavlja stopnjo avtomatiziranega spremljanja IS in je ocena arhitekta. F_A je realno število in ima zalogo vrednosti iz obsega $[0, 1]$, pri čemer vrednost 0 pomeni, da se celotno spremljanje izvaja ročno, vrednost 1 pa, da je spremljanje povsem avtomatizirano.

Potrebna znanja in spretnosti zaposlenih za upravljanje in zagotavljanje obratovanja IS so posledica zgoraj naštetih faktorjev in neposredno vplivajo na stroške dela.

Razpoložljivost ustreznih kadrov oziroma stanje na trgu delovne sile vpliva na ceno in s tem na strošek dela. Več, kot je povpraševanja in nižja, kot je ponudba, višja je cena in s tem strošek dela, ter obratno.

Na stroške dela vplivajo tudi **fluktuacija zaposlenih** ter **absentizem**. Teh dveh faktorjev zaradi

poenostavitve in nizkega vpliva, v tem delu ne upoštevam.

6.2 Povezanost IS

Za izgradnjo modela potrebujem opredelitev pojma *povezanosti IS*. Povezanost lahko opredelim s stopnjami (Worden, 2005, str. 4), za uporabo v zveznem modelu pa je bolj primerna zvezna opredelitev.

Povezanost IS izražam s stopnjo potrebnih podvojenih vnosov podatkov, ki so že bili vneseni v IS drugje ali pa so rezultat obdelave katerega od delov IS. Pri tem so vnosi lahko povsem ročni ali preko namiznih orodij, kot je Microsoft Excel in podobni. V še tako ozaveščenem okolju obstajajo pomembni podatki shranjeni pri uporabnikih v preglednicah, besedilnih dokumentih in drugih namiznih oblikah, dosegljivi samo omejenemu krogu zaposlenih.

Povezanost sistema je pozitivna lastnost in zmanjšuje stroške poslovanja. Ne smemo pa je zamenjati s **sklopljenostjo sistema**. Sklopljenost sistema je negativna posledica povezanosti oziroma implementacije povezav v IS in ima negativen vpliv na stroške. Sklopljenost povzroča medsebojno odvisnost komponent. Za potrebe tega dela naj bo stopnja povezanosti S_p opredeljena kot:

$$S_p = 1 - S_{prv} \tag{12}$$

Pri tem je S_{prv} stopnja podvojenih ročnih vnosov izražena v % glede na vse vnose v IS. Višja, kot je stopnja povezanosti, manj podvojenih ročnih vnosov je v sistem. V primeru, ko se vneseni podatki v sistem večkrat podvojeno vnašajo, štejem vsak podvojen vnos. S_p ima lahko vrednosti iz intervala $[0,1]$, pri čemer je vrednost 0, ko se vsi podatki vnašajo dvakrat ali večkrat in 1, ko takšnih vnosov ni.

7 MODEL VPLIVA GRANULARNOSTI NA STROŠKE OBRATOVANJA

7.1 Omejitve

V tem delu se omejujem na vpliv granularnosti na stroške obratovanja IS. Vrste stroškov IS so naštete v poglavju 6 na strani 66. Granularnost spletnih storitev ima zanemarljiv vpliv na povečanje ali zmanjšanje potrebe po procesorski moči, komunikacijah, hrambi in drugih infrastrukturnih virih, zato bom ta vpliv v tem delu izpustil z namenom poenostavitve modela. Med obratovanje ne sodi niti nakup licenčne programske opreme, ki spada v fazo investicije ob vzpostavitvi IS, upošteval pa bom vzdrževanje in storitve povezane z licenčno programsko opremo, potrebno za obratovanje, oziroma njenim najemom. Upoštevajoč priporočila ITIL, ki

v obratovanje uvršča 9 procesov, se v tem delu omejujem na procese naštete v poglavju 4.1 na strani 57.

Pri stroških dela poenostavim model z linearnim koeficientom zahtevanega kadra, kar pomeni, da po potrebi lahko zaposlimo in plačamo samo del zaposlenega, če obseg dela ne zahteva celote. V realnosti temu seveda ni tako, je pa predpostavka realna, če za delo najamemo zunanjega izvajalca ali če zaposleni, poleg dela na obratovanju IS, opravlja še drugo delo.

Funkcija stroškov obratovanja v odvisnosti od granularnosti je v vsaki točki bodisi padajoča, naraščajoča ali pa ima ekstrem. Minimalni strošek v odvisnosti od granularnosti je torej na mejah funkcije ali v točki, kjer je odvod funkcije stroškov po granularnosti enak 0. Odvod funkcije, ki ni odvisna od granularnosti je vselej 0, zato lahko vse stroške, ki niso odvisni od granularnosti, za potrebe tega dela, zanemarim.

7.2 Izhodiščna enačba

Skupen strošek obratovanja IS naj bo S, sestavljen iz licenčnin S_l in stroškov dela S_d:

$$S = S_l + S_d \tag{13}$$

Najnižji stroški so pri odvodu zgornje vsote po granularnosti. Zato v nadaljevanju izrazim tako stroške licenčnin, kot stroške dela s funkcijo granularnosti in poiščem optimum z odvodom po granularnosti. Pri tem izhajam iz FSG grafa prikazanega na strani 50. Granularnost pa opredelim malo drugače, kot je opredeljena v Krammarjevem modelu: stopnja granularnosti celotnega IS G izhaja iz, oziroma določa število storitev n tako, da je pri najbolj grobi granularnosti število storitev enako številu poslovnih procesov in funkcij P, pri najbolj drobni granularnosti, pa je n enak številu elementarnih funkcionalnosti m:

$$n = Gm + (1-G)P \; ; \; m \geqslant n \geqslant P \tag{14}$$

$$G = \frac{n-P}{m-P} \tag{15}$$

Pri tem je G realno število z zalogo vrednosti iz območja $[0,1]$. Vrednost 0 predstavlja najbolj grobo granularnost, pri kateri je največ povezav ene storitve, vrednost 1 pa predstavlja najbolj drobno granularnost, kjer ima vsaka storitev najmanj povezav. Število poslovnih procesov in funkcij P ter število elementarnih funkcionalnosti m ne smeta biti enaka, kar je smiselna zahteva.

7.3 Strošek licenčnin

Granularnost vpliva na strošek licenčnin z obsegom povezav (integracij) med poslovnimi procesi, storitvami in drugimi gradniki SOA arhitekture. Kot sem zapisal na strani 6, je bilo pri klasičnih aplikacijah število integracij med 5 in 30 odvisno od zrelosti oziroma starosti aplikacije. V SOA arhitekturi je aplikaciji enakovredna podmnožica poslovnih procesov in funkcij skupaj z vsemi njenimi gradniki: storitvami in funkcionalnostmi.

Podobno, kot se aplikacije povezujejo med seboj, se med seboj povezujejo tudi drugi gradniki, da izpolnijo zahteve in ponudijo zahtevano funkcionalnost. Zato predpostavljam, da velja ocena števila integracij aplikacij tudi za število povezav storitev. Običajna življenjska doba IS je med 10 in 15 let (Swanson & Dans, 2000, str. 288), kar potrjujejo tudi moje izkušnje. Zaradi poenostavitve privzamem skrajno mejo 15 let. Povečevanje števila povezav ni linearno, temveč se sprva hitreje povečuje, proti koncu življenjske dobe pa vse počasneje. Upočasnjeno naraščanje povezav prikažem z uporabo funkcije kvadratni koren. Z zapisanimi predpostavkami in omejitvami je enačba števila povezav ene storitve:

$$N_i = 5 + 25\sqrt{\frac{A_i}{A_{\check{z}}}} = 5 + 25\sqrt{\frac{A_i}{15}} \tag{16}$$

Pri tem z N_i označujem število povezav storitve i, z A_i starost storitve i in z $A_{\check{z}}$ pričakovano življenjsko dobo storitev, ki je v tem primeru konstanta in ima vrednost 15. Za izračun dejanskega števila povezav moram upoštevati še faktor granularnosti in sešteti povezave vseh storitev:

$$N = \sum_{i=1}^{n} \left(5 + 25\sqrt{\frac{A_i}{15}}\right)\left(1 - \frac{4G}{5}\right) \tag{17}$$

Pri čemer je N skupno število povezav, n predstavlja število storitev, A_i starost storitve i in G faktor granularnosti. Koeficient je povečan za $0,5$ in odštet zaradi negativnega vpliva vrednosti na število povezav. Po moji oceni, granularnost vpliva na število povezav v takšni meri, da ima storitev pri najbolj drobni granularnosti samo eno povezavo, zato je koeficient pomnožen s faktorjem $\frac{4}{5}$. Mejo n sedaj izrazim z (14). Z upoštevanjem odvisnosti števila storitev od granularnosti, je enačba števila povezav:

$$N = \sum_{i=1}^{Gm+(1-G)P} \left(5 + 25\sqrt{\frac{A_i}{15}}\right)\left(1 - \frac{4G}{5}\right) \tag{18}$$

V posebnem primeru, ko so vse storitve enako stare, je A_i enaka za vse. Označim jo z A. Enačba

za ta poseben primer je:

$$N = \sum_{i=1}^{Gm+(1-G)P} \left(5 + 25\sqrt{\frac{A}{15}}\right)\left(1 - \frac{4G}{5}\right) \tag{19}$$

Glede na to, da je licenčnina za obstoječ sistem že znana, je smiselno računati število povezav le za nov sistem. V takem primeru je $A = 0$ in enačbo lahko poenostavim v kvadratno enačbo:

$$\begin{aligned} N &= \sum_{i=1}^{Gm+(1-G)P} 5\left(1 - \frac{4G}{5}\right) \\ &= 5\left(Gm + (1-G)P\right)\left(1 - \frac{4G}{5}\right) \\ &= 5mG + 5(1-G)P - 4G\left(Gm + (1-G)P\right) \\ &= 5mG + 5P - 5PG - 4mG^2 - 4PG + 4PG^2 \\ &= (4P - 4m)G^2 + (5m - 9P)G + 5P \end{aligned} \tag{20}$$

Prvi odvod funkcije po G pomaga poiskati ekstreme:

$$f'(G) = 2(4P - 4m)G + (5m - 9P) = (8P - 8m)G + (5m - 9P) \tag{21}$$

Ekstrem obstaja pri vrednosti odvoda enakem 0:

$$\begin{aligned} f'(G) &= 0 \\ (8P - 8m)G + (5m - 9P) &= 0 \\ (8P - 8m)G &= 9P - 5m \\ G &= \frac{9P - 5m}{8P - 8m} \end{aligned} \tag{22}$$

Preverim še drugi odvod funkcije, da ugotovim, če gre za minimum ali maksimum:

$$f''(G) = 8P - 8m \tag{23}$$

Drugi odvod (23) bi bil enak 0 le v primeru $P = m$, kar pa je prepovedano v opredelitvi granularnosti (15). P je vedno manjši od m, zato je drugi odvod negativen ne glede na vrednost G. To pomeni, da je v tej točki lokalni maksimum funkcije števila povezav. Ker pa je to edini ekstrem, je to hkrati tudi globalni maksimum. To je nasprotno tistemu, kar iščem: najmanjše

število povezav, zato moram preveriti vrednost funkcije še na obeh mejah:

$$G = 0 \mid (4P - 4m)\, G^2 + (5m - 9P)\, G + 5P = 5P \tag{24}$$

$$G = 1 \mid (4P - 4m)\, G^2 + (5m - 9P)\, G + 5P = m \tag{25}$$

Minimalno število povezav je pri danih predpostavkah in omejitvah torej:

$$N_{min} = \min(5P, m) \tag{26}$$

Strošek licenčnine je tesno odvisen od števila povezav in heterogenosti sistema:

$$S_l = C_l N_p + S_h = C_l \sum_{i=1}^{n} \left(5 + 25\sqrt{\frac{A_i}{15}}\right)\left(1 - \frac{4G}{5}\right) + S_h \tag{27}$$

Pri tem je S_l strošek licenčnin, C_l je cena licenčnine za posamezno povezavo in S_h strošek heterogenosti, ki ni odvisen od granularnosti. V primeru novega sistema ($A = 0$), je strošek licenčnin izražen s kvadratno enačbo:

$$S_l = C_l N_p + S_h = C_l \left((4P - 4m)\, G^2 + (5m - 9P)\, G + 5P\right) + S_h \tag{28}$$

Pri logični predpostavki, da je cena $C_l > 0$, je minimum stroška licenčnin pri enakem G, kot najmanjše število povezav. Optimalno nizki stroški licenčnin so torej:

$$S_{l_{min}} = C_l N_p + S_h = C_l \min(5P, m) + S_h \tag{29}$$

Pri enačbi (28) sem upošteval, da so vse povezave nadzorovane z licenčno programsko opremo. V praksi to le redko drži, predstavlja pa zaželeno ciljno stanje. Za natančnejšo opredelitev moram upoštevati še faktor avtomatizacije F_A:

$$S_{l_{min}} = C_l N_p F_A + S_h = C_l \min(5P, m)\, F_A + S_h \tag{30}$$

Ročni del nadzora upoštevam pri stroških dela v nadaljevanju.

7.4 Strošek dela

Strošek dela bom izrazil posredno kot ceno vloženega dela izhajajoč iz potrebnega vložka dela po posameznih vrstah del:

$$S_d = \sum_{j=1}^{k} \left(C_{d_j}\, Z_{d_j}\right) \tag{31}$$

Pri tem S_d predstavlja skupni strošek dela, k predstavlja število različnih vrst del, C_{d_j} predstavlja ceno dela vrte j in Z_{d_j} obseg dela vrste j.

Najprej izenačim količino zahtevanega vložka in količino opravljenega dela ter izhajam iz osnovne ravnovesne enačbe:

$$Z_v = O_d \tag{32}$$

Pri čemer Z_v predstavlja zahtevan vložek dela in O_d opravljeno delo. Enačaj v enačbi velja v optimalnem primeru. V splošnem velja neenačaj. Ko je opravljeno delo O_d manjše od zahtevanega vložka dela Z_d, delo ne bo v celoti opravljeno in bo razpoložljivost manjša od 1, ko pa bo O_d večji od Z_d, bodo viri neizkoriščeni.

Na podlagi ITIL priporočil, je v okviru zagotavljanja obratovanja potrebno izvajati več različnih vrst del. Le obseg nekaterih od teh del pa je odvisen od granularnosti v SOA arhitekturi. Ta dela so:

- kategorizacija in filtriranje dogodkov za pravilno odločanje,
- upravljanje življenjskega cikla vseh incidentov,
- upravljanje življenjskega cikla vseh težav,
- proaktivno preprečevanje težav,
- spremljanje in nadzor IT storitev in podporne infrastrukture ter
- upravljanje aplikacij v njihovem življenjskem ciklu.

Za vzpostavitev modela, bom zahtevan vložek dela izrazil kot delež polnega delovnega časa zaposlenega z ustreznimi znanji in spretnostmi. Znanja in spretnosti, pa v nadaljevanju uporabim za določanje cene dela. Dobljene rezultate uravnotežim z zahtevano razpoložljivostjo in upoštevam faktor avtomatizacije.

Kategorizacija in filtriranje dogodkov za pravilno odločanje je povezana s številom povezav in intenzivnostjo interakcij na teh povezavah. Intenzivnost interakcij pa je odvisna od števila proženj posameznih poslovnih procesov in funkcij. Zaradi velikega števila dogodkov v večjih IS, kategorizacijo in filtriranje dogodkov izvajamo avtomatizirano z nadzornim sistemom. Zato v tem delu stroška dela iz naslova kategorizacije in filtriranja sporočil ne upoštevam.

Upravljanje življenjskega cikla vseh incidentov zajema delo povezano z incidenti, ki so povezani z zrelostjo posameznega gradnika v SOA arhitekturi in intenzivnostjo interakcij. Več interakcij prinaša več incidentov, manj interakcij prinaša manj incidentov. Količina incidentov je povezana tudi z usposobljenostjo poslovnih uporabnikov sistema. Slabše, kot so usposobljeni, večje je število incidentov in obratno.

Po ITIL priporočilih incidenti izvirajo iz vseh stikov storitvenega centra (angl. *Service Desk*) s končnimi uporabniki. Enostavne incidente in incidente, ki so že bili uspešno obdelani v procesu upravljanja težav, rešijo bodisi člani storitvenega centra ali operativni izvajalci brez dodatne pomoči. V upravljanje incidentov sodi tudi eskalacija na drugi nivo, če incident preide v težavo, ki je na prvem nivoju ni mogoče rešiti. Za poenostavitev modela uporabim povprečni delež delovnega časa enega zaposlenega dnevno za upravljanje življenjskega cikla posameznega incidenta in ga poimenujem Z_{1inc}. Po mojih izkušnjah je povprečno porabljen čas za posamezni incident Z_{1inc} od 15 do 30 minut, kar pri 8-urnem delovniku predstavlja 1/32 do 1/16 oziroma od 3,13% do 6,25% delovnega časa, če ne upoštevam čas za malico in druge odsotnosti zaposlenega.

Vložek potreben za upravljanje življenjskega cikla vseh incidentov Z_{inc} je seštevek vložkov za vse incidente, katerega sestavljajo čas, potreben za upravljanje incidenta Z_{1inc}, povprečne verjetnosti incidenta p_{inc}, števila poslovnih procesov in funkcij P, števila elementarnih funkcionalnosti m, intenzivnosti uporabe I_i ter granularnosti sistema G:

$$\begin{aligned} Z_{inc} &= \sum_{i=1}^{n} Z_{1inc}\ p_{inc}\ I_i \left(5+25\sqrt{\frac{A_i}{15}}\right)\left(1-\frac{4G}{5}\right) \\ &= Z_{1inc}\ p_{inc} \sum_{i=1}^{Gm+(1-G)P} I_i \left(5+25\sqrt{\frac{A_i}{15}}\right)\left(1-\frac{4G}{5}\right) \end{aligned} \tag{33}$$

Za nov sistem ($A = 0$) enačbo poenostavim:

$$Z_{inc} = Z_{1inc}\ p_{inc}\ (5-4G) \sum_{i=1}^{Gm+(1-G)P} I_i \tag{34}$$

Pri tem čas, potreben za upravljanje incidenta, določimo kot povprečje časa iz preteklih izkušenj. Intenzivnost uporabe posamezne storitve je odvisna od intenzivnosti uporabe posameznega poslovnega procesa in funkcije ter granularnosti sistema. Za opredelitev intenzivnosti uporabe posamezne storitve potrebujem še število storitev, ki sestavljajo posamezni poslovni proces oziroma funkcijo P. Število storitev posameznega procesa oziroma funkcije označim z n_p:

$$n_p = 1\ (1-G) + G\frac{m}{P} = 1 - G + G\frac{m}{P} \tag{35}$$

Enačba izhaja iz izenačitve števila storitev v posameznem poslovnem procesu oziroma funkciji z 1 pri najbolj grobi granularnosti ($G = 0$), pri najbolj drobni granularnosti ($G = 1$) pa se omejim na primer, ko so storitve enakomerno porazdeljene med vse poslovne procese oziroma funkcije. Intenzivnost uporabe storitve i označim z I_i in uporabim razmerje med procesi in

storitvami:

$$I_i = \frac{I_p}{n_p} = \frac{I_p}{(1-G)+G\frac{m}{P}} = \frac{I_p}{1-G+G\frac{m}{P}} \tag{36}$$

I_i je v tem primeru povprečje in je enako za vse storitve enega procesa. Sedaj lahko potreben vložek opišem neposredno v odvisnosti od intenzivnosti uporabe poslovnih procesov in funkcij:

$$Z_{inc} = \frac{Z_{1inc}\ p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}} \sum_{i=1}^{P} I_{p_i} \tag{37}$$

Pri tem čas Z_{inc} predstavlja čas vseh incidentov, tudi tistih, ki preidejo v težave. Ni pa zajet čas upravljanja samih težav.

