Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Jaka Močnik

POVEČANJE RAZPOLOŽLJIVOSTI PONUDNIKOV STORITEV S SPROŠČANJEM NEFUNKCIONALNIH ZAHTEV

magistrsko delo

mentor: prof. dr. Borut Robič

Ljubljana, 2007

Izjava

Magistrsko delo z naslovom "Povečanje razpoložljivosti ponudnikov storitev s sproščanjem nefunkcionalnih zahtev" je v celoti rezultat mojega dela pod mentorstvom prof. dr. Boruta Robiča. Vsi dosežki, na katerih sem gradil, so dosledno označeni z navedbo virov literature, pomoč drugih sodelavcev pa je v celoti navedena v zahvali.

Jaka Močnik

Barbari. Ker jê.

Zahvala

Prof. dr. Borutu Robiču za mentorstvo in ustrezno mero potrpljenja z mano.

Prof. dr. Karlu M. Göschki s Tehniške univerze na Dunaju za številne pripombe, kritike in nasvete glede modela sistema.

Piotru Karwaczyńskemu z Wroclawske univerze za tehnologijo za pomoč pri načrtovanju in izvedbi prototipa ter za dostop do omrežja PlanetLab pri preizkušanju prototipa.

Marku Novaku za pomoč pri izvedbi.

Dr. Roku Sosiču in prof. dr. Andreju Brodniku za pomoč pri prevodih novih izrazov v slovenski jezik.

Podjetju XLAB za raziskovalnemu delu prijazno delovno okolje.

Hvala!

Raziskovalno delo je delno financirala Evropska komisija v okviru raziskovalnega projekta DeDiSys (*Dependable Distributed Systems*, št. pogodbe FP6-2003-IST-2-004152).

POVEČANJE RAZPOLOŽLJIVOSTI PONUDNIKOV STORITEV S SPROŠČANJEM NEFUNKCIONALNIH ZAHTEV

Jaka Močnik

Mentor: prof. dr. Borut Robič

Povzetek

Storitveno usmerjene arhitekture postajajo vedno bolj pomemben pristop k povezovanju raznolikih računalniških sistemov v široko porazdeljenih omrežjih. Ob evforičnem uvajanju tega pristopa v poslovna okolja se čedalje hitreje širi t.i. razpoka v trdnosti (ang. dependability gap): raziskave in rezultati, ki bi omogočali načrtovanje trdnih storitveno usmerjenih arhitektur capljajo za tehničnimi novostmi, ki omogočajo čedalje bolj preprosto gradnjo tovrstnih sistemov.

V pričujočem delu predstavimo paradigmo za povečanje razpoložljivosti ponudnikov, te pomembne lastnosti trdnih računalniških sistemov, na račun zadovoljitve nefunkcionalnih zahtev odjemalca. Razvijemo način opisa ponujanih in zahtevanih nefunkcionalnih lastnosti ter metodo za iskanje ponudnikov, ki kar najbolj zadovoljijo odjemalčeve zahteve. S tem, ko omogočimo opis več naborov nefunkcionalnih lastnosti, ki različno zadovoljijo odjemalca, povečamo nabor ustreznih ponudnikov in tako povečamo razpoložljivost. Odjemalcu omogočimo, da zadovoljitev zahtev zamenja za razpoložljivost. Predstavimo abstrakten model razširitev vmesnega sloja za gradnjo porazdeljenih sistemov, udejanjenje tega modela s pomočjo prosto dostopnega vmesnega sloja in izvedbo prototipa. Prototip preizkusimo in rezultate primerjamo z danes uveljavljenimi načini strežbe.

KLJUČNE BESEDE: storitveno usmerjena arhitektura, SOA, spletne storitve, trdnost, razpoložljivost, ujemanje, pogodba o nivoju storitve, SLA

INCREASING AVAILABILITY OF SERVICE PROVIDERS BY RELAXING NON-FUNCTIONAL REQUIREMENTS

Jaka Močnik

Supervisor: prof. dr. Borut Robič

Abstract

Service-oriented architectures are becoming an important approach to interconnection of heterogeneous computer systems in wide area networks. Euphoric implementation of this approach in business environments results in the broadening of the dependability gap: research and results that would enable design of dependable service oriented architectures are well behind the constant influx of technological novelties that allow for very simple building of such systems.

This work presents a new paradigm of increasing the availability of service providers, an important attribute of dependable computer systems, at the expense of satisfaction of the service consumers' non-functional requirements. A method for description of provided and requested non-functional properties and a method for discovering the providers that satisfy the consumer best are developed. Enabling the consumer to describe multiple alternatives of non-functional requirements widens the set of (more or less) satisfactory providers, thus increasing the availability of the providers. The consumer is thus allowed to trade the satisfaction of its requirements for availability if necessary. An abstract model of the relevant extensions to service-oriented middleware is presented, as well as an instantiation of this model by employing freely available web service middleware. A prototype is implemented and tested. Results are compared with service provisioning methods common today.

KEYWORDS: service-oriented architecture, SOA, web services, dependability, availability, matchmaking, service-level agreement, SLA

Kazalo

Slike

Tabele

Izpisi izvorne kode

Poglavje 1

Uvod

V uvodnem poglavju najprej predstavimo pojem porazdeljenega računalniškega sistema in si ogledamo nekaj primerov. Opišemo pristope k gradnji porazdeljenih sistemov ter naredimo kratko primerjavo lastnosti, pomembnih za naše delo. Zaključimo z razlago koncepta trdnosti porazdeljenega sistema ter pojasnimo kaj trdnost ogroža in kako jo zagotovimo.

1.1 Porazdeljeni računalniški sistemi

Uporaba geografsko porazdeljenih računalniških virov je relativna novost. Večji del zgodovine sodobnega računalništva, od štiridesetih let dvajsetega stoletja, ko nastanejo prvi elektronski računalniki, ki sledijo modelu von Neumannove arhitekture, do sredine osemdesetih, so računalniki veliki in predvsem dragi, tako v smislu nakupa kot vzdrževanja in delovanja. Velika večina organizacij premore le enega ali nekaj miniračunalnikov, ki delujejo izolirano, neodvisno.

V osemdesetih letih dvajsetega stoletja se to stanje prične spreminjati predvsem pod vplivom široke uporabe dveh pomembnih tehnoloških novosti. Prva je prihod močnih, a poceni mikroprocesorjev, ki v naslednjih dveh desetletjih dobesedno postavijo računske zmožnosti prejšnjih velikih računalnikov na uporabnikovo mizo. Računalnik postane orodje slehernika. Druga, neodvisna inovacija je računalniško omrežje visoke hitrosti. Lokalna omrežja (ang. local area network, LAN) omogočijo povezovanje stotin računalnikov znotraj posamezne organizacije. Izmenjava manjših količin podatkov med računalniki postane trenutno, neopazno opravilo, ki traja le nekaj mikrosekund. Primerljive hitrosti v omrežjih na širokih območjih (ang. wide-area network, WAN) so zavoljo visoke cene tovrstnega povezovanja sprva domena večjih organizacij, a do druge polovice devetdesetih postane širokopasovni dostop do teh omrežij – in s tem enakopravno vključevanje vanje – dostopen posameznikom.

Zavoljo opisanega razvoja je danes ne le mogoče, temveč celo enostavno povezati veliko število široko, celo globalno porazdeljenih računalnikov v *porazdeljen* računalniški sistem. S pridevkom *porazdeljen* tovrstne sisteme jasno ločimo od poprejšnjih *centraliziranih* računalniških sistemov: slednje sestavlja en sam računalnik in morebiti še množica terminalov, ki služijo dostopu oddaljenih uporabnikov.

Na mestu je kratka in jedrnata definicija porazdeljenega računalniškega sistema. Sposodimo si jo pri Tanenbaumu (Tanenbaum in Steen, 2002):

Definicija 1.1.1 *Porazdeljen računalniški sistem je zbirka* neodvisnih *računalnikov, ki jih uporabnik zaznava kot* enoten *sistem.*

Ponazorimo to suhoparno definicijo z nekaj primeri.

Primer 1.1.1 *Kot prvi primer naj služi računska gruča, ki jo sestavlja množica enakih računalnikov. Neposreden dostop uporabnikov do posameznih računalnikov ni mogoč. Po drugi strani pa lahko uporabnik gručo izkoristi za izvajanje računsko zahtevnih opravil, za katera je njegova delovna postaja preskromna. Opravilo podtakne gruči v izvajanje s pomočjo Condorja (Thain in sod., 2002) ali katerega od sorodnih sistemov za upravljanje z računskim bremenom (PBS, Bayucan in sod. (1999); LSF, Zhou (1992)). Posamezno opravilo je lahko običajen, zaporeden program, ki ga bo Condor dodelil primernemu vozlišču v gruči, lahko pa gre za vzporeden program, ki bo tekel na več vozliščih gruče. Uporabnik gručo s tovrstnim vmesnikom zaznava kot enoten sistem.*

Primer 1.1.2 *Nadaljujmo s primerom široko porazdeljenega sistema: pospeševalnik delcev med poskusi proizvaja izjemno veliko količino podatkov (poizkusi v CERN-ovem pospeševalniku LHC naj bi proizvedli več petazlogov podatkov letno), ki se shranjuje v lokalni podatkovni bazi. Dostop do podatkov je omogočen fizikom z različnih univerz in raziskovalnih inštitutov po svetu s pomočjo storitve GridFTP (Foster, 2006). Analiza zahteva izjemno velike računske zmožnosti: v ta namen uporabimo superračunalnik na eni izmed univerz in več računskih gruč na ostalih; vse te računske vire združimo v eno samo metagručo; sodelujočim organizacijam je dostop do tako združenih računskih virov omogočen s pomočjo Condorja, podobno kot v primeru 1.1.1.*

Primer 1.1.3 *Za konec se ozrimo še na svetovni splet: ta nam ponuja porazdeljeno zbirko dokumentov. Objava dokumentov je preprosta, potrebujemo le spletni strežnik z dostopom do dokumenta, s čimer mu implicitno podelimo naslov URL (ang. Uniform Resource Locator), s pomočjo katerega ga lahko doseže poljuben odjemalec. V resnici se uporabnik zaveda porazdelitve dokumentov in spleta ne zaznava kot enoten sistem kot zahteva definicija 1.1.1, vendar lahko v ta namen služi vmesnik, kakršen je recimo iskalnik Google.*

Namen porazdeljenega računalniškega sistema je, kot lahko ugotovimo iz gornjih primerov, predvsem omogočiti uporabnikom preprost dostop do oddaljenih računalniških virov in delitev lokalnih virov z ostalimi uporabniki. Med vire štejemo procesorske zmogljivosti, pomnilnik, podatke, datoteke, različne periferne enote ipd. Zmožnost deljenja (in seveda tudi združevanja, sočasne uporabe) porazdeljenih virov ima mnogo prednosti: je cenejša kot prisotnost enakih virov na več mestih, kar velja še zlasti za drago opremo, npr. superračunalnike; navidezno združevanje in raba množice porazdeljenih virov omogoča dostop do večje količine virov; deljenje virov je tudi osnova za napredne oblike oddaljenega sodelovanja s pomočjo telekonferenc, prenosa datotek, ogleda in rabe namizja oddaljenega računalnika in sočasnega dela več uporabnikov na istem dokumentu; nenazadnje so porazdeljeni sistemi tudi osnova za e-poslovanje, ki postaja vedno pomebnejši del sodobnih gospodarstev, zlasti na področju trženja storitev.

Porazdeljena sistema v primerih 1.1.2 in 1.1.3 sta:

- velika: računalniki, ki ju sestavljajo, so široko, celo globalno porazdeljeni,
- dinamična: članstvo (nabor sodelujočih računalnikov in virov) in lastnosti članov se hitro in pogosto spreminjajo,
- raznolika: računalniki se močno razlikujejo v strojni in programski opremi,
- odprta: sodelujoči računalniki pripadajo različnim organizacijam, možno pa je pridruževanje novih organizacij. Sodelovanje poteka na podlagi splošno sprejetih, odprtih standardov za komunikacijo, kar ni nujno res v primeru 1.1.1.

Tovrstnim sistemom - velikim, dinamičnim, raznolikim in odprtim - se posvečamo v pričujočem delu. Spričo velikega števila pridevnikov in pomanjkanja uveljavljenega izraza zanje bomo v tem delu kazali nanje z besedno zvezo *velik porazdeljen sistem*.

Oglejmo si, kako se lotiti gradnje porazdeljenega sistema.

1.2 Gradnja porazdeljenih sistemov

Glede na uporabljeno strojno opremo ločimo dva tipa porazdeljenih računalniških sistemov:

- *enovite* (homogene) porazdeljene računalniške sisteme tvorijo *enaki* računalniki s (pretežno) enako strojno in sistemsko programsko opremo, medtem ko
- *raznolike* (heterogene) porazdeljene računalniške sisteme sestavljajo računalniki, ki se med seboj močno razlikujejo tako po strojni kot sistemski programski opremi.

Podobno kot operacijski sistem posameznega računalnika poskrbi za abstraktno, visokonivojsko predstavitev računalniških virov uporabniku[1], tudi v porazdeljenih računalniških sistemih potrebujemo skupno sistemsko programsko opremo, ki skrbi za delovanje in upravljanje sistema, od njene izbire pa so v veliki meri odvisne lastnosti nastalega sistema. Med vrstami sistemske programske opreme za porazdeljene računalniške sisteme ločimo porazdeljene operacijske sisteme, omrežne operacijske sisteme in vmesni sloj (Tanenbaum in Steen, 2002).

1.2.1 Porazdeljeni operacijski sistemi

Porazdeljen operacijski sistem (ang. distributed operating system, DOS) predstavi množico tesno povezanih (ang. tightly coupled) računalnikov kot en sam večprocesorski računalnik z deljenim pomnilnikom. Deljenje pomnilnika je izvedeno programsko kot ena od storitev za upravljanje porazdeljenega sistema in je integrirano v lokalno jedro operacijskega sistema vsakega računalnika v omrežju. Zavoljo tega morajo vsa vozlišča uporabljati enak lokalen operacijski sistem in procesorsko arhitekturo (slednje implicira enako predstavitev podatkov v pomnilniku): DOS so primerni le za enovite porazdeljene računalniške sisteme, saj se zanašajo na skupne značilnosti vseh računalnikov.

Sem prištevamo različne oblike operacijskih sistemov, ki ponujajo *enotno sistemsko sliko* (ang. single system image). Tovrstna operacijska sistema sta derivata Linux jedra Mosix (Barak in La'adan, 1998) in Kerrighed (Morin in sod., 2004).

Porazdeljen operacijski sistem predstavlja skrajnost v upoštevanju načela predstavitve množice računalnikov kot enotnega sistema, ki smo ga podali v definiciji 1.1.1: na takšnem sistemu bo brez dodatnega dela tekel poljuben program, pisan za centraliziran računalniški sistem, in izkoriščal vse vire v porazdeljenem sistemu. Po drugi strani pa ta pristop krši načelo neodvisnosti računalnikov: računalniki v tovrstnih sistemih so tesno sklopljeni (zavoljo deljenja pomnilnika) in močno odvisni med sabo.

1.2.2 Omrežni operacijski sistemi

Omrežni operacijski sistem (ang. network operating system, NOS) se ne zanaša na enovitost računalnikov, temveč se uporablja v sistemih, zgrajenih iz množice računalnikov, ki se razlikujejo tako v strojni kot v sistemski programski opremi. Omrežni operacijski sistemi ponujajo storitve, ki uporabniku omogočajo dostop do oddaljenega računalnika in delo z njim. Med tovrstne storitve štejemo dostop do oddaljene ukazne lupine (telnet, secure shell) in dostop do oddaljenih datotek (FTP, secure copy, omrežni datotečni sistemi).

[1] V tem delu z besedo *uporabnik* običajno naslavljamo programerja, pisca programja za računalniški sistem, bodisi centraliziran ali porazdeljen, ne pa končnega uporabnika programja. V kolikor gre za slednjega, bo to izrecno poudarjeno.

Očitno so omrežni operacijski sistemi primitivnejši od porazdeljenih, saj pred uporabnikom ne skrijejo lokacije vira in načina dostopa do njega. Skrivanje kraja in dostopa sta pri rabi porazdeljenega sistema jako zaželeni lastnosti, saj naj bi se uporabnik ukvarjal predvsem z reševanjem svojega problema, ne pa s pogosto zapletenimi podrobnostmi dostopa do porazdeljenih virov. S tem omrežni operacijski sistemi tudi kršijo načelo enotnosti sistema iz definicije 1.1.1. Prednost je v njihovi odprtosti, saj so storitve dobro definirane in omogočajo, da se tovrstnemu sistemu pridruži skoraj vsak računalnik. Spričo visoke stopnje neodvisnosti posameznih vozlišč so omrežni operacijski sistemi tudi izjemno razširljivi: sistemu je preprosto dodati ali odvzeti posamezne računalnike in obvladovati njihove izpade.

1.2.3 Vmesni sloj

Vmesni sloj (ang. middleware) je dodaten sloj sistemske programske opreme med lokalnim operacijskim sistemom in porazdeljenim uporabniškim programjem. V tem pogledu je podoben preprostim storitvam omrežnih operacijskih sistemov, a s poenotenjem dostopa do virov in skrivanjem lokacije virov poskrbi, da pred uporabnikom skrije razlike v posameznih vozliščih in dejansko mesto, kjer se vir nahaja.

Uporaba vmesnega sloja za gradnjo porazdeljenih sistemov nam ponuja najboljše lastnosti obeh prej omenjenih pristopov: razširljivost in odprtost omrežnih operacijskih sistemov ter preprostost uporabe, značilno za porazdeljene operacijske sisteme.

Vmesni sloj je namenjen predvsem temu, da razvoj porazdeljenega programja čimbolj približa programerju, vajenemu razvoja monolitnega programja. V ta namen ponuja množico storitev, ki so namenjene skrivanju porazdeljene in raznolike narave okolja, v katerem bo tekel program, ter komunikaciji med posameznimi deli programja, ki se izvajajo na različnih računalnikih.

Vrste vmesnega sloja

Kot kriterij za razvrščanje bomo uporabili paradigmo, ki jo vmesni sloj uporablja za komunikacijo med deli porazdeljenega sistema. Običajno skuša vmesni sloj oponašati eno od paradigm, ki se uporabljajo za razvoj programja v centraliziranih računalniških sistemih, in s tem približati razvoj porazdeljenega programja programerju. Na osnovi tega kriterija prepoznamo tri večje družine vmesnih slojev za gradnjo porazdeljenih sistemov (Tanenbaum in Steen, 2002; Curry, 2004):

- *klic oddaljene procedure* (ang. remote procedure call, RPC) oponaša proceduralno programiranje. Oddaljeni deli programja se obnašajo kot moduli, katerih procedure kličemo podobno kot lokalne procedure: vmesni sloj poskrbi za primerno pakiranje

podatkov na strani kličočega sistema, prenos podatkov po omrežju in razpakiranje na strani klicanega sistema, s čimer skrije morebitne razlike v predstavitvi podatkov na obeh sistemih zavoljo različne strojne opreme. Primer tovrstnega vmesnega sloja je Sun RPC (XDR; RPC).

- *porazdeljeni predmeti* (ang. distributed objects) sledijo predmetno usmerjeni paradigmi programiranja. Vsak predmet se lahko izvaja na poljubnem računalniku v sistemu. Programer kliče metode oddaljenih predmetov na enak način kot bi klical metode lokalnih predmetov, vmesni sloj pa poskrbi za omrežno komunikacijo, podobno kot pri klicu oddaljene procedure. Tako kot je predmetno usmerjena paradigma danes najpogostejši pristop k razvoju programske opreme, so tudi izvedbe vmesnega sloja, ki temelji na tem pristopu, danes izjemno pogoste: sem lahko uvrstimo CORBO (Mowbray in Zahavi, 1995), Java RMI (Pitt in McNiff, 2001) in Microsoftovi tehnologiji COM (Brown, 2001) ter .Net Remoting (Rammer, 2002).

- *sporočilno usmerjen vmesni sloj* (ang. message-oriented middleware, MOM) temelji na pošiljanju sporočil med deli porazdeljenega sistema in ustreznih odzivih na prejeta sporočila. Za razliko od prej omenjenih modelov, klica oddaljene procedure in porazdeljenih predmetov, ki sta v svoji zasnovi sinhrona (pokličemo oddaljeno metodo in počakamo na rezultat), je programje, zasnovano na sporočilno usmerjenem vmesnem sloju, asinhrono: ni potrebe, da bi se pošiljatelj ustavil in čakal na rezultat. Kadar semantika to dovoljuje, lahko pošiljatelj nadaljuje z delom in rezultat obdela nekoč kasneje, ko ga prejme v ustreznem povratnem sporočilu. Asinhrona semantika je izjemno pomembna, saj omogoča vsem udeležencem v komunikaciji, da – kolikor je to le mogoče – ohranijo neodvisnost pri obdelavi podatkov: po pošiljanju sporočila lahko pošiljatelj nadaljuje z delom, neodvisno od stanja ostalih udeležencev. Tipičen primer sporočilno usmerjenega vmesnega sloja je Java Messaging Service (JMS) (Hapner in sod., 2002).

Primerjava vrst vmesnega sloja

Izčrpna obravnava in primerjava različnih vrst vmesnega sloja, njihovih prednosti in slabosti je primeren predmet samostojnega dela, tu pa bomo – v luči dejstva, da naše delo cilja na velike, globalno porazdeljene, dinamične in odprte sisteme – naredili le kratko primerjavo, ki bo osvetlila našo izbiro tehnologije: posvetili se bomo povezanosti (ang. coupling) sistemov, ki jih gradimo z omenjenimi vrstami vmesnega sloja.

Paradigmi klica oddaljene procedure in porazdeljenih predmetov sta zasnovani tako, da posamezni deli porazdeljenega sistema, bodisi programski moduli bodisi predmeti, med sabo sodelujejo neposredno. V sistemu, zasnovanem na CORBI, bo metoda enega predmeta neposredno klicala metodo iz vmesnika drugega, morebiti oddaljenega predmeta.

Na ta način dele porazdeljenega sistema *tesno povežemo* (ang. tight coupling), sprememba enega dela (npr. vmesnika enega od predmetov) pa vedno vpliva na druge dele (npr. predmete, ki uporabljajo omenjeni vmesnik), ki jih moramo ustrezno spremeniti.

Zavoljo tesno povezanih delov se tovrsten vmesni sloj izkaže za primernega predvsem za gradnjo manjših porazdeljenih sistemov, ki pripadajo eni sami organizaciji in zato obstaja možnost centralnega nadzora nad razvojem in vzdrževanjem porazdeljenega sistema in spremembami njegovih delov. Zelo zaželen je tudi centralen nadzor nad komunikacijsko infrastrukturo. Predmetno usmerjen vmesni sloj bomo uporabili za npr. gradnjo informacijskega sistema v eni organizaciji, ki jo sestavlja več geografsko ločenih delov, za manj primernega pa se izkaže pri povezovanju več organizacij in njihovih obstoječih sistemov.

Sporočilno usmerjen vmesni sloj sodelujoča dela porazdeljenega sistema – pošiljatelja in prejemnika sporočila – loči tako, da med njiju vrine posrednika, ki skrbi za oddajo sporočil. Tako sta oba dela le šibko povezana (ang. loose coupling), saj ni nujno, da prejemnik nemudoma sprejme in obdela sporočilo – nekatere vrste komunikacije to sicer zahtevajo, a sam vmesni sloj tovrstnega sinhronega obnašanja ne predpisuje. V primeru, ko pride do okvare prejemnika, zavoljo katere je ta nedostopen, lahko posrednik sporočilo shrani v obstojno shrambo (npr. v datoteko ali v podatkovno bazo), dokler ni prejemnik zopet na voljo, ali pa izbere drugega enakovrednega prejemnika. Klic metode oddaljenega predmeta po drugi strani zahteva takojšnjo reakcijo klicanega. Preobremenitev posameznega prejemnika lahko vmesni sloj rešuje s pomočjo porazdeljevanja sporočil med več enakovrednih prejemnikov. Preprosto je dodajati ali odvzemati nove dele porazdeljenega sistema, saj pošiljatelj in prejemnik sporočila nista neposredno povezana. Prav tako ni potrebe, da bi pošiljatelj poznal podrobnosti o prejemniku v času razvoja, tako kot mora v primeru porazdeljenih predmetov kličoči predmet poznati vmesnik klicanega predmeta. V primeru sporočilno usmerjene komunikacije zadošča le konsenz o obliki sporočil. Kot bomo videli v poglavju 2, je ta zahteva precej blažja, prilagajanje spremembam pa ravno zato mnogo lažje.

Ponovno poudarjamo, da tovrstne rešitve niso vedno mogoče, saj so odvisne od vrste programja, ki teče v porazdeljenem sistemu, in od njegove semantike; vsekakor pa obstaja velik razred programja, ki lahko uporabi opisani način asinhrone komunikacije. Še več: v široko porazdeljenih sistemih, kjer so okvare posameznih delov in visoke zakasnitve pri komunikaciji prej pravilo kot izjema, je raba takšne komunikacije celo izjemno zaželena, saj poveča zanesljivost in odzivnost programja (Curry, 2004).

Možnost šibkega povezovanja delov porazdeljenega sistema je izjemno pomembna za gradnjo zanesljivih velikih, globalno porazdeljenih sistemov, v katerih sodeluje množica organizacij, kjer infrastruktura ni centralno nadzorovana in morajo imeti udeleženci možnost samostojnega razvoja in sprememb njim pripadajočih delov porazdeljenega sistema.

1.3 Trdnost računalniških sistemov

Omeniti velja še malce cinično, aforistično definicijo porazdeljenega računalniškega sistema, ki jo pripisujemo Leslieju Lamportu: da *vemo, da gre za porazdeljen sistem, ko okvara računalnika, za katerega nismo nikdar slišali, prepreči, da bi opravili naše delo*[2].

Ta "definicija" poudari, da moramo v porazdeljenih računalniških sistemih prav posebno pozornost posvetiti odpornosti sistema na okvare in na škodljive zunanje vplive – njegovi *trdnosti* (ang. dependability). Razlog je očiten: z rastjo števila posameznih delov sistema – to je v porazdeljen sistemu nedvomno večje kot v izoliranem posameznem računalniku – se veča skupna verjetnost za okvaro poljubnega dela. Sistem, ki ne bo zmožen ustreznega odziva na okvare, bo večino časa neuporaben.

Osnovne pojme trdnosti računalniških sistemov, ki bodo prišli prav pri razumevanju pričujočega dela, povzemimo po Avizienisu (Avizienis in sod., 2004).

1.3.1 Osnovni pojmi

Sistem je entiteta, ki sodeluje z drugimi entitetami, drugimi sistemi, strojno in programsko opremo, človeškimi uporabniki in fizičnim okoljem. Ti ostali sistemi so *okolje* našega sistema. *Meja* sistema ločuje sistem od okolja. Sistem v poljubnem trenutku določa njegovo *stanje*.

Funkcija sistema predstavlja to, kar naj bi sistem počel; opisuje jo funkcijska specifikacija. *Obnašanje* sistema je njegovo delovanje z namenom, da izvede svojo funkcijo; obnašanje opišemo kot zaporedje stanj sistema.

Struktura sistemu omogoča delovanje; sistem je množica povezanih in sodelujočih *komponent*. Vsaka komponenta je lahko samostojen sistem, rekurzivna razgradnja pa se konča pri *atomičnih*, nedeljivih komponentah, ko ne moremo več razločiti notranje strukture ali pa ta ni pomembna in jo smemo prezreti.

Storitev, ki jo ponuja sistem (ponudnik), je njegovo obnašanje, kot ga dojema uporabnik. *Uporabnik* sistema uporablja storitev sistema. Del meje sistema, s pomočjo katerega uporabljamo storitev, imenujemo *storitveni vmesnik*. Del stanja sistema, ki ga uporabnik zaznava preko vmesnika, je njegovo *zunanje stanje*, preostali del pa *notranje stanje*. Ponujena storitev je zaporedje ponudnikovih *zunanjih stanj*. Poljuben sistem je lahko sočasno ali zaporedno tako ponudnik storitev drugim kot odjemalec storitev drugih.

[2] V izvirniku: *you know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done.*

1.3.2 Definicija trdnosti

Kvantitativna definicija trdnosti sistema pravi, da je *trdnost* zmožnost sistema, da se izogne izpadom delovanja, ki bi bili pogostejši in nevarnejši, kot je sprejemljivo za uporabnika.

Trdnost je ločena od funkcionalnosti sistema in jo tudi predpisujemo ločeno: s specifikacijo trdnosti (ang. dependability specification).

V preteklih tridesetih letih je pojem trdnosti zaobjel naslednje lastnosti sistemov:

- *razpoložljivost* (ang. availability): pripravljenost sistema na pravilno delovanje,
- *zanesljivost* (ang. reliability): vzdrževanje pravilnega delovanja,
- *varnost* (ang. safety): preprečevanje katastrofalnih posledic za okolje,
- *zaupnost* (ang. confidentiality): preprečevanje neavtoriziranega dostopa do podatkov,
- *integriteto* (ang. integrity): preprečevanje nepravilnih sprememb stanja sistema
- *zmožnost vzdrževanja* (ang. maintainability): možnost preprostega spreminjanja in popravljanja sistema

1.3.3 Kaj ogroža trdnost?

Storitev deluje pravilno, ko ustreza funkciji sistema. Dogodek, ko storitev prične delovati nepravilno, imenujemo *neuspeh* (ang. failure). Obdobje nepravilnega delovanja, ki se prične z neuspehom, imenujemo *izpad* (ang. outage) storitve. Povrnitev v stanje pravilnega delovanja imenujemo *obnova* (ang. restoration) storitve.

