RAZVOJ UČINKOVITIH PORAZDELJENIH OBJEKTNIH SISTEMOV

Matjaž B. Jurič, Tomaž Domajnko, Marjan Heričko, Ivan Rozman
Univerza v Mariboru, Inštitut za informatiko, Fakulteta za elektrotehniko, računalništvo in informatiko, e-pošta: matjaz.juric@uni-mb.si

Izvleček

V prispevku so prikazani koncepti gradnje učinkovitih velikih informacijskih sistemov na osnovi porazdeljenih objektnih modelov. Opisani pristopi zagotavljajo izdelavo sistemov, ki so uporabni, zanesljivi, razširljivi in nudijo optimalne zmogljivosti. Slednjim je posvečena posebna pozornost. Prikazani so rezultati meritve zmogljivosti in razširljivosti dveh uveljavljenih porazdeljenih objektnih modelov, CORBA in RMI. Članek prispeva k razumevanju pasti in zank izgradnje porazdeljenih objektnih sistemov in vodi razvijalca k programskim praksam, ki zagotavljajo dobre rezultate v smislu zmogljivosti in razširljivosti razvitih sistemov.

Abstract

The paper systematically describes the concepts used for the development of efficient, large-scale information systems, based on distributed object models. The described concepts ensure that the developed systems will be useful, reliable, scalable, and will offer optimal performance. A special emphasis is given to the performance. Different testing scenarios give an overview about the real world performance of two important distributed object models, CORBA and RMI. Based on the comparison, recommendations for selecting the most appropriate model for a given problem domain are presented. Therefore the paper contributes to the understanding of distributed object systems and to the study of distributed object models performance.

■■■

1. UVOD

Sodobni informacijski sistemi so zaznamovani s komponentno zgradbo, ki temelji na objektnih konceptih, in z večslojno arhitekturo, ki omogoča enostavno distribucijo delov aplikacije v lokalnih in v globalnem omrežju. V tem smislu sta zelo pomembna vidika povezljivost in storitve. Povezljivost lahko implementiramo na nizkem nivoju z uporabo komunikacijskega protokola in vtičnic (socket) ali pa uporabimo katerega od porazdeljenih objektnih modelov. V drugem primeru smo odprli vrata do množice objektnih storitev, ki omogočajo razvijalcem hitro implementacijo poimenovanja, dogodkovnih, transakcijskih, varnostnih in drugih mehanizmov.

Gradnja velikih informacijskih sistemov lahko z uporabo porazdeljenih objektnih modelov pridobi hitrost, zanesljivost in kakovost razvoja. Glede na izkušnje pa se tehnologija porazdeljenih objektov pogosto uporablja napačno. Izgradnja robustnih, zanesljivih in razširljivih porazdeljenih sistemov, ki jih bomo lahko enostavno vzdrževali, zahteva velik inženirski vložek, ne glede na tehnologijo implementacije.

Porazdeljeni objektni modeli, kot so CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invocation) in COM+ (Component Object Model), nudijo številne prednosti, če jih uporabljamo pravilno. V tem prispevku so predstavljeni pristopi in spoznanja, kako zgraditi zanesljive porazdeljene sisteme z dobrimi zmogljivostmi.

2. ARHITEKTURA SODOBNIH SISTEMOV

Veliki porazdeljeni, razširljivi in ključni informacijski sistemi morajo zadostiti potrebam po varnosti in zanesljivosti. Posebej je potrebno zagotoviti naslednje:

- zanesljivost aplikacijskih objektov,
- funkcionalnost aplikacijskih objektov,
- zmogljivost aplikacijskih objektov,
- celovitost podatkovnih objektov.

Razširljiv je vsak sistem, katerega arhitektura dopušča z ustrezno porazdelitvijo med procesorske enote zadovoljiti naraščajoče potrebe po storitvah tega sistema. Zelo pomembno je, da arhitektura sistema v ta namen ostaja nespremenjena.

Osnovni cilj pri definiranju sistemske arhitekture je zmanjšanje kompleksnosti. To lahko dosežemo tako, da sledimo naslednjim smernicam:

1. Začnemo z osnovno arhitekturo.
2. Vnašamo preverjene vzorce načrtovanja, s čimer zagotavljamo prilagodljivost.
3. Nastajajočo arhitekturo prototipiramo, da zmanjšamo tveganje.
4. Tehnologije vpeljujemo inkrementalno.
5. Povezujemo zgodaj in pod nadzorom.
6. Načrt gradimo po ravneh.

Pri velikih sistemih je pomembno, da začnemo z osnovno arhitekturo, ki bo vodila razvoj in minimizirala rizike [9, 10]. Na osnovni arhitekturi bodo slonele vse komponente. Če je le mogoče, se odločimo za stabilno in preverjeno arhitekturo in infrastrukturo. V mnogo primerih se je pametneje z nakupom pri uglednem dobavitelju izogniti lastnemu razvoju. Le ta lahko hitro preseže tretjino razvojnega časa celotnega informacijskega sistema, mnogokrat pa je povezan tudi z velikimi dolgoročnimi stroški vzdrževanja.