Za izračun rezultata mora arhitekt ali snovalec izpolniti tabelo 4 v Prilogi 7. Pri novem sistemu je verjetnost nastanka incidenta večja, poznavanje in spretnosti pa nizke. Priporočeni vrednosti za nov sistem sta:

$Z_{1inc} = 1/16$

$p_{inc} = 0{,}05‰$

Poleg samega časa, potrebnega za upravljanje incidentov, so pomembna še potrebna znanja in spretnosti:

- poznavanje sistema za upravljanje incidentov,
- poznavanje baze znanja in
- komunikacijske sposobnosti.

Upravljanje incidentov je ročno delo, zato bom končnemu rezultatu prištel celoten čas, potreben za upravljanje incidentov, brez faktorja avtomatizacije F_A.

Upravljanje življenjskega cikla vseh težav je povezano s številom incidentov in usposobljenostjo zaposlenih za reševanje težav. Težave so podmnožica incidentov z drugačnim procesom reševanja. Težave imajo znatno daljši čas reševanja in zahtevajo večji nabor potrebnih znanj in spretnosti. Delež incidentov, ki postanejo težava je odvisna tudi od zrelosti oziroma starosti sistema. Za nov sistem je čas, potreben za upravljanje vseh težav:

$$Z_{prob} = d_{prob}\ Z_{1prob}\ n_{inc} \tag{38}$$

Kjer je Z_{prob} čas potreben za upravljanje vseh težav, d_{prob} predstavlja delež incidentov, ki preidejo v težavo, Z_{1prob} je povprečen čas, potreben za upravljanje posamezne težave in n_{inc} število vseh incidentov. n_{inc} predstavlja število incidentov, katerega lahko izrazim iz (37), če čas, potreben za

vse incidente Z_{inc} delim s časom, potrebnim za upravljanje enega incidenta Z_{1inc}:

$$n_{inc} = \frac{p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}}\ \sum_{i=1}^{P} I_{p_i} \tag{39}$$

Rezultat sedaj lahko vstavim v enačbo (38) in dobim poenostavljeno enačbo potrebnega časa za upravljanje vseh težav v novem sistemu:

$$Z_{prob} = d_{prob}\ Z_{1prob}\ \frac{p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}}\ \sum_{i=1}^{P} I_{p_i} \tag{40}$$

Oceniti moram še čas, potreben za upravljanje posamezne težave in delež števila težav glede na število vseh incidentov. Tako verjetnost nastanka težave, kot čas za reševanje posamezne težave, sta odvisna od konkretnega okolja, kompleksnosti celotnega IS in usposobljenosti zaposlenih. V tabeli 4 v Prilogi 7 navajam izkustvene vrednosti za področji BSS in OSS v Telekomu. Arhitekt ali snovalec, pa mora tabelo izpolniti na podlagi svojih izkušenj. Na osnovi istih izkušenj traja stabilizacijsko obdobje s povišanim številom incidentov in težav 2 do 3 mesece po prehodu novega sistema v produkcijo. Po tem času število incidentov in težav začne strmo padati zaradi odprave vzrokov, pridobljenega znanja kadrov, ki rešujejo incidente in težave in pridobljenega znanja končnih uporabnikov. To kaže, da je potreben vložek v upravljanje incidentov in težav funkcija časa. Ker pa moramo ob uvedbi novega sistema poskrbeti za zadostne kapacitete za reševanje incidentov in težav, v tem delu upoštevam samo prvo obdobje po prehodu v produkcijsko uporabo.

Poleg samega časa, potrebnega za upravljanje težav, so pomembna še potrebna znanja in spretnosti:

- podrobno poznavanje poslovnih procesov,
- podrobno poznavanje storitev,
- poznavanje baze znanja,
- podrobno poznavanje različnih gradnikov SOA arhitekture in
- tehnična pismenost za vzdrževanje baze znanja.

Upravljanje težav je ročen proces, zato bom končnemu rezultatu prištel celoten čas, potreben za upravljanje težav, brez faktorja avtomatizacije F_A.

Proaktivno preprečevanje težav je povezano z izvajanjem periodičnih vzdrževalnih del na storitvah in infrastrukturi. Ni povezano s številom povezav med gradniki in intenzivnostjo uporabe. Od granularnosti je odvisen posredno, skozi število storitev. Zajema redne preglede dnevniških zapisov storitev, čiščenje in dodajanje diskovnega prostora, prostora za zbirko

podatkov, prenose v podatkovno skladišče, prenose v arhiv, šolanje kadrov, ki izvajajo proces upravljanja incidentov in težav in usposabljanje končnih uporabnikov. Čas potreben za izvajanje proaktivnega preprečevanja težav označim z Z_{prev} in ga zapišem kot:

$$Z_{prev} = \sum_{i=1}^{n} (\phi_i \, Z_{prev_i}) \tag{41}$$

Pri tem je ϕ_i pogostost izvajanja aktivnosti za preprečevanje težav na storitvi i in Z_{prev_i} potreben čas za izvedbo aktivnosti na storitvi i. V enačbi (41) izrazim še n v odvisnosti od granularnosti:

$$Z_{prev} = \sum_{i=1}^{Gm+(1-G)P} (\phi_i \, Z_{prev_i}) \tag{42}$$

V kolikor aktivnosti izvajamo z enako periodo in izračunamo povprečen čas, potreben za izvajanje posamezne aktivnosti, lahko enačbo še poenostavim:

$$Z_{prev} = \phi \, Z_{1prev} \sum_{i=1}^{Gm+(1-G)P} 1 = (Gm + (1-G)P) \, \left(\phi \, Z_{1prev}\right) \tag{43}$$

Pri tem označujem povprečno periodo izvajanja posamezne preventivne aktivnosti s ϕ in povprečen čas, potreben za izvedbo posamezne aktivnosti z Z_{1prev}. Ob uporabi te enačbe, mora arhitekt ali snovalec izpolniti tabelo 4 za izračun potrebnega časa za proaktivno preprečevanje težav.

Arhitektu ali snovalcu prepuščam izbiro enačbe, glede na razpoložljive podatke. Pogosto je mogoče oceniti celoten potreben čas za preventivne aktivnosti kot celoto in uporaba enačbe ni potrebna. Za potrebe tega dela in iskanje optimalne granularnosti pa uporabljam enačbo (43).

Poleg samega časa, potrebnega za proaktivno preprečevanje težav, so pomembna še potrebna znanja in spretnosti:

- poznavanje storitev,
- poznavanje infrastrukture in
- poznavanje različnih gradnikov SOA arhitekture.

Proaktivno preprečevanje težav je lahko delno avtomatiziran proces. Po mojih izkušnjah je mogoče avtomatizirati 1/4 aktivnosti, zato bom končnemu rezultatu prištel 3/4 celotnega časa, potrebnega za proaktivno preprečevanje težav.

Spremljanje in nadzor IT storitev in podporne infrastrukture je tesno povezano s številom

poslovnih procesov, intenzivnostjo interakcij, številom storitev in drugih gradnikov. Spremljanje in nadzor obratovanja je težko opisati glede na posamezni element. Lažje je oceniti, koliko elementov lahko nadzira en zaposleni. Iz lastnih izkušenj, lahko en zaposleni izvaja spremljanje in nadzor na do 15 storitev. Iz te ocene izrazim čas, potreben za spremljanje in nadzor posamezne storitve $Z_{1nadz} = 1/15$. Spremljanje in nadzor pa nista potrebna samo nad storitvami, temveč tudi nad elementarnimi funkcionalnostmi (komponentami). Zato je celoten potreben čas za spremljanje in nadzor Z_{nadz}:

$$\begin{aligned} Z_{nadz} &= Z_{1nadz}(n+m) \\ &= Z_{1nadz}(Gm+(1-G)P+m) \\ &= Z_{1nadz}((G+1)m+(1-G)P) \end{aligned} \tag{44}$$

Ob upoštevanju ocene za Z_{1nadz}, je potreben čas za spremljanje in nadzor:

$$Z_{nadz} = \frac{1}{15}((G+1)m+(1-G)P) \tag{45}$$

Poleg samega časa, potrebnega za spremljanje in nadzor, so pomembna še potrebna znanja in spretnosti:

- poznavanje poslovnih procesov,
- poznavanje storitev,
- poznavanje infrastrukture in
- poznavanje različnih gradnikov SOA arhitekture.

Spremljanje in nadzor sta lahko avtomatizirana, zato bom končnemu rezultatu prištel čas, potreben za spremljanje in nadzor, pomnožen s faktorjem avtomatizacije F_A.

Upravljanje aplikacij v njihovem življenjskem ciklu izhaja iz klasičnega pogleda na IT arhitekturo. V SOA arhitekturi je aplikaciji enakovreden pojem **storitev**. Med upravljanje storitev v njihovem življenjskem ciklu spada nameščanje novih različic in popravkov, konfiguracija, morebitna konverzija podatkov ter usklajevanje namestitev. Obseg in posledično čas trajanja je, po mojih izkušnjah, od 30 minut do več ur s povprečjem okoli ure in pol, pogostost aktivnosti pa je odvisna od uporabljene metodologije za razvoj popravkov in novih funkcionalnosti. Močno orkestrirane storitve zahtevajo manj sprememb programske kode in več konfiguracije ter obratno.

Ob predpostavki, da je mogoče izračunati ali oceniti povprečen čas, potreben za vsako aktivnost,

je celoten čas, potreben za upravljanje storitev Z_{mng}:

$$Z_{mng} = \sum_{i=1}^{n} \left(\gamma_i \, Z_{1mng}\right) = Z_{1mng} \sum_{i=1}^{n} \gamma_i$$
$$Z_{mng} = Z_{1mng} \sum_{i=1}^{Gm+(1-G)P} \gamma_i \tag{46}$$

Pri tem γ_i predstavlja pogostost izvajanja posega i v običajnem delovnem času 8 ur na dan. Posege običajno izvajamo ob istem času, tedaj je pogostost izvajanja posegov γ enaka za vse vrste posegov in predstavlja povprečje. Ob oceni časa za en poseg $Z_{1mng} = \frac{1}{4}$, kar predstavlja 2 uri, in ob običajni pogostosti aktivnosti enkrat vsakih 14 dni ($\gamma = \frac{1}{10}$), enačbo poenostavim v:

$$Z_{mng} = \frac{1}{40} \sum_{i=1}^{Gm+(1-G)P} 1$$
$$= \frac{1}{40} \left(Gm + (1-G)P\right) \tag{47}$$

Arhitekt ali snovalec lahko izbereta enačbo (46) in na podlagi podatkov o pogostosti izvajanja aktivnosti upravljanja storitev, izračunata potreben čas za upravljanje storitev v njihovem življenjskem ciklu.

Poleg samega časa, potrebnega za upravljanje storitev, so pomembna še potrebna znanja in spretnosti:

- poznavanje storitev,
- poznavanje infrastrukture in
- poznavanje različnih gradnikov SOA arhitekture.

Upravljanje storitev je ročen proces, zato bom končnemu rezultatu prištel celoten čas, potreben za upravljanje storitev, brez faktorja avtomatizacije F_A.

Skupen obseg dela odvisen od granularnosti je seštevek:

$$Z'_v = Z_{inc} + Z_{prob} + \frac{3}{4} Z_{prev} + (1 - F_A) \; Z_{nadz} + Z_{mng} \tag{48}$$

Žal pa tak seštevek ne upošteva različnih cen dela. Zato obsega posameznih del ne smem sešteti. Seštejem lahko stroške posameznih del in nato poiščem minimum z odvajanjem po faktorju

granularnosti:

$$S_d = C_{inc}\ Z_{inc} + C_{prob}\ Z_{prob} + C_{prev}\ \frac{3}{4} Z_{prev} + C_{nadz}\ (1 - F_A)\ Z_{nadz} + C_{mng}\ Z_{mng} \tag{49}$$

Pri tem je C_{inc} cena dela za upravljanje incidentov, C_{prob} je cena dela za upravljanje težav, C_{prev} je cena dela proaktivnega preprečevanja težav, C_{nadz} je cena dela spremljanja in nadzora ter C_{mng} je cena dela za upravljanje storitev. Vse cene so izražene kot znesek mesečnega plačila z vsemi dodatnimi stroški delodajalca. Glede na to, da so vsi obsegi opredeljeni kot deleži časa polnega delovnika pri predpostavki 40-urnega delovnega tedna, brez upoštevanja časa za odmor, dopusta, praznikov in drugih odsotnosti, je v tem trenutku še nepomembno, ali gre za ceno na mesec, leto ali teden. Zaradi primerljivosti pa je smiselno uporabljati ceno na leto.

Enačba (49) ima še eno pomanjkljivost: predpostavlja enako razpoložljivost IS, kot je delovni urnik zaposlenih. Ta omejitev se mi zdi prestroga, zato zahtevano razpoložljivost vključim v strošek dela:

$$\begin{aligned} S_d = {} & C_{inc}\ Rf_{inc}\ Z_{inc} + C_{prob}\ Rf_{prob}\ Z_{prob} + C_{prev}\ \frac{3}{4} Z_{prev} + \\ & + C_{nadz}\ Rf_{nadz} (1 - F_A)\ Z_{nadz} + C_{mng}\ Rf_{mng}\ Z_{mng} \end{aligned} \tag{50}$$

Pomen novo uvedenih faktorjev je: Rf_{inc} predstavlja faktor razpoložljivosti upravljanja z incidenti, Rf_{prob} predstavlja faktor razpoložljivosti upravljanja s težavami, Rf_{nadz} predstavlja faktor razpoložljivosti spremljanja in nadzora ter Rf_{mng}, ki predstavlja faktor razpoložljivosti upravljanja storitev. Strošek dela povezan z izvajanjem preventivnih akcij, ni povezan s faktorjem zahtevane razpoložljivosti.

Takšna opredelitev pa ni preveč smiselna, saj le redko srečamo različne zahteve za razpoložljivost znotraj istega IS. Zato različne faktorje razpoložljivosti nadomestim z enim samim:

$$S_d = R_f \left(C_{inc}\ Z_{inc} + C_{prob}\ Z_{prob} + C_{nadz}\ (1 - F_A)\ Z_{nadz} + C_{mng}\ Z_{mng} \right) + C_{prev}\ \frac{3}{4} Z_{prev} \tag{51}$$

Pri tem je R_f faktor razpoložljivosti celotnega IS izražen kot delež delovnega časa. Če je razpoložljivost zahtevana samo med delovnim časom, je R_f enak 1, če je zahtevana razpoložljivost 16 ur na dan ob delovnikih, je R_f enak 2. V splošnem je:

$$R_f = \frac{R\,[ur]}{40} \tag{52}$$

Pri tem $R\,[ur]$ predstavlja zahtevano tedensko razpoložljivost v urah v števcu in predpostavlja 40 urni delovni teden v imenovalcu. Ob omejitvi dolžine tedna, je največja vrednost R_f lahko

$\frac{7\ 24}{40} = 3\frac{34}{40}$ in predstavlja stalno razpoložljivost 24 x 7 x 365, hkrati pa pove, da za takšno razpoložljivost potrebujemo $3\frac{34}{40}$ FTE v primerjavi z razpoložljivostjo samo med delovnim časom, kjer je $R_f = \frac{40}{40} = 1$.

Kompleksnost in obseg potrebnih znanj vplivajo na ceno dela. V nekaterih primerih, je število poslovnih procesov in funkcij, kot tudi orkestriranih storitev in elementarnih funkcionalnosti preveč, da bi jih lahko obvladovala ena oseba. Kje je meja pri številu poslovnih procesov in funkcij, storitev in funkcionalnosti, je lahko predmet samostojnega dela. Končno ceno določa trg dela in trenutne razmere na njem.

Arhitekt in snovalec morata upoštevati vpliv kompleksnosti pri določanju cene dela v tabeli 4 na strani 31 med prilogami.

7.5 Skupni stroški obratovanja

Za skupen strošek obratovanja, seštejem strošek dela in strošek licenčnin:

$$
\begin{aligned}
S &= S_d + S_l \\
&= R_f \ (C_{inc} \ \frac{Z_{1inc}\ p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}} \ \sum_{i=1}^{P} I_{p_i} + \\
&+ C_{prob}\ d_{prob}\ Z_{1prob}\ \frac{p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}} \ \sum_{i=1}^{P} I_{p_i} + \\
&+ C_{nadz}\ (1-F_A)\, Z_{1nadz}\,((G+1)\,m+(1-G)\,P) + \\
&+ C_{mng}\ \frac{1}{40}\,(Gm+(1-G)P)) + \\
&+ C_{prev}\ \frac{3}{4}\,(Gm+(1-G)P)\ \left(\phi\ Z_{1prev}\right) + \\
&+ C_l\left((4P-4m)\,G^2+(5m-9P)\,G+5P\right)F_A + S_h
\end{aligned}
\tag{53}
$$

Za določitev optimalne granularnosti novega sistema, enačbo (53) odvajam in poiščem ničle. Zaradi odvajanja ne potrebujem podatka o strošku heterogenosti, ki je v tem modelu neodvisen od granularnosti. Odvod in izpeljava sta v Prilogi 8. V istem poglavju priloge izračunam še drugi odvod, ki z vstavitvijo vrednosti ničel pove, ali je dana točka minimum, maksimum ali nič od tega. V odvisnosti od konkretnih vrednosti vhodnih parametrov, je mogoče, da minimum stroškov v odvisnosti od granularnosti ne obstaja znotraj intervala $(0,1)$. V tem primeru je potrebno izračunati še vrednosti na robovih intervala pri $G = 0$ in $G = 1$.

Rezultat modela je optimalna stopnja granularnosti G za minimiziranje stroškov obratovanja. Že absolutna vrednost G približno pokaže, kako podrobno je potrebno razdeliti funkcionalnosti

na storitve. Po nekajkratni uporabi modela, bosta arhitekt in snovalec že iz absolutne številke videla, v katero smer ju vodi model. Ker pa sem granularnost opredelil v enačbi (15), ki izhaja iz enačbe števila storitev odvisnih od granularnosti (14), je iz slednje mogoče neposredno izračunati optimalno število storitev v SOA arhitekturi. Pri tem bo za dejansko uporabo potrebno število storitev n, ki bo v splošnem realno število, zaokrožiti na celo število. To pa je že zelo konkreten rezultat modela.

7.6 Izboljšave modela

Model bi bilo mogoče še izpopolniti z upoštevanjem in podrobno razgradnjo dodatnih vplivov.