Neuspeh po definiciji - saj je storitev zaporedje stanj sistema - pomeni, da je vsaj eno od stanj v tem zaporedju nepravilno (ni v skladu s funkcijsko specifikacijo). To zbirko nepravilnih stanj, manifestacijo neuspeha, imenujemo *napaka* (ang. error). Vzrok napake imenujemo *okvara* (ang. fault). Po drugi strani pa se napaka ne konča nujno z neuspehom: je le del stanja sistema, ki more privesti do neuspeha. Pogosto napake ne vplivajo na zunanje stanje sistema in zategadelj ne privedejo do neuspeha.

Povzemimo: okvara je vzrok napake, ta pa lahko privede do neuspeha in s tem povezanega izpada storitve.

1.3.4 Kako zagotovimo trdnost?

Od časov prvih računalniških sistemov, ki so morali skrbeti za kritične procese, se je razvilo mnogo načinov, kako zagotoviti lastnosti, ki jim mora ustrezati trden računalniški sistem. Lahko jih uvrstimo v štiri osnovne kategorije:

- *preprečevanje okvar* (ang. fault prevention): sem spadajo postopki, ki preprečujejo nastanek oziroma vnos okvar v sistem,
- *odpornost na okvare* (ang. fault tolerance): v to kategorijo štejemo postopke, s katerimi se ognemo neuspehom storitev ob prisotnih okvarah,
- *odstranjevanje okvar* (ang. fault removal): zmanjševanje števila in usodnosti okvar,
- *napovedovanje okvar* (ang. fault forecasting) ocenjuje trenutno število okvar, verjetnost njihovega bodočega nastanka in njihove verjetne posledice.

Preprečevanje in sprejemanje okvar omogočita sistemu, da ob prisotnih okvarah ponudi delujočo, zaupanja vredno storitev . Odstranjevanje in napovedovanje okvar pa povečata zaupanje v zmožnost sistema, da bo zadostil specifikacijam glede funkcionalnosti in trdnosti.

Poglavje 2

Storitveno usmerjeni sistemi

Pričujoče poglavje razloži pojem storitveno usmerjene arhitekture. Ponudi pregled tehnologije spletnih storitev (ang. Web Services), nato pa se posveti trdnosti storitveno usmerjenih arhitektur.

2.1 Storitveno usmerjena arhitektura

Literatura pogosto nediskriminatorno izmenjuje pojma storitveno usmerjenega sistema in arhitekture. V tem delu bomo s slednjim pojmom označevali abstrakten koncept, pristop h gradnji porazdeljenih sistemov, s prvim pojmom pa se bomo sklicevali na udejanjenje tega koncepta, na dejanske primerke tovrstnih porazdeljenih sistemov. Začnimo torej z razlago pojma storitveno usmerjene arhitekture (ang. service-oriented architecture, SOA). Osnovne lastnosti, ki določajo takšno arhitekturo, si bomo ogledali v prihodnjih razdelkih. Začnimo s samim pojmom storitve in nadaljujmo z vlogami, ki jih razpoznamo v obnašanju igralcev v storitveno usmerjenih sistemih.

2.1.1 Storitev

V središču storitveno usmerjenih arhitektur je pojem *storitve*. Po WS-Glossary je storitev *abstrakten vir, zmožen opravljati naloge, ki tvorijo koherentno funkcionalnost z vidika ponudnika in odjemalca. Storitev lahko uporabimo, ko jo udejani konkreten ponudnik.*

Storitev je, skratka, možnost izrabe konkretnih računalniških virov. Rezultat storitve (oziroma uporabe virov preko storitve) določa semantika storitve. Ta je lahko formalno opisana na način, ki je razumljiv računalniškemu programju, lahko je le identificirana in brez formalnega opisa, ali pa je neformalno opisana (npr. z ustno razlago uporabniku, kako dostopati do storitve in kaj ta počne). V našem delu nas zanimajo izključno storitve

prve vrste, torej tiste s formalno opisano semantiko, saj le ta omogoča programju, da sklepa o lastnostih storitve.

Vmesnik storitve je abstraktna meja, ki jo storitev kaže okolju. Določa tipe sporočil in načine izmenjave sporočil, s katerimi lahko uporabljamo storitev, prav tako pa tudi semantiko, ki jo ta sporočila implicirajo (WS-Glossary).

2.1.2 Vloge

Storitveno usmerjena arhitektura temelji na sodelovanju treh vlog, v katerih lahko nastopajo deli storitveno usmerjenega sistema: *odjemalcev*, *ponudnikov* in *posrednikov* storitev (Chappell in Jewell, 2002).

Odjemalec

Odjemalec poišče in uporabi eno ali več storitev, ki ustrezajo njegovim zahtevam, s pomočjo katerih opravi potrebno delo.

Ponudnik

Ponudnika lahko dojemamo kot lastnika storitve; ta poskrbi za:

- opis storitve, ki vključuje podatke, s pomočjo katerih lahko odjemalec ugotovi, ali storitev izpolnjuje njegove zahteve in kako dostopa do nje. Opis tipično vsebuje obnašanje oziroma funkcionalne in nefunkcionalne lastnosti storitve, obliko vhodnih in izhodnih sporočil, način dostopa do storitve (protokol, naslov ipd.),
- javno dostopno objavo opisa storitve,
- izvajanje storitve in omogočanje dostopa do nje v skladu z opisom.

Ponujena storitev je lahko poljuben del programske logike, od najpreprostejših, trenutnih akcij do zapletenega poslovnega procesa, ki traja daljši čas, teče na gruči računalnikov in dostopa do porazdeljene podatkovne baze.

Posrednik

Posrednik v storitveno usmerjeni arhitekturi upravlja z zbirko podatkov o obstoječih ponudnikih: zbira njihove opise in omogoča poizvedovanje za storitvami, ki ustrezajo odjemalčevim zahtevam.

Uporabnik lahko s pomočjo posrednika v času razvoja odjemalčevega programja izbere enega od ponudnikov in se *statično* poveže z izbranim ponudnikom. Ta pristop je preprost, a neprimeren za velike, dinamične, odprte sisteme, kjer se razmere neprestano spreminjajo in statična informacija o ponudnikih kaj hitro zastari.

Boljši pristop je odkrivanje ponudnika v samem času izvajanja, kar omogoča izbiro na osnovi ažurne informacije o ponudnikih: temu pristopu pravimo *dinamično* povezovanje s ponudnikom.

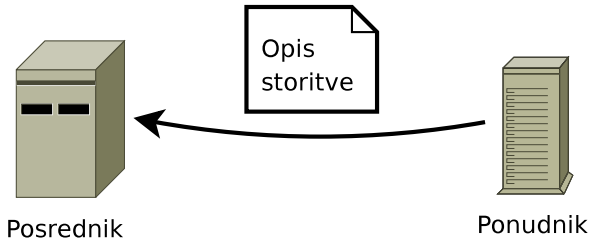
Poudariti velja, da posamezen igralec lahko in pogosto tudi dejansko nastopa v več vlogah: odjemalec ene storitve je lahko ponudnik druge storitve, za izvedbo katere potrebuje prvo. Na ta način sestavljamo preproste storitve v bolj zapletene.

2.1.3 Interakcije

Iz zgornjega opisa vlog lahko izluščimo tri osnovne postopke v storitveno usmerjenem sistemu: objavo storitve, odkrivanje storitve in povezovanje s storitvijo.

Objava storitve

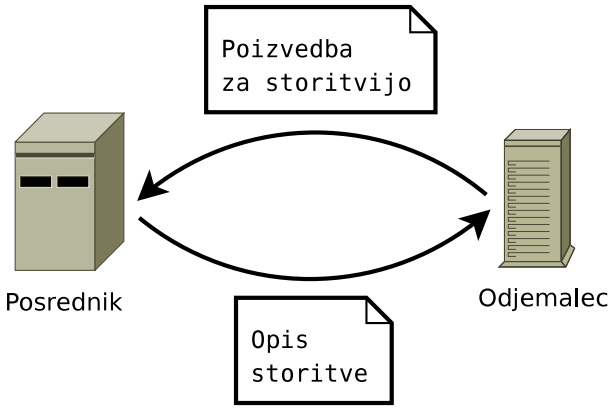
Proces objave prikazuje slika 2.1. Ponudnik mora obnoviti objavljeni opis vsakič, ko se lastnosti storitve spremenijo.



Slika 2.1: Objava storitve

Odkrivanje storitve

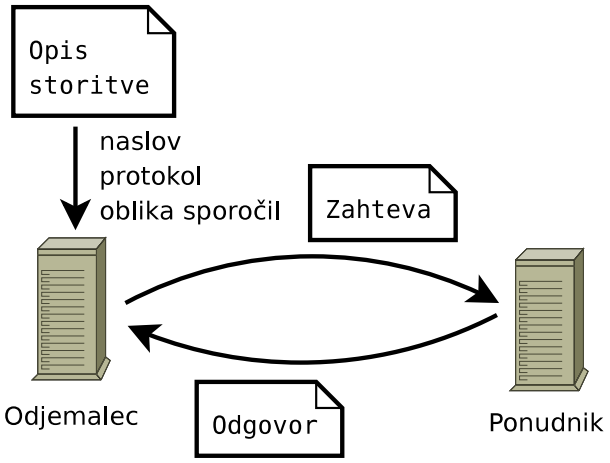
Odjemalec pri odkrivanju storitve navede zahteve, ki naj jih odkriti ponudnik izpolnjuje. Posrednik mora biti zmožen izvesti ustrezno poizvedbo in odjemalcu vrniti množico opisov ustreznih ponudnikov. Odkrivanje prikazuje slika 2.2.



Slika 2.2: Odkrivanje storitve

Povezovanje s storitvijo

Na osnovi opisa odjemalec izbere primeren protokol za komunikacijo s storitvijo, sestavi sporočilo v obliki, ki jo ponudnik pričakuje, in ga pošlje na naslov ponudnika, kot prikazuje slika 2.3.



Slika 2.3: Povezovanje s storitvijo

Zaključimo s kratko definicijo: *storitveno usmerjena arhitektura je množica komponent, ki jih lahko uporabimo v skladu z opisom vmesnika, ki ga ponudnik objavi, odjemalci pa odkrijejo* (WS-Glossary).

Zgornji opis interakcij bo bralcu, vajenemu predmetno usmerjenega pristopa h gradnji porazdeljenih sistemov, npr. s pomočjo CORBE (Mowbray in Zahavi, 1995), znan. Zna ga navesti k misli, da je storitveno usmerjeni pristop le v nove besede zavit koncept predmetno usmerjene gradnje porazdeljenih sistemov. Zavrnimo to tezo z razlago pojma šibko povezanega sistema: povezanost je namreč najpomembnejši ločevalni kriterij med

predmetno in storitveno usmerjenima paradigmama.

2.1.4 Šibko povezan sistem

Pomembna lastnost porazdeljenih sistemov, zgrajenih na osnovi storitveno usmerjene paradigme, je šibko povezovanje posameznih delov sistema, kar pripomore k prožnosti sistema. Razložimo to s pomočjo primerjave s predmetno usmerjeno arhitekturo.

Predmetno usmerjena arhitektura je uveljavljen pristop k načrtovanju in gradnji programja: temelji na močni povezavi med podatki in metodami za njihovo obdelavo. Predmet v predmetno usmerjeni paradigmi vključuje podatke in jih skrije pred programerjem (princip enkapsulacije (Dijkstra, 1982; Booch, 1994)). Programer se pri povezovanju predmetov zanaša na dobro določen vmesnik predmeta, ki se načeloma ne spreminja, namesto da bi se zanašal na nezanesljive podrobnosti implementacije. Na ta način predmetno usmerjen pristop dobro ščiti programerja pred spremembami implementacije, ki so iz teh ali onih vzrokov (optimizacija, odpravljanje hroščev, nove zahteve ipd.) pogoste.

Po drugi strani pa je ozka specializacija in s tem povezana množica vmesnikov, ki je potrebna v večjem, zapletenem sistemu, lahko vzrok za težave, če ni mogoče na enostaven način doseči občega strinjanja o vmesnikih. Ozrimo se ponovno na primera porazdeljenega sistema 1.1.2 in 1.1.3: tu sodelujejo računalniki - če smo natančni, lahko rečemo, da sodeluje programska oprema - množice neodvisnih organizacij. Programska oprema v posameznih organizacijah živi lastno, od ostalih organizacij v veliki meri neodvisno življenje. Vmesniki se lahko pogosto spreminjajo, spričo odprtosti sistema pa se organizacija pogosto niti ne zaveda vseh odjemalcev, ki te vmesnike uporabljajo: če so spremembo povzročile lastne zahteve ali pa zahteve enega od odjemalcev, se je bodo ostali odjemalci pogosto zavedli šele, ko bo nezdružljiva sprememba privedla do napake v njihovem delovanju.

Prilagajanje različnim vmesnikom je naporen in invaziven poseg v samo programje odjemalca. Težava je v tem, da sam vmesnik predpisuje specifično semantiko programja, v ozki specializaciji vmesnikov. Zapovrh nas ta težava tare celo, ko se vmesniki skozi čas ne spreminjajo: če v zgoraj opisanem sistemu nek program uporablja vmesnike predmetov v več organizacijah, se mora prilagoditi vsem. V skrajnem primeru število tovrstnih prilagoditev raste geometrično (Curry, 2004): programje vsake organizacije se mora prilagoditi vmesnikom programja v vseh ostalih organizacijah. Idealna rešitev te težave je standardizacija vmesnikov, žal pa je ta zavoljo množice vmesnikov, specifičnih za vsak programski sistem, praktično nemogoča.

Storitveno usmerjena arhitektura zategadelj predpiše le majhen nabor splošnih vmesnikov za prenos sporočil med odjemalcem in ponudnikom storitve. Zavoljo majhnega števila vmesnikov je standardizacija mogoča. Semantika, specifična za posamezen programski sistem, ki določa funkcionalnost ponudnika storitve, mora biti zato izrecno opisana v

samem sporočilu, saj ni več določena implicitno z vmesnikom.

Na ta način dosežemo *šibko povezavo* odjemalca in ponudnika storitve: načeloma lahko vsak odjemalec na enak način uporabi poljubnega ponudnika. Morebitna sprememba ponudnika ne vpliva na zmožnost, da odjemalec komunicira s ponudnikom: vmesniki, ki jih uporablja za komunikacijo ostanejo enaki. Ob spremembi vmesnika storitve (torej opisa sporočil) mora ponudnik, ki želi ohraniti združljivost za nazaj, le poskrbeti, da razume tudi prejšnjo obliko sporočil. Predmetno usmerjeni sistemi so, po drugi strani, tesno povezani zavoljo vezave na specializirane vmesnike posameznih predmetov.

Seveda pa ostaja težava razumevanja: da ponudnik razume odjemalca in obratno, se morata oba strinjati glede oblike v sporočilu izmenjanih podatkov. Težava, ki smo jo pri predmetno usmerjenem pristopu zasledili v množici vmesnikov, zdaj nastopi pri množici oblik podatkov. Pomembna razlika pa je v tem, da lahko spremembo iz ene v drugo obliko sporočila izvedemo brez invazivnih posegov v programje: lahko jo izvedemo celo v vmesnem sloju, brez posegov v uporabniško programje. Druga prednost je v tem, da število prilagoditev (oziroma v konkretnem primeru storitveno usmerjene arhitekture sprememb med oblikami podatkov) ne raste geometrično (glede na število vmesnikov), temveč linearno (glede na število oblik podatkov).

Ponazorimo gornje s kratkim primerom: denimo, da v sistemu obstaja N storitev z N različnimi oblikami podatkov $O_i, i = 0..N-1$ (po ena oblika za vsako storitev). Očitno tedaj za rabo poljubne storitve s strani poljubne druge storitve zadošča, če poznamo N preslikav $p_{(i,i+1(modN))}$, ki preslikujejo iz oblike O_i v $O_{i+1(modN)}$. Preslikavo med poljubnima dvema oblikama O_i in O_j izvedemo s sestavljanjem osnovnih preslikav: $p_{(i,j)} = p_{(i,i+1(modN))} \circ p_{(i+1(modN),i+2(modN))} \circ \ldots p_{(j-1(modN),j)}$. Programer mora poskrbeti za implementacijo N preslikav. Pri predmetno usmerjenem pristopu pa mora vsak od N predmetov uporabljati vmesnike ostalih $N-1$ predmetov: očitno je potrebno implementirati $N(N-1)$ načinov rabe vmesnikov drugih predmetov.

Tu se pokaže tudi razlog za osrednje mesto, ki ga ima v storitveno usmerjeni arhitekturi proces odkrivanja storitev: le redki izmed vseh ponudnikov storitev v sistemu izvedejo ustrezno semantiko, ki jo zahteva odjemalec. Ustrezne ponudnike mora odjemalec odkriti. Rezultat odkrivanja pa ni le naslov storitve in ostali podatki, s pomočjo katerih se odjemalec poveže s ponudnikom, temveč tudi opis oblike vhodnih in izhodnih podatkov storitve. S pomočjo tega opisa lahko odjemalec ustrezno oblikuje poslano sporočilo in tako zagotovi, da ga bo ponudnik razumel.

Povzemimo: medtem ko je predmetno usmerjen pristop zgrajen okrog paradigme sodelovanja s pomočjo ozko specializiranih vmesnikov predmetov, ki ovijajo podatke, je v središču storitveno usmerjenega pristopa izmenjava sporočil med storitvami, ki jih obdelujejo. V predmetno usmerjenem pristopu semantiko določa vmesnik predmeta, pri storitveno usmerjenem pristopu pa jo določajo sami podatki, poslani v sporočilu. Redukcija vmesnikov na majhno število splošno uporabnih pripomore k prožnosti sistema.

Predmetno usmerjen pristop h gradnji porazdeljenih sistemov se obnese v zaprtih sistemih, ki jih v celoti nadzoruje razvijalec, saj je le tako mogoč konsenz o uporabljanih vmesnikih predmetov. Tipičen primer so lokalna omrežja organizacij s pretežno homogenimi računalniki, majhno zakasnitvijo in majhno verjetnostjo okvar povezav in računalnikov.

Storitveno usmerjen pristop se izkaže v odprtih, široko porazdeljenih sistemih, ki povezujejo več organizacij, kjer je izjemno pomembna zmožnost medsebojnega sodelovanja množice sistemov, ki pripadajo neodvisnim organizacijam, podpora raznolikosti strojne in programske opreme ter omrežij (Vogels, 2003; Chappell in Jewell, 2002) in delovanje sistema navkljub neprestanim spremembam posameznih delov.

2.2 Spletne storitve

Pojem storitveno usmerjene arhitekture ne predpisuje tehnologije, s katero zgradimo ustrezen sistem. Sisteme, ki temeljijo na njem, lahko zgradimo z rabo katerekoli vrste vmesnega sloja (glej razdelek 1.2.3). Danes najpogosteje uporabljamo tehnologijo spletnih storitev (ang. web services) (Cerami, 2002; Chappell in Jewell, 2002; Vogels, 2003; WS-Architecture).

Spletne storitve so od strojne in programske platforme neodvisni standardi, ki določajo splošne vmesnike za povezovanje odjemalca in storitve, predpisujejo protokole za izmenjavo sporočil med odjemalcem in (programsko) storitvijo in jezik za predstavitev sporočil. Osnovni nabor teh standardov bomo predstavili v naslednjih razdelkih. V duhu povedanega v razdelku 2.1.4 ponovimo: *spletne storitve* niso *porazdeljeni predmeti* (Vogels, 2003), četudi jih lahko uporabljamo – med drugim – za izvedbo porazdeljenega predmetno usmerjenega sistema.

2.2.1 Opis podatkov: jezik XML

Jezik XML je meta-jezik, namenjen opisu strukturiranih podatkov. S pomočjo elementov v relaciji oče-sin, njihovih prilastkov (ang. attributes) in besedila, vsebovanega v elementih, oblikuje informacijo v hierarhično obliko (Ray, 2001).

Osnovni namen je omogočiti preprosto deljenje podatkov med heterogenimi sistemi, še prav posebej med sistemi, povezanimi preko Interneta, za kar binarno oblikovanje podatkov ni primerno. Njegove prednosti so preprostost, zmožnost oblikovanja XML dokumentov s preprostimi orodji, kot je običajen urejevalnik besedila, razumljivost človeškemu bralcu, kar olajša diagnosticiranje težav in razhroščevanje, neodvisnost od platforme in predvsem razširljivost.

XML nudi le sintaktične osnove za zapis strukturiranih podatkov: osnovna specifikacija ne pove prav nič o imenih posameznih elementov, njihovih prilastkih ter hierarhičnih

relacijah med elementi. Te lastnosti določi uporabnik skladno s semantiko podatkov, s katerimi dela programje, tako da jih predpiše s shemo. Shema dopolni osnovna pravila za skladnjo dokumentov XML s strogimi sintaktičnimi predpisi, ki določajo prej omenjene podrobnosti: tako hitro določimo nov, strogo omejen jezik, ki ustreza posameznemu programju. V tem delu bomo izraz *dialekt XML* uporabljali za jezike, katerih sintaksa je s pomočjo sheme določena podmnožica splošne sintakse XML, izraz *jezik XML* pa za jezik, ki ga določa splošna sintaksa. Obstaja več načinov, kako opisati shemo, daleč najbolj pogost pristop pa je raba XML Schema jezika, ki je tudi sam le dialekt XML.

Poudarimo še, da je pretvorba med različnimi shemami XML dokumentov enostavna. Jezik XSLT (ang. eXtensible Stylesheet Language Transformation), spet dialekt XML, omogoča opis transformacij za pretvorbo dokumenta med dvema shemama, namenjenima podatkom s sorodno semantiko (Gardner in Rendon, 2002). XSLT opisuje transformacije na deklarativen način: namesto imperativnega zaporedja dejanj, potrebnih za pretvorbo, navaja opis, kako ravnati s posameznim poddrevesom v vhodnem dokumentu. Opis predstavlja množica funkcijskih izrazov, ki določajo izhodno poddrevo dokumenta kot funkcijo vhodnega poddrevesa.

Seveda nobena reč ni zastonj; jeziku XML pogosto očitajo naslednje slabosti:

- dokumenti XML se ponašajo z visoko redundanco informacije. Po eni strani ta pripomore k berljivosti in omogoča splošnost, vsekakor pa negativno vpliva na učinkovitost. Prenos velikih dokumentov, ki vsebujejo relativno malo informacije (v primerjavi z binarnimi, nerazširljivimi oblikami zapisa, ki uspejo taisto informacijo zapisati z mnogo manj redundance), je drag. Po drugi strani pa se ravno zavoljo visoke redundance informacije dobro obnese stiskanje teh dokumentov, s čemer lahko stroške prenosa po mreži, če so ti preveliki, prenesemo na računsko delo pošiljatelja in prejemnika, ki dokumente stiskata oziroma razširjata,
- XML po sebi ne predpisuje nobenih podatkovnih tipov, kar lahko vodi k pogostim napakam. Dosledna raba shem, s katerimi lahko predpišemo tipe, to težavo odpravi,
- hierarhični model predstavitve podatkov je bolj omejujoč od npr. relacijskega.

2.2.2 Prenos sporočil: protokol SOAP

SOAP[1] (Chappell in Jewell, 2002; Cerami, 2002) je protokol za izmenjavo sporočil, zapisanih v dialektih XML, med računalniškimi sistemi. Protokol predpisuje standardno ovojnico sporočil, ki ovija zaglavje (ang. header) sporočila in njegovo telo.

[1]Zanimiva podrobnost: sprva je SOAP predstavljal okrajšavo za *Simple Object Access Protocol*; spričo zavajajočega pomena, saj gre pri SOAP-u predvsem za protokol prenosa sporočil, ne pa za dostop do morebitnih programskih predmetov, se od različice specifikacije 1.2 SOAP ne interpretira več kot kratica.

Zaglavje služi kot odprt, razširljiv mehanizem za dodajanje opcijskih podatkov o samem sporočilu. Namenjeno je predvsem prenosnemu ogrodju, ki je ponavadi del vmesnega sloja. V zaglavje lahko vtaknemo npr. digitalni podpis telesa, morebitno uporabniško ime in geslo ali odjemalčev certifikat, če prejemnik zahteva avtentikacijo, ali drugo informacijo namenjeno pravilnemu prenosu in interpretaciji sporočila. Obdelava elementov zaglavja in upoštevanje informacije, ki jo ponujajo, je privzeto prepuščena presoji posameznega vozlišča na prenosni poti. Posamezen element v zaglavju lahko izrecno namenimo enemu od vozlišč na prenosni poti sporočila, lahko pa tudi zahtevamo obvezno obdelavo.

Telo je obvezen del vsakega SOAP sporočila. Vsebuje poljuben, za vsako posamezno programje specifičen dokument XML. Interpretacija telesa je stvar prejemnika in ne vpliva na prenos do prejemnika. Telo lahko za prenos zakodiramo na različne načine, najpogostejša sta:

- kodiranje SOAP (ang. SOAP encoding) uporabimo predvsem za neposredno preslikavo programskih predmetov v izvedbi storitev v telesa SOAP sporočil. To kodiranje uporablja nadmnožico sistema tipov, ki ga ponuja shema XML, da popiše podatke, ki so pogosti v proceduralnih in predmetno usmerjenih programskih jezikih, od preprostih tipov do polj in strukturiranih podatkov. Vsak element telesa mora vsebovati prilastek, ki izrecno podaja njegov tip, kar omogoča enostavno, avtomatizirano preslikavo iz oblike XML v programske predmete in obratno. Kodiranje SOAP se, kot namiguje opisano preslikovanje podatkov, uporablja predvsem za izvedbo klica oddaljene metode s pomočjo protokola SOAP, kar je bil izvirni namen protokola,
- dobesedno (ang. literal) kodiranje uporabimo za pošiljanje poljubnega dokumenta XML. Za določitev tipov in konstrukcijo dokumenta mora poskrbeti pošiljatelj sam. Tovrstno kodiranje je primernejše za izkoriščanje spletnih storitev v pravem duhu storitveno usmerjenih sistemov: za izmenjavo in obdelavo dokumentov.

SOAP omogoča tudi pošiljanje binarnih priponk dokumenta, vendar se na tem mestu z njimi ne bomo posebej ukvarjali, navrzimo le, da je tovrstna raba SOAP-a v nasprotju z mantro sodelovanja raznolikih sistemov, ki jo ves čas poudarjamo, saj moramo za prenosljivost binarnih priponk poskrbeti na aplikacijskem nivoju. Binarne priponke so smiselne predvsem v primerih, ko obstaja standardizirana binarna oblika za izmenjavo neke vrste podatkov med računalniškimi sistemi (npr. slikovni ali video zapisi ipd.).

Protokol SOAP ne predpisuje transportnega protokola, uporabnik se lahko odloči za poljuben način prenosa, od golih TCP vtičnic (ang. sockets) do prenosa s pomočjo usmerjanja na prekrivnem omrežju (ang. overlay network). Najpogosteje za prenos uporabljamo uveljavljene aplikacijske protokole: HTTP (Fielding in sod., 1999), SMTP (Klensin, 2001), JMS (Hapner in sod., 2002) ipd. Raba standardnih protokolov olajša težave, ki jih pri medorganizacijskem povezovanju predstavljajo restriktivni požarni zidovi in podobne administrativne pregrade.

Za konec povejmo še, da protokol SOAP nikakor ni edina izbira za prenos sporočil: obstaja več alternativ, pri večini pa je vodilo enostavnost, saj protokolu SOAP pogosto očitajo nepotrebno kompleksnost. Sloj med transportnim protokolom in aplikacijskimi sporočili lahko celo popolnoma odpravimo, vendar se moramo v tem primeru odpovedati npr. možnosti, da sporočilo opremimo z meta-podatki na način, neodvisen od sintakse sporočil, ki jih določa aplikacija. Primer takšnega pristopa najdemo v spletnih storitvah, zgrajenih po načelu REST (Khare in Taylor, 2004), katerih operacije prožimo tako, da aplikacijsko sporočilo prenesemo neposredno s protokolom HTTP.

2.2.3 Opis spletne storitve: jezik WSDL

Zdaj, ko smo se poučili, kako določimo obliko aplikacijskih sporočil (glej razdelek 2.2.1) in kako jih prenašamo (glej razdelek 2.2.2), si velja ogledati, kako opišemo zaključeno spletno storitev. V ta namen bomo uporabili jezik WSDL (Chappell in Jewell, 2002; Cerami, 2002), ki določa model za opis spletne storitve: pričakovane oblike vhodnih in izhodnih sporočil, abstraktne definicije operacij, ki jih ponuja storitev, in določitev transportnega protokola za prenos sporočil. Ta opis, ki ga ponudnik objavi, odjemalcu omogoči povezovanje s spletno storitvijo.