Osnovna arhitektura nudi enotno osnovo za implementacijo aplikacijskih objektov. Združuje storitve, ki bi jih v nasprotnem primeru kodirali vsi razvijalci. Takšne storitve so [4, 2]: posrednik zahtev objektov, storitve poimenovanja, dogodkovne storitve, transakcijske storitve, varnostne storitve, storitve trajnega stanja, beleženje dostopa, zmožnost upravljanja in razporeditev obremenitve.

3. PORAZDELJENI OBJEKTNI MODELI

Porazdeljeni objektni modeli implementirajo koncept lokacijsko neodvisne transparentne povezljivosti med programskimi jeziki, operacijskimi sistemi in platformami s striktno uporabo konceptov objektnih vzorcev. Na osnovi jasno definiranih vmesnikov omogočajo komunikacijo med objekti, ki temelji na proženju oddaljenih metod [13]. Pri tem poskrbi vmesni sloj sistemske programske opreme za vse podrobnosti komunikacije, od lociranja ustreznega objekta, do prenašanja podatkov, proženja ustrezne metode in vračanja rezultata. Zaradi narave porazdeljenih objektov je potrebno na novo definirati pojma odjemalca in strežnika, ki ju je v novem okolju potrebno razumeti kot vlogi dveh objektov, ki med seboj sodelujeta in komunicirata. Objekt, ki nudi storitve, se imenuje strežniški objekt. Tisti objekt, ki storitve uporablja (torej proži ustrezne oddaljene metode) pa odjemalni objekt. Vloge odjemalca in strežnika v porazdeljenem objektnem okolju niso fiksne in se lahko zamenjujejo. Le za določeno proženje metode lahko enolično definiramo strežniški in odjemalni objekt. Narava porazdeljenih objektov tudi radikalno spreminja način razmišljanja o tem, kje so naši podatki. Podatki, vgrajeni v objekte, lahko pridejo tja, kjer so najbolj potrebni.

Zaradi striktnega ločevanja vmesnika od implementacije in zaradi upoštevanja načel ograjevanja je relativno enostavno zagotoviti neodvisnost od programskega jezika in povezljivost med različnimi jeziki. Ker odjemalni objekti komunicirajo s strežniškimi izključno preko vmesnika, so na ta način ograjeni od implementacije. Odjemalni objekti ne vedo in jih tudi ne zanima, kako so strežniški objekti implementirani. Zato lahko razvijalci pri ohranitvi enakega vmesnika poljubno zamenjujejo implementacijo objektov. Pri tem lahko uporabijo programski jezik in operacijski sistem, ki najbolje zadoščata potrebam, ali pa implementacijo izvedejo kako drugače – npr. v strojni opremi. Ker odjemalci vidijo celoten sistem izključno skozi množico vmesnikov, so pred njimi skrite vse podrobnosti, kot npr. implementacija, operacijski sistemi in strojne osnove. Zaradi tega govorimo o iluziji ene sistemske slike skozi milijone omreženih računalnikov.

Porazdeljeni objektni modeli dopolnjujejo mehanizme proženja oddaljenih metod s storitvami in skupnimi sredstvi. Proženje oddaljenih metod je ključen koncept, ki zagotavlja delovanje sistema. Objektne storitve, skupna in domenska sredstva pa so vnaprej pripravljene funkcionalnosti, ki dajejo sistemu vsebino. Razlikujemo jih po stopnji abstrakcije in splošnosti njihove namembnosti. Objektne storitve so nižje nivojske funkcionalnosti, ki so uporabne v večini aplikacijskih domen. Primeri zajemajo storitve poimenovanja, licenciranja, povpraševanja, varnostne, transakcijske in relacijske storitve, itd. Skupna sredstva pa so višje nivojske funkcionalnosti, kot na primer uporabniški vmesnik, sestavljeni dokumenti, tiskanje, e-pošta, itd. Domenska sredstva so vnaprej pripravljena ogrodja poslovnih objektov za vertikalne domene, na primer telekomunikacije, zdravstvo, elektronsko poslovanje, arhitekturo sistemov, itd. Aplikacijski objekti predstavljajo uporabniško razvite enote funkcionalnosti – uporabniške aplikacije.

Za zagotavljanje komunikacije uporablja posrednik zahtev objektov storitve operacijskega sistema. Za izmenjavo sporočil med porazdeljenimi objekti se uporablja komunikacijski protokol, ki je lahko standardiziran, kot npr. GIOP/IIOP (General Inter-ORB Protocol / Internet Inter-ORB Protocol) ali specifičen, kot npr. JRMP (Java Remote Method Protocol). Proženje oddaljenih metod porazdeljenih objektov je zelo kompleksno in vpeljuje veliko število slojev [13].