Pri oceni števila povezav in interakcij v modelu ni upoštevana zrelost SOA arhitekture. Zrelost SOA arhitekture je odvisna od uporabe dodatnih mehanizmov, ki zahtevajo dodatne povezave storitev:

- dodatna povezava za iskanje končne točke storitve v imeniku storitev,
- dodatna povezava za iskanje ustrezne storitve v repozitoriju storitev,
- dodatna povezava do sistema varnosti (npr. WS-Security) in
- dodatna povezava za zagotavljanje transakcijske celovitosti orkestriranih storitev (npr. WS-Transaction).

Dodatne povezave naštete zgoraj, zahtevajo razdelitev storitev med orkestrirane in nižjenivojske storitve, kar bi močno povečalo kompleksnost modela.

Model bi bilo mogoče izboljšati tudi z izpopolnitvijo vpliva heterogenosti uporabljenih tehnologij. Ni namreč vseeno, ali uporabljamo npr. samo SOAP spletne storitve, ali kombinacijo SOAP in REST, prav tako ni vseeno, ali so storitve implementirane z .NET, Java, ali kombinacijo obeh. Ni tudi vseeno, ali so storitve implementirane na eni vrsti aplikacijskih strežnikov, ali na več različnih vrstah. Večja heterogenost zahteva zaposlene s širšim naborom znanj, ki so praviloma dražji.

Na stroške dela vplivajo še odmor za malico, dopusti, bolezni in prazniki. Absentizem zaposlenih bi bilo mogoče upoštevati z uporabo statističnih podatkov, ki so na voljo preko javnega portala Statističnega urada RS. V tem delu zaradi poenostavitve, nisem upošteval vpliv nadurnega dela in dežurstev, ki prav tako vplivajo na strošek dela. Razvil bi lahko ločen model vpliva kompleksnosti storitev, gradnikov in pogodb na stroške dela.

Izpopolniti je mogoče model za uporabo pri spremembah obstoječega IS ter uvesti faktorje poslovne panoge, v katerem deluje podjetje.

Vse naštete izboljšave presegajo obseg tega dela, zato jih v tem delu ne obravnavam.

7.7 Uporaba modela na primeru Telekoma Slovenije

V Telekomu Slovenije smo začeli z uvedbo SOA pristopa leta 2001. Sprva so bili to skromni začetki s postavitvijo prvih gradnikov SOA arhitekture in z začetkom izdelave prvih spletnih storitev za novo rešitev „komponenta Customer". Že leta 2002 smo začeli uporabljati obstoječe gradnike, tudi za integracijo starejših sistemov in pri tem kombinirali uporabo sodobnega komponentnega pristopa s tehnikami EAI (Mesarič Kunst, 2005). V letih 2007 in 2008 smo pripravili in sprejeli serijo internih navodil za dokumentiranje in vzpostavitev celovite SOA arhitekture (Heričko & Terbuc, 2007c; Heričko & Terbuc, 2007b; Heričko & Terbuc, 2007d; Heričko & Terbuc, 2007a; Heričko & Terbuc, 2008).

V času pisanja tega dela poteka obširna prenova BSS področja informacijske podpore, s katero bomo konsolidirali in posodobili sisteme, ki so še danes podvojeni po združitvi podjetij Telekom Slovenije in Mobitel. V nadaljevanju pa pripravljamo prenovo na OSS področju. Pri prenovi so naša vodila: uporaba storitvene usmerjenosti in s tem SOA arhitekture na višjem zrelostnem nivoju, kot do sedaj, optimizacija podpore IT storitvam na podlagi ITIL priporočil ter uporaba Frameworx ogrodja, predvsem uporaba standardiziranega podatkovnega modela temelječega na SID, avtomatizacija poslovnih procesov razdeljenih po eTOM ter razdelitev pristojnosti na podlagi TAM.

Za primer uporabe modela v Telekomu Slovenije sem izbral področje OSS, katerega najbolje poznam. Na podlagi eTOM in TAM v področje OSS v grobem spadajo:

- inventar,
- zagotavljanje storitev,
- snovanje storitev in omrežja,
- aktivacija,
- upravljanje terenske delovne sile in
- upravljanje nivoja kakovosti storitev.

V Telekomu področje inventarja upravljamo z dvema ločenima a povezanima rešitvama: fizični inventarni sistem, ki je vpet v GIS platformo in logični inventarni sistem, ki je vpet v poslovne procese. Vsa ostala področja so pokrita vsak s svojim sistemom. Z izjemo snovanja, ki je povsem lastna rešitev, ter aktivacije, ki je delno pokrita z lastno rešitvijo in delno s kupljenim sistemom, so vse ostale rešitve kupljene in z lastnim razvojem integrirane v SOA arhitekturo.

Dobro definirana meja OSS področja, ki jo zagotavlja Frameworx ogrodje, mi omogoča izolirano

obravnavo področja, pri čemer je infrastrukturni del SOA arhitekture deljen z ostalimi deli IS podjetja.

V tabeli 5 v Prilogi 7 sem popisal poslovne procese in funkcije OSS področja ter v tabeli 6 število elementarnih funkcionalnosti po področjih. Z uporabo razpoložljivih podatkov o uporabi poslovnih procesov in funkcij ter ocen cene dela izhajajoč iz internih podatkov, sem izpolnil tabelo 7, potrebno za uporabo modela iz poglavja 7. Zaradi varovanja poslovne skrivnosti Telekoma Slovenije, so podatki pomnoženi s skritim faktorjem. Zbrane podatke vstavim v enačbe (71), (72), (73), (74) in (75), iz katerih dobim:

$$f'(G) = G^3\ (-0,132111) + G^2\ 52116191,45 + G\ 4051414,534 + 32446,66047 \tag{54}$$

Z uporabo orodij za iskanje ničel polinoma tretje stopnje dobim potencialne ničle:

- $G_1 = -0,0682342$;
- $G_2 = 0,05545$;
- $G_3 = 12,339$.

Potencialna ekstrema G_1 in G_3 sta izven definicijskega območja granularnosti. Preostane mi samo še G_2. Zanj izračunam vrednost drugega odvoda s pomočjo enačbe (77) in dobim:

$$f''(0,05545) = 1658569,543 \tag{55}$$

Ker je vrednost drugega odvoda pozitivna, pomeni, da sem našel lokalni minimum in ker je hkrati to edini ekstrem v definicijskem območju granularnosti G, je to tudi globalni minimum.

Posledično to pomeni, da je priporočljivo število storitev v SOA arhitekturi za področje OSS v Telekomu Slovenije, izračunano po enačbi (14) in z uporabo podatkov iz tabele 7:

$$\begin{aligned} n &= G\,m + (1-G)\ P \\ &= 0,05545 * 1847 + 0,94455 * 69 \\ n &= 167,5901 \end{aligned} \tag{56}$$

Za določitev optimalnega števila storitev, moram rezultat še zaokrožiti: $n = 168$, kar je tudi rezultat tega dela in moje priporočilo podjetju.

Večkratni poizkusi z različnimi vrednostmi so pokazali, da je edini minimum vselej zelo blizu

ničelne točke, ki predstavlja grobo granularnost.

SKLEP

Čeprav granularnost storitev v SOA arhitekturi predstavlja relativno majhen delež stroškov IS, ta del stroškov ni nepomemben, je pa skoraj v celoti prezrt. Prikazan model skupaj z ASK metodologijo dokazuje vpliv granularnosti na stroške obratovanja, uporaben pa je tudi v zgodnejših fazah analize in snovanja IS.

Vzpostavitev SOA je dolgotrajen in zahteven proces, ki zahteva spremembo načina razmišljanja in dela v celotnem podjetju. Veliko projektov je bilo tudi neuspešnih, pri čemer je bila stopnja granularnosti identificirana kot eden glavnih vzrokov za neuspeh. Pričujoč model omogoča vnaprejšnjo oceno vpliva granularnosti na stroške obratovanja in s tem zmanjšuje tveganje za SOA projekte. Hkrati pa lahko služi kot podlaga za raziskavo vpliva granularnosti na stroške uvedbe SOA arhitekture.

Pričujoč model je dober pripomoček pri snovanju novega storitveno usmerjenega okolja, hkrati pa praktični primeri uporabe modela kažejo, da so imeli praktiki in avtorji strokovnih prispevkov na temo granularnosti v veliki meri prav, ko so verjeli in trdili, da je optimalna granularnost groba granularnost.

LITERATURA IN VIRI

1. Arsanjani, A. (2002). Developing and Integrating Enterprise Components and Services. *Communications of the ACM*, *45*(10), 30–34.
2. Arsanjani, A., & Holley, K. (2006). The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA. V *2006 IEEE International Conference on Services Computing (SCC'06)* (str. 515–515). Washington: IEEE Computer Society.
3. Astrova, I., Koschel, A., & Kruessmann, T. (2010). Comparison of Enterprise Service Buses Based on Their Support of High Availability. V *Proceedings of the 2010 ACM Symposium on Applied Computing* (str. 2495–2496). SAC '10. Sierre, Switzerland: Association for Computing Machinery.
4. Bahli, B., & Ji, F. (2007). An assessment of facilitators and inhibitors for the adoption of enterprise application integration technology: An empirical study. *Business Process Management Journal*, *13*(1), 108–120.
5. Bakken, D. E. (2003). *Middleware*. Najdeno 25. februarja 2016 na spletnem naslovu http://www.eecs.wsu.edu/~bakken/middleware.pdf
6. Beck, K., & Cunningham, W. (1987). *Using Pattern Languages for Object-Oriented Programs*. Orlando: Association for Computing Machinery.
7. Bell, M. (2008). *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Hoboken: John Wiley & Sons.
8. Box, D. (2001). *A Brief History of SOAP*. Najdeno 15. aprila 2016 na spletnem naslovu http://www.xml.com/pub/a/ws/2001/04/04/soap.html
9. Braithwaite, T. (2002). *Securing E-Business Systems: A Guide for Managers and Executives*. New York: John Wiley & Sons.
10. Brewster, M., Hinderliter, A., Dahlhoff, D., McIlheney, C., Moore, C.-L., Shufflebottom, C., ... & Weseman, K. (2016 februar). *Global Total Retail Survey*. Najdeno 6. marca 2016 na spletnem naslovu https://www.pwc.com/totalretail
11. Britton, C., & Bye, P. (2004). *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems* (Second Edition). Boston: Pearson Addison Wesley.
12. Brooks, Jr., F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Reading: Addison-Wesley.
13. Bruneo, D., Puliafito, A., & Scarpa, M. (2007). *The Handbook of Mobile Middleware*. Boca Raton: Auerbach Publications.

14. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (2000). *Pattern-Oriented Software Architecture: A System of Patterns: Volume 1* (Kindle Edition). Chichester: John Wiley & Sons.
15. Clarke, S., & Baniassad, E. (2005). *Aspect-Oriented Analysis and Design*. Upper Saddle River: Addison-Wesley Professional.
16. Coplien, J. O. (1997). Idioms and Patterns as Architectural Literature. *IEEE Software*, *14*(1), 36–42.
17. Cotton, J. L., & Tuttle, J. M. (1986). Employee Turnover: A Meta-Analysis and Review with Implications for Research. *Academy of Management Review*, *11*(1), 55–70. Najdeno 19. junija 2016 na spletnem naslovu http://www.jstor.org/stable/258331
18. Cowan, D. F. (2005). Total Cost of Ownership. V D. F. Cowan (Ur.), *Informatics for the Clinical Laboratory* (str. 87–97). Health Informatics. New York: Springer Science+Business Media.
19. DAML Services - Coalition. (2008). *OWL-S 1.2 Release*. Najdeno 3. maja 2016 na spletnem naslovu http://www.daml.org/services/owl-s/
20. Davis, J. (2009). *Open Source SOA* (1st). Greenwich: Manning Publications Co.
21. Derler, P., & Weinreich, R. (2006). Models and Tools for SOA Governance. V *Trends in Enterprise Application Architecture* (str. 112–126). Berlin: Springer.
22. *Distributed TP: The XA Specification*. (1991). Reading: X/Open Company Ltd.
23. *Distributed TP: The XA+ Specification* (Version 2). (1994). Reading: The Open Group.
24. *Distributed Transaction Processing: Reference Model* (Version 3). (1996). Reading: X/Open Company Ltd.
25. *Distributed Transaction Processing: The TX (Transaction Demarcation) Specification*. (1995). Reading: X/Open Company Ltd.
26. Drucker, P. F. (2007). *The Practice of Management* (2nd Revised Edition). Oxford: Butterworth-Heinemann.
27. EAI. (2015). V *iSlovarju*. Najdeno 9. junija 2015 na spletni strani http://www.islovar.org
28. Emmerich, W., Ellmer, E., & Fieglein, H. (2001). TIGRA - An Architectural Style for Enterprise Application Integration. V *Proceedings of the 23rd International Conference on Software Engineering* (str. 567–576). Toronto: IEEE Computer Society Press.
29. Erl, T. (2005). *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. New Jersey: Prentice Hall.
30. Erl, T. (2008). *SOA Principles of Service Design*. New Jersey: Prentice Hall.
31. Erl, T. (2009). *SOA Design Patterns*. New Jersey: Prentice Hall.

32. Erl, T. (2011). The Four Pillars of Service-Orientation. V *SOA in Healthcare 2011*. Herndon: OMG. Najdeno 14. novembra 2015 na spletnem naslovu http://www.omg.org/news/meetings/workshops/SOA-HC/presentations-2011/13_K1_Erl.pdf
33. Erl, T., Carlyle, B., Pautasso, C., & Balasubramanian, R. (2013). *SOA with REST: Principles, Patterns and Constraints for Building Enterprise Solutions with REST* (Kindle Edition). New Jersey: Prentice Hall.
34. ERP RIP? (1999). *The Economist*, *351*(8125), 29–34.
35. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Irvine: University of California.
36. Florjančič, J., Ferjan, M., & Bernik, M. (1999). *Planiranje in razvoj kadrov*. Kranj: Moderna organizacija.
37. Friedman, M. (2009). *Capitalism and Freedom: Fortieth Anniversary Edition*. Chicago: University of Chicago Press.
38. Gacek, C., Abd-Allah, A., Clark, B., & Boehm, B. (1995). On the definition of software system architecture. V *Proceedings of the First International Workshop on Architectures for Software Systems* (str. 85–94). Seattle: University of Southern California.
39. Galster, M., Lapre, L., & Avgeriou, P. (2014 januar). SOA in Variability-Intensive Environments: Pitfalls and Best Practices. *IEEE Software*, *31*(1), 77–84.
40. González, L., & Ruggia, R. (2011). Addressing QoS Issues in Service Based Systems Through an Adaptive ESB Infrastructure. V *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing* (4:1–4:7). MW4SOC '11. Lisbon: Association for Computing Machinery.
41. Goodhue, D. L., Wybo, M. D., & Kirsch, L. J. (1992). The Impact of Data Integration on the Costs and Benefits of Information Systems. *MIS Quarterly*, *16*(3), 293–311.
42. Greiner, L. (2007, 7. marec). IT Infrastructure Library (ITIL) Definition and Solutions. *CIO*. Najdeno 15. aprila 2016 na spletnem naslovu http://www.cio.com/article/2439501/infrastructure/it-infrastructure-library--itil--definition-and-solutions.html
43. Groznik, A., Gradišar, M., Indihar Štemberger, M., Jaklič, J., Kovačič, A., Turk, T., ... & Manfreda, A. (2010) Stanje poslovne informatike v Sloveniji – kje smo in kam gremo? V *Uravnotežite naložbe, tveganja in razvoj za uspeh/Zbornik prispevkov DSI 2010 (14.-16. april 2010)* [zgoščenka]. Portorož: Slovensko društvo Informatika.
44. Guevara, J. K. (2016). *2016 Sample IT Budget Output Report*. Stamford: Gartner Group.
45. Guevara, J. K., Hall, L., & Stegman, E. (2012). *IT Key Metrics Data 2013: Executive Summary*. Stamford: Gartner Group.

46. Haesen, R., Snoeck, M., Lemahieu, W., & Poelman, S. (2008). On the Definition of Service Granularity and Its Architectural Impact. *Advanced Information Systems Engineering Lecture Notes in Computer Science*, *5047*(20), 375–389.
47. Handy, C. (2002). What is a Business For? *Harvard Business Review*, 49–55.
48. Hately, A. (2015). *UDDI Business Registry Shutdown and Transition*. Najdeno 30. aprila 2016 na spletnem naslovu https://lists.oasis-open.org/archives/uddi-spec/200512/msg00020.html
49. He, H. (2003). *What Is Service-Oriented Architecture*. Najdeno 15. aprila 2016 na spletnem naslovu http://www.xml.com/pub/a/ws/2003/09/30/soa.html
50. Hérault, C., Thomas, G., & Lalanda, P. (2007). A Distributed Service-Oriented Mediation Tool. V *IEEE International Conference on Services Computing, SCC 2007* (str. 403–409). Los Alamitos: IEEE Computer Society.
51. Heričko, M., & Terbuc, L. (2007a) *Arhitekturni dokument za SOA rešitve* (interno gradivo). Maribor: COT in Telekom Slovenije.
52. Heričko, M., & Terbuc, L. (2007b) *Metodološki pristopi za uveljavljanje storitvenih arhitektur in izvedbo korakov* (interno gradivo). Maribor: COT in Telekom Slovenije.
53. Heričko, M., & Terbuc, L. (2007c) *Priporočila za dokumentiranje SOA arhitekture* (interno gradivo). Maribor: COT in Telekom Slovenije.
54. Heričko, M., & Terbuc, L. (2007d) *Priporočila za implementacijo spletnih storitev* (interno gradivo). Maribor: COT in Telekom Slovenije.
55. Heričko, M., & Terbuc, L. (2008) *Priporočila za dokumentiranje IT arhitekture* (interno gradivo). Maribor: COT in Telekom Slovenije.
56. Hirzalla, M., Cleland-Huang, J., & Arsanjani, A. (2009). A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures. V *International Conference on Service-Oriented Computing* (str. 41–52). Berlin: Springer-Verlag.
57. Hölzle, U. (1994). *Adaptive Optimization for SELF: Reconciling High Performance with Exploratory Programming*. Stanford: Stanford University.
58. Hribar Rajterič, G. (2012). Automation Strategies: Evolving OSS to Increase Productivity. V *OSS/BSS Summit 2012*. Budapest: Ericsson.
59. Hribar Rajterič, G. (2013). Surviving the Storm. V *2nd Annual Optimizing OSS/BSS in Telecom*. Barcelona: Allan Lloyds Group.
60. Hribar Rajterič, G., & Terbuc, L. (2010) Vpeljava celovite rešitve upravljanja naročil telekomunikacijskih storitev. V *Uravnotežite naložbe, tveganja in razvoj za uspeh/Zbornik prispevkov DSI 2010 (14.-16. april 2010)* [zgoščenka]. Portorož: Slovensko društvo Informatika.