Primer dokumenta v jeziku WSDL, ki opisuje preprosto spletno storitev, klasični "Hello, world!" primer, najdemo na izpisu izvorne kode 2.1. Zavoljo berljivosti smo izpustili določitve imenskih prostorov, ki so nujne, da bi opis ustrezno prevedli.

```
<definitions name="Hello">

  <!-- podatkovni tipi -->
  <types>
    <schema>
      <!-- definicije kompleksnih tipov -->
      <element name="hello-request" type="string"/>
      <element name="hello-response" type="string"/>
    </schema>
  </types>

  <!-- definicija tipa vrat -->
  <message name="HelloRequest">
    <part name="body" element="hello-request"/>
  </message>
  <message name="HelloResponse">
    <part name="body" element="hello-response"/>
  </message>
  <portType name="HelloPortType">
    <operation name="hello">
      <input message="HelloRequest"/>
```

```
      <output message="HelloResponse"/>
    </operation>
  </portType>

  <!-- povezava s transportnim protokolom (SOAP preko HTTP) -->
  <binding name="HelloSOAPBinding"
           type="HelloPortType">
    <binding style="document"
             transport="http://schemas.xmlsoap.org/soap/http/"/>
    <operation name="hello">
      <operation soapAction="hello" style="document"/>
      <input>
        <body use="literal"/>
      </input>
      <output>
        <body use="literal"/>
      </output>
    </operation>
  </binding>

  <!-- storitev dostopna s SOAP preko HTTP -->
  <service name="HelloService">
    <port name="HelloPort"
          binding="HelloSOAPBinding">
      <address location="http://service.somewhere.org/hello/"/>
    </port>
  </service>

<definitions>
```

Izpis izvorne kode 2.1: Primer dokumenta WSDL

Dokument WSDL v grobem razdelimo na štiri dele:

1. v prvem delu, znotraj elementa `types`, določimo podatkovne tipe, ki jim morajo ustrezati vhodni in izhodni dokumenti. V našem primeru uporabljamo le osnovne tipe, ki jih predpisuje shema XML, zato zadošča, da določimo samo korenske elemente vhodnega in izhodnega dokumenta,

2. drugi del določa abstraktni tip vrat (ang. port). Posamezen tip vrat združuje sorodne operacije. Operacija izvede dobro določeno funkcionalnost, ki jo ponuja storitev: v splošnem preoblikuje - v najširšem pomenu te besede, od preproste zamenjave vrednosti dveh elementov do zapletenega izračuna, ki vpreže računsko gručo in porazdeljeno podatkovno bazo - vhodni dokument v izhodnega. Ta del vsebuje tudi definicijo vhodnih in izhodnih sporočil (elementi `message`),

3. v tretjem delu povežemo abstraktni tip vrat s transportnim protokolom. S povezovanjem predpišemo, kako bo odjemalec dostopal do storitve: transportni protokol, kodiranje sporočil ipd. V našem primeru uporabljamo SOAP preko HTTP (`<binding transport=...`), slog povezovanja (`<binding style=...`) je prirejen izmenjavi dokumentov, sporočila pa so kodirana dobesedno (in ne s SOAP kodiranjem, glej razdelek 2.2.2). V sloge povezovanja, izbire protokolov in kodiranja se ne bomo podrobno spuščali: več bralec zve v (Chappell in Jewell, 2002; Cerami, 2002; WS-Architecture),

4. v zadnjem delu opišemo storitev: vratom in njihovim povezavam s transportnimi protokoli določimo naslove, na katerih so dosegljiva. V našem primeru (uporabljamo SOAP preko HTTP) je to HTTP naslov, na katerega odjemalec pošlje zahtevo. Na ta način udejanimo abstraktno storitev[2]. Posamezna storitev lahko podpira več protokolov, tako da združuje povezave za te protokole (npr. SOAP1.1, SOAP1.2, REST ipd.) v enem samem elementu `service`.

2.2.4 Nad osnovami...

Opisani trije gradniki, jezik XML, protokol SOAP in jezik WSDL, tvorijo jedro tehnologije spletnih storitev in zadoščajo tako za izvedbo ponudnika kot odjemalca storitve. Vrh omenjenih osnovnih gradnikov najdemo množico drugih specifikacij, ki so namenjene standardizaciji postopkov, ki sežejo onkraj preproste komunikacije med ponudnikom in odjemalcem storitve. Spričo tega, da so spletne storitve namenjene predvsem povezovanju raznolikih sistemov množice neodvisnih organizacij, je standardizacija izjemno pomembna za razumevanje različnih sistemov med sabo.

Med pomembnejšimi omenimo naslednje specifikacije:

- Universal Description, Discovery and Integration (UDDI) je standard, ki predpisuje vmesnike in arhitekturo imenika ponudnikov storitev. UDDI registri zbirajo podatke o ponudnikih storitev, jih uvrstijo v primerne kategorije in odjemalcem ponujajo opise storitev tako v človeškemu bralcu kot programju prijazni obliki, ter podatke o tem, kako se s storitvijo povezati,

- WS-Addressing (Box in sod., 2004) je standard za izmenjavo naslovnih podatkov spletnih storitev. Določa XML opis reference končne točke (ang. endpoint reference, EPR), ki vsebuje ciljni naslov sporočil, dodatne podatke, potrebne za usmerjanje sporočila, in morebitne meta-podatke (npr. dokument WSDL ipd.),

- WS-Coordination (Cabrera in sod., 2005) je standard za usklajevanje porazdeljenega programja, ki obsega več spletnih storitev. Uporaben je npr. za vzpostavitev

[2]Za pravo udejanjenje manjka še izvedba storitve, ki bo na podanem naslovu prav zares poslušala in obdelovala zahteve.

transakcije med odjemalcem in ponudnikom, ki zaobjema več operacij,

- WS-Security in sorodne specifikacije (WS-SecureConversation, WS-Authorization, WS-Privacy ipd.) opisujejo standardne varnostne mehanizme, kot je enkripcija sporočil, avtorizacija odjemalcev in ponudnikov ipd.,
- WS-Policy (Bajaj in sod., 2004b,a) ponuja standarden način za oglaševanje politik (varnostnih parametrov, kakovosti storitve ipd.) storitev,
- WS-Notification in WS-BaseNotification opisujeta standardne prijeme za izvedbo dogodkovno gnanega (ang. event driven) programja s pomočjo tehnologije spletnih storitev.

2.3 Trdnost storitveno usmerjenih arhitektur

Spomnimo se osnov trdnosti računalniških sistemov, opisanih v razdelku 1.3: trdnost zagotovimo s kombinacijo tehnik za preprečevanje, odstranjevanje in predvidevanje okvar in zagotavljanje odpornosti na okvare. Trdnost tradicionalnih, tesno povezanih porazdeljenih sistemov je dobro raziskano področje. Za zagotavljanje trdnosti imamo tam na voljo množico arhitektur, protokolov in programskih ogrodij, ki služijo skupinski komunikaciji (ang. group communication, group membership), replikaciji podatkov in ugotavljanju neuspeha (ang. failure detection).

Po eni strani lahko del teh rezultatov na precej neposreden način uporabimo tudi v storitveno usmerjenih sistemih. Primer takšnega recikliranja znanja je replikacija podatkov, eden osnovnih mehanizmov za zagotavljanje odpornosti na okvare (Birman in sod., 1985). Danes poznamo sisteme za replikacijo spletnih storitev, ki na ta način povečajo razpoložljivost in zanesljivost posamezne storitve (Ye in Shen, 2005; Osrael in sod., 2006; Salas in sod., 2006). Uporaba znanega pristopa je možna, ker gre za replikacijo znotraj organizacije, ki ponuja izvedbo storitve: okolje je pretežno homogeno, centralno nadzorovano in relativno majhno.

Po drugi strani pa tradicionalni pristopi niso uporabni v primeru velikih porazdeljenih sistemov. Danes na področju storitveno usmerjenih arhitektur nimamo mehanizmov, ki bi zagotavljali trdnost, primerljivo s tisto, ki jo lahko dosežemo v pravilno izvedenih tradicionalnih, tesno povezanih porazdeljenih sistemih (Birman, 2006). Še več, veliki, zapleteni, visoko dinamični in heterogeni programski sistemi, ki neprestano delujejo, s časom pogosto postanejo krhki in ranljivi. Po Mary Shaw (Shaw, 2002) gre iskati razlago v naslednjih dejstvih: (i) večina uporabnikov slabo artikulira natančne kriterije trdnosti in ostalih želenih lastnosti sistema; (ii) nemogoče je predvideti vse kombinacije razvoja okolja sistema in scenarije okvar med načrtovanjem, izvedbo in namestitvijo sistema; (iii) veliki sistemi so pogosto prezapleteni, da bi bilo mogoče natančno predvideti njihovo notranje obnašanje.

Na osnovi ugotovitev (ii) in (iii) v našem delu trdimo, da velja trdnost storitveno usmerjenih sistemov povečati tako, da sistem naredimo prožnejši glede nefunkcionalnih zahtev in mu tako omogočimo, da se je zmožen prilagoditi večjemu naboru možnih scenarijev okvar, stanj sistema in okolja. Bolj prožen porazdeljen sistem omogoči povečanje razpoložljivosti ponudnikov storitev na račun izpolnitve njihovih nefunkcionalnih zahtev. V naslednjem poglavju bomo podrobneje popisali ta razmislek, na katerem temelji naše delo, in motivacijo, ki nas je vodila pri našem delu.

Poglavje 3

Osnovni pojmi in sorodno delo

V razdelku 1.1 smo popisali osnovne lastnosti velikih porazdeljenih sistemov, kakršnim se posvečamo v tem delu, in dva primera, 1.1.2 in 1.1.3. Motivacija za naše delo izvira iz želje omogočiti zanesljivo uporabo porazdeljenih računalniških sistemov v skladu z modelom oskrbovalnega računanja. V tem poglavju opišemo ta model in razložimo motivacijo za naše delo. Nadaljujemo z razlago osnovnih pojmov našega dela in preučitvijo obstoječega dela na področjih, ki se jih dotaknemo.

3.1 Oskrbovalno računanje

V oskrbovalnem poslovnem modelu (ang. utility business model, (Rappa, 2004)) so dobrine, npr. električna energija, plin ali voda, dobavljane glede na potrebe porabnikov in plačane na osnovi dejanske porabe. Oskrbovalno računanje (ang. utility computing) predstavlja uporabo te paradigme pri računanju: programska oprema in računski viri tvorijo storitev, uporabljano na osnovi potrebe odjemalca. Razlaga pojma *programska oprema kot storitev* (ang. Software as a Service, SaaS) izrecno ugotavlja, da moremo oskrbovalni poslovni model uporabiti pri programskih storitvah (Rappa, 2004).

Uporaba zunanjega ponudnika storitev omogoči podjetju, da se ogne vlaganjem v drago lastno programsko in strojno opremo, namenjeno nalogam, ki so sicer nujne za podporo poslovnega procesa, a niso del glavnih kompetenc podjetja. Zapovrh se storitve zunanjih ponudnikov ponašajo s povečano dostopnostjo in zanesljivostjo (spričo specializacije ponudnika storitev), a so vseeno cenejše kot vzdrževanje lastne infrastrukture v ta namen (spričo ekonomije obsega). Po drugi strani pa mora odjemalec zaupati zunanjemu ponudniku, da bo zmožen storitev opraviti v skladu z odjemalčevimi zahtevami. Poleg očitnih funkcionalnih zahtev – kakšno delo naj bo opravljeno! – velja poudariti tudi pomen nefunkcionalnih zahtev – kako naj bo delo opravljeno, kakšna naj bo kakovost storitve. Ponazorimo povedano s primeroma 3.1.1 in 3.1.2.

Primer 3.1.1 *Ocenjevanje bodoče vrednosti naložbenih portfeljev in tveganj, povezanih z njimi, ter konstrukcija optimalnih portfeljev so tipična opravila finančnih institucij. Gre za optimizacijski postopek, ki je spričo obsežnega nabora spremenljivk in zapletenih modelov neprimeren za deterministično reševanje. Pogosta metoda reševanja je Monte Carlo simulacija, ki preveri množico možnih poti naložbe (npr. portfelja delnic) glede na verjetne vrednosti neodvisnih spremenljivk (npr. obrestne mere ipd.). Metoda zahteva veliko računsko moč, saj je potrebno za zaupanja vreden rezultat preveriti velik del prostora možnih poti, rezultat pa mora biti na voljo v relativno kratkem času, sicer je brez vrednosti. Namesto nakupa in vzdrževanja obsežnih računskih gruč velja to opravilo zaupati bodisi ponudniku računske infrastrukture, ki bo na svoji strojni opremi pognal odjemalčevo programsko opremo, ali pa kar ponudniku zaključene storitve, ki bo priskrbel tako ustrezno programsko opremo kot strojno opremo, na kateri bo ta tekla. Odjemalec pričakuje, da bo v danem času ponudnik zmožen analizirati neko minimalno število možnih rešitev, kar bo zagotavljalo primerno zaupanje v rezultat. Ponudnik mora poleg funkcionalne zahteve* izvajanja ustrezne Monte Carlo simulacije *biti zmožen zadovoljiti tudi nefunkcionalno zahtevo* analizirati več kot N rešitev v omejenem času.

Primer 3.1.2 *Sodelovanje na daljavo je danes mogoče ne le na rudimentarne načine, npr. z e-pošto, temveč na množico bolj ali manj zapletenih načinov, ki vključujejo prenos video, zvočnih in besedilnih tokov med sodelujočimi, pogled na oddaljeno namizje, deljenje datotek ipd. Izvedba tovrstnih storitev se praviloma spopada s tem, kako omrežni promet prenesti preko administrativnih ovir posameznih organizacij - požarnih zidov, NAT domen ... Ena od najbolj splošnih in pogosto uporabljanih rešitev je raba zunanjega strežnika, ki posreduje podatkovne tokove med sodelujočimi. Za zanesljivo delovanje tovrstnih sistemov potrebujemo povečkraten strežnik, da lahko v primeru izpada enega od primerkov povezave nemudoma prenesemo na drugega. Namesto da organizacija upravlja in vzdržuje lasten strežnik ter se ukvarja z namestitvijo in nastavljanjem ter rednimi nadgradnjami ustrezne programske opreme, velja v času sodelovanja zakupiti primerno podatkovno širino na strežnikih podjetja, ki nudi posredovanje omrežnega prometa med sodelujočimi, ki na svojih namiznih računalnikih potrebujejo le tankega odjemalca. Takšna dejavnost je odlična dopolnitev ponudbe kar za podjetje, ki razvija programsko opremo za sodelovanje. Odjemalec, ki vzpostavlja sodelovanje (npr. video konferenco), pričakuje, da mu bo ponudnik posredovalnega strežnika zagotovil zadovoljivo pasovno širino za prenos video in avdio tokov med vsemi sodelujočimi. Odjemalec išče ponudnike, ki zmorejo zadostiti funkcionalni zahtevi po* posredovanju podatkovnih tokov *in nefunkcionalni zahtevi po* zagotovljeni skupni pasovni širini N Mbps.

Razmislek o navedenih primerih z vidika podjetja v vlogi odjemalca postreže z naslednjim očitnim dejstvom: da je v porazdeljenem sistemu, v katerem nastopa množica neodvisnih ponudnikov in odjemalcev storitev, raba storitev drugih ponudnikov pa je kritičnega pomena za uspešno izvajanje poslovnih procesov, trdnost sistema izjemno pomembna.

Izpadi storitev, njihova nedostopnost in tudi nezmožnost zadostiti odjemalčevim zahtevam v času, ko jih odjemalci potrebujejo, povzročijo ustavitev poslovnih procesov. Trdnost in iz nje izhajajoča zmožnost odjemalca, da zaupa porazdeljenemu sistemu, da bo v primernem trenutku lahko uporabil potrebne storitve na želen način, je osnovni pogoj za zanašanje na opisani poslovni model. Uporaba storitveno usmerjene arhitekture in modela oskrbovalnega računanja v sistemih s kritično namembnostjo (ang. mission-critical) postavlja še strožje zahteve glede trdnosti. Sistemi za kontrolo nevarnih postopkov (jedrske elektrarne, kemične tovarne ipd.), nadzorni sistemi v zdravstvu ali javnem prometu (npr. nadzor in usmerjanje letalskega prometa) si ne morejo privoščiti izpada delovanja.

3.2 Motivacija za delo

Težava pri izrabi opisanega modela je predvsem v tem, da spričo velikosti, dinamičnosti in odprtosti teh sistemov pri načrtovanju in razvoju programja zanje ne moremo uporabiti podrobnega *a priori* znanja o programski in strojni opremi, ki bo porazdeljenemu programju v takšnem sistemu na voljo. Ponudniki storitev neprestano vstopajo v sistem in ga zapuščajo. Prav tako funkcionalno enakovredno storitev ponujajo različni ponudniki, ti pa morejo spričo razlik v izvedbi storitve in uporabljane strojne opreme različno zadovoljiti zahteve odjemalca. Celo zmožnosti istega ponudnika se spreminjajo skozi čas zavoljo različnega števila sočasno streženih odjemalcev in tretjih opravil, ki prav tako porabljajo računske vire, namenjene delovanju storitve (npr. vzdrževalna dela, nadgradnja programja, izdelava varnostnih kopij podatkov ipd.).

Dinamika in odprtost sistema ne onemogočata le predvidljivega zanašanja na vnaprej znane ponudnike. Še več, iz taistih vzrokov ne moremo predvideti niti števila in natančnih zahtev sočasnih odjemalcev posamezne vrste storitev. Poznavanje teh podatkov bi nam namreč omogočilo, da v sistem vnesemo primerno količino ponudnikov storitev, ki bi s svojimi viri zmogli zadostiti skrajnim zahtevam odjemalcev, četudi bi bil večji del virov večino časa neizkoriščen (ang. overprovisioning of resources). Lep primer tovrstnega izpada je zavrnitev strežbe (ang. denial of service, DoS) v primeru popularnega spletnega mesta, ki ga gostijo strežniki, ki ne zmorejo postreči množice sočasnih odjemalcev (t.i. *slashdot effect*, imenovan po novičarskem spletnem mestu `http://www.slashdot.org/`).

Tretji dejavnik, poleg spreminjanja množic ponudnikov in odjemalcev, njihovih zmožnosti in zahtev, so okvare - izpadi programske in strojne opreme ter omrežnih povezav -, ki so v tako velikih sistemih prej pravilo, običajen dogodek, kot izjema. Okvare lahko pomembno zmanjšajo število ponudnikov storitev in s tem računskih virov, do katerih more dostopati odjemalec.

Dinamika, odprtost in raznolikost – predvsem raznolikost programske opreme, izvedb iste vrste storitev – ponudnikov silijo programje v odkrivanje primernega ponudnika in povezovanje z njim v času izvajanja. Očitno je, da je storitveno usmerjena paradigma,

pri kateri je takšen pristop vtkan v same temelje, primeren način za gradnjo tovrstnih sistemov. Omogočila bo prilagajanje programja stanju sistema.

V razdelku 3.1 smo predstavili model oskrbovalnega računanja s pomočjo primerov 3.1.1 in 3.1.2. V obeh primerih se odjemalec zanaša, da bo eden od odkritih ponudnikov funkcionalno ustrezne storitve zmožen ponuditi tudi zahtevane nefunkcionalne lastnosti: ustrezno število analiz možnih rešitev v primeru 3.1.1 oziroma pasovno širino za posredovanje podatkov v primeru 3.1.2. Kadar ustreznih ponudnikov ni na voljo, ko so *nerazpoložljivi*, odjemalec ne more nadaljevati z delom. Pride do napake v delovanju programja, sistem pa smemo z vidika odjemalca – upoštevajoč terminologijo iz razdelka 1.3.3 – proglasiti za okvarjen. Okvaro predstavlja odsotnost ponudnika, ki bi mogel ponuditi zahtevane nefunkcionalne lastnosti storitve.

Oglejmo si sorodna, a malce drugačna scenarija.

Primer 3.2.1 *Odjemalcu storitve simulacije Monte Carlo zna zadoščati tudi ponudnik, ki lahko analizira manj kot optimalno želeno število N potencialnih rešitev v omejenem času. Večje število analiziranih rešitev odjemalcu res zagotavlja večje zaupanje v kakovost rešitve, a v primeru, ko optimalnega ponudnika ni na voljo, bi znal biti odjemalec zadovoljen tudi z nižjim številom analiziranih rešitev, da je le večje od neke spodnje meje M. Vsekakor pa se mora odjemalec zavedati, da je bilo analizirano manjše število rešitev in da je treba v tem pogledu rezultate interpretirati z ustrezno večjim zrnom soli. Alternativna, celo boljša rešitev izvira iz dejstva, da je mogoče postopek simulacije Monte Carlo trivialno povzporediti: odjemalec lahko uporabi več neodvisnih ponudnikov, katerih skupna računska moč zadošča za analizo želenih N rešitev v omejenem času. V tem primeru lahko odjemalec svoje nefunkcionalne zahteve do ponudnika izrazi kot* zmožnost analizirati vsaj M, najraje pa N potencialnih rešitev v omejenem času.

Primer 3.2.2 *Udeleženec videokonferenčne seje, odjemalec storitve posredovanja podatkovnih tokov, ki ostane brez ponudnika, ki bi zmogel ponuditi ustrezno pasovno širino, načeloma ne more delovati. Po drugi strani pa lahko shaja z manj kot optimalnimi N Mbps, če je voljan npr. znižati kakovost video toka ali se mu celo popolnoma odreči in se zadovoljiti le s prenosom zvoka. Denimo, da v prvem primeru zadošča M Kbps, v slednjem pa L Kbps. Nefunkcionalne zahteve velja torej izraziti kot* zmožnost posredovati L Kbps ali M Kbps prometa, najraje pa kar N Mbps.

Izkaže se, da bi odjemalca v obeh primerih, ko ne obstajajo ponudniki, ki bi zmogli izpolniti stroge nefunkcionalne zahteve, kar je poprej predstavljalo okvaro sistema, pravzaprav lahko nadaljevala z delom, resda za ceno slabšega delovanja. Težavo so povzročile prestroge nefunkcionalne zahteve, ki so vsebovale le optimalno stopnjo njihove zadovoljitve in niso upoštevale morebitnih rešitev, ko optimalne zadovoljitve ni moč doseči.

Iz primerov 3.2.1 in 3.2.2 lahko izluščimo, da velja pri iskanju primernega ponudnika podati množico naborov nefunkcionalnih zahtev (v primeru 3.2.1 so zahteve podane kot zvezen interval; takšen primer lahko interpretiramo kot neštevno množico različnih naborov nefunkcionalnih zahtev). Izpolnitev nekaterih naborov predstavlja optimalno zadovoljitev odjemalca, izpolnitev drugih nižjo zadovoljitev, izpolnitev spet tretjih pa najnižjo še sprejemljivo zadovoljitev. Preučitev obsežnega nabora primerov uporabe programja v široko porazdeljenih sistemih, od koder izvirata tudi zgornja primera, daje vedeti, da obstaja velik razred programja, kjer je tovrstno obnašanje možno in tudi zaželeno.

Izrecno poudarimo še, da različni nabori nefunkcionalnih zahtev med sabo nikakor niso enakovredni: tisti, katerih izpolnitev odjemalca bolj zadovolji, imajo višjo vrednost, in takšni ponudniki naj imajo pri odkrivanju in izbiri ponudnika prednost.

S tem, ko se bo odjemalec zatekel k rabi manj kakovostnega ponudnika, a vseeno nadaljeval delo, bomo povečali razpoložljivost ponudnikov na račun zadovoljitve odjemalca.

3.3 Osnovni pojmi

Opišimo osnovne pojme, ki jih bomo uporabljali v razlagi modela našega sistema.

Zmožnosti

Zmožnost (ang. capability) je imenovana funkcionalnost ali lastnost, ki jo agent zahteva ali navede kot podprto (WS-Glossary).

V našem delu zmožnost definiramo kot imenovano lastnost, ki jo ponudnik storitve lahko navede kot podprto oziroma jo odjemalec lahko navede kot zahtevano. Zmožnost predstavimo kot urejen par $(ime, vrednost)$. Zmožnosti lahko označujejo bodisi funkcionalnost storitve (t.j. kaj storitev izračuna) bodisi nefunkcionalne lastnosti (t.j. varnostne parametre, kakovost opravljene storitve ipd.). Pomen posamezne zmožnosti in domeno njenih vrednosti določa semantika programja, ki uporablja (bodisi ponuja ali zahteva) to zmožnost. V primeru 3.2.1 bo zaloga vrednosti zmožnosti *število rešitev na uro* (SR) kar množica realnih števil.

Izrecno bomo ločevali med *ponujenimi* in *zahtevanimi* zmožnostmi.

Ponujene zmožnosti

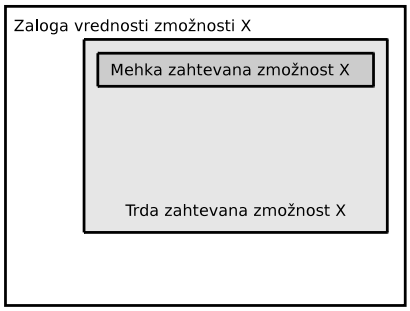
Ponujene zmožnosti (ang. provided capabilities) so lastnosti storitve, ki jih ponudnik lahko zagotovi pri strežbi odjemalca. V primeru 3.2.1 je ponujena zmožnost ponudnika storitve simulacije Monte Carlo $(SR, [0, N])$.

Zahtevane zmožnosti

Zahtevane zmožnosti so lastnosti storitve, ki jih odjemalec zahteva od ponudnika, ki mu streže. V primeru zahtevanih zmožnosti bomo – skladno z razmislekom v razdelku 3.2 – ločili

- *trde zahtevane zmožnosti*: te predstavljajo minimalne zahteve odjemalca, ki jih ustrezen ponudnik mora izpolniti, sicer je za odjemalca neuporaben,
- *mehke zahtevane zmožnosti*: predstavljajo zahteve odjemalca, ki naj jih ponudnik izpolni, da bi optimalno zadovoljil odjemalca.

Velja, da so mehke zahtevane zmožnosti podmnožica trdih. Razmerje prikazuje slika 3.1



Slika 3.1: Mehke in trde zahtevane zmožnosti

Odjemalec storitve simulacije Monte Carlo iz primera 3.2.1 npr. določi trde zahtevane zmožnosti $SR \geq M$ in mehke zahtevane zmožnosti $SR = N$.

Ujemanje zmožnosti

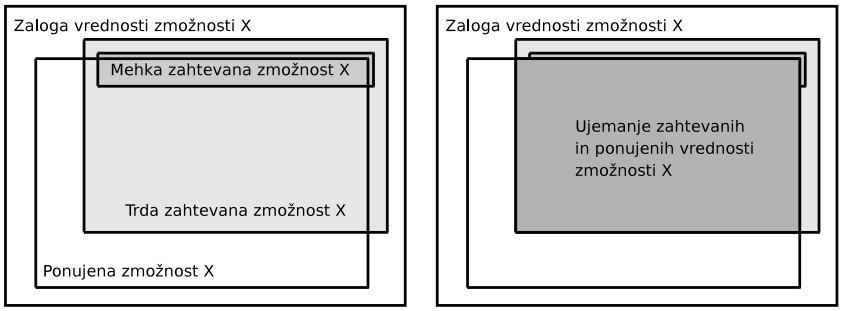
Ponujene in zahtevane zmožnosti se ujemajo, če ponujene zadostijo vsaj zahtevam, izraženim s trdimi zahtevanimi zmožnostmi. To velja natanko takrat, ko je presek trdih zahtevanih zmožnosti in ponujenih zmožnosti neprazen (sliki 3.2 (a) in (b)).

Funkcija zadovoljstva

Funkcija zadovoljstva preslika ponujene zmožnosti v realno število in tako vzpostavi linearno urejenost na množici ponudnikov ter omogoči ovrednotenje ponudnikov, katerih ponujene zmožnosti se ujemajo z odjemalčevimi zahtevanimi. S pomočjo te funkcije odjemalec določi in izbere *najboljšega* ponudnika.

Primerna funkcija zadovoljstva za primer 3.2.1 bo

$$f_z(SR) = \begin{cases} 1, & SR \geq N \\ \frac{SR}{N}, & SR < N \end{cases}$$



(a) (b)

Slika 3.2: Ujemanje ponujenih in zahtevanih zmožnosti

Takšno funkcijo zadovoljstva lahko uporabimo za požrešno dodeljevanje ponudnikov: vsak odjemalec bo zasegel v tistem trenutku zanj najboljšega ponudnika.

Navkljub temu, da smo v dosedanjih primerih vztrajno rabili eno samo zmožnost na storitev in tej podelili številsko, realno domeno, velja izrecno poudariti, da funkcija zadovoljstva preslikuje *množico ponujenih zmožnosti* v realno število, pri tem pa lahko uporablja tudi poljubne podatke iz okolja (npr. latenco dostopa do storitve, število obstoječih odjemalcev, globalno stanje sistema ipd.), če obstaja možnost dostopa do tovrstne informacije. Upoštevajoč tako splošno definicijo ugotovimo, da lahko s primerno izvedbo funkcije zadovoljstva uveljavimo poljubno politiko dodeljevanja ponudnikov.

Oglas storitve

Oglas storitve vsebuje opis funkcionalnosti storitve, njene nefunkcionalne lastnosti (zmožnosti) in naslov, s pomočjo katerega lahko odjemalec dostopa do izvedbe storitve.

Pogodba

Pogodba je dogovor med odjemalcem in ponudnikom glede zmožnosti, ki jih mora ponudnik zagotavljati odjemalcu storitve. Vsebuje naslov storitve (ki enolično določa izvedbo storitve), opis zmožnosti, ki jih bo ponudnik storitve zagotavljal odjemalcu, in veljavnost pogodbe. Po preteku veljavnosti pogodba poteče in ponudnik sme sprostiti vire, ki jih je zasegel za odjemalca. Odjemalec lahko pogodbo pred potekom veljavnosti podaljša. Ta pristop zagotavlja, da viri ne bodo ostali zaseženi v primeru, ko odjemalec ne zaključi svoje uporabe storitve z izrecnim preklicem pogodbe, npr. zavoljo napake v odjemalčevem programju ali zavoljo prekinitve omrežnih povezav med odjemalcem in ponudnikom. Pogodba s ponudnikom v našem delu predstavlja (resda ne pravno zavezujoč) ekvivalent pogodbe o nivoju storitve (ang. service level agreement, SLA), pojma, znanega npr. s področja telekomunikacijskih storitev.

3.4 Sorodno delo

V tem razdelku si ogledamo obstoječe delo na področjih, ki so pomembna za naše delo, ki omogoči povečanje razpoložljivosti na račun zadovoljitve odjemalca: odkrivanje virov, predstavitev zmožnosti in iskanje ujemajočih se zmožnosti.

3.4.1 Odkrivanje virov

Velika večina porazdeljenih sistemov potrebuje način za iskanje različnih virov (primerne strojne opreme, programskih predmetov, komponent ipd.), verjetno razpršenih preko večjega števila vozlišč in lokalnih omrežij. Iskanje v splošnem poteka tako, da določimo lastnosti, ki naj jih najdeni vir ima, tako da določimo zmožnosti, ki jih ponuja. Zapletenost teh pogojev določa primerne načine iskanja.

Imenske storitve

Najosnovnejša lastnost vsakega vira je njegovo ime, ki je od nekdaj tudi najpogostejši kriterij za iskanje. Storitev, ki poišče vir na osnovi njegovega imena, imenujemo imenska storitev (ang. naming service). Običajno skrbi za neposredno, enonivojsko preslikavo med imenom in (računalniku domačim) naslovom vira. Ponavadi imenska storitev predpiše množico pravil, ki jim mora ustrezati vsako ime. Primere imenskih storitev najdemo v skoraj vsakem porazdeljenem sistemu, omenimo npr. CORBA Naming Service (CORBA-NS) in Domain Name System (Mockapetris, 1987) ter javanski RMI Registry (Pitt in McNiff, 2001).

Imeniki

Iskanje na osnovi preprostega imena se izkaže za jako omejujoč način. Večina programja potrebuje možnost iskanja na osnovi poljubne množice lastnosti. Takšno iskanje omogočajo imeniki, ki so pravzaprav nadgradnja imenskih storitev: *imenik je poseben primer imenske storitve, s katero lahko odjemalec poišče želeno entiteto na osnovi opisa njenih lastnosti namesto polnega imena. Ta pristop je podoben temu, kako ljudje uporabljajo rumene strani...* (Tanenbaum in Steen, 2002). Imeniki so preproste podatkovne baze – pogosto za shranjevanje podatkov tudi uporabljajo relacijsko podatkovno bazo –, ki omogočajo iskanje virov na osnovi množice iskalnih kriterijev.

Iskalne storitve

Tako imenske storitve kot imeniki so običajno načrtovani in izvedeni za pretežno statična okolja. Vstavitev nove ali posodabljanje obstoječe informacije je naporen proces, ki ponavadi zahteva poseg administratorja. Spričo teh lastnosti je imenik primeren za npr.

dobro nadzorovano okolje lokalnega omrežja ene organizacije, izkaže pa se za neprimernega v dinamičnem, široko porazdeljenem, v vseh pogledih decentraliziranem okolju, saj je pogostost sprememb tam tako velika, da jim administratorski posegi ne morejo slediti. Smiselna rešitev je pristop, v katerem viri v porazdeljenem sistemu sami obnavljajo lastne podatke v imeniku. Na pomoč nam priskočijo iskalne storitve (ang. discovery service): *iskalna storitev je imenik, v katerega se prijavijo storitve v spontanem omrežnem okolju. (...) iskalna storitev ponuja vmesnik za samodejno prijavo in odjavo storitev, kot tudi vmesnik, ki odjemalcem omogoča, da med storitvami, ki so na voljo, poiščejo ustrezne.* (Coulouris in sod., 2001).

Podatki, ki jih hrani iskalna storitev, se samodejno obnavljajo, ko se okolje – omrežje – spreminja. Bodisi za obnovo skrbijo prijavljene storitve (potisni model, ang. push model) same ali pa jih iskalna storitev občasno povprašuje po njihovem stanju (povpraševalni model, ang. polling model). Primeri iskalnih storitev so Monitoring and Discovery Service (MDS, Foster (2006)), del programskega ogrodja Globus Toolkit 4, CORBA Trading Object Service (CORBA-TS) in Jini (Jini).

Spričo lastnosti okolij, ki jih kanimo nasloviti z našim delom, predvsem njihove visoke dinamike, se za primerno izbiro izkaže slednji pristop. Izvedba iskalne storitve same ni v domeni tega dela, zato bomo naš model opremili le s primernimi abstraktnimi vmesniki za prijavo in odkrivanje storitev ter omogočili rabo poljubne primerno splošno zasnovane iskalne storitve za dejansko hrambo podatkov in povpraševanje.

3.4.2 Predstavitev zmožnosti

Uporabniki vseh – z izjemo najpreprostejših – porazdeljenih okolij potrebujejo način, da opišejo zahteve opravil, ki jih kanijo izvesti na oddaljenih računalnikih. Uporabljajo različne jezike, s katerimi opišejo, npr. koliko CPE potrebuje njihovo opravilo, koliko pomnilnika, časovne omejitve, lastnosti sistemske programske opreme (npr. zahtevani operacijski sistem) ipd. Na osnovi teh podatkov lahko razporejevalnik opravil (ang. scheduler) sprejme ustrezno odločitev, na katerem računalniku bo posel izveden. Obstoječe delo prihaja s področij računskih gruč, omrežij peer-to-peer in hrambe podatkov. Lastnosti virov in zahteve poslov v računskih gručah se običajno opisujejo na relativno nizkem nivoju, npr. na nivoju lastnosti strojne in sistemske programske opreme (hitrost in vrsta CPE, količina pomnilnika in obstojne shrambe, operacijski sistem ipd.). V teh razmerah je kaj lahko določiti majhno, zaprto množico lastnosti, ki jih mora takšen sistem v računskih gručah podpreti.

Jeziki za opis poslov

Pogosto uporabljana sistema za nadzor in upravljanje računskih gruč in opravil v njih sta Load Sharing Facility (LSF, glej Zhou (1992)) in družina Portable Batch System (PBS,

glej Bayucan in sod. (1999)). Oba uporabljata koncept datoteke z opisom opravila: ta vsebuje zahteve, ki jih mora izpolnjevati vozlišče, na katerem bo opravilo izvedeno. Preprost primer takšne datoteke, namenjene orodju *qsub*, ki služi vnosu opravila v razporejevalnik v okolju PBS, najdemo na izpisu 3.1. Primer ilustrira opis opravila z imenom *Ime_opravila*, ki bo zahtevalo deset minut in pol računskega časa, četrt GB pomnilnika, pred (b) in po (e) izvajanju pa naj razporejevalnik pošlje e-pošto lastniku.

```
#PBS -N Ime_opravila
#PBS -l walltime=10:30, mem=256MB
#PBS -m be
```

Izpis izvorne kode 3.1: Opis opravila v PBS

Ti preprosti jeziki temeljijo na vnaprej določeni množici lastnosti, ki jo poznajo tako programska oprema na vozliščih (ki objavlja ustrezne vrednosti teh lastnosti za vsako vozlišče) kot razporejevalnik. Niso razširljivi. Vse zahteve so trde (glej razdelek 3.3): opravilo bo teklo le, če obstaja vsaj eno vozlišče, ki lahko popolnoma zadosti zahtevam. Nemogoče je v opis vtkati povezave med posameznimi zahtevami (npr. če je na voljo vsaj pol GB pomnilnika, bo opravilo trajalo le pet minut, sicer deset). Zavoljo teh lastnosti, predvsem nerazširljivosti, so ti pristopi neprimerni za naše delo. Razširljivost je nujen pogoj, saj v odprtem sistemu po definiciji ne moremo vnaprej predvideti vseh možnih lastnosti, ki jih bodo ponujali oziroma zahtevali ponudniki in odjemalci storitev.

Informacijski model LDAP

Za hrambo lastnosti predmetov v porazdeljenih okoljih in iskanje primernih predmetov pogosto uporabljamo informacijski model LDAP (Hodges in Morgan, 2002). Protokol LDAP je namenjen pridobivanju podatkov iz imenikov (glej razdelek 3.4). Imenik LDAP je razdeljen v razrede, ki definirajo imena hranjenih predmetov in njihove lastnosti. Razredi – podobno kot v predmetno usmerjenem programiranju – skupaj z relacijo dedovanja inducirajo hierarhijo nad hranjenimi predmeti in njihovimi lastnostmi. Lastnosti predmeta so unija lastnosti razreda, ki mu pripada, vseh nadrazredov in posebnih pomožnih razredov. Slednji služijo preprostemu dodajanju nespremenljivih lastnosti. Standard LDAP že določa osnovno hierarhijo razredov, ki naj bi bila prisotna v privzetih namestitvah, uporabnik pa jo lahko po potrebi širi.

Intentional Naming System

Intentional Naming System (INS) (Balazinska in sod., 2002), namenjen odkrivanju virov v peer-to-peer omrežjih, uporablja podoben pristop kot LDAP, a dovoli tudi odvisnosti med lastnostmi (npr. lastnost *soba* sme imeti vrednost *2xy* le, če ima lastnost *nadstropje* vrednost *2*).

Jini

Jini (Jini) ponuja rešitev za iskanje storitev, ki je tesno vezana na izvedbeno tehnologijo. Ponudniki sestavijo storitveni predmet (ang. service item), ki združuje množico javanskih predmetov, ki opisujejo storitev. Odjemalci, ki iščejo ustrezno storitev, pa sestavijo storitveni vzorec (ang. service template). Slednji vsebuje opise razredov in njihovih prilastkov (ang. attributes): ti se morajo ujemati s storitvenimi predmeti, ki ustrezajo odjemalcu.

Condor in jezik ClassAd

Med bolj prilagodljive pristope spada jezik ClassAd (Raman, 2000), uporabljen za opis ponujanih in zahtevanih zmožnosti v sistemu za upravljanje računskih bremen Condor (Thain in sod., 2002). Dokumente v jeziku ClassAd imenujemo oglasi (ang. classified advertisement). Jezik ClassAd ne določa množice znanih lastnosti vnaprej: predpisuje le znane tipe (številske, logične in znakovne) ter množico operacij nad primerki teh tipov (osnovne aritmetične in logične operacije). Primer preprostega oglasa (ang. classified advertisement) za vozlišče v računski gruči podaja izpis 3.2.

```
[
  Type = "Machine";
  Activity = "Idle";
  KeybrdIdle = RelTime("00:12:13");
  Disk = 218340.5M;
  Memory = 1024M;
  State = "Unclaimed";
  LoadAvg = 0.039;
  Mips = 958;
  Arch = "INTEL";
  OpSys = "SOLARIS7";
  KFlops = 73498;
  Name = "node17.somecluster.org";
  Rank = 1;
  Requirements = other.Type == "Job" &&
                 LoadAvg < 0.5 &&
                 KeybrdIdle > RelTime("00:10:00") &&
                 Memory - other.ImageSize >= 32M &&
                 other.Owner == "jaka";
]
```

Izpis izvorne kode 3.2: Condorjev oglas vozlišča v računski gruči

Oglasi odjemalcev in ponudnikov so popolnoma simetrični: eni in drugi lahko navajajo ponujene lastnosti in zahtevane lastnosti. Več o iskanju ujemanj med dokumenti ClassAd bomo izvedeli v razdelku 3.4.3.

Web Service Policy Framework

Specifikacija Web Service Policy Framework (WS-Policy) je nastala z namenom standardizirati opis (želene ali možne) interakcije med odjemalci in ponudniki spletnih storitev. Ponuja abstrakten, splošen model in ustrezen dialekt XML za opis politik (ang. policy) sodelovanja ponudnika in odjemalca storitve v storitveno usmerjenem sistemu.

Politika je množica medsebojno izključujočih se alternativ (ang. policy alternative). Pri sodelovanju vsakega para (*ponudnik*, *odjemalec*) velja natanko ena od možnih alternativ, ki mora biti sprejemljiva tako za ponudnika kot za odjemalca.

Posamezna alternativa vsebuje množico trditev (ang. policy assertions). Vsaka trditev opisuje neko lastnost interakcije. Specifikacija ne predpisuje trditev, saj so te odvisne od aplikacije: ta uvede trditve, ki jih potrebuje glede na svojo semantiko.

Tarča politike je lahko celotna storitev, posamezna operacija ali celo posamezno sporočilo med odjemalcem in ponudnikom.

Dokumente WS-Policy danes pogosto uporabljamo za določanje pretežno statičnih lastnosti in zahtev odjemalcev in ponudnikov, npr. podprtih varnostnih parametrov komunikacije. Primer takšnega dokumenta, s katerim ponudnik naznanja, da more podpreti dve alternativni varnostni konfiguraciji, prikazuje izpis 3.3. Specifikacija WS-Policy določa le elemente `Policy`, `ExactlyOne` in `All`; ostale elemente (trditve) določajo neodvisne varnostne specifikacije.

```
<Policy
  xmlns:sp="..."
  xmlns:wsp="...">
  <!-- Policy P1 -->
  <wsp:ExactlyOne>
    <wsp:All> <!-- Alternative A1 -->
      <sp:SignedElements>
        <sp:XPath>/S:Envelope/S:Body</sp:XPath>
      </sp:SignedElements>
      <sp:EncryptedElements>
        <sp:XPath>/S:Envelope/S:Body</sp:XPath>
      </sp:EncryptedElements>
    </wsp:All>
    <wsp:All> <!-- Alternative A2 -->
      <sp:SignedParts>
        <sp:Body />
        <sp:Header
          Namespace="..."/>
      </sp:SignedParts>
      <sp:EncryptedParts>
        <sp:Body />
```

```
      </sp:EncryptedParts>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Izpis izvorne kode 3.3: Primer ponudnikovega dokumenta WS-Policy

3.4.3 Pristopi k iskanju ujemanj

Iskanje virov, katerih lastnosti ustrezajo zahtevam iščočega, imenujemo iskanje ujemanj (ang. matchmaking). Iskanje ujemanj je eno od osnovnih opravil iskalnih storitev: odkriti mora vire z lastnostmi, ki ustrezajo zahtevam iskalca oziroma. Pravimo, da se takšni viri ujemajo z zahtevami.

Proces iskanja ujemanj se vedno začne z opisom zahtev, ki naj jim odkriti viri ustrezajo. Način opisa zahtev in posledično jezik, s katerim zahteve opisujemo, šteje za najpomembnejšo lastnost, po kateri lahko ločimo različne pristope k iskanju ujemanj.

Najosnovnejši pristop uporabljajo nekateri razporejevalniki opravil v računskih gručah, ki lastnosti vira opisujejo implicitno s pomočjo množice opravilnih vrst (ang. job queues). Različnim virom ali lastnostim virov ustrezajo različne vrste, v katere odjemalec uvrsti svoje opravilo. Z rastjo števila vrst virov raste tudi število ustreznih vrst. Seveda ta pristop omogoča le iskanje ujemanj na osnovi ene same lastnosti: razširitev na več lastnosti zahteva uvedbo vrst za vse mogoče kombinacije lastnosti. V primeru, ko lastnosti niso tipa 0/1 (vir neko lastnost lahko ima ali pa ne), temveč imajo npr. številsko vrednost, je potrebno postaviti vrsto za vsako vrednost vsake lastnosti ali vsaj za primerno število intervalov vrednosti vsake lastnosti (Raman in sod., 1998). Brez dvoma gre za jako nepraktičen pristop, primeren le za zelo majhno število zanimivih lastnosti in statične sisteme.

Razporejevalnika opravil PBS in LSF (glej razdelek 3.4.2) omogočata opis zahtev vsakega posla v jeziku, ki vnaprej določa množico lastnosti virov. To po eni strani omogoča izjemno preprosto iskanje ujemanj, algoritmi za iskanje pa so lahko izjemno učinkoviti, saj preprostost in zaprtost jezika omogočata mnoge optimizacije. Množica lastnosti mora biti znana vnaprej in se med življenjem sistema ne sme spreminjati, kar je za naše namene neprimerno.

Sistem Condor razširi zgoraj opisani pristop tako, da sicer uvede vnaprej določen sistem tipov in osnovnih operacij nad posameznimi tipi, nabor lastnosti (torej primerkov tipov) pa prepusti uporabniku. Ujemanja iščemo tako, da oglas, ki opisuje vir, *ovrednotimo* v kontekstu oglasa, ki opisuje zahteve uporabnika in obratno. Pri ovrednotenju se izračunajo vrednosti vseh lastnosti v oglasu, ki niso konstante, temveč izrazi. S pomočjo konstrukta `other.lastnost` se v izrazih sklicujemo na lastnosti iz oglasa, v katerega kontekstu izvajamo ovrednotenje.

Za določanje ujemanj na osnovi tako ovrednotenih dokumentov Condor predpisuje dve odlikovani lastnosti:

- `Requirements` vsebuje logičen izraz, zasnovan tako, da se ovrednoti kot resničen, če se oglas, ki ga vsebuje, ujema s tistim, v katerega kontekstu računamo vrednost izraza. Z izračunom izraza `Requirements` iz odjemalčevega oglasa v kontekstu oglasa nekega vira ugotovimo, ali se vir ujema z odjemalčevimi zahtevami,

- `Rank` vsebuje številski izraz, ki ga uporabljamo za ocenjevanje virov, tako da lahko Condor izbere odjemalcu najbolj primeren vir izmed vseh ujemajočih se. Odjemalec, ki bi rad dobil čim hitrejši procesor, izbere izraz `Rank` tako, da ta dobi višjo vrednost, ko ga izračunamo v kontekstu oglasa vira, ki ponuja hitrejšo CPE.

Ujemanje je v sistemu Condor simetrična relacija: oglasa A in B se ujemata natanko tedaj, ko se izraz `Requirements` v oglasu A v kontekstu B ovrednoti kot resničen in obratno. S tem dosežemo, da lahko zahteve predpisujejo tudi viri in ne le odjemalci: oglas vira na izpisu 3.2 z izrazom `other.Owner == "jaka"` omeji posle, ki jih sprejema, na tiste, katerih lastnik je uporabnik jaka.

Podkrepimo povedano še s primerom: na izpisu 3.4 najdemo oglas posla. Ta kot zahteve navaja, da pričakuje računalnik (`other.Type == "Machine"`) z Intelovo arhitekturo (`other.Arch == "INTEL"`) in operacijskim sistemom Solaris 7 (`other.OpSys == "SOLARIS7"`). Izraz `Requirements`, ovrednoten v kontekstu oglasa na izpisu 3.2, je resničen: oglaševani računalnik je torej primeren za posel.