61. IEEE. (1998). ISO/IEC 8802-5:1998, Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 5: Token ring access method and physical layer specifications. V *IEEE Std 802.5 1998 Edition (ISO/IEC 8802-5:1998)* (str. 1–256). Piscataway: IEEE Standards Association.
62. *In A Nutshell: A Short History of ITIL*. (2016). Najdeno 15. aprila 2016 na spletnem naslovu http://itsm.fwtk.org/History.htm
63. Jurič, M. B. (2005). Storitvena arhitektura – zgolj kompozicija spletnih storitev? V *Zbornik prispevkov desete konference OTS'2005* (str. 7–12). OTS'2005. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko: Center odličnosti za sodobne informacijske tehnologije in storitve.
64. Jurič, M. B., Heričko, M., & Rozman, I. (2000). Legacy Systems in Distributed Object Architecture. *ACM Software Engineering Notes*, *25*(2), 35–39.
65. Jurič, M. B., Saša, A., Brumen, B., & Rozman, I. (2009). WSDL and UDDI extensions for version support in web services. *The Journal of Systems and Software*, *82*(8), 1326–1343.
66. Krafzig, D., Banke, K., & Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Englewood Cliffs: Prentice Hall.
67. Král, J., & Žemlička, M. (2009). Popular SOA Antipatterns. V *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns* (str. 271–276). Athens, Greece: IEEE Computer Society.
68. Krammer, A., Heinrich, B., Henneberger, M., & Lautenbacher, F. (2011 december). Granularity of Services. *Business & Information Systems Engineering*, *3*(6), 345–358. Najdeno 9. aprila 2016 na spletnem naslovu https://ideas.repec.org/a/spr/binfse/v3y2011i6p345-358.html
69. Kruchten, P. (1995). Architectural Blueprints - The "4+1" View Model of Software Architecture. *IEEE Software*, *12*(6), 42–50.
70. Kulkarni, N., & Dwivedi, V. (2008). The role of service granularity in a successful SOA realization a case study. V *IEEE Congress on Services, Part I* (str. 423–430). Honolulu: IEEE.
71. Lee, J., Siau, K., & Hong, S. (2003 februar). Enterprise Integration with ERP and EAI. *Communications of the ACM*, *46*(2), 54–60.
72. Lehmann, C. (2004). *Business Transparency for Distributed Organizations*. Stamford: META Group.
73. Lewis, G. A., Smith, D. B., & Kontogiannis, K. (2010). A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems. V

Research Showcase. Pittsburgh: Carnegie Mellon University. Najdeno 10. aprila 2016 na spletnem naslovu http://repository.cmu.edu/sei/39/

74. Liles, D. (2000). The Zero Latency Enterprise. V A. El Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, & K. Whang (Ur.), *Proceedings of 26th International Conference on Very Large Data Bases, VLDB 2000, September 10-14, 2000, Cairo, Egypt* (str. 674–676). San Francisco: Morgan Kaufmann Publishers.
75. Linthicum, D. S. (1999). *Enterprise Application Integration* (First Edition). Boston: Addison-Wesley.
76. Longwell, J., & Ratta, A. (2015). *IT Spending and Staffing Benchmarks 2015/2016: Executive Summary* (B. Newton, & M. Longwell, Ur.). Irvine: Computer Economics.
77. Longwell, J., Ratta, A., & Senia, A. (2014). *IT Spending and Staffing Benchmarks 2014/2015: Executive Summary* (B. Newton, & M. Longwell, Ur.). Irvine: Computer Economics.
78. Mahmoud, K. M. (2013). A Unified Messaging-Based Architectural Pattern for Building Scalable Enterprise Service Bus. V *Proceedings of International Conference on Information Integration and Web-based Applications & Services* (str. 697–701). IIWAS '13. Vienna, Austria: Association for Computing Machinery.
79. Maréchaux, J.-L. (2006). *Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus*. Najdeno 23. aprila 2016 na spletnem naslovu http://immagic.com/eLibrary/ARCHIVES/GENERAL/IBM/I060328M.pdf
80. Marx, K. (1967). *Capital: A Critique of Political Economy, Volume I: The Process of Capitalist Production* (English Edition) (F. Engels, Ur.). New York: International Publishers.
81. Al-Masri, E., & Mahmoud, Q. H. (2007). Crawling Multiple UDDI Business Registries. V *Proceedings of the 16th International Conference on World Wide Web* (str. 1255–1256). Banff, Canada: International World Wide Web Conference Committee.
82. May, R. M. (2004). Uses and Abuses of Mathematics in Biology. *Science*, *303*, 790–793.
83. McRae, M. (2008). *Closure of OASIS UDDI Specification TC*. Najdeno 30. aprila 2016 na spletnem naslovu https://lists.oasis-open.org/archives/uddi-spec/200807/msg00000.html
84. Medvidovic, N., & Taylor, R. N. (2000). A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, *26*(1), 70–93.
85. Merrifield, R., Calhoun, J., & Stevens, D. (2008). The Next Revolution in Productivity. *Harvard Business Review*, *86*(6), 72–80.
86. Mesarič Kunst, U. (2005). Integracija v praksi. V *Zbornik prispevkov desete konference OTS'2005* (str. 25–32). OTS'2005. Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko: Center odličnosti za sodobne informacijske tehnologije in storitve.

87. Microsoft. (2010). *Removal of UDDI Services from Server Operating System*. Najdeno 30. aprila 2016 na spletnem naslovu https://msdn.microsoft.com/en-us/library/dd464641.aspx
88. Middleware. (2016a), V *PC Magazine Encyclopedia*. Najdeno 25. februarja 2016 na spletni strani http://www.pcmag.com/encyclopedia/term/47013/middleware
89. Middleware. (2016b), V *BusinessDictionary.com*. Najdeno 25. februarja 2016 na spletni strani http://www.businessdictionary.com/definition/middleware.html
90. Mohapatra, S. (2009). *Business Process Automation*. Delhi, India: PHI Learning Private Limited.
91. Morris, B. (2013). *Business Architecture Made Easy* (Kindle Edition). Philadelphia: BookBaby.
92. Müller, H. (2009). SOA Governance. V *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research* (str. 339–340). CASCON '09. Ontario, Canada: IBM Corporation.
93. OASIS. (2007). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Najdeno 28. aprila 2016 na spletnem naslovu http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
94. Ott, C., Korthaus, A., Böhmann, T., Rosemann, M., & Krcmar, H. (2011). Foundations of a Reference Model for SOA Governance. V P. Soffer, & E. Proper (Ur.), *Information Systems Evolution: CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers* (str. 44–59). Berlin: Springer Berlin Heidelberg.
95. Papazoglou, M. P., & Heuvel, W.-J. van den. (2006). Service-Oriented Design and Development Methodology. *International Journal of Web Engineering and Technology*, *2*(4), 412–442.
96. Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, *40*(11), 38–45.
97. Pardon, G., & Pautasso, C. (2014). Atomic Distributed Transactions: A RESTful Design. V *Proceedings of the 23rd International Conference on World Wide Web* (str. 943–948). WWW '14 Companion. Seoul, Korea: Association for Computing Machinery.
98. Peklaj, R. (2003). Koliko stane informacijski sistem. *Manager: revija za podjetne*, *2003*(3), 48–51.
99. Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, *17*(4), 40–52.
100. Pratt, L., & Hribar Rajterič, G. (2010). Customers first: an object lesson in achieving multiple goals. V *Case Study Handbook 2010* (str. 14–16). Morristown: TeleManagement Forum.

101. Puschmann, T., & Alt, R. (2004). Enterprise application integration systems and architecture – the case of the Robert Bosch Group. *Journal of Enterprise Information Management*, *17*(2), 105–116.
102. Reldin, P., & Sundling, P. (2007). *Explaining SOA Service Granularity: How IT-strategy shapes services* (magistrsko delo). Stockholm: Linköping University.
103. Rosen, M., Lublinsky, B., Smith, K. T., & Balcer, M. J. (2008). *Applied SOA: Service-Oriented Architecture and Design Strategies*. Indianapolis: John Wiley & Sons.
104. Schmidt, M. T., Hutchison, B., Lambros, P., & Phippen, R. (2005). The Enterprise Service Bus: Making Service-Oriented Architecture Real. *IBM Systems Journal*, *44*(4), 781–797.
105. Shooman, M. L. (2003). *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. New York: John Wiley & Sons.
106. SmartBear. (2014). *OpenAPI Specification*. Najdeno 1. junija 2016 na spletnem naslovu http://swagger.io/specification/
107. Söderström, E., & Meier, F. (2007). Combined SOA Maturity Model (CSOAMM): Towards a Guide for SOA Adoption. V R. J. Gonçalves, J. P. Müller, K. Mertins, & M. Zelm (Ur.), *Enterprise Interoperability II: New Challenges and Approaches* (str. 389–400). London: Springer-Verlag London Limited.
108. Sonic Software Corporation, AmberPoint Inc., BearingPoint Inc., & Systinet Corporation. (2005). *A New Service-Oriented (SOA) Maturity Model*. Najdeno 17. junija 2016 na spletnem naslovu http://www.omg.org/soa/Uploaded%20Docs/SOA/SOA_Maturity.pdf
109. Steghuis, C. (2006). *Service Granularity in SOA Projects: A Trade-off Analysis* (magistrsko delo). Enschede: University of Twente.
110. Stern, N. (1979). The BINAC: A Case Study in the History of Technology. *Annals of the History of Computing*, *1*(1), 9–20.
111. Swanson, E. B., & Dans, E. (2000). System Life Expectancy and the Maintenance Effort: Exploring Their Equilibration. *Management Information Systems Quarterly*, *24*(2), 277–297.
112. Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks* (Fifth Edition). Boston: Prentice Hall.
113. *The USMBOK Navigator*. (2016). Najdeno 15. aprila 2016 na spletnem naslovu http://www.usmbok.com/
114. Themistocleous, M., & Irani, Z. (2004). Evaluating the Integration of Supply Chain Information Systems: A case study. *European Journal of Operational Research*, *159*(2), 393–405.
115. Thomas Manes, A. (2009). *SOA is Dead; Long Live Services*. Najdeno 24. aprila 2016 na spletnem naslovu http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html

116. TM Forum, & itSMF. (2009). *Building Bridges: ITIL and eTOM* (Release 1.0). Morristown: TM Forum.
117. Tracy, L., Guevara, J. K., & Stegman, E. (2008). *IT Key Metrics Data 2009: Key Industry Measures: Multi Industry Spending Overview*. Stamford: Gartner Group.
118. Trotta, G. (2003). *Business Process Management (BPM) Best Practices*: *Dancing Around EAI 'Bear Traps'*. Najdeno 15. junija 2015 na spletnem naslovu http://www.ebizq.net/topics/int_sbp/features/3463.html
119. Turk, T. (2005). Analiza stroškov in koristi naložb v informatiko. *Uporabna informatika*, *13*(3), 153–169.
120. Tziner, A., & Birati, A. (1996). Assessing employee turnover costs: A revised approach. *Human Resource Management Review*, *6*(2), 113–122.
121. Webber, J., Parastatidis, S., & Robinson, I. (2010). *REST in Practice: Hypermedia and Systems Architecture* (Kindle Edition). Sebastopol: O'Reilly Media.
122. Winer, D. (1999). *XML-RPC Specification*. Najdeno 27. aprila 2016 na spletnem naslovu http://xmlrpc.scripting.com/spec.html
123. Worden, R. (2005). *Four Levels of Data Integration*. Najdeno 8. junija 2015 na spletnem naslovu http://www.charteris.com/insight/four-levels-of-data-integration
124. World Wide Web Consortium. (1998a). *Extensible Markup Language (XML) 1.0*. Najdeno 9. aprila 2016 na spletnem naslovu https://www.w3.org/TR/1998/REC-xml-19980210
125. World Wide Web Consortium. (1998b). *The World Wide Web Consortium Issues XML 1.0 as a W3C Recommendation*. Najdeno 9. aprila 2016 na spletnem naslovu https://www.w3.org/Press/1998/XML10-REC
126. World Wide Web Consortium. (2000). *Simple Object Access Protocol (SOAP) 1.1*. Najdeno 22. aprila 2016 na spletnem naslovu http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
127. World Wide Web Consortium. (2004). *Web Services Architecture*. Najdeno 12. junija 2015 na spletnem naslovu http://www.w3.org/TR/ws-arch
128. World Wide Web Consortium. (2005). *Web Services Choreography Description Language Version 1.0*. Najdeno 2. maja 2016 na spletnem naslovu https://www.w3.org/TR/ws-cdl-10/
129. World Wide Web Consortium. (2007a). *SOAP Version 1.2 Part 1 Second Edition*: Messaging Framework. Najdeno 27. aprila 2016 na spletnem naslovu https://www.w3.org/TR/soap12-part1/
130. World Wide Web Consortium. (2007b). *Web Services Addressing Current Status*. Najdeno 2. maja 2016 na spletnem naslovu https://www.w3.org/standards/techs/wsaddr
131. World Wide Web Consortium. (2007c). *Web Services Description Language (WSDL) Version 2.0*: Core Language. Najdeno 27. aprila 2016 na spletnem naslovu https://www.w3.org/TR/2007/REC-wsdl20-20070626/

132. World Wide Web Consortium. (2007d). *Web Services Policy Current Status*. Najdeno 2. maja 2016 na spletnem naslovu https://www.w3.org/standards/techs/wspolicy
133. World Wide Web Consortium. (2009). *Web Application Description Language (WADL)*. Najdeno 1. junija 2016 na spletnem naslovu https://www.w3.org/Submission/wadl/
134. World Wide Web Consortium. (2016). *Facts About W3C*. Najdeno 27. aprila 2016 na spletnem naslovu https://www.w3.org/Consortium/facts
135. Xiao-Jun, W. (2009). Metrics for Evaluating Coupling and Service Granularity in Service Oriented Architecture. V *International Conference on Information Engineering and Computer Science, 2009. ICIECS 2009.* (Str. 1–4). IEEE. Wuhan, China.
136. Zhengyu, X., Baotian, D., & Li, W. (2009). Research of Service Granularity Base on SOA in Railway Information Sharing Platform. V F. Yu, J. Shu, & G. Yue (Ur.), *Proceedings of the 2009 International Symposium on Information Processing (ISIP 2009)* (str. 391–395). Oulu, Finska: Academy Publisher.

PRILOGE

KAZALO PRILOG

PRILOGA 1: Seznam kratic

Tabela 1: Pregled uporabljenih kratic

Kratica	Izvirni pomen	Prevod ali pojasnilo
ACID	Atomicity Consistency Integrity Durability	lastnosti transakcij: atomarnost, konsistentnost, integriteta in trajnost
ADL	Architecture Description Language	jezik za opis arhitekture informacijskih rešitev
API	Application Programming Interface	programski vmesnik aplikacij za dopolnjevanje in povezovanje z zaprto aplikacijo
ASK	analiza stroškov in koristi	metoda za izračun upravičenosti investicije z upoštevanjem pridobljenih koristi in stroškov
B2B	Business-to-Business	Poslovanje ali komunikacija med podjetji
BCV	Business Continuance Volume	Mehanizem zagotavljanja neodvisnih kopij podatkov znotraj diskovnega polja podjetja EMC^2
BRMS	Business Rule Management Systems	sistemi za upravljanje poslovnih pravil
BSS	Business Support Systems	sistemi za podporo poslovanju
BPA	Business Process Automation	Avtomatizacija poslovnih procesov
BPEL	Business Process Execution Language	Jezik za opis izvedbe poslovnih procesov
BPEL4WS	Business Process Execution Language for Web Services	Razširitev jezika za opis izvedbe poslovnih procesov z uporabo spletnih storitev. Predstavlja orodje za orkestracijo storitev.
BPM	Business Process Management	Upravljanje poslovnih procesov
BPMN	Business Process Management Notation	Jezik za modeliranje poslovnih procesov.

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
BPMS	Business Process Management System	sistem za upravljanje poslovnih procesov
CCTA	Central Computer and Telecommunications Agency	britanska vladna agencija, ki je zbrala in objavila priporočila za upravljanje informacijskih sistemov ITIL
CEO	Chief Executive Officer	najvišji predstavnik vodstva, predsednik uprave
CFO	Chief Financial Officer	najvišji predstavnik vodstva zadolžen za finance v najširšem smislu, praviloma član uprave
CIO	Chief Information Officer	najvišji predstavnik vodstva zadolžen za IT, praviloma član uprave
CMMI	Capability Maturity Model Integration	model zrelosti in ogrodje za izboljšanje učinkovitosti podjetij
COM	Component Object Model	Microsoftov model izpostavljanja internih objektov kot komponent drugim programom
CORBA	Common Object Request Broker Architecture	Standard združenja OMG. Določa funkcije, komunikacijske protokole in informacijske objekte/storitve za omogočanje heterogenim aplikacijam, napisanim v različnih programskih jezikih in na različnih sistemih, da lahko medsebojno komunicirajo.
COTS	Commercial off-the-shelf Software	komercialni programski paket
CRM	Customer Relationship Management	upravljanje razmerij s strankami

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
CSOAMM	Combined Service-Oriented Architecture Maturity Model	kombinacija zrelostnih modelov SOAMM in SIMM za področje SOA arhitekture
CTO	Chief Technology Officer	najvišji predstavnik vodstva zadolžen za tehnološki del poslovanja, praviloma član uprave
CXF		Kombinacija imen dveh projektov, iz katerega je nastal Apache CXF: Celtix in XFire.
DAML	DARPA Agent Markup Language	združenje za razvoj jezika in orodij za podporo semantičnemu spletu
DARPA	Defense Advanced Research Projects Agency	vladna agencija Združenih držav Amerike za raziskavo in razvoj naprednih tehnologij v vojaške namene
DCOM	Distributed Component Object Model	dopolnitev Microsoftovega modela izpostavljanja internih objektov kot komponent drugim programom z možnostjo uporabe preko omrežja
DNS	Domain Name System/Service/Server	storitev (ali strežnik) za preslikavo ljudem prijaznega imena v IP naslov ter obratno
DRMS	Decision Rules Management System	sistemi za upravljanje odločitvenih pravil, pogosto sinonim za BRMS
EAI	Enterprise Application Integration	Integracija aplikacij v podjetju
ebXML	Electronic Business using eXtensible Markup Language	družina standardov združenja OASIS in UN/CEFACT z namenom ponuditi odprto, standardizirano infrastrukturo za elektronsko poslovanje temelječo na XML

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
ESB	Enterprise Service Bus	Storitveno vodilo, ki kot osrednji gradnik SOA arhitekture služi za integracijo distribuiranih komponent in gradnikov.
ETL	Extract-Transform-Load	Skupno ime za orodja, ki izvajajo paketne obdelave nad velikimi količinami podatkov. Osnovne funkcije teh orodij so zajem iz izvora, transformacija in polnjenje ponora. Najpogosteje so uporabljena v povezavi s podatkovnimi skladišči.
eTOM	Enhanced Telecom Operations Map	Ogrodje združenja TM Forum za razdelitev operativnega dela poslovanja na posamezne sklope s poudarkom na procesih. Ogrodje se sedaj imenuje Business Process Framework.
FSG	Functionality and Service Graph	graf funkcionalnosti in storitev kot osnova za matematični model granularnosti storitev
FTE	Full Time Employee	enakovredna mera za zaposlenega s polnim delovnim časom
FTP	File Transfer Protocol	Protokol za prenos datotek po internetnem omrežju.
GIS	Geographical Information System	geografski informacijski sistem
HTML	Hyper Text Markup Language	Jezik za predstavitev vsebin z interaktivnimi povezavami.
HTTP	Hyper Text Transfer Protocol	Protokol za prenos vsebin preko svetovnega spleta
HTTPS	Hyper Text Transfer Protocol Security	Varen protokol za prenos vsebin preko svetovnega spleta

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
IBM	International Business Machines	Eno največjih podjetij s področja informacijskih tehnologij na svetu.
ICT	Information and Communication Technology	informacijsko-komunikacijske tehnologije
IDL	Interface Description Language	jezik za opis vmesnikov v CORBA arhitekturi
IEEE	Institute of Electrical and Electronics Engineers	Največje svetovno združenje tehničnih strokovnjakov področij elektrike, elektronike in računalništva.
IETF	Internet Engineering Task Force	odprta mednarodna skupnost snovalcev, ponudnikov rešitev, operaterjev in raziskovalcev iz področja spletnih tehnologij, katerih skupna skrb je za razvoj arhitekture medmrežja in nemotenega delovanja medmrežja
IIOP	Internet Inter-ORB Protocol	internetni protokol za komunikacijo s posrednikom v CORBA arhitekturi
IKT	informacijsko-komunikacijske tehnologije	
IP	Internet Protocol	internetni protokol
IR	informacijska rešitev	
IS	Information System	informacijski sistem
ISO	International Standards Organization	mednarodno združenje za standardizacijo
IT	Information Technology	informacijska tehnologija
ITIL	Information Technology Infrastructure Library	izbor najboljših praks za upravljanje informacijskih sistemov, ki je postal de facto standard in se rahlo razlikuje od ISO/IEC 20000 standarda za ITSM

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
ITSM	The IT Service Management Forum	združenje strokovnjakov iz področja upravljanja IT storitev
itSMF	Information Technology Service Management	področje upravljanja informacijskih sistemov standardiziran v ISO/IEC 20000 in ITIL
J2EE	Java 2 Enterprise Edition	Poslovna različica Jave kot specifikacija in referenčna implementacija. Kasneje je izgubila številko 2 v imenu, s katero se je v začetku ločevala od manj uspešne različice Jave, uporabljene predvsem v brskalnikih.
JAX-WS	Java Api for Xml Web Services	Programski vmesnik za jezik Java za izdelavo in uporabo XML spletnih storitev
JCR	Java Content Repository	standardiziran programski vmesnik za repozitorij vsebine v Javi
JEE	Java Enterprise Edition	novejše ime za J2EE
JMS	Java Message Service	Java programski vmesnik (API) za asinhrono izmenjavo sporočil med dvema odjemalcema
JSON	JavaScript Object Notation	Notacija za opis parametrov in rezultatov metod objektov temelječih na JavaScript sintaksi
LDAP	Lightweight Directory Access Protocol	odprt, od proizvajalcev neodvisen protokol programskega vmesnika za dostop in vzdrževanje splošno namenskega imenika praviloma organiziranega v drevesno strukturo
LOC	Lines of Code	število vrstic kode, kot metrika za obseg funkcionalnosti programske opreme

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
MoWS	Management of Web Services	standard združenja OASIS za upravljanje spletnih storitev
MTOSI	Multi-Technology Operations Systems Interfaces	Tehnološko neodvisni sistemski vmesniki za operacije
MUWS	Management Using Web Services	standard združenja OASIS za upravljanje virov organizacije z uporabo spletnih storitev
OASIS	Organization for the Advancement of Structured Information Standards	Eno največjih svetovnih združenj za sprejemanje in promocijo odprtih standardov.
OGC	Office of Government Commerce	vladno telo v okviru finančnega ministrstva Velike Britanije, ki skrbi za ITIL
OMG	Object Management Group	Mednarodni, odprt, neprofitni konzorcij za sprejemanje tehničnih standardov.
OSS	Operations Support Systems	sistemi za podporo operacijam
OSS/J	Operations Support Systems through Java	OSS/J je nabor API specifikacij za sisteme za podporo operacijam in poslovanju ponudnikov storitev in omrežij.
OWL	Ontology Web Language	semantični jezik, ki omogoča razumevanje svetovnega spleta in obdelavo na podlagi vsebine
PTT	Pošta, telefon in telegraf	v preteklosti pogosta kratica za poštno in telekomunikacijsko podjetje praviloma v državni lasti
QoS	Quality of Service	Kakovost storitve v smislu zagotavljanja dogovorjenih nivojev storitve med ponudnikom in odjemalcem storitve.

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
REST	REpresentational State Transfer	REST je arhitekturni stil distribuiranih sistemov za povezane vsebine, ki opisuje temeljne principe informacijskega inženirstva tega stila in izbrane omejitve interakcij, ki omogočajo uveljavitev teh principov v primerjavi z omejitvami drugih arhitekturnih stilov. (Fielding, 2000, str. 76)
RMI	Remote Method Invocation	oddaljen klic metod
RPC	Remote Procedure Call	oddaljen klic procedur
RS	Republika Slovenija	
SAML	Security Assertion Markup Language	OASIS standard za podporo enkratni prijavi (single sign-on)
SGML	Standard Generalized Markup Language	Standardiziran posplošen jezik za predstavitev vsebin z interaktivnimi povezavami
SID	Shared Information and Data	Podatkovni model za uporabo v telekomunikacijskih ponudnikih, ki ga je pripravil TM Forum.
SIMM	Service Integration Maturity Model	zrelostni model integracije storitev
SiOL	Slovenija On-Line	Hčerinsko podjetje Telekoma Slovenije ustanovljeno po zgledu ameriškega podjetja America On-Line, ki je bil eden največjih ponudnikov interneta v začetnem obdobju. Leta 2007 je bil pridružen matičnemu podjetju in je prenehal obstajati kot podjetje, ostala je samo blagovna znamka.
SMTP	Simple Mail Transfer Protocol	Protokol za prenos elektronske pošte preko svetovnega spleta.

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
SOA	Service-Oriented Architecture	Storitveno usmerjena arhitektura. V tem delu je kratica SOA uporabljena kot sinonim za storitveno usmerjenost, za opis dejanske arhitekture pa uporabljam besedno zvezo SOA arhitektura.
SOAMM	Service-Oriented Architecture Maturity Model	zrelostni model SOA arhitekture
SOAP	Simple Object Access Protocol	Priporočilo W3C za format pošiljanja parametrov oddaljenim metodam in format odgovorov temelječ na XML dokumentih. V priporočilu verzija 1.2, ki je veljavna še danes, je bil pomen kratice opuščen.
SQL	Structured Query Language	Standardiziran jezik za poizvedovanje in spreminjanje podatkov v relacijskih zbirkah podatkov.
SRDF	Symmetrix Remote Data Facility	Funkcionalnost izdelkov podjetja EMC^2, ki omogoča sprotno ustvarjanje konsistentne kopije podatkov na oddaljeni lokaciji.
TAM	Telecom Applications Map	Specifikacija združenja TM Forum za opredelitev glavnih sklopov aplikacij uporabljenih v telekomunikacijskem operaterju.
TCC	Try-Cancel/Confirm pattern	vzorec za programiranje nezanesljivih transakcij s preklicom in potrditvijo
TCO	Total Cost of Ownership	Metodologija izračuna stroškov lastništva z upoštevanjem vseh stroškov v življenjski dobi osnovnega sredstva

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
TMF	TeleManagement Forum	Združenje telekomunikacijskih operaterjev, proizvajalcev opreme, dobaviteljev in izvajalcev z namenom vzpostavitve standardizacije.
UDDI	Universal Description Discovery and Integration	Protokol za opis in odkrivanje storitev na omrežju.
UML	Unified Modeling Language	Splošnonamenski modelirni jezik za snovanje programske opreme usmerjen v razvoj. Poskuša poenotiti grafično predstavitev snovanja računalniškega sistema.
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business	medvladno telo pod okriljem Združenih narodov z namenom vzpodbujanja trgovanja in odpravljanja ovir med članicami Združenih narodov
URI	Uniform Resource Identifier	enolična identifikacija vira
URL	Uniform Resource Locator	enolični določevalnik vira
USMBOK	Universal Service Management Body of Knowledge	metodologija za upravljanje IS
W3C	World Wide Web Consortium	Neprofitno združenje za standardizacijo tehnologij svetovnega spleta.
WADL	Web Application Description Language	jezik za opis pogodb REST storitev
WS-BPEL	Web Services Business Process Execution Language	Razširitev jezika za opis izvedbe poslovnih procesov z uporabo spletnih storitev. Predstavlja orodje za orkestracijo storitev.
WS-CDL	Web Services Choreography Description Language	Jezik za opis interakcij preko meja domenskih območij v SOA arhitekturi.

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	**Izvirni pomen**	**Prevod ali pojasnilo**
WS-TX	Web Services Transactions	Nabor standardov organizacije OASIS za zagotavljanje konsistentnosti transakcij ob uporabi spletnih storitev.
WSDL	Web Services Description Language	Jezik za opis spletnih storitev in njihovih vmesnikov. Predstavlja pogodbo med odjemalcem in storitvijo, ki je strojno berljiva in natančno opisuje operacije in njihove parametre ter rezultate, ki jih vrača. Opredeljuje lahko tudi kardinalnosti in potek interakcije.
WSDM	Web Services Distributed Management	delovno telo združenja OASIS za opredelitev standardov na področju arhitekture spletnih storitev za upravljanje virov v organizaciji
WSRR	Web Services Registry and Repository	register in repozitorij spletnih storitev
XACML	eXtensible Access Control Markup Language	OASIS standard kot varnostna razširitev SOAP za podporo avtorizaciji
XML	eXtensible Markup Language	Razširljiv standardiziran jezik za izmenjavo podatkov med računalniki in drugimi napravami.
XMLENC	XML ENCoding	postopek enkripcije dela ali celotnega XML dokumenta
XML-RPC	eXtensible Markup Language Remote Procedure Call	protokol za oddaljen klic funkcij in procedur z uporabo XML kodiranja klica in parametrov
XMLSIG	XML SIGnature	tehnični postopek za digitalno podpisovanje celotnega ali dela XML dokumenta

se nadaljuje

Tabela 1: Pregled uporabljenih kratic (nad.)

Kratica	Izvirni pomen	Prevod ali pojasnilo
XSLT	eXtensible Stylesheet Language Transformations	jezik za transformacijo XML dokumentov z uporabo predlog
ZLE	Zero Latency Enterprise	stanje v podjetju, ko so pomembne informacije na voljo takoj, ko nastanejo, vsem ki jih potrebujejo

PRILOGA 2: WS-* in REST razširitve

Vsi opisi standardov in priporočil organizacije OASIS, ki sledijo, so povzeti po dokumentih dosegljivih na njihovi spletni strani.

WS-Security je standard organizacije OASIS. Prvič je bil sprejet leta 2004, zadnjič pa dopolnjen maja 2012. Predstavlja razširitev SOAP izmenjave sporočil z namenom zagotavljanja zaupnosti in integritete. Zaupnost predstavlja zaščita pred nezaželenim razkritjem vsebine SOAP sporočila. Integriteta zagotavlja zaščito pred nepooblaščenim spreminjanjem poljubnega dela SOAP sporočila na poti od pošiljatelja do prejemnika z detekcijo morebitnih sprememb. WS-Security omogoča šifriranje in digitalno podpisovanje glave, telesa ali delov glave in telesa SOAP sporočil.

Integriteta sporočil je podprta z XML podpisovanjem (XMLSIG) v povezavi z varnostnimi žetoni (angl. *Security Tokens*), ki zagotavljajo detekcijo sprememb sporočil. Podpira večkratno podpisovanje in je razširljiva za dodajanje novih oblik podpisov.

Zaupnost sporočil uporablja XML šifriranje (XMLENC) v povezavi z varnostnimi žetoni, ki lahko zaščitijo dele SOAP sporočila pred nezaželenim razkritjem.

WS-Addressing je priporočilo konzorcija W3C za naslavljanje spletnih storitev in sporočil neodvisno od transportnega medija. Predlog so posredovali BEA, IBM, Microsoft, SAP in Sun Microsystems avgusta 2004 konzorciju W3C. Kot priporočilo W3C je bil potrjen septembra 2007 (World Wide Web Consortium, 2007b).

Opredeljuje dva prepletajoča se konstrukta, ki opredeljujeta informacijo opredeljeno običajno s transportnim protokolom ali sporočilnim sistemom. Ta konstrukta sta **referenca končne točke** in **glave sporočila**. Oba konstrukta sta v predlogu razširljiva in zmožna ponovne uporabe.

WS-Trust je standard organizacije OASIS. Predlagan je bil julija 2008 in sprejet kot standard februarja 2009. Zadnji popravki so bili sprejeti aprila 2012. Specifikacija opredeljuje razširitve **WS-Security** specifikacije z opredelitvijo ogrodja z metodami za izdajo, obnovo in preverjanje veljavnosti varnostnih žetonov ter načinov ugotavljanja veljavnosti in posredovanja relacij zaupanja med udeleženci. Implementacija lahko za delo z varnostnimi žetoni uporablja **WS-SecureConversation** specifikacijo, vendar to ni obvezno.

WS-SecureConversation je standard organizacije OASIS. Predlagan je bil septembra 2006 in sprejet kot standard februarja 2009. Specifikacija opredeljuje razširitve **WS-Security** specifikacije z opredelitvijo načina izdelave varnostnih žetonov, način obdelave varnostnih žetonov ter postopke izračuna in izmenjave izvedenih ključev. Pri izmenjavi zahteva uporabo **WS-Addressing** specifikacije.

Slabost WS-Security standarda je potencialna visoka obremenitev ob izmenjavi večjega števila zaščitenih sporočil. Kot izboljšava standarda in hkrati alternativa za okolja in uporabo pri izmenjavi večjega števila praviloma krajših sporočil, je bil sprejet standard WS-SecureConversation. WS-SecureConversation omogoča vzpostavitev varne seje med odjemalcem in ponudnikom storitve in potem izmenjavo sporočil znotraj vzpostavljene varne seje. Pri tem se integriteta in izmenjava ključev izvedeta samo enkrat v seji, kar znatno zmanjša potrebo po dodatnem procesiranju.

WS-Policy je priporočilo konzorcija W3C. Prvi predlog je bil posredovan konzorciju W3C julija 2006, kot priporočilo pa je bil potrjen septembra 2007 (World Wide Web Consortium, 2007d).

Ogrodje opredeljuje splošni model in ustrezno sintakso jezika za opis politik entitet v okolju spletnih storitev. Ogrodje opredeljuje osnovni nabor konstruktov za uporabo in razširitev v drugih standardih za opis širokega nabora zahtev in zmogljivosti spletnih storitev:

- subjekt politike je entiteta (npr. končna točka, sporočilo, vir, operacija), katero lahko asociiramo s politiko;
- zahteva politike (angl. *Policy Assertion*) predstavlja zahtevo, zmogljivost ali drugo lastnost obnašanja;
- neobvezna zahteva politike (angl. *Ignorable Policy Assertion*) je neobvezna zahteva politike, ki jo lahko prezremo;
- tip zahteve politike (angl. *Policy Assertion Type*) predstavlja razred zahtev politik in je povezan s shemo zahtev in specifično vsebino zahtev (angl. *Assertion-Specific Semantic*);
- parameter zahteve politike (angl. *Policy Assertion Parameter*) opredeljuje obnašanje povezano

z zahtevo politike;
- alternativna politika (angl. *Policy Alternative*) je potencialno prazen nabor zahtev politik;
- politika je potencialno prazen nabor alternativnih politik;
- izraz politike (angl. *Policy Expression*) je predstavitev politike v XML obliki;
- priloga politike (angl. *Policy Attachment*) je mehanizem povezovanja politike s področjem veljavnosti politike;
- področje veljavnosti politike (angl. *Policy Scope*) je nabor subjektov politike, na katerega se nanaša politika.

V preprostem jeziku, **WS-Policy** omogoča opis zahtev in omejitev spletnih storitev.

WS-Transaction znan tudi pod krajšim imenom **WS-TX**, je standard organizacije OASIS, ki vključuje **WS-Coordination**, **WS-AtomicTransaction** in **WS-BusinessActivity**. Prvič je bil predlagan kot standard avgusta 2006 in sprejet aprila 2007, zadnja potrjena posodobitev standarda pa je bila opravljena februarja 2009.

Spletne storitve vse bolj in bolj povezujejo veliko število sodelujočih v velike enote obdelav imenovanih aktivnosti. Takšne aktivnosti imajo pogosto zapleteno strukturo in vsebujejo zapletene relacije med sodelujočimi. Izvedba takšnih aktivnosti pogosto zahteva daljši čas izvedbe zaradi poslovnih procesov in zahtevanih človeških interakcij. **WS-Coordination** je specifikacija, ki opredeljuje razširljivo ogrodje za koordinacijo aktivnosti z uporabo **koordinatorja** in množico **koordinacijskih protokolov**. Ogrodje zagotavlja sodelujočim konsistentno predstavitev pričakovanih rezultatov distribuiranih aktivnosti. Pri tem pa ogrodje ni omejeno samo na izvajanje transakcij, ampak podpira širok nabor protokolov za distribuirane aplikacije. WS-Coordination temelji na infrastrukturi temelječi na **WS-Security**, **WS-Trust**, **WS-Policy** specifikacij in pri opredelitvi končnih točk zahteva uporabo glave sporočil skladno z **WS-Addressing** specifikacijo.

WS-AtomicTransaction definira atomarno transakcijo kot koordinacijski tip za uporabo v koordinacijskem ogrodju opisanem v **WS-Coordination**. Specifikacija definira 3 specifične koordinacijske protokole za atomarno transakcijo: zaključek (angl. *Completion*), neobstojni dvofazni zapis (angl. *Volatile Two-Phase Commit*) in obstojni dvofazni zapis (angl. *Durable Two-Phase Commit*). Specifikacija temelji na varnostnih specifikacijah: **WS-Security**, **WS-Trust** in **WS-SecureConversation** ter uporablja glave sporočil skladnimi z **WS-Addressing** specifikacijo.

WS-BusinessActivity definira dva koordinacijska tipa poslovnih aktivnosti za uporabo v koordinacijskem ogrodju opisanem v **WS-Coordination**: AtomicOutcome in MixedOutcome.

Za definirana tipa opredeljuje tudi dva specifična koordinacijska protokola: BusinessAgreementWithParticipantCompletion in BusinessAgreementWithCoordinatorCompletion. Oba tipa in protokola sta namenjena uporabi pri dlje trajajočih transakcijah. Varnost je zasnovana na specifikacijah **WS-Security**, **WS-Trust** in **WS-SecureConversation**. Temelji še na naslednjih specifikacijah: **WS-Policy**, **WS-SecurityPolicy** in **WS-Addressing**.

WS-SecurityPolicy je standard organizacije OASIS. Predlagan je bil decembra 2006 in sprejet kot standard februarja 2009. Zadnji popravki standarda so bili objavljeni aprila 2012. Standard specificira opis posebnih zahtev in omejitev spletnih storitev, skladen z **WS-Policy** vezanih na specifikacije **WS-Security**, **WS-Trust** in **WS-SecureConversation**.

WS-BPEL razširitev je znana tudi pod imenom BPEL4WS. Je standard organizacije OASIS. Predlagan je bil novembra 2006 in kot standard sprejet aprila 2007. Specificira jezik za opis poslovnih procesov z uporabo spletnih storitev. Jezik je uporaben tako za abstrakten opis procesa, kot za izvajanje procesa. Temelji na **WSDL** in **SOAP** ter omogoča orkestracijo storitev.