```
[
  Type = "Job";
  QueueDate = AbsTime("2007-03-29T12:34:56-7:00");
  CompletionDate = undefined;
  Owner = "jaka";
  Cmd = "do_something";
  WantRemoteSyscalls = true;
  WantCheckpoint = true;
  Iwd = "/usr/home/jaka/something";
  Args = "-Q␣17␣3200␣10";
  ImageSize = 512M;
  Rank = other.KFlops/1000 + other.Memory/32;
  Requirements = other.Type == "Machine" &&
                 other.Arch = "INTEL" &&
                 other.OpSys = "SOLARIS7";
]
```

Izpis izvorne kode 3.4: Condorjev oglas posla v računski gruči

Na koncu omenimo še semantično ujemanje, nedavno predlagan pristop (Klein in König-Ries, 2004; Harth in sod., 2004). Pri tem pristopu ni potrebe, da bi se posamezni udeleženci v porazdeljenem sistemu strinjali glede poimenovanja lastnosti. To naredi proces ujemanja izjemno prilagodljiv in primeren za velike, nenadzorovane sisteme. Težava je predvsem v tem, da potrebujejo za dobro delovanje podroben opis semantike, obsežno bazo semantičnih pravil, ki je ob zagonu sistema praviloma nimamo na voljo.

Poglavje 4

Model sistema

Zdaj, ko smo podrobno razložili in na primerih utemeljili uporabno vrednost našega dela, uvedli osnovne pojme in preučili sorodno delo, si oglejmo, kaj pravzaprav potrebujemo za izdelavo razširitev vmesnega sloja za gradnjo storitveno usmerjenih sistemov, ki bodo omogočale *menjavo zadovoljitve odjemalcev za razpoložljivost ponudnikov* (Močnik in Karwaczyński, 2006).

4.1 Zahteve

Najprej si oglejmo zahteve, ki jim mora ustrezati sistem, ki ga razvijamo.

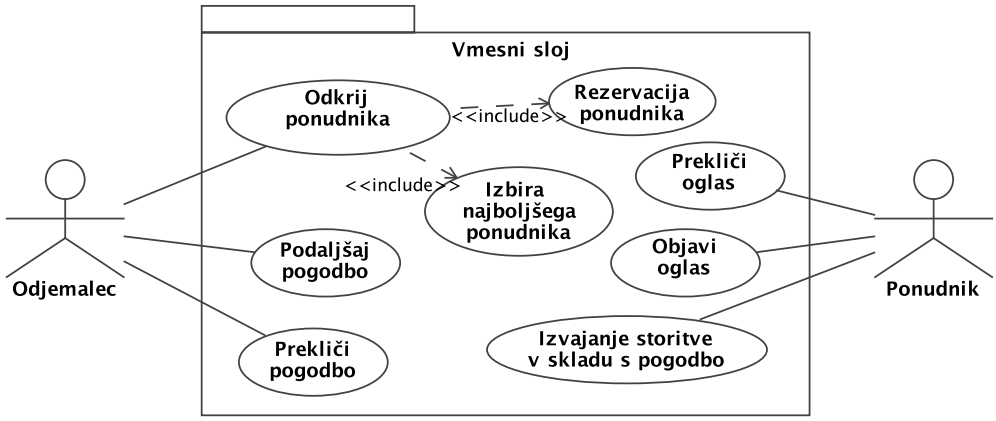
4.1.1 Funkcionalne zahteve

Funkcionalne zahteve na osnovi primerov uporabe prikazuje slika 4.1.

Sistem mora omogočati naslednje:

- opisovanje ponujenih in zahtevanih zmožnosti na formalen, programju razumljiv način,
- objavo in preklic oglasa storitve,
- odkrivanje ponudnikov, ki morejo zadostiti zahtevam odjemalca, in v okviru tega izbiro najboljšega ponudnika ter njegovo rezervacijo,
- vzpostavitev dogovora, pogodbe med odjemalcem in odkritim ponudnikom, s katerim se ponudnik obveže, da bo odjemalcu zagotavljal dogovorjene zmožnosti.

Prva točka je predmet definicije primernega jezika, ostale pa mora ponujati iskalnik, osrednja komponenta našega sistema.



Slika 4.1: Primeri uporabe

4.1.2 Nefunkcionalne zahteve

Zanesljivost

Zanesljivost sistema lahko obravnavamo z dveh vidikov: uporabniškega in sistemskega.

Z vidika uporabnikov sistema se zanesljivost nanaša predvsem na zmožnost ponudnikov, da strežejo v skladu z oblikovanimi pogodbami z odjemalci. V tem pogledu je zanesljivost zunaj domene našega dela: ponudniki morajo poskrbeti za primerno robustno izvedbo svojih storitev. Posamezne storitve so običajno izvedene znotraj ene same organizacije, npr. s predmetno usmerjenim pristopom, zato lahko za zagotavljanje zanesljivosti uporabimo že znane rezultate s tega področja.

Po drugi strani moramo zagotoviti tudi zanesljivo delovanje iskalne storitve, pomembnega dela iskalnika. Omogočili bomo rabo različnih iskalnih storitev, zato mora za zanesljivost poskrbeti izvedba uporabljene iskalne storitve. Tiste, zasnovane na principih peer-to-peer omrežij, ponujajo zanesljivost v primeru izpadov posameznih vozlišč na osnovi inherentne replikacije podatkov in redundantnih povezav (Androutsellis-Theotokis in Spinellis, 2004; Močnik in sod., 2006). V primeru hierarhičnih (npr. MDS, glej Foster (2006)) ali celo centraliziranih (npr. UDDI register, glej UDDI) izvedb moramo izrecno poskrbeti za redundanco in replikacijo podatkov, ki jih iskalna storitev hrani.

Razpoložljivost

Ponudnik lahko odjemalcu ponudi storitev, ki bo zadovoljila odjemalčeve zahteve le, ko ima na voljo dovolj računskih virov, ki jih za izvedbo storitve v skladu z zahtevami potrebuje (procesorski čas, pomnilnik, shramba, pasovna širina ipd.). V primeru, ko ti viri niso na voljo, zahtevana storitev odjemalcu ne bo na voljo: bodisi ne bo zadovoljila

zahtev ali pa bo ponudnik - če npr. izvaja nadzor dostopa (ang. admission control) - odjemalca zavrnil.

Najpomembnejši cilj našega dela je, da z enako količino virov postrežemo odjemalce tako, da bodo zadovoljene zahteve čim večjega števila sočasnih odjemalcev. Za uspeh bomo šteli, če nam uspe zadovoljiti več odjemalcev, kot to omogočajo danes najpogostejši načini strežbe v sorodnih sistemih, npr. strežba po najboljših močeh (ang. best-effort service) ali optimalna strežba z nadzorom dostopa. Tako bomo povečali razpoložljivost ponudnikov pri enaki količini virov.

Razpoložljivost lahko ocenjujemo tudi z vidika dostopnosti iskalne storitve v sistemu, vendar je ta del v domeni uporabljene izvedbe iskalne storitve: veljajo podobni pomisleki kot pri zanesljivosti.

Varnost

Primerno pozornost velja posvetiti tudi varnostnim ukrepom, s katerimi onemogočimo, da bi škodoželjni udeleženci v našem sistemu onemogočili njegovo delovanje. Spričo dejstva, da v splošnem ne obstaja koncept neprekinjene seje med odjemalcem in ponudnikom (odjemalec lahko povezavo s ponudnikom prekine in kasneje vzpostavi novo), bi bilo v popolnoma odprtem sistemu izjemno preprosto v sistem vnesti škodljiva omrežna sporočila. S pomočjo teh bi lahko napadalec v sistem vnesel nepravilne podatke o zmožnostih ponudnikov, preklical obstoječe pogodbe ipd.

Vsekakor je mogoče osnovno zaupanje v sistem doseči z rabo uveljavljenih pristopov k varovanju računalniške komunikacije: odločiti se velja za rabo infrastrukture javnih ključev (ang. public key infrastructure, PKI) za ugotavljanje istovetnosti ponudnikov (ang. authentication) ter tehnik digitalnega podpisovanja in zakrivanja izmenjanih sporočil za zagotavljanje integritete ter zaupnosti izmenjanih sporočil.

Varnost je na področju spletnih storitev razmeroma dobro obdelana tema z množico uveljavljenih rešitev (WS-Security). Naš sistem bomo zasnovali na odprtih standardih, tako da bo omogočil izrabo teh rešitev, posebej pa se varnosti ne bomo posvečali.

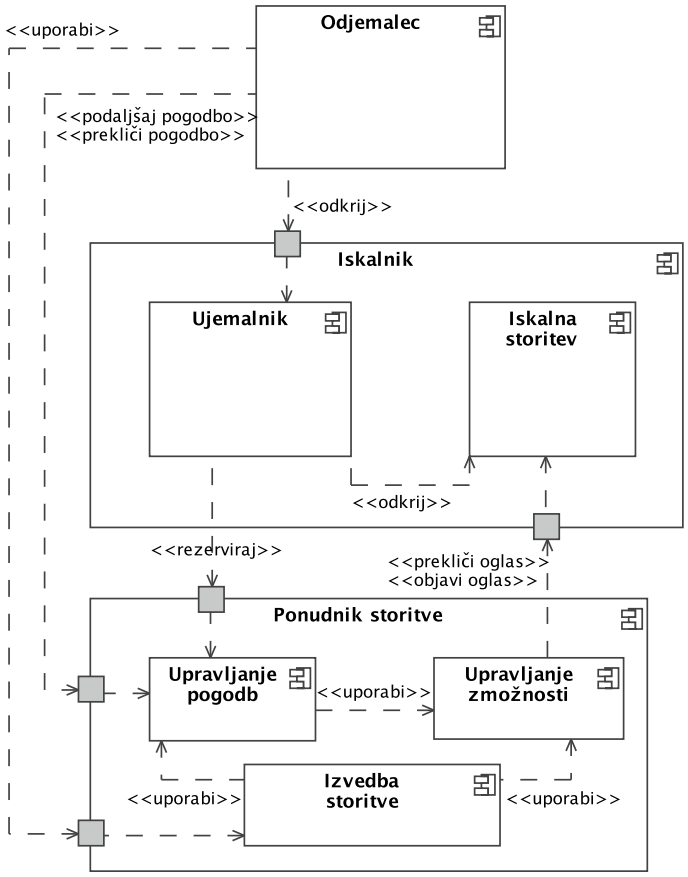
Razširljivost

Ciljna omrežja so izjemno velika, celo globalna (svetovni splet), in lahko vsebujejo sto tisoče ali celo milijone ponudnikov in odjemalcev. Rešitev mora biti zmožna učinkovito delovati s tako velikimi omrežji. Po eni strani mora razširljivost omogočati uporabljena iskalna storitev, za kar bomo v našem delu poskrbeli z abstraktnimi vmesniki, ki bodo omogočali rabo različnih iskalnih storitev: v kolikor je ciljno okolje manjši sistem, se uporabnik lahko odloči za npr. centraliziran imenik, sicer pa velja razmisliti o porazdeljeni

rešitvi. Po drugi strani pa moramo za razširljivost poskrbeti tudi pri izvedbi procesa ujemanja ponujenih in zahtevanih zmožnosti, ki ima osrednje mesto v opisovanem sistemu.

4.2 Arhitektura sistema

Arhitekturo zasnovanega sistema prikazuje slika 4.2. Prikazana arhitektura se ne osredotoča na konkretne izvedbene komponente, ki so odvisne od izbranih izvedbenih tehnologij, temveč na abstraktne komponente sistema.



Slika 4.2: Abstraktna arhitektura sistema

V okviru našega dela bomo izvedli naslednje komponente, ki skrbijo za tri ločene funkcionalne sklope predlaganega sistema:

- iskalnik (*ang. dependable discovery service*),
- upravljanje zmožnosti (*ang. policy manager*),
- upravljanje pogodb (*ang. contract manager*).

Oglejmo si njihov namen.

4.2.1 Iskalnik

V samem središču arhitekture je *iskalnik*, sestavljen iz dveh delov: iskalne storitve in ujemalnika.

Iskalna storitev hrani oglase ponudnikov storitev in omogoča iskanje oglasov ponudnikov, ki ponujajo želeno funkcionalnost. Imenik bo v našem delu le abstrakten vmesnik za prijavo in odjavo oglasov ter iskanje oglasov. S tovrstno abstrakcijo bomo omogočili rabo poljubne iskalne storitve (npr. UDDI registra, storitve MDS, porazdeljene razpršene tabele ipd.) na enak način, ne glede na izvedbene razlike med njimi.

Iskalna storitev ponudniku razkriva vmesnik, ki omogoča prijavo in odjavo storitve. Po drugi strani razkriva vmesnik, ki omogoča iskanje storitev na osnovi želene funkcionalnosti – tega uporablja ujemalnik.

V demonstracijske namene predvidevamo izvedbo tega abstraktnega vmesnika s pomočjo Globusove iskalne storitve MDS (Foster, 2006) in s pomočjo porazdeljene razpršene tabele (ang. distributed hash table, DHT), izvedene s peer-to-peer omrežjem (Močnik in sod., 2006). Prva izvedba bo namenjena omrežjem Grid (Foster in sod., 2001), zgrajenim s pomočjo ogrodja Globus Toolkit 4 (Foster, 2006), druga pa globalnim, odprtim omrežjem, npr. kar svetovnemu spletu.

Ujemalnik uporabljajo odjemalci za iskanje storitev, katerih ponujene zmožnosti se ujemajo z odjemalčevimi zahtevami. Ujemalnik uporabi iskalno storitev, da odkrije oglase storitev, ki ponujajo zahtevano funkcionalnost. Nato med njimi poišče oglase, ki se ujemajo z zahtevanimi zmožnostmi, ki jih je navedel odjemalec. Najboljšega (glede na kriterij, ki ga predpiše odjemalec) zaseže, rezervira zahtevane zmožnosti ter nastalo pogodbo vrne odjemalcu.

4.2.2 Upravljanje zmožnosti

Komponenta za upravljanje zmožnosti je namenjena enostavnemu sestavljanju oglasov, ki vsebujejo ponujene zmožnosti ponudnika.

Izračun dejanskih zmožnosti je odvisen od semantike ponujane storitve, običajno pa bodo vrednosti osnovane na stanju virov, ki jih storitev porablja. Storitev simulacije Monte Carlo iz primera 3.2.1 bo vrednost zmožnosti SR določila kot funkcijo računskih zmogljivosti CPE in obremenitve CPE v bližnji preteklosti, ob upoštevanju računske zahtevnosti izvedbe simulacije. Storitev posredovanja multimedijskih tokov iz primera 3.2.2 pa bo

svoje zmožnosti izračunala na osnovi celotne prepustnosti omrežnih vmesnikov in trenutne porabe pasovne širine.

Programsko ogrodje, ki ga bomo razvijalcem ponudili, bo omogočalo preprosto računanje zmožnosti tako, da bo ponudnik določil ime zmožnosti in kodo, ki izračuna njeno trenutno vrednost. Ogrodje bo iz teh podatkov sestavilo oglas in ga samodejno objavilo s pomočjo vmesnikov iskalne storitve. Skrbelo bo tudi za ponovno objavo, ko se vrednost katere izmed zmožnosti spremeni.

4.2.3 Upravljanje pogodb

Komponenta za upravljanje pogodb hrani vse trenutno veljavne pogodbe z odjemalci.

Po eni strani ponuja dostop do zmožnosti storitve, ki se jih je ponudnik s pogodbo zavezal ponuditi posameznemu odjemalcu. Na ta način lahko ponudnik vsakokrat, ko odjemalec zahteva izvedbo storitve, preveri in zaseže primeno količino virov in tako zagotavlja zanesljivo strežbo v skladu z odjemalčevimi zahtevami. Po drugi strani pa minimalne zahtevane zmožnosti iz vseh pogodb služijo tudi kot vhod za oblikovanje oglasa. Očitno je namreč, da sme ponudnik oglaševati le zmožnosti, ki zahtevajo največ toliko virov, da še vedno lahko (vsaj v skladu z minimalnimi zahtevami) postreže vse odjemalce z že sklenjenimi pogodbami.

4.2.4 Ostali deli sistema

Izvedba storitve

Izvedba storitve je v domeni semantike posamezne storitve. Za rabo našega sistema jo mora razvijalec prilagoditi tako, da:

- registrira vse ponujane zmožnosti in metode za njihov izračun pri komponenti za upravljanje z zmožnostmi,
- upošteva zahtevane zmožnosti, določene v pogodbi odjemalca, ki zahteva izvedbo storitve. Zahtevane zmožnosti so izvedbi storitve na voljo preko komponente za upravljanje pogodb, enolična oznaka pogodbe pa bo vsebovana v odjemalčevih sporočilih, s katerimi bo zahteval izvajanje storitve.

Odjemalec

Odjemalec storitve mora odkriti primernega ponudnika s pomočjo vmesnikov našega iskalnika, nato pa potrditi predlagano pogodbo s pomočjo posebnega vmesnika na strani

ponudnika storitve. Vsako sporočilo, ki ga pošlje ponudniku mora vsebovati enolično oznako pogodbe, s pomočjo katere bo ponudnik lahko prepoznal odjemalca in mu zagotovil personalizirano storitev glede na zmožnosti v pogodbi.

4.3 Interakcije v sistemu

V tem razdelku opišemo najpomembnejše interakcije v našem sistemu:

- *objava oglasa* omogoči odkritje ponudnika,
- *iskanje ponudnikov*, ki ustrezajo odjemalčevim zahtevam,
- *rezervacijo ponudnika*: rezultat rezervacije je krajši čas veljavna pogodba z opisom zmožnosti, ki jih bo ponudnik zagotavljal odjemalcu,
- *potrjevanje pogodbe* podaljša njeno veljavnost,
- *preklic pogodbe* obvesti ponudnika, da sme sprostiti vire, ki jih je zasegel za odjemalca.

Omenjene interakcije bomo opisali v okviru treh aktivnosti v sistemu: odkrivanja ponudnika ter življenjskega cikla ponudnika in odjemalca.

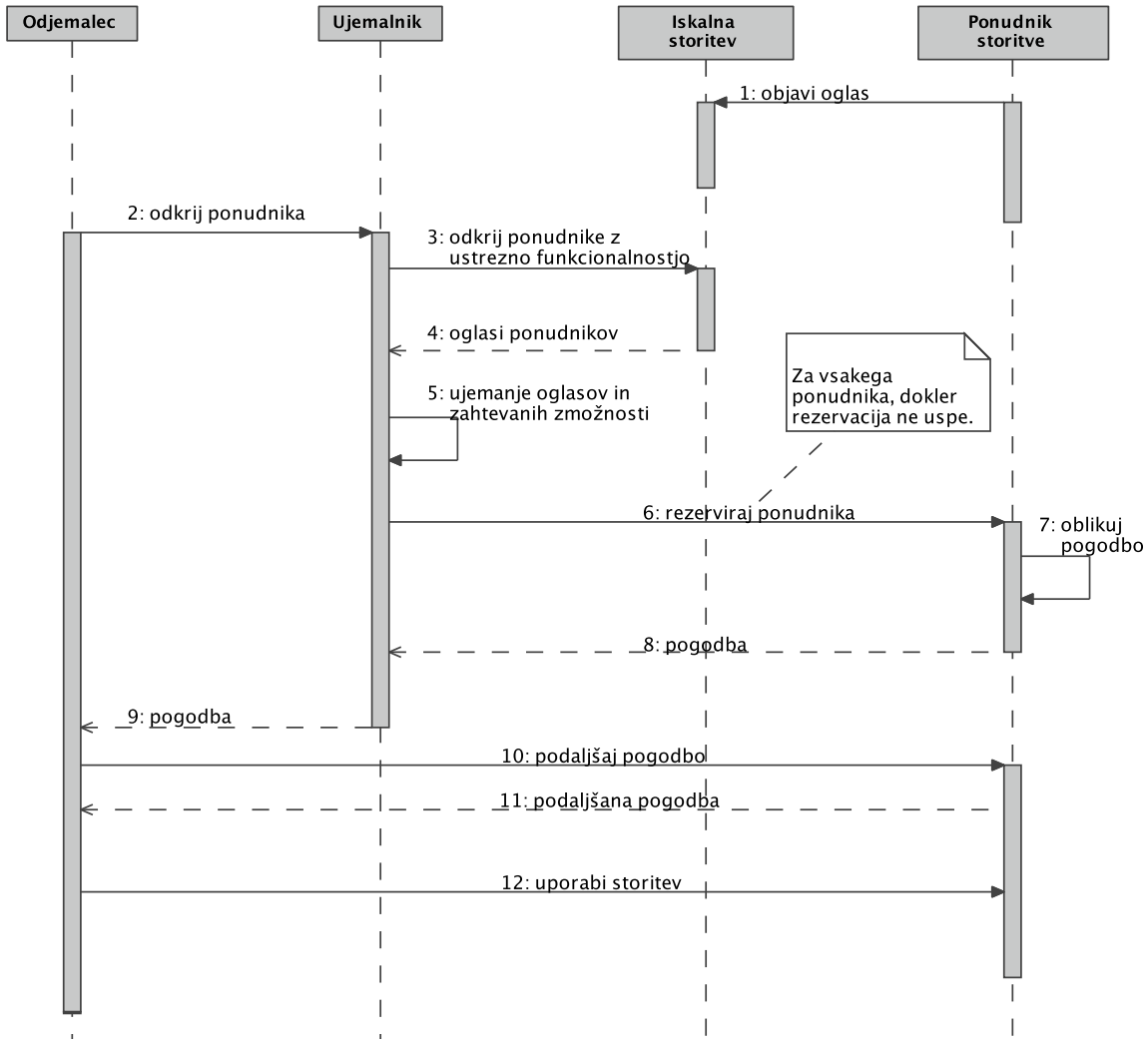
4.3.1 Odkrivanje ponudnika storitve

Slika 4.3 prikazuje proces odkrivanja ponudnika storitve. Predpogoj za odkrivanje je objava oglasa storitve s strani ponudnika. Oglas vsebuje naslov storitve, opis njene funkcionalnosti in seznam ponujenih zmožnosti. Ponudnik uporabi vmesnik iskalnika za objavo oglasa, ta pa objavo posreduje iskalni storitvi, ki podatke shrani.

Odjemalec, ki želi odkriti primernega ponudnika, uporabi za odkrivanje vmesnik iskalnika in kot parametre navede želeno funkcionalnost, zahtevane zmožnosti in funkcijo zadovoljstva. Opis funkcionalnosti je v domeni izvedbe storitve, da pa ohranimo kar največjo svobodo glede različnih opisov, predvidevamo v ta namen poljuben dokument XML - tako lahko iščemo glede na npr. imenski prostor opisa WSDL storitve, seznam ključnih besed ipd.

Iskalnik posreduje poizvedbo iskalni storitvi, ki poišče vse oglase storitev, katerih opis funkcionalnosti se natančno ujema z opisom funkcionalnosti, ki jo zahteva odjemalec.

Odkrite oglase iskalna storitev posreduje ujemalniku, ki izvede proces ujemanja ponujenih in zahtevanih zmožnosti. Oglase, ki ne ustrezajo niti trdim zahtevanim zmožnostim



Slika 4.3: Odkrivanje ponudnika storitve

in so zato neprimerni za odjemalca, iskalnik zavrže. Ostale oceni na osnovi funkcije zadovoljstva in jih uredi po padajoči vrednosti ocene.

Iskalnik nato poskusi rezervirati ponudnike, enega za drugim, v urejenem vrstnem redu. Takoj ko prva rezervacija uspe, iskalnik odjemalcu vrne pogodbo, ki je rezultat rezervacije. Pogodba opisuje zmožnosti, ki jih bo ponudnik zagotavljal odjemalcu.

Odjemalec mora pogodbo še potrditi s ponovnim klicem ponudnika, nato pa sme od ponudnika zahtevati izvajanje poljubne operacije, ki sodi v domeno uporabljane storitve.

Vmesni korak med odkrivanjem in potrditvijo pogodbe, rezervacijo ponudnika, uvedemo zato, da preprečimo siceršnje težave v primeru sočasnih odkrivanj ponudnikov s strani več odjemalcev. Oglejmo si naslednji primer: dva odjemalca sočasno odkrivata ponu-

dnike z enakimi funkcionalnimi zahtevami in določita enake (ali vsaj podobne) zahtevane zmožnosti. Iskanje primernih oglasov bo obema odjemalcema vrnilo isti nabor oglasov. Predvidevamo, da bomo ujemanje izvajali na strani odjemalca (centralizacija ujemanja je resna grožnja tako razširljivosti kot zanesljivosti sistema, zato ne razmišljamo v tej smeri). Vsak od procesov ujemanja, ne vedoč za drugega, bo izbral istega ponudnika. Oba odjemalca bosta poskusila z izbranim ponudnikom skleniti pogodbo, a uspel bo le prvi, drugi pa bo moral ponoviti odkrivanje.

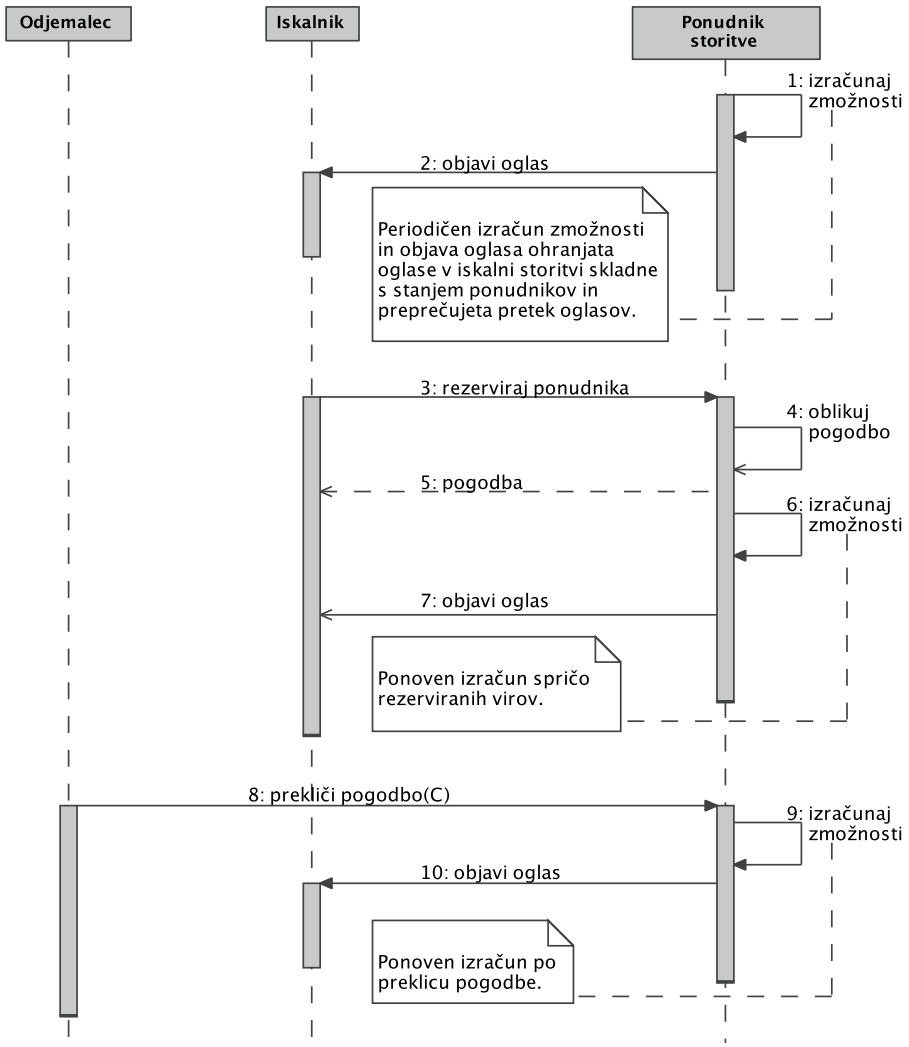
Opisani scenarij – predvsem prepuščanje ponovnega odkrivanja odjemalcu – je neprimeren in je posledica dejstva, da postopek *odkrij-in-skleni-pogodbo* ni atomičen: med odkritjem in sklepanjem pogodbe pride do tekmovanja (ang. race condition) med obema odjemalcema. Za zagotavljanje atomičnosti bi potrebovali bodisi mehanizem za izvajanje porazdeljenih transakcij ali pa globalno zaklepanje iskalnika. Prva možnost zahteva zapleteno izvedbo, druga pa izjemno poslabša zmogljivosti sistema, saj zaporedno razvrsti poizvedbe. Z uvedbo rezervacije premaknemo skrb za morebitne posledice tekmovanja v naš vmesni sloj, saj zanje poskrbi ujemalnik. Namesto da bi proces iskanja vrnil ustrezen oglas ponudnika, na osnovi katerega bi odjemalec zahteval oblikovanje pogodbe, se pogodba oblikuje že med procesom iskanja, v koraku rezervacije. Proces iskanja zato vrne kar sklenjeno pogodbo (resda s krajšim časom veljavnosti). V primeru, ko najbolje ocenjeni ponudnik zavrne oblikovanje pogodbe spričo v času med odkritjem in rezervacijo spremenjenih zmožnosti (npr. zavoljo sprejetja rezervacije za drugega odjemalca), ujemalnik preprosto poskusi z rezervacijo naslednjega ponudnika in tako razbremeni odjemalca skrbi za ta primer.

4.3.2 Življenjski cikel ponudnika

Slika 4.4 prikazuje življenski cikel ponudnika. Ponudnik najprej izračuna zmožnosti ponujane storitve in objavi oglas. Ponudnik mora oglas ponovno objaviti ob vsaki spremembi v njem navedenih zmožnosti. Ponovno objavo zahteva sprememba zmožnosti spričo:

- spremenjenega stanja virov, na katerih temeljijo zmožnosti (npr. zavoljo zunanje obremenitve virov),
- zaseganja virov spričo oblikovanja nove pogodbe,
- sprostitve virov ob poteku ali izrecnem preklicu pogodbe.

Skrb za obnavljanje oglasov bomo naložili na pleča ponudnika storitve: tovrstno *potiskanje* (ang. push) oglasa v iskalno storitev je primernejše od rednega *povpraševanja* iskalne storitve po oglasu (ang. polling). Ponudnik namreč natančno ve, kdaj so se zmožnosti spremenile (glej zgornje alineje) in osveži oglas le ob teh dogodkih, s čimer se zmanjša obremenitev tako omrežja kot ponudnikov in iskalne storitve.



Slika 4.4: Življenjski cikel ponudnika storitve

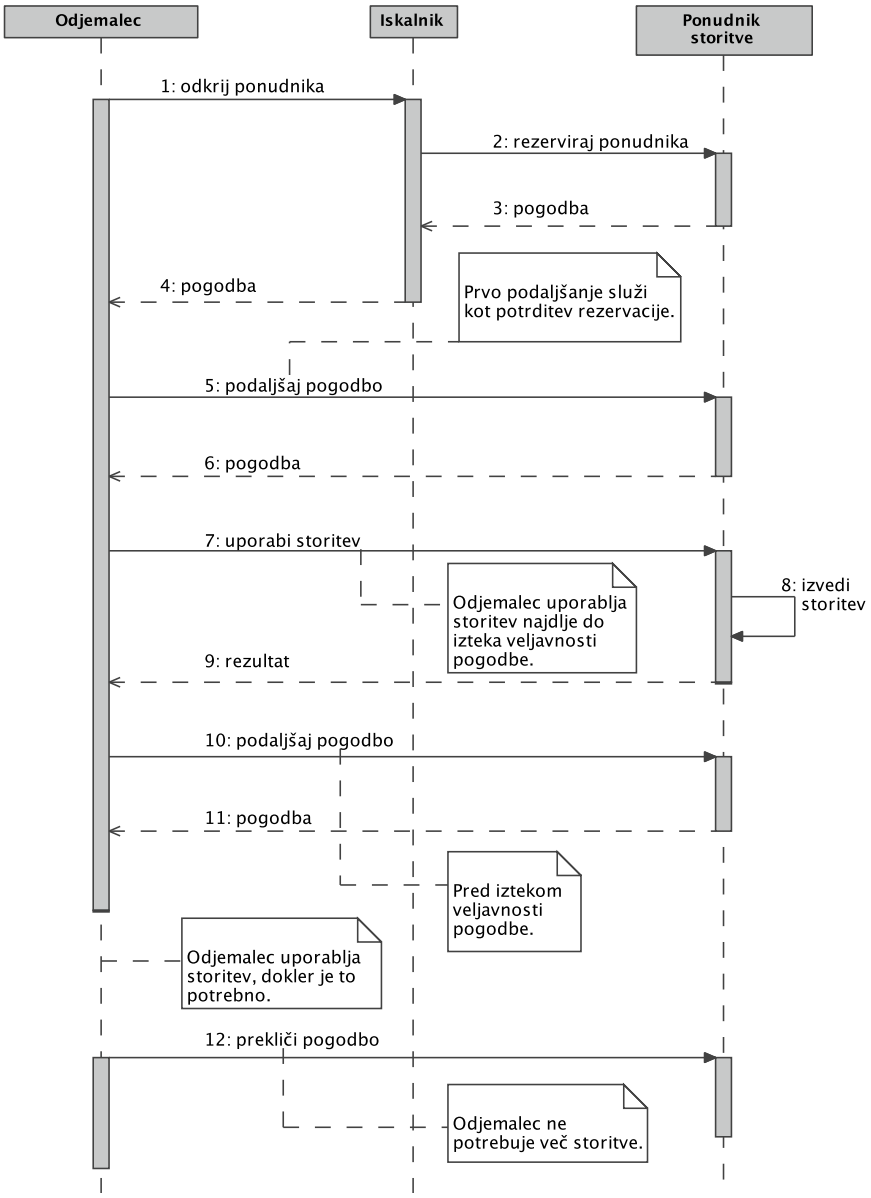
Poudariti velja še, da oglasi - podobno kot pogodbe - po določenem času potečejo: tako preprečimo, da bi sistem smetili oglasi ponudnikov, ki niso več na voljo (npr. zavoljo napake v programju ali zavoljo okvare omrežnih povezav).

Ob vsakem klicu operacij storitve ponudnik preveri, ali obstaja veljavna pogodba z odjemalcem, in v tem primeru izvede ustrezno računanje glede na zmožnosti, ki jih predpisuje pogodba.

4.3.3 Življenjski cikel odjemalca

Odjemalec, njegovo aktivnost prikazuje slika 4.5, uporabi iskalnik, da odkrije ponudnike, ki ustrezajo njegovim zahtevam, ter z njimi sklene pogodbo. Veljavna pogodba zagotavlja

odjemalcu možnost dostopanja do storitve in zagotovljene lastnosti opravljene storitve (zmožnosti), ki ustrezajo vsaj minimalnim zahtevam (trdim zahtevanim zmožnostim). Slednji proces je podrobno popisan v razdelku 4.3.1.



Slika 4.5: Življenjski cikel ponudnika storitve

Rezervirana pogodba velja le kratek čas, zato jo mora odjemalec po prejetju potrditi in tako podaljšati njeno veljavnost.

Do poteka pogodbe sme odjemalec klicati operacije storitve. Pred potekom pogodbe mora pogodbo ponovno potrditi in tako podaljšati, če želi storitev uporabljati še naprej. Po poteku mora ponoviti celoten proces odkrivanja ustreznega ponudnika.

V naslednjem razdelku bomo predstavili prototipno izvedbo opisanega sistemskega modela, temelječo na izvajalnem okolju javanskega virtualnega stroja, programskem jeziku Java in referenčni izvedbi tehnologije spletnih storitev, Apache Axis.

Poglavje 5

Izvedba prototipa

V tem poglavju opišemo prototipno izvedbo našega sistema. Začnemo z razlago izbire tehnologij, na katerih bomo prototip zasnovali. Sledi preslikava abstraktne arhitekture, podane v poglavju 4, na izbrane tehnologije. Nadaljujemo z opisom posameznih izvedenih komponent in zaključimo z razlago sprememb obstoječih odjemalcev in ponudnikov spletnih storitev, ki so potrebne, da se ti vključijo v naš sistem.

5.1 Izbrane tehnologije

Prototip bomo zgradili vrh tipičnih primerkov tehnologij, ki se danes uporabljajo za gradnjo storitveno usmerjenih sistemov. Pomembno vodilo pri izvedbi prototipa je, da obstoječemu programju omogočimo rabo prototipa s kar najmanj spremembami odjemalčeve ali ponudnikove kode.

5.1.1 Izvedba spletnih storitev

Prototip bomo zasnovali na tehnologiji spletnih storitev. Za izvedbo spletnih storitev bomo uporabili referenčno implementacijo standardov spletnih storitev, strežnik spletnih storitev Apache Axis[1]. Odločitvi zanj so botrovali naslednji razlogi:

- sodi med prosto programje (ang. free software),
- podpora velikemu naboru dodatnih standardov,
- velika baza uporabnikov in skupnost razvijalcev zagotavljata dobro podporo,

[1] Vsa progamska oprema, uporabljena pri izdelavi prototipa, ki je delo Apache Foundation (`http://www.apache.org/`), je dostopna na `http://ws.apache.org/`.

- je del okolja Globus Toolkit 4 (GT4, Foster (2006)), zato lahko na njem zasnovan prototip namestimo v omrežja Grid, zgrajena s pomočjo GT4.

5.1.2 Programski jezik

Izbira strežnika spletnih storitev Apache Axis in poudarek na rabi v okolju GT4 implicirata rabo programskega jezika Java. Poudariti velja, da so tipi vrat spletnih storitev, ki jih bomo določili in so pravzaprav edina stična točka med ponudniki in odjemalci, prenosljivi med izvajalnimi okolji in programskimi jeziki. To nam omogoča, da spletno storitev, napisano npr. v Javi s pomočjo našega prototipa, brez težav uporabljamo v poljubnem drugem izvajalnem okolju in s poljubnim drugim jezikom, npr. v okolju .NET z odjemalcem, pisanim v jeziku C#.

5.1.3 Iskalna storitev

V okviru izvedbe prototipa bomo podprli dve iskalni storitvi: prva je Monitoring and Discovery Service (MDS) iz okolja GT4, druga pa porazdeljena razpršena tabela, zasnovana na algoritmih Tapestry (Zhao in sod., 2004) z izboljšavami, ki znižajo stroške vzdrževanja tabele spričo sprememb v omrežju (Karwaczyński in Močnik, 2007; Karwaczyński in sod., 2007).

5.1.4 Predstavitev zmožnosti

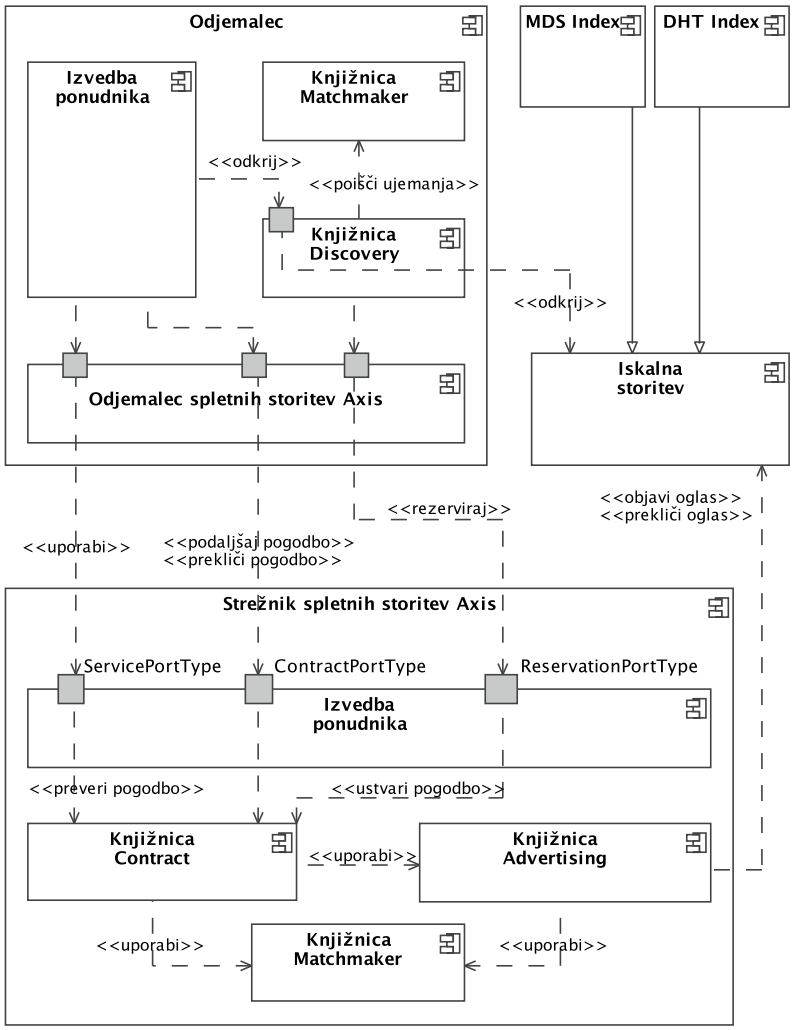
Zmožnosti bomo opisali s pomočjo dokumentov WS-Policy, ki smo jih predstavili v razdelku 3.4.2. Na ta način bo mogoče našo izvedbo ujemanja in ocenjevanja zmožnosti ter izbiro ponudnika, osnovano na teh ocenah, uporabiti za poljubno spletno storitev, ki svoje lastnosti oglašuje z dokumentom WS-Policy.

5.2 Projekcija arhitekture na izbrane tehnologije

Slika 5.1 prikazuje arhitekturo prototipa, ki je nastala iz abstraktne arhitekture (glej sliko 4.2) ob upoštevanju poprej navedenih tehnologij, na katerih bomo zgradili prototip.

Prototip bodo tvorile štiri programske knjižnice, namenjene bodisi izvedbi ponudnika ali odjemalca spletne storitve:

- knjižnica `Matchmaker` bo namenjena analizi dokumentov WS-Policy, preslikavi leteh v programske objekte, operacijam nad temi dokumenti (npr. ujemanju dveh dokumentov) in ocenjevanju dokumentov v skladu z odjemalčevimi kriteriji (t.j. funkcijo zadovoljstva),



Slika 5.1: Arhitektura prototipa

- knjižnica `Advertising` omogoča programerju, da navede podatke o storitvi, poimenuje zmožnosti storitve in postopke za izračun zmožnosti ter na osnovi teh podatkov oblikuje oglase in jih objavi v iskalni storitvi,
- knjižnica `Contract` vodi seznam trenutno veljavnih pogodb z odjemalci, omogoča dostop do njih in oblikuje nove pogodbe na osnovi trenutnih zmožnosti ponudnika in zahtevanih zmožnosti odjemalca,
- knjižnica `Discovery` odjemalcu omogoči, da na osnovi zahtevanih zmožnosti, želene funkcionalnosti storitve in funkcije zadovoljstva poišče primernega ponudnika in sklene pogodbo z njim.

Predvideni sta dve specializaciji abstraktnega vmesnika iskalne storitve:

- `MDS Index` je preprosta ovojnica okrog storitve MDS iz ogrodja GT4; namenjen je manjšim sistemom, ki jim lahko postrežemo s hierarhično urejeno iskalno storitvijo,
- `DHT Index` (Močnik in sod., 2006) bo uporabljal porazdeljeno razpršeno tabelo, zgrajeno na omrežju peer-to-peer; ta izvedba je namenjena večjim, globalnim sistemom.

Ponudniku storitve mora poleg specifičnih operacij, značilnih za konkretno ponujano storitev (te določa tip vrat `ServicePortType`), podpreti še vrata za rezervacijo in upravljanje s pogodbami (`ReservationPortType` in `ContractPortType`).

Posamezne komponente in vmesnike podrobno popišemo v naslednjih razdelkih, poglavje pa zaključimo z opisom sprememb, ki so potrebne, da obstoječega ponudnika in odjemalca spletne storitve priredimo za rabo prototipnega sistema.

5.3 Knjižnica Matchmaker

Knjižnica `Matchmaker` služi analizi (ang. parsing) dokumentov WS-Policy in njihovi preslikavi v programske predmete, ki omogočajo enostavno programsko upravljanje s temi dokumenti. Na osnovi tega poskrbi tudi za operacije, ki jih predpisuje specifikacija WS-Policy, predvsem za presek dokumentov, na katerem utemeljimo proces ujemanja.

5.3.1 Predstavitev zmožnosti

Zmožnosti, tako zahtevane kot ponujene, bomo opisali v dokumentih WS-Policy, standardu za opis lastnosti spletnih storitev in komunikacije med odjemalcem in ponudnikom storitve.

Specifikacija WS-Policy predpisuje abstrakten, splošnonamenski model za opis spletnih storitev in ustrezen dialekt XML za predstavitev tega modela. Določi pojem *politike spletne storitve* (ang. policy), ki je zbirka lastnosti storitve. Politika vsebuje množico *alternativ* (ang. policy alternative), te pa vsebujejo *trditve* (ang. policy assertions): vsaka trditev opisuje neko lastnost spletne storitve.

Ponudnik storitve lahko ponudi storitev, ki ustreza trditvam natanko ene od alternativ, ki jih navaja v politiki. Odjemalec lahko politiko preuči in se odloči za alternativo, ki mu najbolj ustreza.

Specifikacija ne predpisuje sintakse za opis konkretnih trditev: trditev more biti poljuben element XML, ki ga vsebuje neka alternativa. Tako ostane določanje trditev – te so nenazadnje stvar semantike posamezne storitve – v domeni uporabniškega programja, v domeni ponudnika storitve. Naša izvedba analize in preslikovanja v programske predmete bo sledila temu zgledu: vključevala bo le izvedbo predmetov, ki ustrezajo splošnim elementom dokumentov WS-Policy, in operacij nad njimi. Predvideli pa bomo mehanizem, s

katerim bo programer lahko analizi podtaknil postopke za preslikavo od storitve odvisnih trditev v ustrezne programske predmete in omogočil operacije nad njimi.

Scenarija 3.2.1 in 3.2.2, na osnovi katerih gradimo predlagani sistem, se ukvarjata z zmožnostmi, ki temeljijo na računskih virih, te pa ponavadi predstavimo kot vrednosti s številsko domeno. Scenarij 3.2.1 nam bo služil tudi kot testni primer, zato bomo določili trditev za zmožnosti s številsko domeno. To lahko v splošnem uporabimo za velik razred zmožnosti. To vrsto trditve opišemo v prihodnjih razdelkih.

Element *Capability*

Element `Capability`, primer je na izpisu 5.1, opisuje posamezno ponujeno ali zahtevano zmožnost s številsko domeno. Zmožnost je določena z:

- imenom (element `Name`),
- številsko domeno zmožnosti (element `Domain`), ki je lahko poljubna podmnožica realnih števil. Ta predstavlja bodisi vrednosti, ki jih lahko storitev ponudi (ponujena zmožnost), bodisi vrednosti, ki jih odjemalec zahteva (zahtevana zmožnost).

Na nivoju zapisa ne bomo ločevali med trdimi in mehkimi zahtevanimi zmožnostmi, kot jih opisuje razdelek 3.3: domena ustreza trdim zahtevanim zmožnostim, izmed teh pa bomo mehke (torej tiste, ki optimalno zadovoljijo odjemalca) ugotovili s pomočjo funkcije zadovoljstva. Optimalno zadovoljijo tiste vrednosti zmožnosti, pri katerih funkcija zadovoljstva doseže maksimum.

```
<Capability>
  <Name>A</Name>
  <Domain>A >= 1000 && A <= 1000</Domain>
</Capability>
```

Izpis izvorne kode 5.1: Primer zapisa zmožnosti

Element *Rank*

Element `Rank` (glej izpis 5.2) uporabimo, da navodila za ocenjevanje (funkcijo zadovoljstva) ponudnikovih zahtevanih zmožnosti podamo kar kot del samih zahtevanih zmožnosti. Takšen pristop, deklarativna navedba kriterija, nam omogoča, da funkcijo zadovoljstva preprosto spreminjamo brez posegov v programsko kodo, kar bi sicer zahtevala imperativna določitev funkcije neposredno v programski kodi. Funkcijo zadovoljstva lahko sestavimo iz nekaj preprostih matematičnih operatorjev in funkcij nad realnimi števili ter

if-else konstrukta (zapisanega v C-jevskem slogu, kot trojiški operator `?:`) tako, da se sklicujemo na imena zmožnosti.

```
<Rank>
  (max(A) >= 800)?1:(max(A)/800)
</Rank>
```

Izpis izvorne kode 5.2: Primer zapisa funkcije zadovoljstva

Element *RCapabilities*

Ta element je namenjen združitvi vseh številskih zmožnosti v eno samo trditev v dokumentu WS-Policy. Primer elementa, ki vsebuje opis dveh zahtevanih zmožnosti in ustrezno funkcijo zadovoljstva, je podan na izpisu 5.3.

```
<RCapabilities>
  <Capability>
    <Name>A</Name>
    <Domain>A >= 200 && A <= 5000</Domain>
  </Capability>
  <Capability>
    <Name>B</Name>
    <Domain>B >= 20 && B <= 500</Domain>
  </Capability>
  <Rank>
    max(A)*max(B)/2500000
  </Rank>
</RCapabilities>
```

Izpis izvorne kode 5.3: Primer zapisa trditve o zmožnostih

Celotna shema XML, ki opisuje navedene elemente, je podana v dodatku A.1.

5.3.2 Ujemanje politik

Ujemanje ponujanih in zahtevanih politik (in posledično zmožnosti) izvedemo z operacijo preseka dokumentov WS-Policy (kot presek določa specifikacija, Bajaj in sod. (2004b)). Rezultat preseka dveh politik je spet politika, ki je unija presekov vsake od alternativ iz prvega dokumenta z vsako od alternativ iz drugega dokumenta. Denimo, da je prva politika množica alternativ $P_1 = \{A_{11}, A_{12}, \ldots A_{1N}\}$ in druga politika $P_2 = \{A_{21}, A_{22}, \ldots A_{2M}\}$, tedaj je njun presek $P_1 \cap P_2 = \{A_{1n} \cap A_{2m} | n = 1 \ldots N, m = 1 \ldots M\}$.

Presek dveh alternativ je določen podobno kot presek dveh politik. Denimo, da je prva alternativa množica trditev $A_1 = \{T_{11}, T_{12}, \ldots T_{1N}\}$ in druga alternativa $A_2 = \{T_{21}, T_{22}, \ldots T_{2M}\}$. Tedaj je njun presek $A_1 \cap A_2 = \{T_{1n} \cap T_{2m} | n = 1 \ldots N, m = 1 \ldots M, T_{1n} \cap T_{2m} \neq \emptyset\}$. Presek alternativ je nova alternativa, določena kot unija presekov vsake od trditev iz A_1 z vsako od trditev iz A_2. Prazne preseke zavržemo, saj predstavljajo "trditev o ničemer" in ne ponujajo nobene informacije o delovanju storitve.

Presek dveh trditev je – tako kot določitev različnih vrst trditev samih in njih semantike – zunaj domene osnovne specifikacije. Specifikacija priporoča le, da je presek dveh trditev različnih vrst (zapisanih z različnima elementoma XML) prazen, presek dveh trditev enake vrste pa unija njunih vsebin. Nadaljna interpretacija takšnega preseka (ali je smiselen in zatorej neprazen) je odvisna od semantike vrste trditve in zato prepuščena interpretaciji uporabniškega programja.

Prikažimo povedano na preprostem primeru. Za prvo trditev uporabimo kar gornji primer zahtevanih številskih zmožnosti na izpisu 5.3. Njen presek s trditvijo `<Bla/>` je – glede na to, da gre za trditvi različnih vrst – prazen. Presek te trditve s trditvijo na izpisu 5.4 je neprazen in je podan na izpisu 5.5.

```
<RCapabilities>
  <Capability>
    <Name>A</Name>
    <Domain>A >= 100 && A <= 1000</Domain>
  </Capability>
  <Capability>
    <Name>B</Name>
    <Domain>B >= 10 && B <= 100</Domain>
  </Capability>
</RCapabilities>
```

Izpis izvorne kode 5.4: Trditev o ponujenih zmožnostih

```
<RCapabilities>
  <Capability>
    <Name>A</Name>
    <Domain>A >= 100 && A <= 1000</Domain>
  </Capability>
  <Capability>
    <Name>A</Name>
    <Domain>A >= 200 && A <= 5000</Domain>
  </Capability>
  <Capability>
    <Name>B</Name>
    <Domain>B >= 10 && B <= 100</Domain>
  </Capability>
```

```
  <Capability>
    <Name>B</Name>
    <Domain>B >= 20 && B <= 500</Domain>
  </Capability>
  <Rank>
    max(A)*max(B)/2500000
  </Rank>
</RCapabilities>
```

Izpis izvorne kode 5.5: Presek trditev

Glede na to, da poznamo semantiko trditve `RCapabilities`, lahko zapis preseka poenostavimo, tako da izračunamo najmanjšo ustrezno domeno zmožnosti A in B. Poenostavljen zapis prikazuje izpis 5.6.

```
<RCapabilities>
  <Capability>
    <Name>A</Name>
    <Domain>A >= 200 && A <= 1000</Domain>
  </Capability>
  <Capability>
    <Name>B</Name>
    <Domain>B >= 20 && B <= 100</Domain>
  </Capability>
  <Rank>
    max(A)*max(B)/2500000
  </Rank>
</RCapabilities>
```

Izpis izvorne kode 5.6: Poenostavljen presek trditev

Prazna alternativa – ta nastane, ko je presek vseh trditev iz izvirnih alternativ prazen – predstavlja nabor lastnosti, ki ni kompatibilen ne z zahtevano politiko odjemalca ne s ponujeno politiko ponudnika. V primeru, ko politika, ki nastane s presekom, vsebuje le prazno alternativo, sta izvirni politiki *nekompatibilni* in ustrezna odjemalec in ponudnik ne moreta sodelovati.

V nasprotnem primeru politika, ki je rezultat preseka, vsebuje množico alternativ, ki so sprejemljive tako odjemalcu kot ponudniku. Tako politiko imenujemo *kompatibilna*.

5.3.3 Ocenjevanje politik

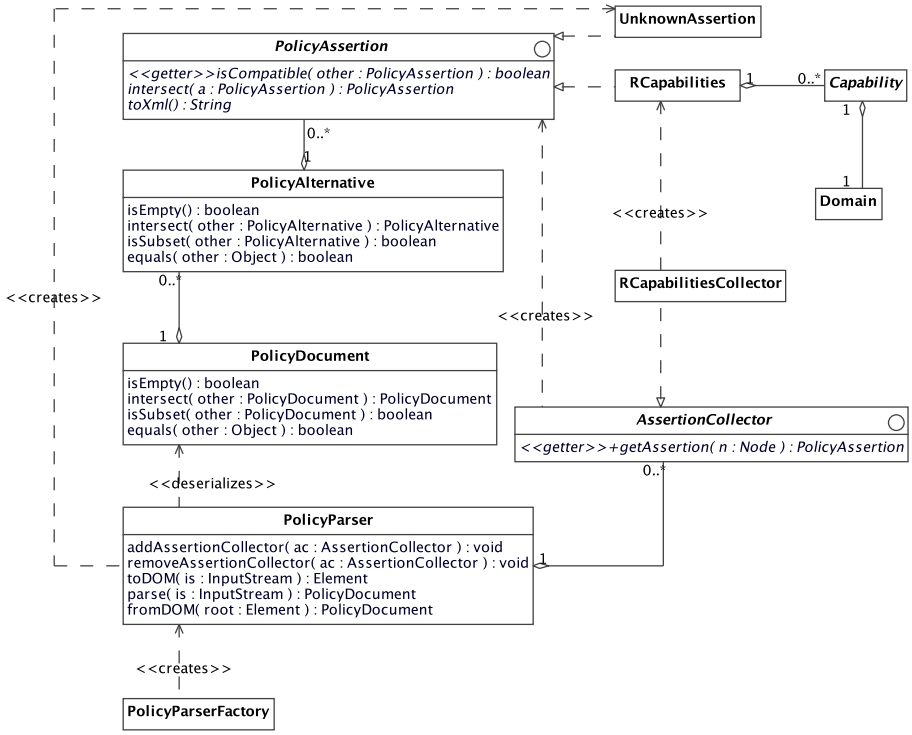
Politiko ocenimo tako, da ocenimo vse alternative znotraj nje s pomočjo funkcije zadovoljstva. Funkcijo zadovoljstva bo podal odjemalec v podobi postopka, ki kot vhod prejme posamezno alternativo, edini izhod pa je ocena. V primeru, ko imamo opravka

le z eno samo trditvijo o številskih zmožnostih, kakršne so predstavljene zgoraj, je ocena alternative kar enaka oceni trditve, ki ustreza izračunu izraza, podanega z elementom `Rank`.

5.3.4 Izvedba knjižnice

Analiza dokumentov WS-Policy

Statični model analize dokumentov prikazuje slika 5.2.



Slika 5.2: Statični model analize dokumentov WS-Policy

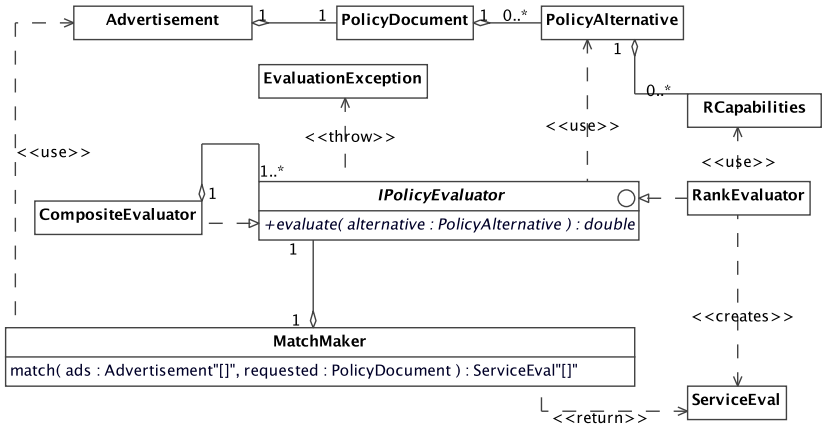
Razred `PolicyParser` analizira vhodni dokument WS-Policy in ga - v kolikor ustreza shemi - preslika v primerek razreda `PolicyDocument`. Ta vsebuje poljubno število alternativ, te pa poljubno število trditev. Analizator sam ne pozna posameznih vrst trditev: vse preslika v razred `UnknownAssertion`. Slednji ponuja le splošno funkcionalnost preseka trditev v skladu s predpisi specifikacije, ki smo jih opisali v razdelku 5.3.2.

Uporabnik mora priskrbeti logiko za preslikavo trditev, ki ga zanimajo: pomembna je predvsem izvedba preseka teh trditev. To stori tako, da pri analizatorju registrira ustrezne primerke vmesnika `AssertionCollector`. Ti vračajo primerke uporabnikovih razredov, ki ustrezajo zanimivim trditvam. Diagram prikazuje razrede, ki poskrbijo za primerno

preslikavo trditve `RCapabilities`: `CapabilitiesCollector` in `RCapabilities`. Njihova izvedba poskrbi za presek domen ustreznih zmožnosti, tako kot to kaže primer na izpisu 5.6.

Ujemanje in ocenjevanje

Razredni diagram izvedbe ujemanja in ocenjevanja prikazuje slika 5.3.



Slika 5.3: Statični model ujemanja in ocenjevanja politik

Za iskanje ujemanj med oglasi storitev in zahtevanimi zmožnostmi ter ocenjevanje leteh skrbi razred `MatchMaker`. Vhod metode `match` je polje oglasov (primerkov razreda `ServiceAdvertisement`), ki vsebujejo politike storitev s ponujenimi zmožnostmi, in dokument z zahtevano politiko odjemalca, ki vsebuje zahtevane zmožnosti.

Ujemanje je izvedeno s pomočjo preseka dokumenta z zahtevanimi zmožnostmi in dokumentov s ponujenimi zmožnostmi (po eden iz vsakega oglasa). Neprazni preseki vsebujejo alternative, sprejemljive za odjemalca.

Sprejemljive alternative ocenimo s pomočjo primerka vmesnika `IPolicyEvaluator`. Prototip priskrbi dva takšna primerka: `RankEvaluator` oceni alternativo na osnovi izraza `Rank` v trditvi o številskih zmožnosti (če ta obstaja). `CompositeEvaluator` združuje več primerkov vmesnika `IPolicyEvaluator` in sešteje njihove ocene ter vsoto normalizira na interval $[0, 1]$.

Rezultat ocenjevanja je urejeno polje primerkov razreda `ServiceEval`, ki vsebuje oglas ustreznega ponudnika, ocenjevano alternativo in oceno. Polje, ki ga vrne metoda `match`, je že urejeno po padajočih vrednostih ocen: prvi element polja je ocenjen najviše, zadnji najslabše, vsi pa so sprejemljivi za odjemalca. Na osnovi rezultata proces odkrivanja poskusi rezervirati najbolje ocenjenega ponudnika (glej razdelek 5.6).

5.4 Knjižnica Advertising

Knjižnica `Advertising` skrbi za oglaševanje politike (in posredno ponujenih zmožnosti), ki jo v danem trenutku lahko ponudi storitev.

5.4.1 Postopek izračuna zmožnosti

Na osnovi analize scenarija iz primera 3.2.1 ugotovimo, da so zmožnosti osnovane na računskih virih. Največjo vrednost zmožnosti lahko storitev ponudi, kadar je posvečena enemu samemu odjemalcu, ki mu zagotavlja vse vire, ki so na voljo. Ustrezne vrednosti zmožnosti lahko storitev izračuna na osnovi modela lastne izvedbe in lastnosti virov (hitrosti CPE, količine pomnilnika, hitrosti dostopa do diska ipd.) ali pa na osnovi meritve lastnega delovanja v neobremenjenem stanju (ko ne streže nikomur, takisto pa virov ne obremenjuje morebitno tretje breme).

Prikažimo to na omenjenem primeru: maksimalna vrednost zmožnosti SR temelji na računski hitrosti računalnika, ki gosti storitev (oziroma na hitrosti njene CPE). Denimo, da je ta vrednost, ki jo bodisi izračunamo ali izmerimo, SR_b. Imenujmo jo *osnovna vrednost zmožnosti*: to vsebuje *osnovna politika*. Dejanska vrednost zmožnosti bi bila enaka osnovni vrednosti natanko tedaj, ko bi se storitev v celoti posvečala enemu samemu odjemalcu. Temu seveda ni nujno tako: upoštevati je potrebno npr. minimalne zmožnosti, ki jih mora storitev zagotoviti drugim odjemalcem z obstoječimi pogodbami, in morebitno obremenitev računalnika s strani tretjih procesov. Dejansko vrednost dobimo tako, da osnovno vrednost zmožnosti (oziroma osnovno politiko, ki združuje vse zmožnosti) *transformiramo* glede na trenutno stanje ponudnikovega okolja (obstoječih odjemalcev, tretjih procesov ipd.). Rezultat je *trenutna vrednost zmožnosti* (oziroma *trenutna politika*).

Sam postopek transformacije posamezne zmožnosti je – tako kot operacija preseka – v domeni programja, ki to zmožnost predpiše, zato bo knjižnica izvedla le splošne postopke za transformacijo politik in posameznih alternativ, postopke za transformacijo posameznih trditev (torej z njimi popisanih zmožnosti) pa bo priskrbel uporabnik knjižnice, programer storitve.

Na primeru zmožnosti SR bi morali od osnovne vrednosti odšteti minimalne zmožnosti, zagotovljene vsakemu od N obstoječih odjemalcev. Trenutna vrednost je torej $SR_t = SR_b - \sum_{k=1}^{N} min(SR_k)$. Pri tem predpostavimo, da je računalnik namenjen izključno izvajanju storitve in zatorej ni možnosti nepredvidljive tretje obremenitve.

5.4.2 Vsebina oglasa