WS-CDL razširitev je znana tudi pod imenom WS-Choreography. Je kandidat za priporočilo konzorcija W3C. Prvič je bil predlagan aprila 2004, kot kandidat za priporočilo pa je bil potrjen novembra 2005 (World Wide Web Consortium, 2005). WS-CDL je opisni jezik temelječ na XML za opis sodelovanja udeležencev iz globalnega stališča. Predvsem v scenarijih e-poslovanja, je potreba po izvajanju dolgo trajajočih sodelovanj med udeleženimi spletnimi storitvami. Sodelovanje lahko poteka znotraj zaupanja vrednih domen ali preko mej domen izven podjetja. WS-Choreography omogoča sestavo interoperabilnih sodelovanj vsak z vsakim med poljubnimi vrstami udeležencev neodvisno od tehnologije in programskega modela uporabljenega za implementacijo. Opis koreografije je pogodba med večjim številom udeležencev, ki opisuje kompozicijo iz globalnega stališča. Z WS-Choreography lahko opišemo tudi zahtevano zaporedje klicev metod storitve, kar je ena večjih pomanjkljivosti WSDL specifikacije.

WS-ReliableMessaging je standard organizacije OASIS. Predlagan je bil avgusta 2006, zadnja potrjena revizija pa je bila sprejeta februarja 2009. Opisuje protokol, ki omogoča zanesljivo dostavo sporočil med vozlišči tudi v primeru napak in izpadov programskih komponent, sistemov ali napak v omrežju.

WS-Reliability je standard organizacije OASIS. Standard je bil sprejet novembra 2004. Predstavlja razširitev SOAP spletnih storitev, ki zagotavlja: zanesljivo dostavo sporočil v skladu z **WS-ReliableMessaging**, preprečevanje podvajanja sporočil in pravilno zaporedje sporočil.

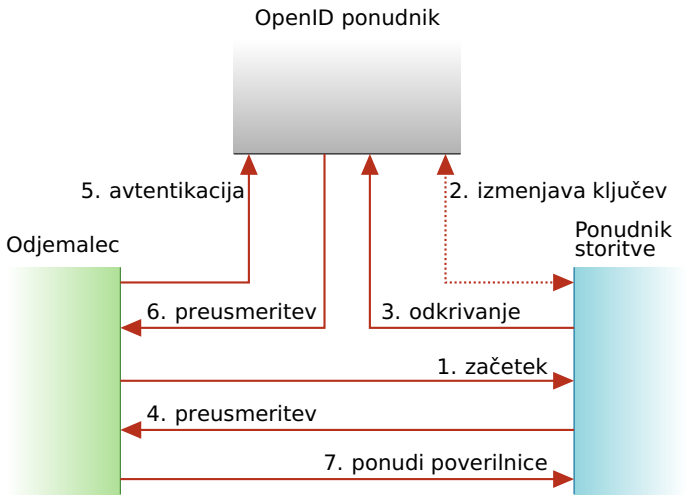
WS-Federation je standard organizacije OASIS. Predlagan je bil junija 2008, kot standard pa je bil sprejet maja 2009. Specifikacija je namenjena razširitvi spletnih storitev temelječih na

SOAP, WSDL in XML in opisuje mehanizme, ki omogočajo dostop do virov med različnimi varnostnimi domenami. Temelji na standardih **WS-Security**, **WS-SecurityPolicy** in **WS-Trust**. Je osnova drugim standardom na področju varnosti spletnih storitev.

OpenID Authentication 2.0 (v nadaljevanju OpenID) je specifikacija OpenID fundacije, ki decentralizira preverjanje identitete v spletu in je uporabljena v REST okolju. Je alternativa standardni možnosti predstavitve odjemalca s certifikatom, kjer je breme preverjanja identitete na sami storitvi. Specifikacija je bila sprejeta decembra 2007.

Ob uporabi OpenID je preverjanje identitete delegirano **OpenID ponudniku**, kar omogoča, da so podrobnosti odjemalca skrite ponudniku storitve. Osnova za medsebojno zaupanje odjemalca in ponudnika storitve je zaupanje OpenID ponudniku, ki pa mora biti vzpostavljeno z ločenim mehanizmom in ga OpenID ne predpisuje. Potek OpenID avtentikacije prikazuje slika 10.

Slika 10: OpenID avtentikacija



Vir: J. Webber, S. Parastatidis & I. Robinson, REST in Practice, 2010, str. 296.

Atom Syndication Format (v nadaljevanju Atom) temelji na XML in je format za predstavitev časovno žigosanih seznamov spletnih vsebin in meta podatkov. Je prilagodljiv, interoperabilni format za izmenjavo podatkov med aplikacijami. Ima status predloga standarda pri združenju IETF od decembra 2005. Predlog standarda je opredeljen v dokumentu RFC 4287.

Predvsem ob uporabi REST storitev, Atom ponuja standardizirano osnovo za izmenjavo podatkov med odjemalci in ponudniki storitev. S časovnim žigosanjem preprečuje napake, ki bi jih predpomnenje lahko povzročilo nad zastarelimi podatki. S tem je omogočena uporaba predpomnilnikov, ki so v REST storitvah ključnega pomena za izvedbo hitrih in skalabilnih rešitev. To je tudi povzročilo veliko priljubljenost formata med razvijalci in ponudniki storitev.

PRILOGA 3: Najpogostejši gradniki SOA arhitekture

3.1 Storitveno vodilo

Mahmoud (2013, str. 697) trdi, da je „storitveno vodilo ogrodje, ki pomaga pri kreiranju nameščanju in orkestraciji (medsebojni komunikaciji) servisnih komponent v distribuiranem sistemu".

Schmidt, Hutchison, Lambros & Phippen (2005, str. 781) opredeli storitveno vodilo, kot „Storitveno vodilo je infrastruktura, ki podpira polno integrirano in prilagodljivo SOA arhitekturo skozi celotno podjetje".

ESB oziroma storitveno vodilo je srce SOA arhitekture (Hérault, Thomas & Lalanda, 2007). Kot osrednja točka SOA arhitekture, storitveno vodilo opravlja pomembne funkcije:

- usmerjanje sporočil,
- transformacijo sporočil,
- izenačevanje časovno neenakomerne obremenjenosti z uporabo sporočilnih vrst,
- koordinacijo transakcij,
- orkestracijo storitev,
- centralizacijo poslovnih pravil in
- pomožne storitve: obravnava dogodkov, varnostne storitve, uveljavljanje politik, zagotavljanje kakovosti storitev (QoS) in pretvorba med protokoli.

Poleg naštetih sta pogosto v storitveno vodilo vključeni tudi imenik storitev in repozitorij storitev. V takem primeru storitveno vodilo izvaja še funkciji nameščanja storitev in nadzor verzij, če je slednje v uporabi.

Usmerjanje sporočil temelji na metapodatkih klicateljev in objavljenih storitev v SOA arhitekturi (Schmidt, Hutchison, Lambros & Phippen, 2005, str. 782–783). Z usmerjanjem na podlagi metapodatkov zapisanih v imeniku storitev in repozitoriju storitev ter v opisih politik je dosežena šibka sklopljenost, ki naprej omogoča lažje vzdrževanje, upravljanje in povečuje preglednost sistema.

Skozi **transformacijo sporočil** storitveno vodilo uveljavlja enoten podatkovni model za vse storitve v konkretni implementaciji SOA arhitekture. V funkcijo transformacije so vključeni tudi prilagojevalniki za komercialne in starejše IR, kar kaže, da je storitveno vodilo evolucija vodila iz EAI arhitekture.

Večje število slojev abstrakcije pomeni tudi sodelovanje večjega števila distribuiranih storitev

in komponent ob izvedbi posameznih poslovnih funkcij. Slednje pa zahteva **koordinacijo transakcij**, ki jo v SOA arhitekturi izvaja in zagotavlja storitveno vodilo. Poleg transakcijskih koordinatorjev, kot gradnikov vključenih v storitveno vodilo, so za koordinacijo transakcij ob uporabi več storitev pomembne razširitve WS-Transaction za SOAP storitve, oziroma vzorec Pardon & Pautasso (2014, str. 943–948) za REST storitve. Storitveno vodilo skozi naštete mehanizme zagotavlja ACID lastnosti transakcije na nivoju orkestriranih storitev.

Orkestracija storitev v poslovne procese se izvaja v okviru BPMS, drugi del orkestracije storitev v višjenivojske storitve pa se izvaja na storitvenem vodilu. To je skladno s centralizacijo poslovne logike in pravil. Orkestrirane storitve so namreč nosilec poslovne logike. Pri tem je lahko del poslovnih pravil uporabljen iz BRMS, drugi del je izveden opisno z metapodatki, del pa je neposredno zakodiran v logiki orkestrirane storitve.

Storitveno vodilo centralizira poslovna pravila skozi sistem za upravljanje poslovnih pravil (v nadaljevanju BRMS), v obliki kodirane logike ali v obliki poslovnih diagramov v strojno berljivi obliki. Najpogosteje pa so poslovna pravila zapisana kot kombinacija vseh naštetih. Podobno kot pri poslovnih procesih, želimo postavljanje poslovnih pravil približati poslovnim uporabnikom in jih razbremeniti tehničnih podrobnosti. To je mogoče doseči z uporabo BRMS. Ker pa vseh pravil ni mogoče predstaviti z BRMS oziroma predstavlja uporaba BRMS iz performančnega vidika nesprejemljivo ali preobsežno obremenitev za preprost namen, je del poslovnih pravil zakodiran v komponentah na storitvenem vodilu in izpostavljen kot nabor storitev. Najbolj znano komercialno BRMS orodje je Websphere ILOG JRules, konkurenti pa so še Progress Corticon BRMS, Fico Blaze Advisor DRMS in PegaRULES. Odprtokodni rešitvi pa sta Drools in OpenRules.

Storitveno vodilo lahko ob pravilni implementaciji, konfiguraciji in uporabi, močno poveča zanesljivost in razpoložljivost IS (González & Ruggia, 2011, str. 1). To je posledica šibke sklopljenosti storitev in možnost dodajanja virov, ki naredi sistem skalabilen in omogoča podvajanje kritičnih gradnikov SOA arhitekture.

Na trgu je poleg komercialnih tudi več odprtokodnih storitvenih vodil, praviloma pa odprtokodni produkti ne dosegajo komercialnih vsaj na področju zanesljivosti (Astrova, Koschel & Kruessmann, 2010, str. 2495). Vzpostaviti celoten sklad SOA gradnikov na podlagi odprtokodnih rešitev je lahko velik izziv (Davis, 2009).

3.2 Imenik storitev

Imenik storitev implementira preprosto funkcijo preslikave identifikacije storitve na njen vmesnik, ki je točka implementacije storitve. Poleg tega opravlja še funkcijo preverjanja avtorizacije. Tako

je za SOAP storitve opredeljeno tudi v UDDI specifikaciji. Opisani gradniki in postopek je lahko uporabljen tudi za REST storitve, vendar izkorišča le majhen del funkcionalnosti in je zato preveč zapleten. Bolj smiselno je uporabiti preprostejše mehanizme, kot sta DNS in LDAP. Modernejši imeniki storitev podpirajo tudi več različic posameznega vmesnika. Standardnega načina zapisa različic v UDDI specifikaciji ni. Drugi protokoli in sistemi pa imajo različno podprte zapise različic in jih prav tako smatram za nestandardne. Najpreprostejši način je pohabljanje imen (angl. *Name Mangling*), ki izhaja iz programskega jezika C++. V tem primeru imenu storitve dodamo številko različice v naprej predpisani obliki (Jurič, Saša, Brumen & Rozman, 2009). Dejanska storitev ima sicer drugačno ime, česar se „zaveda" imenik storitev in zato pri vračanju odgovora odjemalcu, prilagodi URL dejanskemu imenu storitve in njenega vmesnika.

Odjemalec se predstavi imeniku storitev in zahteva informacijo o določeni spletni storitvi na podlagi njenega imena. Če je v uporabi še več različic, mora odjemalec podati tudi različico. Imenik storitev preveri, ali storitev obstaja in ali ima odjemalec pravico dostopa do storitve. V primeru pozitivnih testov, odjemalcu vrne naslov vmesnika, ki je tako v primeru SOAP, kot v primeru REST storitev, URL vmesnika storitve.

V SOA arhitekturi je imenik storitev sicer preprost, vendar pogosto uporabljen, zato je ključnega pomena, da je zanesljiv in hiter. Trajanje vsakega klica imenika storitev se prišteje k trajanju izvedbe storitve, kar lahko znatno upočasni celoten sistem. Zanesljivost imenika storitev lahko dosežemo s federacijo imenika in postavitev na več ločenih, zmogljivih strežnikih, ki so „blizu" odjemalcem, da je klic čim hitrejši in s čim manjšo zakasnitvijo. Replikacija vsebine imenika pa poskrbi za skoraj sinhronizirano stanje med vsemi vozlišči imenika. V primeru odpovedi enega ali več vozlišč imenika, njihovo funkcijo prevzamejo preostala vozlišča. V idealnem primeru je določitev nadomestnih vozlišč narejena na podlagi obremenitve delujočih vozlišč in „oddaljenosti" odjemalca. V praksi pa se pogosto uporabljajo preprostejše metode s HTTP prehodi na podlagi krožnega dodeljevanja (angl. *Round Robin*).

3.3 Repozitorij storitev

Odjemalci SOAP storitev potrebujejo za uporabo storitve njen opis v WSDL obliki, odjemalci REST storitev pa imajo na voljo opis v WADL ali Swagger obliki. Opisovanje REST storitev je v uporabi praktično samo v poslovnih okoljih in večjih informacijskih ekosistemih. V manjših okoljih in javnih, bolj tehnično usmerjenih primerih, pa so REST storitve uporabljene brez opisov in posrednih gradnikov arhitekture, kar povečuje sklopljenost odjemalca in ponudnika storitve. Opisi storitev omogočajo pozno povezovanje odjemalca in ponudnika storitve ter s tem šibko sklopljenost in prilagodljivost celotnega sistema. Zato je opis storitve v SOA arhitekturi obvezni del vsake storitve.

WSDL, WADL in Swagger pa niso edini dokumenti o storitvi vpisani v repozitorij. Vsak repozitorij opredeljuje še dodatne lastnosti, kar predstavlja nestandardno rešitev. Standardi se za doseganje višje stopnje natančnega opisa in s tem možnosti iskanja storitev, stalno dopolnjujejo. V SOA arhitekturi najpogosteje v repozitorij shranjujemo meta podatke po standardu JCR 2.0 ali Microsoft Managed Services Engine. Na trgu je več komercialnih izdelkov: IBM WSRR, Oracle Service Registry, Sentinet Repository ter odprtokodnih rešitev, kot je JBoss Metadata Repository.

3.4 Storitve

Storitve kot gradnik SOA arhitekture, opredeljujejo končni IS. Vsi ostali gradniki so samo pomoč pri povezovanju, izgradnji in izvajanju storitev. Storitve so tudi nosilec vsebine in poslovnih pravil, s katerimi podjetje dosega konkurenčno prednost na trgu. To je tudi vzrok, da vseh poslovnih storitev ne moremo kupiti že izdelanih na trgu, temveč jih je potrebno zgraditi, skonfigurirati in orkestrirati v vsakem okolju posebej. SOA arhitekture zato ne moremo kupiti. Potrebno jo je zgraditi v vsakem okolju posebej.

Storitve so lahko tehnično implementirane na različne načine: kot SOAP spletne storitve, kot REST storitve, kot poslovni proces ali katerakoli druga oblika avtomatizirane logike. Za storitve v SOA arhitekturi je pomembno, da je storitev opisana in izpolnjuje pogoje opisane na strani 41.

3.5 Sistem za upravljanje poslovnih procesov

Orkestracija storitev tesno povezuje SOA arhitekturo in sisteme za upravljanje poslovnih procesov (v nadaljevanju BPMS). Področje upravljanja poslovnih procesov ima dobro opredeljeni standardni notaciji BPMN in BPEL. BPMS je orodje, skozi katerega lahko uporabniki orkestrirajo storitve z uporabo BPEL4WS (Papazoglou, Traverso, Dustdar & Leymann, 2007, str. 64) oziroma WS-BPEL (OASIS, 2007). S tem je SOA najboljša osnova za avtomatizacijo poslovnih procesov (angl. *Business Process Automation*) (Mohapatra, 2009, str. 55), ki odpravlja napake in zakasnitve kot posledice človeškega vpliva ter zmanjšuje potrebno število kadrov, ki so največji strošek podjetja (Peklaj, 2003, str. 50–51) ter tako omogoča doseganje osnovnega cilja podjetja: povečevanje vrednosti za lastnike (Mohapatra, 2009, str. 5).

Orkestracija je odvisna od dveh gradnikov SOA arhitekture: imenika storitev in repozitorija storitev. Brez slednjih, bi bili poslovni uporabniki omejeni na vnaprej znane storitve na točno določenih strežnikih. Poleg togosti, bi bila takšna arhitektura občutljiva na izpade posameznih storitev, njena zanesljivost bi bila nižja, skalabilnost v smislu zmožnosti prilagajanja obremenitvi pa na zelo nizki ravni.

Preprost uporabniški vmesnik v kombinaciji z dovolj preprosto notacijo in grafičnim urejevalnikom procesov v primeru uporabe BPMN, oziroma preproste sintakse za opis procesov v primeru uporabe BPEL, omogočajo prenos urejanja procesov na poslovne uporabnike, s čimer so razbremenjeni IT kadri, skrajšana komunikacijska veriga, zmanjšan prepad med poslovanjem in informacijsko tehnologijo ter tako omogočeno hitrejše prilagajanje IS spremembam v poslovanju. Dodatne tehnologije, kot je WS-BPEL pa omogočajo uporabo drugih storitev v SOA arhitekturi in s tem zlitje s storitveno usmerjenostjo.

3.6 Uporabniški vmesnik

Uporabniški vmesnik je vstopna točka v poslovne procese. V SOA arhitekturi je pogosto uporabniški vmesnik izveden kot spletna aplikacija oziroma s portalskim sistemom. S tem je dosežena neodvisnost od operacijskega sistema odjemalca. To pa ni pravilo ali celo zahteva. Prav lahko je uporabniški vmesnik implementiran kot težka aplikacija pisana za konkretni operacijski sistem, kar pa prinaša vezavo na točno določen operacijski sistem.

Uporabniški vmesnik je lahko ločen za vsako poslovno funkcijo posebej ali poenoten zaradi enotnega izgleda in enotnega načina uporabe, kar je za uporabnike, ki uporabljajo več poslovnih funkcij, prijaznejše in zahteva manj učenja. V obeh primerih pa mora implementacija uporabniškega vmesnika spoštovati arhitekturna načela SOA: do funkcij sme dostopati le preko najvišjega nivoja storitev, nefunkcionalne zahteve pa prepustiti drugim gradnikom SOA arhitekture, kot je storitveno vodilo.