```
<Advertisement>
  <Interface>
    http://somewhere.org/namespaces/SomeService/SomeService
  </Interface>
  <Address>
    https://somewhere.org/services/SomeService
  </Address>
  <Policy>
    <ExactlyOne>
      <All>
        <RCapabilities>
          <Capability>
            <Name>A</Name>
            <Domain>(A >= 10 && A <= 100)</Domain>
          </Capability>
        </RCapabilities>
      </All>
    </ExactlyOne>
  </Policy>
</Advertisement>
```

Izpis izvorne kode 5.7: Primer oglasa storitve

Oglas storitve - primer je prikazan na izpisu 5.7 - vsebuje:

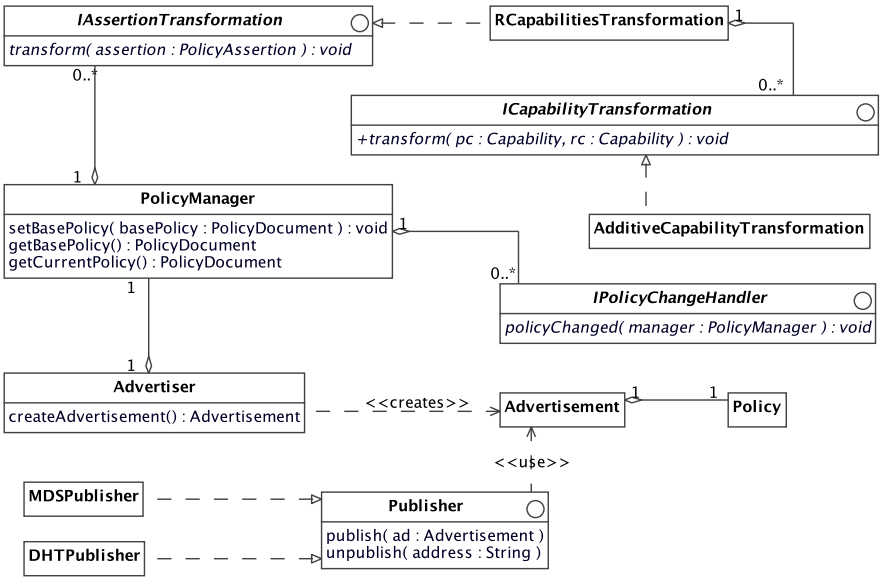
- opis funkcionalnosti storitve: prototip v ta namen uporabi imenski prostor dokumenta WSDL storitve (element `Interface`),
- naslov storitve: služi dostopu do storitve; zapisan je po standardu WS-Addressing (Box in sod., 2004),
- politiko storitve: celoten dokument WS-Policy, ki opisuje zmožnosti storitve.

Shema oglasa je predstavljena v dodatku A.2.

5.4.3 Izvedba knjižnice

Osrednjo vlogo, kot kaže statični model na sliki 5.4, ima razred `PolicyManager`: temu nastavimo osnovno politiko z metodo `setBasePolicy` in registriramo ustrezne transformacije za trditve o zmožnostih. Slednje so primerki vmesnika `IAssertionTransformation`.

V okviru prototipa smo razvili transformacijo za trditev o številskih zmožnostih (razred `RCapabilitiesTransformation`). Ta lahko transformira poljubno število posameznih



Slika 5.4: Statični model knjižnice Advertisement

zmožnosti: za vsako zmožnost znotraj trditve (glede na njeno ime) transformaciji dodamo po en primerek vmesnika `ICapabilityTransformation`. `AdditiveCapabilityTransformation` npr. skrbi za transformacijo zmožnosti, ki se ponašajo z lastnostjo, da se preprosto seštevajo (npr. zmožnost SR iz našega primera 3.2.1).

Ob vsaki spremebi trenutne politike sproži `PolicyManager` uporabnikove rokovalnike (ti izvedejo vmesnik `IPolicyChangeHandler`). Tipičen rokovalnik ob spremembi uporabi primerek razreda `Advertiser`, da sestavi nov oglas, ki vsebuje spremenjeno politiko, in ga objavi v iskalni storitvi s pomočjo vmesnika `Publisher`. Prototip vsebuje dve izvedbi tega vmesnika: `MDSPublisher` za objavo v GT4 MDS in `DHTPublisher` za objavo v porazdeljeni razpršeni tabeli.

5.5 Knjižnica Contract

Knjižnica `Contract` služi oblikovanju pogodb na osnovi zahtev odjemalcev ter upravljanju z obstoječimi pogodbami: pregledovanju, podaljševanju in prekinitvi.

5.5.1 Vmesniki spletnih storitev

Knjižnica določi dva tipa vrat spletnih storitev, ki ju mora podpreti vsak ponudnik. Prvi tip vrat, `Reservation`, je namenjen oblikovanju nove pogodbe na osnovi zahtevanih zmožnosti. V ta namen ponuja eno samo operacijo, `reserve`: njen edini vhodni parameter

je dokument WS-Policy, ki opisuje zahtevane zmožnosti, izhod pa je nova rezervirana pogodba, dokument XML, ki ustreza tipu `Contract`.

Ta vrata uporabi odjemalec, da rezervira ponudnika, ki ga je proces odkrivanja, ujemanja in ocenjevanja ponudnikov določil za najbolj ustreznega. Rezultat rezervacije je pogodba s kratkim rokom veljavnosti. Odjemalec jo lahko pregleda, da ugotovi, ali ponujene zmožnosti res ustrezajo zahtevanim, nato pa podaljša njeno veljavnost in prične z uporabo storitve.

Drugi tip vrat, `Contract`, je namenjen odjemalčevemu upravljanju s pogodbo. Ponuja dve operaciji: `contract`, ki obstoječi pogodbi podaljša veljavnost, in `cancel`, ki prekine obstoječo pogodbo.

Rezervacija vsebuje:

- naslov rezerviranega ponudnika,
- dokument WS-Policy, z eno samo alternativo, ki opisuje zahtevane zmožnosti ponudnika, ki naj jih ta rezervira; ta dokument je običajno rezultat ujemanja v procesu odkrivanja.

Rezervacijo oblikuje odjemalec (oziroma v njegovem imenu knjižnica Discovery, glej razdelek 5.6) in jo preko vrat `Reservation` pošlje ponudniku.

Pogodba vsebuje:

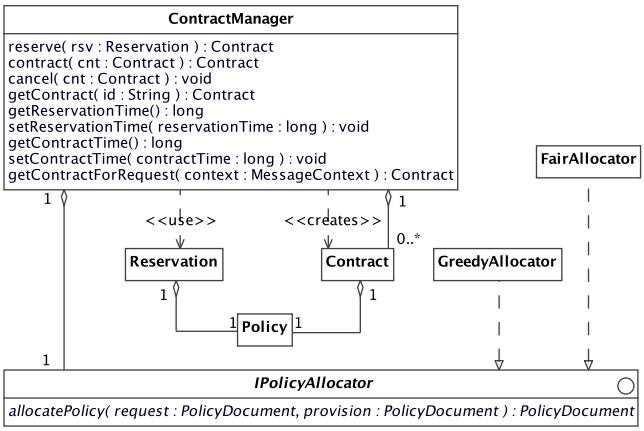
- globalno enolično oznako pogodbe,
- referenco na vstopno točko ponudnika storitve (ang. endpoint reference), ki služi za dostop do storitve v skladu s pogodbo,
- dokument WS-Policy, ki opisuje zmožnosti, ki jih bo ponudnik zagotovil odjemalcu,
- rok veljavnosti pogodbe: pogodba je veljavna in jo sme ponudnik upoštevati le pred iztekom roka.

Pogodbo na osnovi rezervacije oblikuje ponudnik in jo vrne odjemalcu. Zmožnosti, ki jih pogodba zagotavlja, morajo ustrezati zahtevanim, znotraj tega okvira pa ima ponudnik vso svobodo, koliko zmožnosti bo za nekega odjemalca zasegel.

Oba tipa vrat in dialekta XML za opis rezervacije in pogodbe sta podana v dodatku, v razdelkih A.3, A.4, A.5 in A.6.

5.5.2 Izvedba knjižnice

Statični model knjižnice prikazuje slika 5.5.



Slika 5.5: Statični model knjižnice Contract

Za vse zgoraj navedene operacije, rezervacijo, podaljšanje in prekinitev pogodbe, skrbi razred `ContractManager`. Ponudnik mora le poskrbeti, da sporočila, poslana vratom `Reservation` in `Contract`, pokličejo ustrezne metode v razredu `ContractManager`: `reserve`, `contract` in `cancel`.

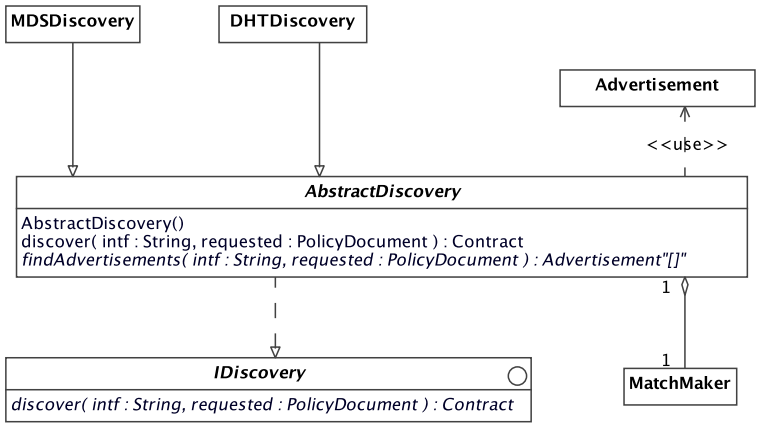
Za oblikovanje pogodbe, določitev v pogodbi zagotovljenih zmožnosti (kar je del metode `reserve`), uporabimo primerek vmesnika `IPolicyAllocator`. Izvedbe tega vmesnika odjemalcu dodelijo ustrezne zmožnosti. Ponudnik, ki bo uporabil `GreedyAllocator`, bo vsakemu odjemalcu zagotovil zmožnosti, ki so kar najbliže optimalnim, a bo zato lahko postregel ustrezno manjše število enakih odjemalcev. Raba razreda `FairAllocator` bo po drugi strani poskušala zagotoviti kar najbolj "pošteno" strežbo vseh odjemalcev, seveda v okviru minimalnih zahtevanih zmožnosti. Za ilustracijo predpostavimo, da odjemalec zahteva zmožnost SR v domeni $100 \leq SR \leq 1000$, ponudnik pa more zagotoviti $SR \leq 1000$. Raba prvega načina dodeljevanja bo v pogodbi odjemalcu zagotovila $SR = 1000$ in mu toliko v vsakem trenutku tudi zagotovila, a ponudnik ne bo mogel postreči nobenega odjemalca več. Raba drugega načina dodeljevanja zmožnosti pa bo v pogodbi zagotovila $100 \leq SR \leq 1000$ in odjemalcu dodelila $SR = 1000$, dokler bo edini odjemalec, ob prihodu novih odjemalcev pa se bo dodeljena vrednost manjšala, a ne niže kot $SR = 100$.

Možne so seveda implementacije še drugih načinov dodeljevanja, za katerega pa se bo odločil, je stvar ponudnika. Možno je celo, da ponudnik oglašuje več načinov dodeljevanja, odjemalec pa zahteva enega od njih na popolnoma enak način, kot se oglašujejo in zahtevajo ostale zmožnosti. Nenazadnje je način dodeljevanja zmožnosti spet le ena od nefunkcionalnih lastnosti storitve.

5.6 Knjižnica Discovery

Knjižnico Discovery uporabi odjemalec, da odkrije svojim zahtevam primernega ponudnika storitve. Odkrivanje poteka v treh korakih, kot to prikazuje diagram aktivnosti na sliki 4.3 v prejšnjem poglavju:

- iskanje oglasov vseh funkcionalno ustreznih ponudnikov (povpraševanje iskalne storitve),
- odkrivanje ponudnikov z ustreznimi zmožnostmi (ujemanje zahtevanih in ponujenih zmožnosti) in njihovo ocenjevanje,
- rezervacija najbolje ocenjenega ponudnika.



Slika 5.6: Statični model knjižnice Discovery

Najpomembnejše dele knjižnice prikazuje razredni diagram na sliki 5.6.

Odjemalec uporabi vmesnik `IDiscovery`, ki predpisuje eno samo metodo, `discover`. Tej podamo opis funkcionalnosti (v našem prototipu imenski prostor opisa storitve v jeziku WSDL) in dokument z zahtevanimi zmožnostmi.

Izvedba tega vmesnika, `AbstractDiscovery`, izvede ujemanje in ocenjevanje (s pomočjo razreda `MatchMaker`, opisanega v razdelku 5.3.4) in rezervacijo ponudnika (z rabo vmesnika `Reservation` na strani izbranega ponudnika). Prvi korak, iskanje oglasov funkcionalno ustreznih ponudnikov, je deklariran kot abstraktna metoda `findAdvertisements`, saj je odvisen od uporabljene iskalne storitve. Specializirana razreda `MDSDiscovery` in `DHTDiscovery` določita to operacijo za rabo GT4 MDS iskalne storitve oziroma porazdeljene razpršene tabele.

5.7 Uporaba prototipa

V tem razdelku opišemo spremembe, ki so potrebne, da obstoječo kodo odjemalca in ponudnika prilagodimo tako, da bosta uporabljala naš prototip.

5.7.1 Izvedba ponudnika

Denimo, da obstaja izvedba ponudnika poljubne spletne storitve, ki bi jo radi oglaševali s pomočjo našega prototipa in ji omogočili dinamično dodeljevanje zmožnosti odjemalcem na osnovi njihovih zahtev. Prilagoditev zahteva naslednje spremembe:

1. izvedba ponudnika mora ustvariti po en primerek razredov `PolicyManager`, `ContractManager`, `Advertiser` in ustrezno izvedbo vmesnika `Publisher` (glede na uporabljano iskalno storitev, v kateri oglašujemo),
2. ponudnik mora določiti vse ponujane zmožnosti in postopke za njihov izračun ter transformacije,
3. spletna storitev mora podpreti tipa vrat `Reservation` in `Contract` (glej dodatka A.4 in A.6),
4. sporočila, ki prispejo na ta vrata, mora podati metodam `reserve`, `contract` in `cancel` razreda `ContractManager`.

5.7.2 Izvedba odjemalca

Obstoječega odjemalca moramo prirediti tako, da namesto trenutnega načina povezovanja s ponudnikom (npr. naslov storitve zapečen v kodo ali zapisan v nastavitveni datoteki ipd.) uporabi primerno izvedbo vmesnika `IDiscovery`, odvisno od uporabljane iskalne storitve. V ta namen mora določiti zahtevane zmožnosti in sestaviti ustrezen dokument WS-Policy, ki jih opisuje.

Po sklenitvi pogodbe mora skrbeti za njeno podaljševanje (če je to potrebno), vsak klic ponudnika pa mora opremiti z oznako pogodbe, ki se prenese kot poseben element zaglavja sporočila SOAP.

Poglavje 6

Preizkus prototipa

V tem poglavju predstavimo rezultate preizkusa prototipa, s katerim potrdimo, da naš pristop in njegova prototipna izvedba omogočata povečanje razpoložljivosti ponudnikov na račun zadovoljitve odjemalcev.

6.1 Preizkusni scenariji

V namen preizkusa bomo uporabili ponudnike in odjemalce storitve Monte Carlo simulacije, kakršno smo opisovali že v primerih 3.1.1 in 3.2.1. Odkrivanje ponudnikov in dodeljevanje njihovih virov na osnovi zmožnosti s pomočjo našega prototipa bomo primerjali z dvema danes uveljavljenima principoma strežbe: strežbo po najboljših močeh (ang. best-effort service) in optimalno strežbo z nadzorom dostopa (ang. admission control).

Strežba po najboljših močeh je princip, ki je danes najpogosteje uporabljan pri storitvah na svetovnem spletu. Ponudnik vedno poskusi postreči odjemalca, ne glede na svoje zmožnosti in odjemalčeve zahteve. Pri tem pristopu je v primeru preobremenitve ponudnikov spričo velikega števila odjemalcev zelo verjetno, da bodo vsi odjemalci postreženi neustrezno (slabše od minimalnih zahtevanih zmožnosti).

Optimalna strežba z nadzorom dostopa je pogost pristop pri storitvah, kjer je pomembno zagotavljanje ustrezne kakovosti storitev. Ponudnik sprejema odjemalce le, v kolikor jih je zmožen optimalno postreči. Ostale zavrne. Na ta način sicer sprejetim odjemalcem zagotovi ustrezno kakovost storitve, razpoložljivost ponudnika pa se pomembno zmanjša.

Pričakujemo, da se bo naš pristop k dodeljevanju ponudnikov in njihovi strežbi izkazal za

boljšega od obeh omenjenih. Ponudniki, ki strežejo po najboljših močeh, v primeru preobremenitve namreč postanejo nerazpoložljivi za vse odjemalce (ker ne strežejo v skladu z minimalnimi zahtevami). Ponudniki, ki nadzirajo dostop odjemalcev, pa v primeru preobremenitve sicer uspešno postrežejo manjši del odjemalcev, a ostanejo nerazpoložljivi večjemu delu, ki ga *a priori* zavrnejo.

Oglejmo si najprej delovanje preizkusnega ponudnika in odjemalca.

6.1.1 Ponudnik

Preizkusni ponudnik s pomočjo metode Monte-Carlo ocenjuje pričakovano vrednost strukturiranih obveznic z možnostjo predčasnega odkupa (ang. callable structured bonds). Zavoljo velike računske zahtevnosti te naloge se ogibamo deterministični analizi in kot sprejemljiv kompromis med računsko zahtevnostjo in praktično uporabnostjo rezultatov izberemo pristop Monte-Carlo simulacije. Ta oceni vrednost obveznice glede na množico možnih poti obveznice skozi čas (vsako pot določajo spremembe vrednosti parametrov, ki vplivajo na bodočo vrednost obveznice (npr. obrestnih mer), čas odkupa ipd.). Izvedba konkretne simulacije uporablja metodo *Least Squares Monte-Carlo (LSM)*.

Storitev je računsko zahtevna (saj verodostojna rešitev zahteva simulacijo velikega števila poti) in zato odvisna predvsem od zmožnosti CPE ponudnika storitve. Ponudnik oglašuje zmožnost SR: ta predstavlja število različnih scenarijev (v našem primeru različnih *poti obveznice*, Longstaff in Schwartz (2001)), ki jih je ponudnik zmožen analizirati v eni uri.

Ponudnikova storitvena vrata vsebujejo eno samo operacijo, `simulate(string env)`. Njen edini parameter `env` vsebuje opis obveznice, spremenljivk, ki vplivajo na ocenjevanje, in časovno omejitev za vračanje rezultata. Semantika simulacije nima posebnega pomena za pričujoče delo, o samem postopku pa se radovedni bralec lahko podrobno pouči v Longstaff in Schwartz (2001).

Na testni infrastrukturi, ki je bila sestavljena iz računalnikov podobnih zmogljivosti, je bila vrednost zmožnosti SR na popolnoma neobremenjenem računalniku med 3500 in 3600 simuliranih poti v eni uri (približno ena simulacija na sekundo).

6.1.2 Odjemalec

Odjemalec zahteva, da ponudnik vrne oceno pred iztekom neke časovne omejitve. V konkretnem primeru rabe Monte-Carlo simulacije za analizo finančnih trgov je npr. zaželeno, da so rezultati na voljo pred začetkom naslednjega trgovalnega dne. Po drugi strani pa mora ponudnik simulirati dovolj veliko število poti, da bo ocena primerno zanesljiva. V primeru, ko ni na voljo ponudnika, ki bi zmogel simulirati želeno število poti v omejenem času, bi prišla prav tudi manj zanesljiva analiza (t.j. rezultat simulacije manjšega števila poti).

Odjemalci v našem preizkusu bodo od ponudnika zahtevali simulacijo vsaj desetih in največ (optimalno) šestdesetih poti: $10 \leq SR \leq 60$. V resnici so zahteve za zanesljivo analizo nekaj velikostnih redov večje (npr. nekaj deset tisoč poti), za namene preizkusa smo se odločili za nižje vrednosti in tako skrajšali trajanje. Števila so izbrana tako, da optimalen izračun na neobremenjem ponudniku, ki služi enemu samemu odjemalcu, traja eno minuto, kar je tudi še sprejemljiva meja za vrnitev rezultata (časovna omejitev).

6.1.3 Mere

Prototip in njegovo uspešnost pri doseganju cilja, večanju razpoložljivosti ponudnikov na račun zadovoljitve odjemalčevih zahtev, bomo presojali s pomočjo naslednjih mer:

1. delež uspešno postreženih odjemalcev. Uspešno postrežen je odjemalec, ki je prejel rezultat v skladu z zahtevanimi zmožnostmi ($10 \leq SR \leq 60$) pred iztekom časovne omejitve. Ta mera neposredno ocenjuje razpoložljivost ponudnikov v sistemu, saj je vsaj eden izmed ponudnikov odjemalcu na voljo (je razpoložljiv) natanko tedaj, ko uspe odjemalca postreči v skladu z njegovimi zahtevami,
2. kakovost opravljene storitve po odjemalcu. Kakovost predstavlja v našem primeru število simuliranih poti in neposredno meri zadovoljitev odjemalčevih zahtev. Za predstavitev bomo odjemalce razporedili v več razredov, od neuspešnih (zavrnjeni odjemalci ali manj kot deset simuliranih poti) do optimalno postreženih (šestdeset simuliranih poti),
3. povprečna kakovost opravljene storitve je povprečje prejšnje mere čez vse odjemalce. Meri povprečno zadovoljitev naključnega odjemalca v sistemu,
4. delež optimalno postreženih odjemalcev.

Mere zbiramo tako, da preverimo dnevniške zapise ponudnikov in odjemalcev, kamor zapisujemo vse podatke, ki jih potrebujemo za izračun mer.

6.1.4 Scenarij 1: preobremenitev ponudnikov

Ta scenarij razišče obnašanje našega prototipa v primeru preobrementive ponudnikov spričo velikega števila sočasnih odjemalcev. V vsakem trenutku je velika verjetnost, da je v sistemu toliko odjemalcev, da skupaj za optimalno strežbo zahtevajo več virov, kot jih morejo zagotoviti vsi ponudniki v sistemu.

Po uspešnem zagonu vseh ponudnikov v sistemu pričnemo z zagonom odjemalcev. Vstop odjemalcev v sistem (štejemo, da odjemalec vstopi v sistem, ko sproži zahtevo za odkrivanje ponudnika) modeliramo kot homogen Poissonov proces (Snyder in Miller, 1991)

z intenziteto $\lambda = 0.2/s$: v povprečju nov odjemalec vstopi v sistem vsakih pet sekund. Vsak odjemalec poskusi odkriti in uporabiti ponudnika storitve, ki lahko ponudi zahtevano zmožnost $10 \leq SR \leq 60$. Časovna omejitev za vrnitev rezultata simulacije je ena minuta.

Preizkus zaključimo, ko mine ena minuta od vstopa zadnjega odjemalca v sistem.

6.1.5 Scenarij 2: zunanja obremenitev

Ta scenarij nadgradi prvega (glej razdelek 6.1.4) z uvedbo naključne zunanje obremenitve. Zunanjo obremenitev predstavljajo računsko intenzivni procesi, ki tečejo na istem računalniku kot ponujana storitev. Ob predpostavki, da ima proces storitve enako prioriteto kot ti procesi, in ob rabi poštenega razporejanja (ang. fair scheduling) procesov v operacijskem sistemu, vsak tak proces zmanjša osnovne vrednosti ponudnikove zmožnosti SR: prvi na polovico, drugi na tretjino ipd.

Prihod (zagon) in odhod (ustavitev) teh procesov modeliramo kot homogena Poissonova procesa z intenziteto $\lambda = 0.01/s$. Torej: v povprečju na vsakih sto sekund poženemo po en zunanji proces, enega pa ustavimo.

6.1.6 Scenarij 3: izpadi ponudnikov

Tudi ta scenarij je nadgradnja prvega z možnostjo odpovedi vozlišča, ki gosti ponudnika storitve. V primeru odpovedi morajo vsi odjemalci ponudnika, ki je odpovedal, poiskati novega ponudnika in mu posredovati isto nalogo, a s krajšo časovno omejitvijo (očitno je del časa, ki je bil izvirno na voljo, že potekel). Dokler se vozlišče ne zažene ponovno, ni na voljo odjemalcem, ki morajo shajati s preostalimi ponudniki.

Odpoved in ponoven zagon odpovedanega vozlišča ponovno modeliramo kot Poissonova procesa z intenziteto $\lambda = 0.01/s$: v povprečju na vsakih sto sekund odpove eno vozlišče, eno od nedelujočih vozlišč pa ponovno oživi.

6.2 Preizkusno okolje

Preizkus smo izvedli na omrežju PlanetLab (glej PlanetLab Consortium). Omrežje PlanetLab sestavljajo vozlišča, opremljena z običajno strojno opremo višjega razreda (ob času tega pisanja so bile minimalne zahteve CPE Intel P4 s frekvenco 3.2GHz ali AMD Athlon 3200+ in 4GB fizičnega pomnilnika), ki jo gostijo sodelujoče ustanove s celega sveta. Vozlišča so opremljena s posebej prirejeno različico operacijskega sistema Linux, ki omogoča postavitev poljubnega števila virtualnih strežnikov na vsakem fizičnem vozlišču in enostavno oddaljeno upravljanje z njimi.

Uporabnik dobi *rezino* omrežja (ang. slice), ki predstavlja podmnožico vseh fizičnih vozlišč omrežja, na vsakem od teh vozlišč pa ima na voljo (od ostalih uporabnikov) ločen virtualni strežnik in dostop do njegove ukazne vrstice s pomočjo storitve SSH.

Prototip smo preizkušali na omrežju sto petdesetih vozlišč. Uporabili smo iskalno storitev, zasnovano kot porazdeljeno razpršeno tabelo (glej razdelke 5.1.4, 5.2 5.4.3 in 5.6 in članek (Močnik in sod., 2006) za razlago implikacij). Štiri vozlišča so gostila ponudnike Monte-Carlo storitve, ostala pa druge storitve, ki so služile za simulacijo resničnega okolja, v katerem obstaja množica različnih storitev. Odjemalce smo poganjali z enega samega vozlišča zunaj omrežja PlanetLab. Z istega vozlišča smo sprožali tudi zahteve za zagon in prekinitev zunanjih procesov (glej razdelek 6.1.5) in odpoved oziroma oživitev vozlišča (glej razdelek 6.1.6).

6.3 Rezultati

Vsakega od testnih scenarijev, opisanih v razdelkih 6.1.4, 6.1.5 in 6.1.6, smo izvedli z rabo vsakega od naslednjih treh načinov odkrivanja ponudnikov in dodeljevanja ponudnikovih virov odjemalcem (glej razdelek 6.1):

1. strežba s povečano razpoložljivostjo je uporabljala naš prototip v celoti, z namenom, da v primerih preobremenitve ponudnikov omogoči povečanje razpoložljivosti na račun zadovoljitve odjemalcev. V naslednjih grafih in tabelah je označena kot *povečana razpoložljivost*,

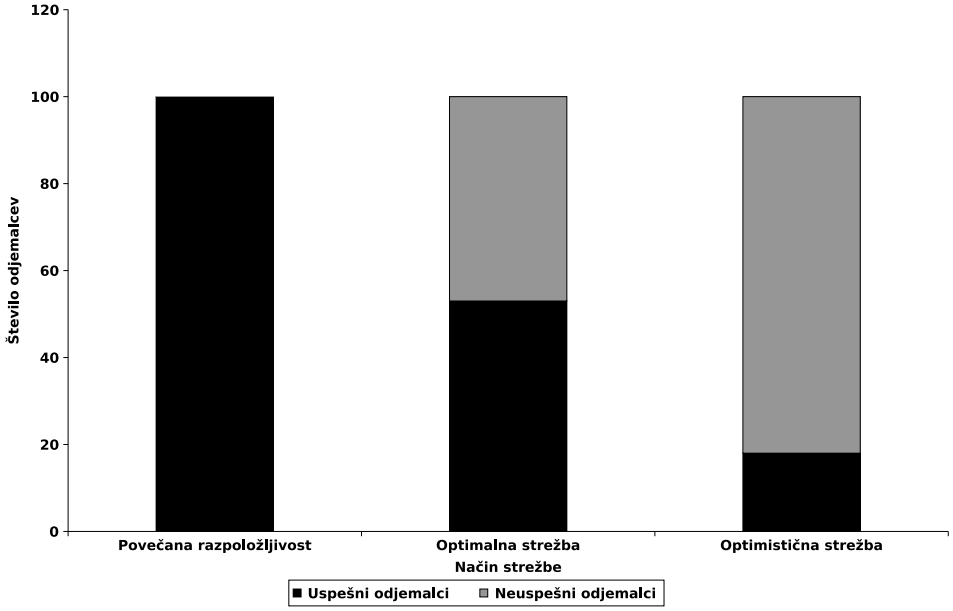
2. optimalna strežba z nadzorom dostopa uporablja naš prototip le za nadzor dostopa glede na ujemanje ponujenih in optimalnih zahtevanih zmožnosti, ne pa tudi za povečanje razpoložljivosti. Ta način je označen kot *optimalna strežba*,

3. strežba po najboljših močeh uporablja krožni (ang. round-robin) princip dodeljevanja ponudnikov in ne upošteva zahtevanih zmožnosti. Označena je kot *optimistična strežba*.

Za vsakega od scenarijev bomo predstavili razdelitev odjemalcev glede na kakovost storitve (torej glede na to, koliko poti je ponudnik simuliral). Z razredom *0* označimo odjemalce, ki jih ponudniki *a priori* zavrnejo (razen v primeru optimistične strežbe, ki sprejme vse odjemalce, ne glede na trenutno stanje virov). Ostali razredi označujejo intervale števila simuliranih poti, ki tvorijo končno rešitev. Razreda *0* in *1-10* skupaj vsebujeta *neuspešne* odjemalce: tiste, ki bodisi niso našli primernega ponudnika ali pa je ponudnik storitev zanje izvedel slabše od minimalnih zahtev. Za te odjemalce so bili ponudniki *nerazpoložljivi*. Ostali odjemalci se uvrstijo med *uspešne*, odkrili so *razpoložljivega* ponudnika.

V prihodnjih razdelkih za vsak scenarij prikažemo porazdelitev stotih odjemalcev, ki vstopijo v sistem med preizkušanjem, v vsakem od scenarijev in za vse tri načine odkrivanja ponudnikov in dodeljevanja virov. Pričakujemo, da bo število uspešnih odjemalcev največje pri odkrivanju ponudnikov in dodeljevanju virov s pomočjo našega prototipa (oznaka *povečana razpoložljivost*). Za vsak scenarij navedemo še povprečno kakovost rezultata storitve preko vseh odjemalcev in delež optimalno postreženih odjemalcev.

6.3.1 Scenarij 1: preobremenitev ponudnikov

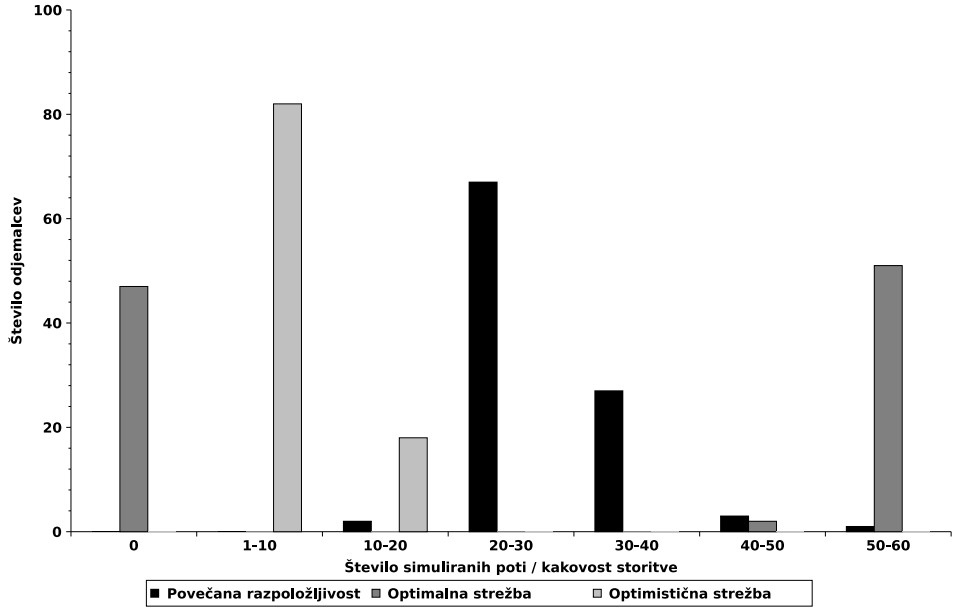


Slika 6.1: Scenarij 1: uspeh odjemalcev

	Povprečna kakovost storitve (SR)	Delež optimalno postreženih odjemalcev (%)	Razpoložljivost (%)
Povečana razpoložljivost	27,84	1	100
Optimalna strežba	28,64	51	53
Optimistična strežba	7,72	0	18

Tabela 6.1: Scenarij 1: kakovost storitve

Rezultati, prikazani na slikah 6.2 in 6.1 ter v tabeli 6.1, kažejo, da uspe v preobremenjenem sistemu naš prototip postreči vseh sto odjemalcev v skladu z njihovimi zahtevami. Večina prejme rezultate na osnovi med 20 in 40 simuliranih poti, s povprečjem 27,48. Optimalna strežba se sicer ponaša z boljšo povprečno kakovostjo storitve, a je ta nerazpoložljiva skoraj polovici (47) odjemalcev. Ostali so, seveda, postreženi optimalno ali skoraj optimalno (majhna odstopanja so posledica nenatančne umeritve zmožnosti ob



Slika 6.2: Scenarij 1: razdelitev odjemalcev glede na prejete zmožnosti

zagonu storitve). Optimistična strežba je popolnoma neprimerna za takšen scenarij: le 18 izmed 100 odjemalcev je primerno postreženih, vsi ostali pa so postreženi preslabo in zatorej označeni kot neuspešni.

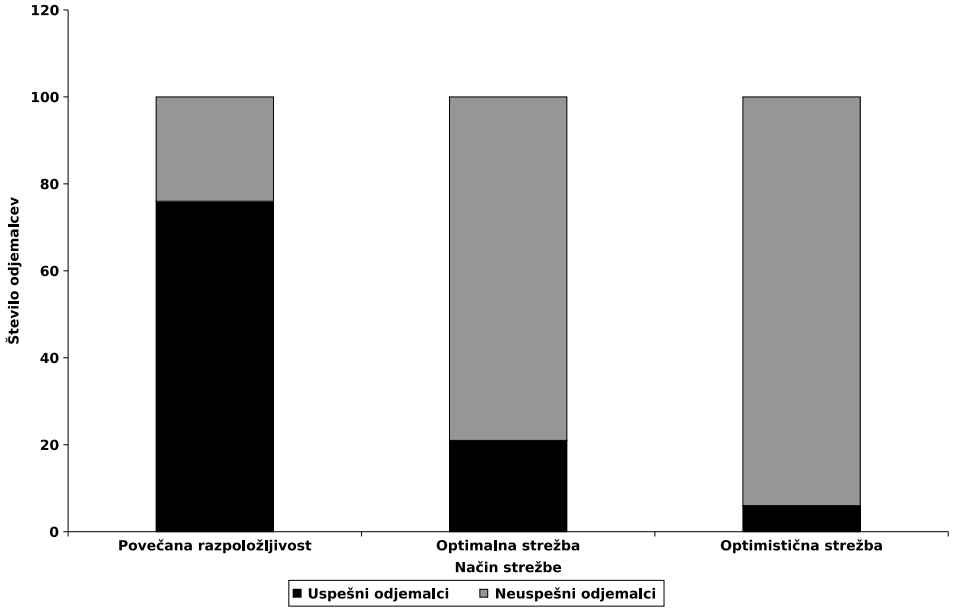
Rezultati jasno pokažejo, da raba naše paradigme večanja razpoložljivosti na račun zadovoljitve zahtev prav zares poveča razpoložljivost ponudnikov v preobremenjenem sistemu v primerjavi z ostalima preverjenima pristopoma.

6.3.2 Scenarij 2: zunanja obremenitev

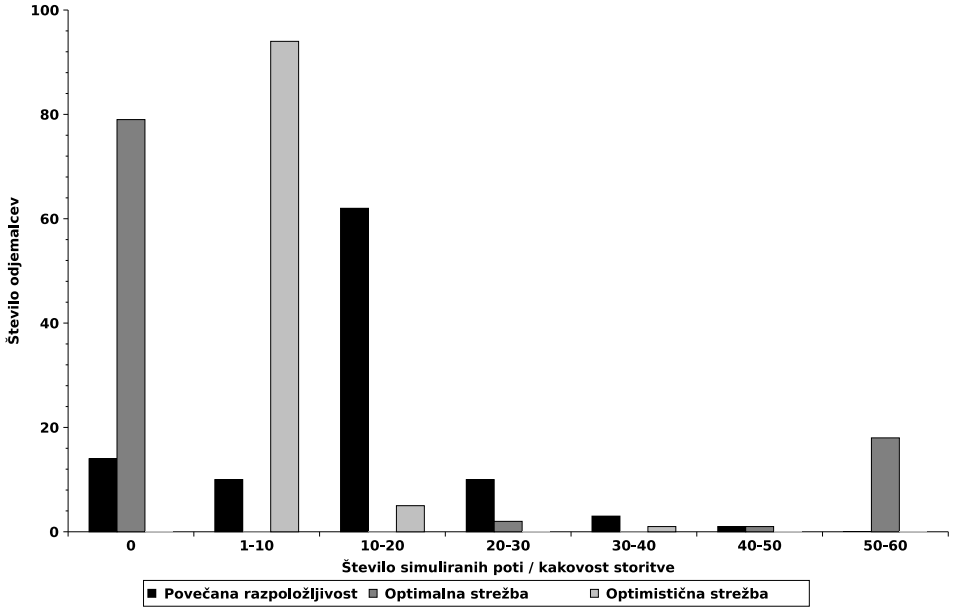
	Povprečna kakovost storitve (SR)	Delež optimalno postreženih odjemalcev (%)	Razpoložljivost (%)
Povečana razpoložljivost	13,46	0	76
Optimalna strežba	12,16	18	21
Optimistična strežba	4,09	0	6

Tabela 6.2: Scenarij 2: kakovost storitve

V primeru, ko so viri ponudnikov obremenjeni še z zunanjimi procesi, na katere ponudnik nima vpliva, nam uspe z rabo prototipa uspešno postreči 76 izmed 100 odjemalcev. Po drugi strani se v teh razmerah močno poslabša pristop z *optimalno strežbo*: na ta način uspemo uspešno postreči le 21 odjemalcev. Razlog za to poslabšanje gre iskati v dejstvu, da ponudniki storitev, ki so obremenjeni z vsaj enim zunanjim procesom, zavračajo več



Slika 6.3: Scenarij 2: uspeh odjemalcev



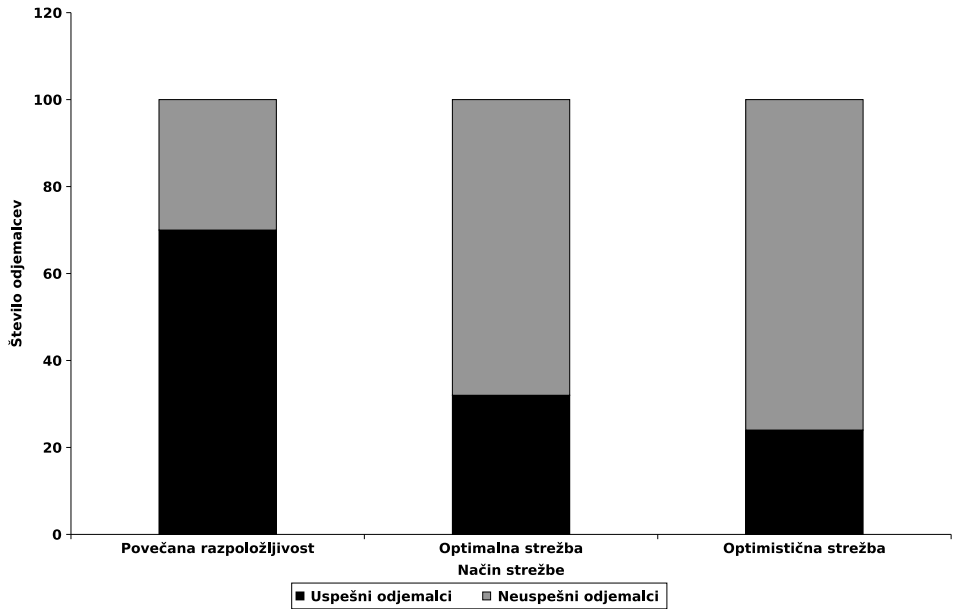
Slika 6.4: Scenarij 2: razdelitev odjemalcev glede na prejete zmožnosti

odjemalcev, saj jih spričo manjšega deleža CPE, ki jim je na voljo, ne zmorejo optimalno postreči. Ob rabi naše paradigme ponudnik po drugi strani pravilno ugotovi, da je še vedno zmožen ponuditi storitev (resda slabše kakovosti), in tudi ustrezno prilagodi dodelitve virov posameznim odjemalcem spričo rabe poštenega dodeljevanja (glej razdelek 5.5.2). Res je povprečna kakovost storitve nižja, a razpoložljivost vseeno ostane visoka.

Optimistična strežba uspe v tem scenariju postreči le 6 izmed 100 odjemalcev.

Spričo velikega števila zavrnjenih odjemalcev pri optimalni strežbi tokrat prototip ponuja tudi boljšo povprečno kakovost storitve.

6.3.3 Scenarij 3: izpadi ponudnikov



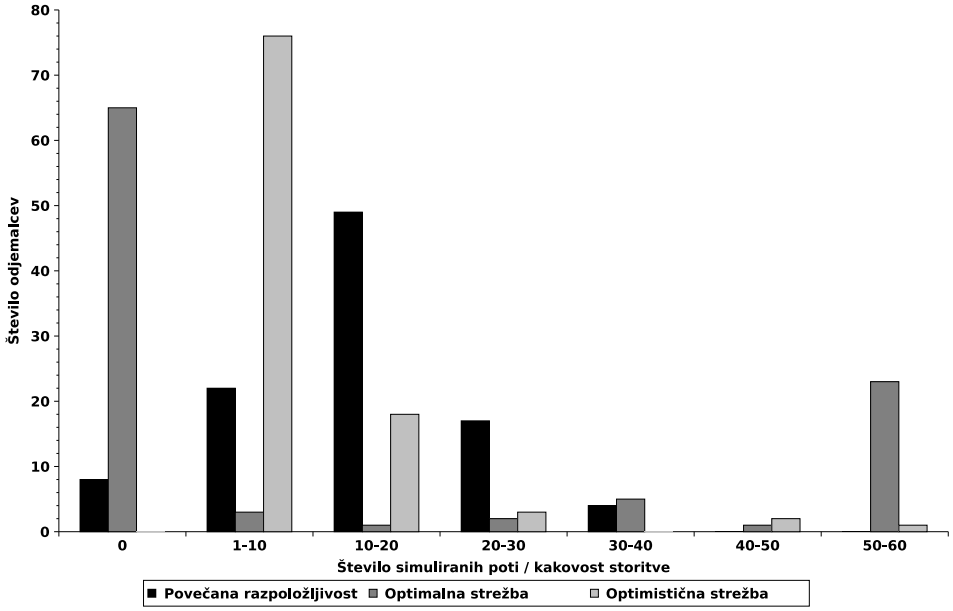
Slika 6.5: Scenarij 3: uspeh odjemalcev

	Povprečna kakovost storitve (SR)	Delež optimalno postreženih odjemalcev (%)	Razpoložljivost (%)
Povečana razpoložljivost	13,86	0	70
Optimalna strežba	17,23	23	32
Optimistična strežba	8,96	1	24

Tabela 6.3: Scenarij 3: kakovost storitve

Zadnji preizkusni scenarij razišče vpliv našega pristopa v primeru odpovedi ponudnikov. Ob odpovedi mora odjemalec poiskati drugega ponudnika, a tokrat z ustrezno višjimi zahtevanimi zmožnostmi, saj ima na voljo manj časa za enako število poti, ki jih mora ponudnik simulirati.

Storitev ne izroča vmesnih rezultatov (npr. po vsaki simulirani poti) odjemalcu, zato mora ta celoten izračun ponoviti s pomočjo drugega ponudnika. Če ponudnik odpove pozno v odjemalčevem življenju, tik pred iztekom roka, mora za optimalno strežbo (v



Slika 6.6: Scenarij 3: razdelitev odjemalcev glede na prejete zmožnosti

predpisanem roku, seveda) zahtevati mnogo višje zmožnosti ponudnika. Takšnih ponudnikov bo manj, zato se optimalna strežba ponovno obnese slabo. Uporaba našega prototipa, ki omogoči slabšo zadovoljitev odjemalca, pa po drugi strani poveča razpoložljivost ponudnikov tudi v takšnih razmerah.

Poglavje 7

Zaključek

V tem poglavju si ogledamo izpolnitev izvirnih ciljev in prispevke tega dela k področju storitveno usmerjenih porazdeljenih sistemov. Na koncu podamo smernice za bodoče delo na obdelanem področju.

7.1 Prispevki dela

Pričujoče delo je uvedlo paradigmo povečanja razpoložljivosti na račun izpolnitve zahtev odjemalca, nov pristop k načrtovanju trdnih široko porazdeljenih računalniških sistemov. Paradigmo smo razložili in utemeljili v razdelku 3.2. V razdelku 3.3 smo uvedli osnovne pojme, uporabljene v našem delu. Razdelek 3.4 je razložil razmerje predstavljene paradigme do obstoječega sorodnega dela na področju porazdeljenih računalniških sistemov.

Proces odkrivanja ponudnikov smo razširili z možnostjo odkrivanja na podlagi nefunkcionalnih zahtev odjemalca s pomočjo postopka ujemanja ponujanih in zahtevanih zmožnosti. Ocenjevanje ujemanj smo uporabili kot osnovo za dodeljevanje ponudnikov odjemalcem. Oba postopka sta bila opisana v razdelkih 4.2.1 in 4.3.1. Ujemanje služi tudi kot osnova za vpeljavo paradigme povečanja razpoložljivosti na račun izpolnitve zahtev odjemalca.

V poglavju 4 smo predstavili abstrakten model sistema in ga v razdelku 5 preslikali na konkretne tehnologije (spletne storitve in z njimi povezane standarde, glej razdelek 5.1). To nam omogoča izdatno pouporabo obširnega obstoječega dela na področju vmesnega sloja in preprosto prilagoditev obstoječih odjemalcev in ponudnikov storitev. V poglavju 5 podrobno predstavimo izvedeni prototip sistema.

Preizkus prototipa v realnem okolju in na realnem problemu – rezultate smo predstavili v poglavju 6 – je potrdil našo tezo: z rabo paradigme povečanja razpoložljivosti na račun izpolnitve zahtev odjemalca lahko pomembno povečamo razpoložljivost porazdeljenega programja. Ta pristop se obnese v primerih, ko je razpoložljivost ogrožena: ob preo-

bremenitvi sistema zavoljo velikega števila sočasnih odjemalcev, nepredvidljivih zunanjih obremenitvah virov in ob izpadih ponudnikov.

Proces ujemanja ponudnikov in odjemalcev na osnovi nefunkcionalnih zahtev in ocenjevanje ujemanj odjemalcu omogoči, da izbere najboljšega ponudnika storitve iz množice primernih. Metoda za opis zahtevanih zmožnosti nam omogoča, da določimo več alternativnih naborov zahtevanih zmožnosti, ki različno zadovoljijo odjemalca. V primeru, ko ni na voljo ponudnika, ki bi zmogel optimalno zadovoljiti zahteve odjemalca, lahko odjemalec izbere ponudnika, ki bo zadovoljil vsaj minimalne zahteve, ne pa optimalnih, in tako poveča razpoložljivost ponudnikov. Predstavljeni pristop je resda uporaben le pri razredu storitveno usmerjenega programja, ki lahko ponuja oziroma uporablja različne nabore nefunkcionalnih lastnosti (npr. različne stopnje kakovosti storitve), a kot pokažejo primeri, izpeljani iz resničnih aplikacij, teh ni malo. Spričo dejstva, da v široko porazdeljenih sistemih, ki zaobjemajo omrežja več organizacij, ni preprosto in še manj poceni priskrbeti primerno količino redundantnih virov, ki bi zmogli zagotavljati razpoložljivost, verjamemo, da naš pristop predstavlja pomemben način povečevanja razpoložljivosti.

7.2 Bodoče delo

V našem delu smo predpostavili, da so ponudniki "pošteni" in odjemalcu vedno zagotavljajo zmožnosti (in posledično vire) v skladu s pogodbo. Zlonameren ponudnik, ki oglaševanih in s pogodbo obljubljenih zmožnosti pri uporabi ne bi zagotovil odjemalcu, bi otežil ali celo onemogočil uporabo sistema. Na osnovi tega razmisleka je jasno, da moramo odjemalcu omogočiti sklepanje o tem, koliko je oglaševani ponudnik *vreden zaupanja* (ang. trustworthy). S to težavo se srečujejo vsa velika, odprta, decentralizirana omrežja. Rešujejo jih z modeli *porazdeljega zaupanja* (ang. distributed trust, Abdul-Rahman in Hailes (1997); Suryanarayana in sod. (2006); Dimmock in sod. (2004)). V okviru bodočega dela bi veljalo razmisliti o uvedbi sorodnega sistema v našo arhitekturo.

Trenutno predstavljena arhitektura in prototip podpirata le preprosto povezovanje enega odjemalca z enim samim ponudnikom. Problem sestavljanja storitve iz več osnovnih storitev ostaja odjemalcu, ravno tako kot sklepanje o nefunkcionalnih lastnostih sestavljene storitve. Naš sistem bi veljalo razširiti tako, da bi omogočal sestavljanje storitev na osnovi zahtevanih nefunkcionalnih lastnosti končne, sestavljene storitve. Ta problem je neprimerno težji, saj zahteva preiskovanje prostora vseh sestavljenih storitev (ta je številno neskončen, saj lahko v splošnem isto osnovno storitev v eni sami sestavljeni uporabimo poljubnokrat) z namenom najti (čimbolj) optimalno sestavo. Spričo očitne kombinatorične eksplozije izčrpno preiskovanje ne pride v poštev, po drugi strani pa odprtost sistema, najbolj dejstvo, da lastnosti storitev niso vnaprej določene, zbuja dvom, da bi bilo mogoče poiskati primerno kakovostno splošno hevristiko, ki bi vedno delovala razumno dobro. Najbolj obetaven se zdi pristop z rabo metahevristik (Blum in Roli, 2003), ki bi obstoječe hevristike prilagajale spremembam sistema (novim storitvam, novim lastnostim

storitev) na osnovi povratne informacije uporabnikov o zadovoljstvu s poprej najdenimi rešitvami.

Dodatek A

Programska koda

Dodatek vsebuje dele kode, na katere se sklicujemo v magistrskem delu. Koda je namenjena zgolj in le boljšemu razumevanju dela, zato ji pogosto manjkajo nekateri deli, ki v ta namen niso nujni. Pričakovati, da se bo koda prevedla, zategadelj ne gre. Celotna izvorna koda je na voljo na spletnem naslovu

`http://www.gmajna.net/svojat/jaka/code/matchmaker/`

V primeru dokumentov XML (npr. shem) so zavoljo berljivosti izpuščene določitve imenskih prostorov in komentarji.

A.1 Opis številskih zmožnosti: shema XML