Prav pri implementaciji uporabniškega vmesnika prihaja najpogosteje do kršitev politik in pravil SOA arhitekture ter s tem do postopne arhitekturne erozije in arhitekturnega odmika, kar s časom pripelje do težko obvladljivega in nepreglednega IS.

PRILOGA 4: Kriteriji in stopnje zrelostnih modelov

Tabela 2: SOA Maturity Model kriteriji in stopnje

Nivo zrelosti	**Primarne poslovne koristi**	**Področja**	**Kritični tehnološki faktorji uspeha**	**Kritični faktorji uspeha zaposlenih in organizacije**	**Izbrani pomembni standardi**
1. Začetne storitve	nova funkcionalnost	preizkušanje v R&D, spletna stran za pilotne projekte, portal, prilagojene integracije, majhno število storitev	standardi, integracija obstoječih sistemov	razvijalci pridobijo znanja za razvoj storitev, razvijalci so gonilna sila	XML, XSLT, WSDL, SOAP, Java, .NET
2. Storitve v arhitekturi	znižanje IT stroškov in nadzor	več integriranih aplikacij	podpora heterogenosti in distribuiranim sistemom, zanesljiv sporočilni sistem, mediacija, olajšane namestitve, integracija zbirk podatkov, podpora različicam, sestavljene aplikacije	arhitekturna skupina prevzame vodilno vlogo, SOA kompetenčni center, podpora najvišjega vodstva IT (CIO)	UDDI, WS-Reliable-Messaging, WS-Policy, WS-Addressing, XQuery, WS-Security, SAML
3.a poslovne storitve	poslovna odzivnost - hitro in učinkovito spreminjanje poslovnih procesov	poslovni proces poteka preko meja organizacijskih enot v celotnem podjetju	ponovna uporaba, potreben vložek za spremembe, razpoložljivost, pravila za poslovne procese, dogodkovno vodeni procesi, sestavljene aplikacije	partnerstvo med IT in poslovnim delom, partnerstvo med podjetji, Obvladovanje SOA, sponzorstvo vodstva, znanje snovanja dogodkovno vodenih sistemov,	WS-BPEL
3.b skupne storitve	poslovna odzivnost - sodelovanje s poslovnim delom in partnerji	razpoložljive storitve zunanjim partnerjem v celotnem podjetju	odprtje zunanjih storitev, enotna varnost v podjetju, poenotenje protokolov v podjetju, dolgo trajajoče transakcije	sponzorstvo vodij organizacijskih enot	RosettaNet, ebXML, WS-Trust

se nadaljuje

Tabela 2: SOA Maturity Model kriteriji in stopnje (nad.)

Nivo zrelosti	**Primarne poslovne koristi**	**Področja**	**Kritični tehnološki faktorji uspeha**	**Kritični faktorji uspeha zaposlenih in organizacije**	**Izbrani pomembni standardi**
4. merljive poslovne storitve	sprememba poslovanja iz reakcijskega v realni čas, izpolnjevanje poslovnih metrik zmogljivosti	organizacijska enota ali celotno podjetje, preko celotnega podjetja	spremljanje poslovnih aktivnosti, obdelovanje toka dogodkov, obdelovanje kompleksnih procesov, dogodkovno proženi portali in alarmi	stalno vrednotenje poslovnih procesov in odzivov, sponzorstvo CFO	
5. optimizirane poslovne storitve	optimizacija poslovanja - avtomatska reakcija in odziv	organizacijska enota ali celotno podjetje, preko celotnega podjetja	dogodkovno prožena avtomatizacija za optimizacijo	stalno izboljševanje kulture, sponzorstvo najvičjega vodstva (CEO)	

PRILOGA 5: Vzorci

Vzorci so naraven način človeškega razmišljanja in delovanja. Na ta način delujejo tudi človeški možgani. Predstavljajo ponovno uporabo preteklih pozitivnih izkušenj in izogibanje stanjem, ki so povzročili negativne izkušnje. Pozitivne izkušnje lahko uporabimo v podobnih, novo nastalih okoliščinah. Negativnim izkušnjam se poskušamo izogniti, pozitivne pa ponoviti. Proces vzpostavljanja vzorcev imenujemo učenje. Učimo se lahko sami, pri čemer imamo omejen nabor učnih primerov v omejenem času, lahko pa nas nauči kdo drug, ki je že zbral nabor vzorcev in nam jih lahko posreduje. Ta proces imenujemo šolanje.

Na področju IT uporabljamo vzorce zaradi kompleksnosti problemov: izredno težko ali celo nemogoče je proučiti medsebojni vpliv vseh lastnosti sistema na končni rezultat. Z vzorci zmanjšamo nabor potrebnih odločitev in uporabljamo najboljše prakse ter s tem povečamo verjetnost uspešne vzpostavitve sistema s pozitivnimi učinki. Pri tem pa je pomembno, da za dani problem izberemo pravi vzorec.

Vzorci v inženirstvu izhajajo iz gradbene stroke. Na področju programiranja so se uveljavili vzporedno z razmahom objektno usmerjenega programiranja (Beck & Cunningham, 1987). Sprva so vzorci zajemali način reševanja najpogostejših problemov pri implementaciji programske kode, kasneje pa so se pojavili še na področju arhitekture, kot snovalski vzorci.

Da bi razumeli, kako izbrati ustrezne snovalske vzorce, najprej opredelimo, kaj snovalski vzorci so:

„Vzorec je lahko tako gradnik sistema, kot opis, kako takšen gradnik izdelati.” (Coplien, 1997, str. 37)

„Vzorec programske opreme opisuje konkreten ponavljajoč problem snovanja, ki izhaja iz specifičnega konteksta snovanja in predstavlja preizkušeno generično shemo rešitve problema. Shema rešitve je opisana z njenimi sestavnimi komponentami, njihovimi odgovornostmi, relacijami in načinom medsebojnega sodelovanja.” (Buschmann, Meunier, Rohnert, Sommerlad & Stal, 2000, str. 23)

„Vzorec opredeljuje proces reševanja problema z upoštevanjem zaporedja odločitev snovanja in implementacije.” (Fielding, 2000, str. 20)

Fielding opredeli tudi arhitekturni slog, kot „poimenovan nabor omejitev arhitekturnih gradnikov, ki vodijo k želenemu naboru lastnosti arhitekture”.

Vzorec opredeljujejo: **kontekst**, ki je okoliščina, ki povzroča **problem**. **Problem** je ponavljajoče

stanje v danem kontekstu, ki potrebuje **rešitev**. **Rešitev** je preverjena razrešitev problema (Buschmann, Meunier, Rohnert, Sommerlad & Stal, 2000, str. 24).

Za obravnavo arhitekturnih vzorcev moramo ločiti med arhitekturo in infrastrukturo (Erl, 2009, str. 27). Za obravnavo SOA področja, infrastrukturo predstavljajo gradniki opisani v poglavju 2.3 brez samih storitev. Storitve spadajo v arhitekturo in so skupaj z drugimi programskimi komponentami in medsebojnimi povezavami, predmet snovalskih vzorcev v SOA arhitekturi.

Vzorci so opisani z imenom, zahtevo, problemom, rešitvijo, uporabo in vplivi. Pogosto so vzorcu pripisane še druge lastnosti, kot so npr. ikona, uporabljeni principi in sloj arhitekture. **Zahteva** v enem stavku opisuje osnovno zahtevo, na katero se vzorec nanaša, v obliki vprašanja. **Problem** opisuje težavo, ki povzroča problem in posledice problema. **Rešitev** predstavlja snovalsko rešitev, ki jo ponuja vzorec za rešitev problema in izpolnitev zahtev. **Uporaba** opisuje način uporabe vzorca. Uporaba lahko vključuje splošne napotke, kot tudi tehnične podrobnosti in občasno predlaga še proces. **Vplivi** opisujejo pogoste posledice, stroške in zahteve povezane z uporabo vzorca. Lahko ponujajo tudi alternativne vzorce.

Dva reprezentativna primera sta: **preusmeritev končne točke** (Erl, Carlyle, Pautasso & Balasubramanian, 2013, poglavje 14.1) in **centralizacija pogodbe** (Erl, Carlyle, Pautasso & Balasubramanian, 2013, poglavje 14.2). Primera sta podana v prilogi 5.1.

Arhitekt mora pri izbiri in uporabi vzorcev paziti na ustreznost vzorca za rešitev konkretnega problema ter na medsebojno nezdružljivost vzorcev, pri čemer je lahko nezdružljivost povezana bodisi s kontekstom vzorcev ali njihovimi vplivi.

5.1 Primera snovalskih vzorcev v SOA arhitekturi

Ime	Preusmeritev končne točke
Zahteva	Kako naj se odjemalec določene končne točke storitve prilagodi ob spremembi ali umiku končne točke?
Problem	Opis storitvene končne točke vsebuje razdelke, ki se sčasoma lahko spremenijo. Včasih ni mogoče zamenjati vse reference na posodobljeno končno točko, kar povzroči, da odjemalec ne more več komunicirati s končno točko storitve.
Rešitev	Samodejna preusmeritev odjemalcev storitve, ki poskušajo dostopati do prejšnje končne točke storitve k prenovljeni končni točki.
Uporaba	Uporabi funkcijo pogodbe storitve za preusmeritev končne točke storitve. Ko odjemalec poskuša uporabiti staro metodo storitve, vrni preusmeritveni odgovor. Odjemalec bo upošteval preusmeritev in ponovil zahtevo preko nove končne točke. Preusmeritve so lahko začasne ali stalne. Stalne preusmeritve si odjemalec zapomni v svoji konfiguraciji in ne uporablja več stare končne točke. Storitve, ki uporabljajo identifikatorje virov za izpostavljanje funkcionalnosti, sprememba identifikatorja povzroči spremembo odkrite pogodbe med izvajanjem. Preusmeritev je zmožna reševati tudi takšne scenarije.
Vplivi	Odjemalec storitve mora vsebovati logiko, ki zazna in obdela preusmeritvene odgovore. Usklajene nadgradnje odjemalca in storitev sta v celoti nepotrebna ob uporabi preusmeritve končne točke. Preusmeritve pa povzročajo ponavljajoče podaljšanje časa izvajanja, katerim se je mogoče izogniti z nadgradnjo odjemalcev. Vračanje preusmeritve odjemalcu zahteva od odjemalca, da vnaprej določi stopnjo zaupanja viru zahteve za preusmeritev. Storitev, ki je bila kompromitirana s stališča varnosti, lahko pri odjemalcu povzroči stalno preusmeritev identifikatorja k neveljavni ali škodljivi storitvi. Napačno skonfigurirana storitev lahko povzroči enak učinek. Pozorni moramo biti tudi na izogibanje neskončnim zankam preusmeritev.

Ime	Centralizacija pogodbe
Zahteva	Kako se lahko izognemo neposredni sklopljenosti odjemalca z implementacijo storitve?
Problem	Odjemalci so lahko zasnovani tako, da dostopajo do storitve preko različnih točk, kar vodi v večkratno odvisnost od implementacije. To pa ovira razvoj storitve pri odzivu na spremembe.
Rešitev	Dostop do logike storitve je omejen na storitveno pogodbo, kar prisili odjemalce v izogibanje sklopljenosti z implementacijo.
Uporaba	Ta vzorec je izveden skozi formalne standarde snovanja v podjetju, pri čemer mora biti končna storitev zasnovana in implementirana v skladu s principom abstrakcije storitev.
Vplivi	Siljenje odjemalcev, da dostopajo do funkcionalnosti in virov storitve izključno preko pogodbe, predstavlja dodaten čas za procesiranje v času izvajanja, kar znižuje zmogljivost storitve, zato zahteva stalni nadzor in vložek v standardizacijo.

PRILOGA 6: Fluktuacija zaposlenih

Stroškom dela lahko prištejemo še stroške fluktuacije, do katerih prihaja iz različnih razlogov. Cotton & Tuttle (1986, str. 57) navajata dejavnike fluktuacije, ki so predstavljeni v tabeli 3.

Tabela 3: Dejavniki fluktuacije zaposlenih

zunanji dejavniki	**z delom povezani dejavniki**	**osebnostni dejavniki**
zaznavanje zaposlenosti	plačilo	starost
stopnja brezposelnosti	učinkovitost	mandat
stopnja zaposlovanja	jasnost zadolžitev	spol
moč sindikatov	ponavljajoče delo	biografija
	splošno zadovoljstvo z zaposlitvijo	izobrazba
	zadovoljstvo s plačilom	zakonski status
	zadovoljstvo z delom	število vzdrževanih članov
	zadovoljstvo z vodstvom	sposobnosti in zmožnosti
	zadovoljstvo s sodelavci	inteligenca
	možnost napredovanja	vedenjski vzorci
	pripadnost organizaciji	izpolnitev pričakovanj

Vir: J. L. Cotton & J. M. Tuttle, Employee Turnover: A Meta-Analysis and Review with Implications for Research, 1986, str. 57.

Cotton & Tuttle (1986, str. 64) na podlagi korelacijske analize določita tudi dejavnike, ki pomembno vplivajo na fluktuacijo zaposlenih:

- zaznavanje zaposlenosti $(+)$
- moč sindikatov $(-)$
- plačilo $(-)$
- splošno zadovoljstvo z zaposlitvijo $(-)$
- zadovoljstvo z delom $(-)$
- zadovoljstvo s plačilom $(-)$
- zadovoljstvo z vodstvom $(-)$
- starost $(-)$
- mandat $(-)$
- izobrazba $(+)$
- število vzdrževanih članov $(-)$
- biografija
- pripadnost organizaciji $(+)$
- izpolnitev pričakovanj $(-)$
- vedenjski vzorci $(+)$

V seznamu so označeni dejavniki, ki prispevajo k večji fluktuaciji $(+)$ in dejavniki, ki zmanjšujejo fluktuacijo $(-)$. Biografija pomembno vpliva na fluktuacijo v obeh smereh.

Stroški fluktuacije zajemajo še stroške izobraževanja, rekrutiranja in uvajanja. Do fluktuacije pride lahko v dveh primerih: povečanje obsega dela ali kot posledica odhoda obstoječega kadra. Do odhoda obstoječega kadra lahko pride na željo zaposlenega ali zaradi potreb podjetja.

Verjetnost odhoda torej lahko povežem z obsegom in zahtevnostjo dela ter s plačilom zanj na eni strani in verjetnostjo napačne ocene v fazi rekrutiranja:

$$p_{odhoda} = \frac{f(O*K)}{plačilo} + p_{napačne\ ocene} \tag{57}$$

Če verjetnost odhoda pomnožimo s stroški rekrutiranja, šolanja in uvajanja, dobimo enačbo za strošek zamenjave zaposlenega:

$$C_{zamenjave} = \left(\frac{f(O*K)}{plačilo} + p_{napačne\ ocene}\right) * \left(C_{rekrutiranja} + C_{šolanja} + C_{uvajanja}\right) \tag{58}$$

Kolikšni so torej $C_{rekrutiranja}$, $C_{šolanja}$ in $C_{uvajanja}$? Strošek zamenjave zaposlenega je naslednji (Florjančič, Ferjan & Bernik, 1999, str. 62):

$$EF = \frac{D*(\check{c} + i\check{c} + \check{c}a)}{z*r} * O \tag{59}$$

V enačbi (59) je EF ekonomski učinek, D dohodek, $\check{c}$ izgubljeni delovni čas zaradi fluktuacije, $\check{c}a$ izgubljeni čas čakanja do dopolnitve delovnega časa, $i\check{c}$ je izgubljeni čas, ki je potreben za sprejem, izobraževanje in odpoved delavca, z povprečno število zaposlenih med letom, r letni neto fond delovnih ur enega zaposlenega in O število flukturirajočih delavcev med letom.

Tziner & Birati (1996, str. 166) vzpostavita model, v katerem so stroški grupirani v 3 skupine:

- **direktni stroški procesa zamenjave zaposlenega**, kamor spadajo: rekrutiranje, najemanje, šolanje in vključevanje (socializacija) v delovno okolje;
- **posredni stroški** kot posledica prekinitve v delovnem procesu in
- **finančna vrednost učinka na produktivnost** kot posledica padca morale preostalih zaposlenih v primeru odpuščanja.

Direktni stroški D po Tziner in Birati so sestavljeni iz:

- razlika med stroški zaposlitve novega delavca $R(m)$ v primerjavi z vsemi izplačili odhajajočega delavca $R(o)$:

$$C = \sum_{z=i}^{t} \frac{R(m) - R(o)}{(1+i)^z} \tag{60}$$

, V zgornji enačbi C predstavlja razliko v stroških za celotno obdobje v letih, v katerem je bilo od delavca v odhodu pričakovati normalne rezultate, če ne bi odšel (t), i pa je strošek podjetja izražen v denarju. Tako bo C pozitiven (izguba podjetja), če bo $R(m) - R(o) > 0$ in negativen

(prihranek podjetja), če bo $R(m) - R(o) < 0$;

- vsi stroški povezani z rekrutiranjem novega zaposlenega, ki vključujejo oglaševanje in proces selekcije S,
- direktni stroški šolanja novega zaposlenega T in
- stroški procesa vključitve novega zaposlenega v delovni proces, dokler ne postane polno produktiven, vključujoč čas nadrejenih in sodelavcev za socializacijo novega zaposlenega U.

Direktni stroški so torej:

$$D = C + S + T + U \tag{61}$$

Neposredni stroški (I) predstavljajo dodatne stroške, kot posledica odhoda zaposlenega:

- denarna nadomestila za dodatno delo preostalih zaposlenih in/ali začasni najem zunanjih zaposlenih za čas zamenjave zaposlenega O,
- finančna vrednost zaradi znižanja proizvodnje ali izgube strank, kot posledica odhoda zaposlenega F do polne produktivnosti novega zaposlenega in
- učinek fluktuacije na moralo M, v primeru odhoda dobrega zaposlenega. V ekstremnih primerih lahko odhod dobrega zaposlenega sproži plaz odhodov drugih dobrih zaposlenih.

Indirektni stroški so:

$$I = O + F + M \tag{62}$$

Novo zaposleni v povprečju ostanejo v podjetju manj časa od obstoječih zaposlenih. Razlog je v tem, da novo zaposleni šele po določenem času dela v podjetju pridobi dovolj informacij o samem podjetju, delu, ki je zahtevano od njega in . Poleg tega pa so novo zaposleni bolj izpostavljeni preverjanju in strožjemu nadzoru v začetnem obdobju po zaposlitvi. Zato je verjetnost odhoda novo zaposlenega višja od verjetnosti odhoda dlje časa zaposlenega v podjetju. Verjetnost odhoda lahko predstavimo s faktorjem fluktuacije f, ki predstavlja dodaten strošek fluktuacije in je za vsako podjetje specifično. Oceni ga lahko samo računovodstvo na podlagi preteklih stroškov in statistik.

Skupen strošek fluktuacije L po Tziner in Biratiju predstavlja seštevek neposrednih in posrednih stroškov pomnoženih s faktorjem fluktuacije:

$$L = (D + I)\,(1 + f) = (C + S + T + U + O + F + M)\,(1 + f) \tag{63}$$

Prikazan izračun stroška fluktuacije v tem delu ni upoštevan zaradi poenostavitve modela in majhnega vpliva na končni rezultat.

PRILOGA 7: Tabele parametrov za izračun optimalnih stroškov

Tabela 4: Parametri za izračun optimalnih stroškov

Opis	**Parameter**	**Vrednost**	**Standardna vrednost ali obseg**
Število elementarnih funkcionalnosti	m		
Število poslovnih procesov in funkcij	P		
Čas, potreben za upravljanje enega incidenta	Z_{1inc}		od 1/32 do 1/16
Verjetnost nastanka incidenta glede na število interakcij	p_{inc}		ocena: 0,05‰
Čas, potreben za upravljanje ene težave	Z_{1prob}		
Delež težav glede na število incidentov	d_{prob}		
Pogostost izvajanja preventivnih aktivnosti	ϕ		
Povprečno trajanje izvajanja ene preventivne aktivnosti	Z_{1prev}		
Delež časa za nadzor ene storitve	Z_{1nadz}		
Faktor zahtevane razpoložljivosti	R_f		
Število interakcij poslovnih procesov in funkcij	$\sum_{i=1}^{P} I_{p_i}$		
Faktor avtomatizacije	F_A		
Cena licence na povezavo	C_l		
Cena dela upravljanja incidentov	C_{inc}		
Cena dela upravljanja težav	C_{prob}		
Cena dela izvajanja preventivnih aktivnosti	C_{prev}		
Cena dela upravljanja in nadziranja storitev	C_{nadz}		
Cena dela upravljanja storitev	C_{mng}		

Za izračun optimalnih stroškov je potrebno izpolniti vse vrednosti v tabeli 4.

Tabela 5: Seznam poslovnih procesov in funkcij v Telekomu

Naziv procesa ali funkcije	**Število interakcij**	**Naziv procesa ali funkcije**	**Število interakcij**
Analiza zahtev za nove storitve	0,604	Zajem pomanjkljivosti obstoječih storitev	2,198
Pridobitev odobritve za investicijo v novo storitev	0,44	Zasnova zmogljivosti nove storitve	0,352
Zagotovitev podpore in obratovanja storitve	0,44	Upravljanje dobave kakovosti storitve	0,44
Upravljanje predaje storitve v obratovanje	1,319	Analiza zahtev za vire	26,374
Zajem pomanjkljivosti obstoječih virov	13,187	Pridobitev odobritve za investicijo v nov vir	10
Zasnova zmogljivosti novega vira	17,397	Zagotovitev podpore in obratovanja vira	1,74
Upravljanje dobave kakovosti vira	10	Upravljanje predaje vira v obratovanje	40
Upravljanje inventarja storitev	1211	Podpora upravljanju težav na storitvah	112
Podpora konfiguraciji in aktivaciji storitev	12	Podpora upravljanju kakovosti storitev	24
Upravljanje performans virov	116,667	Upravljanje težav na virih	3916,667
Zbiranje in distribucija podatkov o virih	31	Upravljanje inventarja virov	1715
Upravljanje logistike	2950	Zasnova rešitev	1025
Določanje parametrov storitev	1025	Spremljanje in upravljanje provizioniranja storitve	3075

se nadaljuje

Tabela 5: Seznam poslovnih procesov in funkcij v Telekomu (nad.)

Naziv procesa ali funkcije	**Število interakcij**	**Naziv procesa ali funkcije**	**Število interakcij**
Rezervacija in namestitev vira	641,667	Konfiguracija in aktivacija vira	683,333
Preveritev vira	133,333	Spremljanje in upravljanje provizioniranja vira	683,333
Poročanje o provizioniranju vira	342	Zapiranje naročila vira	444
Izdaja naročila vira	403	Odstranitev vira	135
Upravljanje zahtev dobaviteljev in partnerjev	644	Prijava težave na storitvi	993
Diagnoza težave na storitvi	2907	Odprava težave na storitvi	1032
Spremljanje in upravljanje težave na storitvi	1032	Zapiranje prijave težave na storitvi	1032
Pregled in analiza težav na storitvah	335	Poročanje o težavah na storitvah	7
Nadziranje kakovosti storitev	6	Analiza kakovosti storitev	2
Dvig kakovosti storitev	39	Poročanje o kakovosti storitev	1
Prijava padca kakovosti storitve	3	Spremljanje in upravljanje prijave o padcu kakovosti storitev	27
Zaključitev prijave padca kakovosti storitve	3,5	Pregled in analiza težav na virih	125
Lokaliziranje težave na viru	125	Prijava težave na viru	17
Spremljanje in upravljanje prijave težave na viru	17	Odprava težave na viru	19

se nadaljuje

Tabela 5: Seznam poslovnih procesov in funkcij v Telekomu (nad.)

Naziv procesa ali funkcije	**Število interakcij**	**Naziv procesa ali funkcije**	**Število interakcij**
Zaključitev prijave težave na viru	19	Izdelava poročila o težavi na viru	19
Spremljanje delovanja vira	0,19	Analiza delovanja vira	0,19
Nadzor delovanja vira	0,333	Poročanje o delovanju vira	0,333
Prijava padca performans vira	8	Spremljanje in upravljanje prijave o padcu performans vira	10
Odprava vzroka za padec performans vira	10	Zapiranje prijave o padcu performans vira	10
Administracija in konfiguriranje upravljanja delovne sile	30	Upravljanje življenjskega cikla delovnega naloga	946
Upravljanje urnikov in aktivnosti	267	Planiranje in napovedovanje upravljanja delovne sile	4
Poročanje o upravljanju delovne sile	0,333		
		Skupaj število procesov P:	**69**
		Skupaj število interakcij $\sum_{i=1}^{P} I_{p_i}$:	**28.844,37**

Številke so zaradi varovanja poslovne skrivnosti, pomnožene s skritim faktorjem.

Tabela 6: Število elementarnih funkcionalnosti po področjih v Telekomu

Naziv področja	**Število funkcionalnosti**
Inventar	732
Zagotavljanje storitev	417
Snovanje storitev in omrežja	165
Aktivacija	118
Upravljanje terenske delovne sile	269
Upravljanje nivoja kakovosti storitev	146
Skupaj število funkcionalnost m:	**1847**

Številke so zaradi varovanja poslovne skrivnosti, pomnožene s skritim faktorjem.

Tabela 7: Parametri za OSS področje v Telekomu Slovenije

Opis	**Parameter**	**Vrednost**
Število elementarnih funkcionalnosti	m	1847
Število poslovnih procesov in funkcij	P	69
Čas, potreben za upravljanje enega incidenta	Z_{1inc}	0,0625
Verjetnost nastanka incidenta glede na število interakcij	p_{inc}	0,004
Čas, potreben za upravljanje ene težave	Z_{1prob}	0,25
Delež težav glede na število incidentov	d_{prob}	0,05
Pogostost izvajanja preventivnih aktivnosti	ϕ	0,26
Povprečno trajanje izvajanja ene preventivne aktivnosti	Z_{1prev}	0,75
Delež časa za nadzor ene storitve	Z_{1nadz}	0,083
Faktor zahtevane razpoložljivosti	R_f	3,85
Število interakcij poslovnih procesov in funkcij	$\sum_{i=1}^{P} I_{p_i}$	28.844,37
Faktor avtomatizacije	F_A	0,45
Cena licence na povezavo	C_l	0,45 €
Cena dela upravljanja incidentov	C_{inc}	96,55 €
Cena dela upravljanja težav	C_{prob}	144,83 €
Cena dela izvajanja preventivnih aktivnosti	C_{prev}	112,64 €
Cena dela upravljanja in nadziranja storitev	C_{nadz}	96,55 €
Cena dela upravljanja storitev	C_{mng}	112,64 €

PRILOGA 8: Izpeljava optimalnih stroškov

Iz enačbe (51) sledi:

$$\begin{aligned}
S_d &= R_f\ (C_{inc}\ Z_{inc} + C_{prob}\ Z_{prob} + C_{nadz}\ (1-F_A)\ Z_{nadz} + \\
&+ C_{mng}\ Z_{mng}) + C_{prev}\ \frac{3}{4} Z_{prev} \\
&= R_f\ (C_{inc}\ \frac{Z_{1inc}\ p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}}\ \sum_{i=1}^{P} I_{p_i} + \\
&+ C_{prob} \left(d_{prob}\ Z_{1prob}\ \frac{p_{inc}\ (5-4G)}{1-G+G\frac{m}{P}}\ \sum_{i=1}^{P} I_{p_i} \right) + \\
&+ C_{nadz}\ (1-F_A) Z_{1nadz} ((G+1)\,m + (1-G)\,P) + \\
&+ C_{mng}\ \frac{1}{40} (Gm + (1-G)P)) + \\
&+ C_{prev}\ \frac{3}{4} (Gm + (1-G)P)\ \left(\phi\ Z_{1prev}\right)
\end{aligned} \tag{64}$$

Prva pomožna enačba za poenostavitev:

$$\begin{aligned}
&Gm + (1-G)\,P = \\
&= G\,(m-P) + P
\end{aligned} \tag{65}$$

Druga pomožna enačba za poenostavitev:

$$\begin{aligned}
&(G+1)\,m + (1-G)\,P = \\
&= Gm + m + P - GP = \\
&= G\,(m-P) + m + P
\end{aligned} \tag{66}$$

Tretja pomožna enačba za poenostavitev:

$$\begin{aligned}
&1 - G\left(1 - \frac{m}{P}\right) = \\
&= 1 - G + G\frac{m}{P} = \\
&= G\left(\frac{m}{P} - 1\right) + 1
\end{aligned} \tag{67}$$

Sedaj izločim granularnost G iz posameznih členov:

$$\begin{aligned}
S_d = & \frac{5\,R_f\,C_{inc}\,Z_{1inc}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} - \frac{4\,R_f\,C_{inc}\,Z_{1inc}\,p_{inc}\sum_{i=1}^{P} I_{p_i}\,G}{G\left(\frac{m}{P}-1\right)+1} + \\
& + \frac{5\,R_f\,C_{prob}\,d_{prob}\,Z_{1prob}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} - \\
& - \frac{4\,R_f\,C_{prob}\,d_{prob}\,Z_{1prob}\,p_{inc}\sum_{i=1}^{P} I_{p_i} Z_{1prob}\,G}{G\left(\frac{m}{P}-1\right)+1} + \\
& + G\,(m-P)\,C_{nadz}\,R_f\,(1-F_A)\,Z_{1nadz} + (m+P)\,C_{nadz}\,R_f\,(1-F_A)\,Z_{1nadz} + \\
& + G\,(m-P)\,C_{mng}\,R_f\,\frac{1}{40} + C_{mng}\,R_f\,\frac{1}{40}P + \\
& + G\left(\frac{3}{4}\,C_{prev}\,\phi\,Z_{1prev}\,(m-P)\right) + \frac{3}{4}\,C_{prev}\,P\,\phi\,Z_{1prev}
\end{aligned} \tag{68}$$

Drugi in četrti člen enačbe (68) lahko seštejem med seboj in izločim granularnost G:

$$\begin{aligned}
S_d = & \frac{5\,R_f\,C_{inc}\,Z_{1inc}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} + \frac{5\,R_f\,C_{prob}\,d_{prob}\,Z_{1prob}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} + \\
& + G\frac{\left(-4\,R_f\,p_{inc}\sum_{i=1}^{P} I_{p_i}\right)\left(C_{inc}\,Z_{1inc} + C_{prob}\,d_{prob}\,Z_{1prob}\right)}{G\left(\frac{m}{P}-1\right)+1} + \\
& + G\,(m-P)\,C_{nadz}\,R_f\,(1-F_A)\,Z_{1nadz} + (m+P)\,C_{nadz}\,(1-F_A)\,Z_{1nadz} + \\
& + G\,(m-P)\,C_{mng}\,R_f\,\frac{1}{40} + C_{mng}\,R_f\,\frac{1}{40}P + \\
& + G\left(\frac{3}{4}\,C_{prev}\,\phi\,Z_{1prev}\,(m-P)\right) + \frac{3}{4}\,C_{prev}\,P\,\phi\,Z_{1prev}
\end{aligned} \tag{69}$$

Za celotne stroške obratovanja IS prištejem še stroške licenčnin:

$$\begin{aligned}
S = & S_d + S_l = \\
= & \frac{5\,R_f\,C_{inc}\,Z_{1inc}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} + \frac{5\,R_f\,C_{prob}\,d_{prob}\,Z_{1prob}\,p_{inc}\sum_{i=1}^{P} I_{p_i}}{G\left(\frac{m}{P}-1\right)+1} + \\
& + G\frac{\left(-4\,R_f\,p_{inc}\sum_{i=1}^{P} I_{p_i}\right)\left(C_{inc}\,Z_{1inc} + C_{prob}\,d_{prob}\,Z_{1prob}\right)}{G\left(\frac{m}{P}-1\right)+1} + \\
& + G\,(m-P)\,C_{nadz}\,R_f\,(1-F_A)\,Z_{1nadz} + (m+P)\,C_{nadz}\,(1-F_A)\,Z_{1nadz} + \\
& + G\,(m-P)\,C_{mng}\,R_f\,\frac{1}{40} + C_{mng}\,R_f\,\frac{1}{40}P + \\
& + G\left(\frac{3}{4}\,C_{prev}\,\phi\,Z_{1prev}\,(m-P)\right) + \frac{3}{4}\,C_{prev}\,P\,\phi\,Z_{1prev} + \\
& + C_l\left((4P-4m)\,G^2 + (5m-9P)\,G + 5P\right)F_A + S_h
\end{aligned} \tag{70}$$

Za ugotovitev točke z minimalnimi stroški v odvisnosti od granularnosti G najprej izračunam odvod $f'(G)$:

$$\begin{aligned} f'(G) = & - \frac{\left(5\ R_f\ C_{inc}\ Z_{1inc}\ p_{inc} \sum_{i=1}^{P} I_{p_i}\right)\left(\frac{m}{P}-1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^2} - \\ & - \frac{\left(5\ R_f\ C_{prob}\ d_{prob}\ Z_{1prob}\ p_{inc} \sum_{i=1}^{P} I_{p_i}\right)\left(\frac{m}{P}-1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^2} - \\ & - \frac{\left(4\ R_f\ p_{inc} \sum_{i=1}^{P} I_{p_i}\right)\left(C_{inc}\ Z_{1inc} + C_{prob}\ d_{prob}\ Z_{1prob}\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^2} + \\ & + (m-P)\, C_{nadz}\ R_f\,(1-F_A)\, Z_{1nadz} + \\ & + (m-P)\, C_{mng}\ R_f\ \frac{1}{40} + \\ & + \frac{3}{4}\, C_{prev}\ \phi\ Z_{1prev}\,(m-P) + \\ & + 2\ G\ C_l\,(4P-4m) + C_l\ F_A\,(5m-9P) \end{aligned} \tag{71}$$

Za poenostavitev označim člene 4, 5, 6 in 8 enačbe (71) z α, v prvih treh členih pa po vrsti imenovalce skupaj s predznakom z β, Γ in Δ, za dodatno poenostavitev označim izraz $\frac{m}{P}-1$ s τ ter izraz $2\ C_l\,(4P-4m)$ s ψ ter ponovno zapišem enačbo:

$$f'(G) = \frac{\beta}{(G\tau+1)^2} + \frac{\Gamma}{(G\tau+1)^2} + \frac{\Delta}{(G\tau+1)^2} + \alpha + G\ \psi \tag{72}$$

Člene s skupnim imenovalcem seštejem:

$$f'(G) = \frac{\beta+\Gamma+\Delta}{(G\tau+1)^2} + \alpha + G\ \psi \tag{73}$$

Izraz $\beta+\Gamma+\Delta$ lahko nadomestim z znakom ω:

$$f'(G) = \frac{\omega}{(G\tau+1)^2} + \alpha + G\ \psi \tag{74}$$

Pri neničelni vrednosti izraza $(G\tau+1)^2$ lahko za iskanje ničel celotno enačbo pomnožim z

imenovalcem prvega člena enačbe in dobim:

$$\begin{aligned} f'(G) &= 0 \\ 0 &= \omega + (\alpha + G\psi)(G\tau+1)^2 \\ &= \omega + (\alpha + G\psi)(G\tau+1)(G\tau+1) \\ &= \omega + (\alpha + G\psi)\left(G^2\tau^2 + 2\,G\tau + 1\right) \\ &= \omega + G^3\tau^2\psi + 2\,G^2\tau\psi + G\psi + G^2\alpha\tau^2 + 2\,G\alpha\tau + \alpha \\ &= G^3\tau^2\psi + G^2\left(\tau\psi + \alpha\tau^2\right) + G(\psi + 2\,\alpha\tau) + \omega + \alpha \end{aligned} \tag{75}$$

V tej točki vstavim vrednosti iz tabele 4 in izračunam ničle polinoma, kjer so potencialni ekstremi.

Za preveritev, ali je v danih točkah maksimum, minimum, prevoj ali nedefinirana vrednost funkcije, izračunam še drugi odvod funkcije po granularnosti:

$$\begin{aligned} f''(G) = &\frac{\left(5\,R_f\,C_{inc}\,Z_{1inc}\,p_{inc}\sum_{i=1}^{P} I_{p_i}\right)\left(\frac{m}{P}-1\right)\;2\;\left(\frac{m}{P}-1\right)\left(G\left(\frac{m}{P}-1\right)+1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^4} \\ &+ \frac{\left(5\,R_f\,C_{prob}\,d_{prob}\,Z_{1prob}\,p_{inc}\sum_{i=1}^{P} I_{p_i}\right)\left(\frac{m}{P}-1\right)\;2\;\left(\frac{m}{P}-1\right)\left(G\left(\frac{m}{P}-1\right)+1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^4} \\ &+ \frac{\left(4\,R_f\,p_{inc}\sum_{i=1}^{P} I_{p_i}\right)\left(C_{inc}\,Z_{1inc}+C_{prob}\,d_{prob}\,Z_{1prob}\right)\;2\;\left(\frac{m}{P}-1\right)\left(G\left(\frac{m}{P}-1\right)+1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^4} \\ &+ 2\,C_l\,(4P-4m) \end{aligned} \tag{76}$$

Kar je lepše zapisano kot:

$$\begin{aligned} f''(G) = &\frac{R_f\,p_{inc}\sum_{i=1}^{P} I_{p_i}\;2\;\left(\frac{m}{P}-1\right)\left(G\left(\frac{m}{P}-1\right)+1\right)}{\left(G\left(\frac{m}{P}-1\right)+1\right)^4}\;\left(\left(5\,C_{inc}\,Z_{1inc}\left(\frac{m}{P}-1\right)\right)\right. \\ &\left.+\left(5\,C_{prob}\,d_{prob}\,Z_{1prob}\left(\frac{m}{P}-1\right)\right)+\left(4\left(C_{inc}\,Z_{1inc}+C_{prob}\,d_{prob}\,Z_{1prob}\right)\right)\right) \\ &+ 8\,C_l\,(P-m) \end{aligned} \tag{77}$$

V posamezni točki je lokalni maksimum, če je vrednost drugega odvoda manjša od 0 in lokalni minimum, če je vrednost drugega odvoda večja od 0. Če je vrednost drugega odvoda enaka 0, potem v tej točki ni niti lokalni maksimum, niti lokalni minimum.

PRILOGA 9: Excel preglednica za pomoč pri izračunu

 (Dvojni klik na ikono)