```
<xsd:schema>

  <complexType name="capability">
    <sequence>
      <element name="Name" type="string"
               minOccurs="1" maxOccurs="1"/>
      <element name="Domain" type="string"
               minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

  <complexType name="rcapabilities">
    <all>
      <element name="Capability" type="capability"
               minOccurs="0" maxOccurs="unbounded"/>
      <element name="Rank" type="string"
               minOccurs="0" maxOccurs="1"/>
```

```
    </all>
  </complexType>

  <element name="RCapabilities" type="rcapabilities"/>

</xsd:schema>
```

A.2 Oglas storitve: shema XML

```
<schema>

  <complexType name="advertisement">
    <sequence>
      <element name="Interface" type="string"
               minOccurs="1" maxOccurs="1"/>
      <!-- element Address je del WS-Addressing standarda -->
      <element ref="Address"
               minOccurs="1" maxOccurs="1"/>
      <!-- element Policy je del WS-Policy standarda -->
      <element ref="Policy"
               minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

  <element name="Advertisement" type="advertisement"/>

</schema>
```

A.3 Rezervacija: shema XML

```
<schema>

  <complexType name="reservation">
    <sequence>
      <element name="Id" type="string"
               minOccurs="1" maxOccurs="1"/>
      <!-- element Address je del WS-Addressing standarda -->
      <element ref="Address"
               minOccurs="1" maxOccurs="1"/>
      <!-- element Policy je del WS-Policy standarda -->
```

```
      <element ref="Policy"
               minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

</schema>
```

A.4 Rezervacija: opis vrat v WSDL

```
<definitions name="Reservation">

  <types>
    <schema>
      <element name="InRes" type="reservation"/>
      <element name="OutRes" type="contract"/>
      <element name="Fault" type="string"/>
    </schema>
  </types>

  <message name="reserveInputMessage">
    <part name="parameters" element="InRes"/>
  </message>

  <message name="reserveOutputMessage">
    <part name="parameters" element="OutRes"/>
  </message>

  <message name="reserveFaultMessage">
    <part name="exception" element="Fault"/>
  </message>

  <portType name="ReservationPortType">
    <operation name="reserve">
      <input message="reserveInputMessage"/>
      <output message="reserveOutputMessage"/>
      <fault name="error" message="reserveFaultMessage"/>
    </operation>
  </portType>

</definitions>
```

A.5 Pogodba: shema XML

```
<schema>

  <complexType name="contract">
    <sequence>
      <element name="Id" type="string"
               minOccurs="1" maxOccurs="1"/>
      <!-- tip je del WS-Addressing standarda -->
      <element name="EndpointReference" type="EndpointReferenceType"
               minOccurs="1" maxOccurs="1"/>
      <element name="ContractExpiration" type="dateTime"
               minOccurs="1" maxOccurs="1"/>
      <!-- element Policy je del WS-Policy standarda -->
      <element ref="Policy"
               minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>

</schema>
```

A.6 Pogodba: opis vmesnika v WSDL

```
<definitions name="Contract">

  <types>
    <schema>
      <element name="InCnt" type="contract"/>
      <element name="OutCnt" type="contract"/>
      <element name="InCancel" type="contract"/>
      <element name="OutCancel" type="boolean"/>
      <element name="Fault" type="string"/>
    </schema>
  </types>

  <message name="contractInputMessage">
    <part name="parameters" element="InCnt"/>
  </message>

  <message name="contractOutputMessage">
    <part name="parameters" element="OutCnt"/>
  </message>
```

```
  <message name="cancelInputMessage">
    <part name="parameters" element="InCancel"/>
  </message>

  <message name="cancelOutputMessage">
    <part name="parameters" element="Void"/>
  </message>

  <message name="contractFaultMessage">
    <part name="exception" element="Fault"/>
  </message>

  <message name="cancelFaultMessage">
    <part name="exception" element="Fault"/>
  </message>

  <portType name="ContractPortType">
    <operation name="contract">
      <input message="contractInputMessage"/>
      <output message="contractOutputMessage"/>
      <fault name="error" message="contractFaultMessage"/>
    </operation>
    <operation name="cancel">
      <input message="cancelInputMessage"/>
      <output message="cancelOutputMessage"/>
      <fault name="error" message="cancelFaultMessage"/>
    </operation>
  </portType>

</definitions>
```

Literatura

A. Abdul-Rahman in S. Hailes (1997) A distributed trust model. V: *NSPW '97: Proceedings of the 1997 workshop on New security paradigms*, str. 48–60, New York, NY, USA. ACM Press.

S. Androutsellis-Theotokis in D. Spinellis (2004) A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371.

A. Avizienis, J. Laprie, B. Randell in C. Landwehr (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

S. Bajaj in sod. (2004a) Web Services policy attachment (WS-PolicyAttachment) framework. Na naslovu `ftp://www6.software.ibm.com/software/developer/library/ws-polat.pdf`.

S. Bajaj in sod. (2004b) Web Services policy (WS-Policy) framework. Na naslovu `ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf`.

M. Balazinska, H. Balakrishnan in D. Karger (2002) INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. V: *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, str. 195–210, London, UK. Springer-Verlag.

A. Barak in O. La'adan (1998) The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, 13(4–5):361–372.

A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett in D. Tweten (1999) Portable batch system: External reference specification. Tehnično poročilo, MRJ Technology Solutions.

K. Birman (2006) The untrustworthy Web Services revolution. *IEEE Computer*, 39(2): 98–100.

K. Birman, T. Joseph, T. Raeuchle in A. E. Abbadi (1985) Implementing fault-tolerant distributed objects. *IEEE Transactions on Software Engineering*, 11(6):502–508.

C. Blum in A. Roli (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308.

G. Booch (1994) *Object-oriented analysis and design with applications (2nd ed.)*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA.

D. Box in sod. (2004) Web Services Addressing (WS-Adressing). Na naslovu `http://www.w3.org/Submission/ws-addressing/`.

W. R. Brown (2001) *Designing Solutions with COM+ Technologies*. Microsoft Press, Redmond, WA.

L. F. Cabrera in sod. (2005) Web Services coordination (WS-Coordination). Na naslovu `ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf`.

E. Cerami (2002) *Web Services Essentials*. O'Reilly & Associates, Inc., Sebastopol, CA.

D. Chappell in T. Jewell (2002) *Java Web Services*. O'Reilly & Associates, Inc., Sebastopol, CA.

CORBA-NS (2004) CORBA naming service specification v1.3. Na naslovu `http://www.omg.org/cgi-bin/apps/doc?formal/04-10-03.pdf`.

CORBA-TS (2000) CORBA trading object service specification v1.0. Na naslovu `http://www.omg.org/cgi-bin/apps/doc?formal/00-06-27.pdf`.

G. Coulouris, J. Dollimore in T. Kindberg (2001) *Distributed Systems - Concepts and Design*. Addison Wesley, Reading, Massachusetts, 3rd edition.

E. Curry Message-oriented middleware. V: Q. H. Mahmoud, editor, *Middleware for Communications*. John Wiley & Sons (2004).

E. W. Dijkstra On the role of scientific thought. V: *Selected Writings on Computing: A Personal Perspective*, str. 60–66. Springer-Verlag, London, UK (1982).

N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon in K. Moody (2004) Using trust and risk in role-based access control policies. V: *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, str. 156–162, New York, NY, USA. ACM Press.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach in T. Berners-Lee (1999) RFC 2616: Hypertext transfer protocol – HTTP/1.1. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc2616.txt`.

I. Foster (2006) Globus toolkit version 4: Software for service-oriented systems. V: *Proceedings of the IFIP International Conference on Network and Parallel Computing*, Lecture Notes in Computer Science, str. 2–13, London, UK. Springer-Verlag.

I. Foster, C. Kesselman in S. Tuecke (2001) The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222.

J. R. Gardner in Z. L. Rendon (2002) *XSLT and XPath: A Guide to XML Transformations*. Prentice Hall, Upper Saddle River, NJ.

M. Hapner, R. Sharma, R. Burridge, J. Fialli in K. Haase (2002) *Java Message Service API tutorial and reference: messaging for the J2EE platform*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

A. Harth, S. Decker, Y. He, H. Tangmunarunkit in C. Kesselman (2004) A semantic matchmaker service on the grid. V: *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, str. 326–327, New York, NY, USA. ACM Press.

J. Hodges in R. Morgan (2002) RFC 3377: Lightweight directory access protocol (v3): Technical specification. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc3377.txt`.

Jini (2003) Jini(tm) technology core specification platform v2.0 - lookup service. Na naslovu `http://java.sun.com/products/jini/`.

P. Karwaczyński in J. Močnik (2007) Self-optimization of a dht-based discovery service. V: *ICCGI 07: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, stran 7, Washington, DC. IEEE Computer Society.

P. Karwaczyński, D. Konieczny, J. Močnik in M. Novak (2007) Dual proximity neighbour selection method for peer-to-peer-based discovery service. V: *Applied computing 2007: Proceedings of the 2007 ACM Symposium on Applied Computing*, stran 9, New York, NY. ACM Press.

R. Khare in R. N. Taylor (2004) Extending the representational state transfer (REST) architectural style for decentralized systems. V: *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, str. 23–28.

M. Klein in B. König-Ries (2004) Coupled signature and specification matching for automatic service binding. V: *ECOWS*, str. 183–197.

J. Klensin (2001) RFC 2821: Simple mail transfer protocol. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc2821.txt`.

F. A. Longstaff in E. S. Schwartz (2001) Valuing american options by simulation: A simple least squares approach. *Review of Financial Studies*, 14(1).

P. Mockapetris (1987) RFC 1035: Domain names – implementation and specification. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc1035.txt`.

J. Močnik in P. Karwaczyński (2006) An architecture for service discovery based on capability matching. V: *ARES 2006: Proceedings of the First International Conference on Availability, Reliability, and Security*, str. 824–831, Washington, DC. IEEE Computer Society.

J. Močnik, M. Novak, G. Pipan in P. Karwaczynski (2006) A discovery service for very large, dynamic grids. V: *Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, Washington, DC. IEEE Computer Society.

C. Morin, P. Gallard, R. Lottiaux in G. Vallee (2004) Towards an efficient single system image cluster operating system. *Future Generation Computer Systems*, 20(4):505–521.

T. J. Mowbray in R. Zahavi (1995) *The Essential CORBA: Systems Integration Using Distributed Objects*. John Wiley & Sons, Inc., New York, NY.

J. Osrael, L. Froihofer in K. M. Goeschka (2006) What service replication middleware can learn from object replication middleware. V: *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing in conjuction with the 7th Int. ACM/IFIP/USENIX Middleware Conference*, str. 18–23, New York, NY. ACM Press.

E. Pitt in K. McNiff (2001) *Java.RMI: The Remote Method Invocation Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

PlanetLab Consortium (2006) Planetlab consortium. Na naslovu `http://www.planetlab.org/`.

R. Raman (2000) *Matchmaking Frameworks for Distributed Resource Management*. Doktorska disertacija, University of Wisconsin, Madison. Na naslovu `http://www.cs.wisc.edu/condor/doc/rajesh.dissert.pdf`.

R. Raman, M. Livny in M. Solomon (1998) Matchmaking: Distributed resource management for high throughput computing. V: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing,*, Washington, DC. IEEE Computer Society.

I. Rammer (2002) *Advanced .Net Remoting*. Apress, Berkely, CA.

M. A. Rappa (2004) The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42.

E. T. Ray (2001) *Learning XML: (Guide to) Creating Self-Describing Data*. O'Reilly & Associates, Inc., Sebastopol, CA.

RPC (1988) RFC 1057: RPC: Remote procedure call protocol specification version 2. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc1057.txt`.

J. Salas, F. Perez-Sorrosal, M. Patino-Martinez in R. Jimenez-Peris (2006) WS-Replication: a framework for highly available Web Services. V: *Proceedings of the 15th Int. Conference on the World Wide Web*, str. 357–366, New York, NY. ACM Press.

M. Shaw (2002) Self-healing: softening precision to avoid brittleness. V: *Proceedings of the 1st workshop on self-healing systems*, str. 111–114, New York, NY. ACM Press.

D. L. Snyder in M. I. Miller (1991) *Random Point Processes in Time and Space*. Springer-Verlag, London, UK.

G. Suryanarayana, M. H. Diallo, J. R. Erenkrantz in R. N. Taylor (2006) Architectural support for trust models in decentralized applications. V: *ICSE '06: Proceeding of the 28th international conference on Software engineering*, str. 52–61, New York, NY, USA. ACM Press.

A. S. Tanenbaum in M. V. Steen (2002) *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ.

D. Thain, T. Tannenbaum in M. Livny Condor and the grid. V: F. Berman, G. Fox in T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., New York, NY (2002).

UDDI (2004) UDDI technical white paper. Na naslovu `http://uddi.org/pubs/uddi-tech-wp.pdf`.

W. Vogels (2003) Web Services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66.

WS-Architecture (2004) Web Services architecture. W3C WG Note. Na naslovu `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.

WS-Glossary (2004) Web Services glossary. W3C WG Note. Na naslovu `http://www.w3.org/TR/ws-gloss/`.

WS-Security (2006) Web Services security. Na naslovu `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`.

XDR (1987) RFC 1014: XDR: External data representation standard. Network Working Group Request for Comments. Na naslovu `http://www.ietf.org/rfc/rfc1014.txt`.

X. Ye in Y. Shen (2005) A middleware for replicated Web Services. V: *Proceedings of the 3rd Int. Conference on Web Services*, str. 631–638, Washington, DC. IEEE CS.

B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph in J. Kubiatowicz (2004) Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53.

S. Zhou (1992) LSF: Load sharing in large-scale heterogenous distributed systems. V: *Proceedings of International Workshop on Cluster Computing.*

Stvarno kazalo