4. DOSEGANJE UČINKOVITE PORAZDELJENOSTI OBJEKTOV

4.1. Identiteta objekta

Referenca objekta je začasna in za razvijalca neprozorna ročica, ki identificira primerek objekta. Reference lahko preprečijo zadostno razširljivost aplikacij, če jih ne uporabljamo pravilno. Referenca objekta je sama

zase lahko precej velik objekt (slika 1). Če mora odjemalec uporabljati veliko takih referenc, potem njegova velikost naraste in lahko odpove zaradi premalo pomnilnika na odjemalnem računalniku. Primerjanje referenc objektov je počasno – potreben je oddaljeni klic. Odvisno od modela nitenja lahko takšna preverjanja povzročijo mrtve zanke, zato jih je potrebno natančno načrtovati [7].

Včasih aplikacije uporabljajo objektne reference, spremenjene v nize, kot ključe na objekte in jih hranijo v podatkovni bazi (slika 2). Takšne podatkovne baze ponavadi potrebujejo precej prostora. Posebej pereč je problem pri relacijskih shemah, ki uporabljajo redundanco v obliki tujih ključev za prikaz relacij med zapisi tabel. Alternativa je uporaba tako imenovanih lahkih objektnih identifikatorjev. Le-ti temeljijo na celih številih in jih lahko tudi učinkovito primerjamo. Za dosego unikatnosti identifikatorjev uporabljamo odgovarjajočo shemo.

Vmesnike do objektov moramo zgraditi tako, da bodo podpirali dostop do objektov prek lahkih objektnih identifikatorjev ali referenc objektov. To poveča število operacij, ki jih je potrebno implementirati, zato moramo biti preudarni. Kot rešitev se ponuja storitev, ki je sposobna pretvarjati med referencami objektov in lahkimi identifikatorji.

4.2. Porazdelitev

Osnovna arhitektura ne sme diktirati porazdelitve. O porazdelitvi naj se odloča le na osnovi analize problemske domene. Porazdeljevanje majhnih objektov brez ozira na prednosti vodi do problemov. Zelo pomembno se je zavedati, da porazdelitev vnaša ogromno kompleksnost v sistem, kar kažejo tudi meritve zmogljivosti v nadaljevanju prispevka. Zato jo je potrebno uporabljati premišljeno. Porazdelitev uporabljamo, da zadostimo sistemskim zahtevam. Takrat uporabljamo večje objekte na nivoju objektov poslovnega procesa in storitev.

Vsak objekt ne more biti porazdeljen objekt. Takšne rešitve so sicer elegantne, vendar neuporabne z vidika razširljivosti in zmogljivosti. Oblikovanje, brisanje in uporaba porazdeljenih objektov so neprimerno počasnejši kot uporaba lokalnih objektov. Rešitve najdemo v naslednjih točkah:

1. Uporabljamo večje objekte v smislu komponentne arhitekture – na nivoju poslovnih procesov in storitev. Takšni večji objekti nudijo dostop do velikega števila entitetnih objektov.
2. Do določenih objektov dostopamo prek vmesnika dinamičnega proženja.
3. Uporabljamo pretvornike za porazdelitev, ki temeljijo na istoimenskem vzorcu. Le-ti omogočajo lokalnim objektom pridobiti distribucijske lastnosti po potrebi. Lep primer pretvornika, ki to podpira, je prenosljiv objektni pretvornik (POA – Portable Object Adapter) modela CORBA.

4.3. Atributi

Uporabi atributov v vmesnikih IDL (Interface Definition Language) se izogibamo iz dveh razlogov:

- niso v skladu z objektnimi koncepti glede ograjevanja vmesnika in
- zahtevajo oddaljen klic, kar vpliva na zmogljivosti.

Različni podatkovni tipi se v različnih posrednikih zahtev objektov različno dobro upravljajo [8]. Običajno pri osnovnih podatkovnih tipih ni velikih razlik. Razlike pa nastopajo pri nizih, poljih, strukturah in množicah. Skozi prototipno rešitev je primerno poiskati predstavitev podatkov, ki zadovoljivo deluje.

4.4. Vmesniki

Načrtovanje vmesnikov aplikacijskega sistema vključuje tako vmesnike abstrakcij aplikacije kot tudi vmesnike za podporo operacijam na veliko. Predvsem je potrebna podpora za paketno in skupinsko izvajanje in za povpraševanja in iteratorje. Paketno izvajanje

Tip	Identifikator repozitorija	Podatki o protokolu	Podatki o naslovu	Ključ objekta	Ime adapterja	Ime objekta

Slika 1: Zgradba objektne reference

```
IOR:000000000000002e49444c3a766973696765 6e69632e636f6d2f676174656b656570657
22f416c6961734d616e616765723a312e300000000000000200000000000006d0001000000
00000d3136342e382e3235332e323100003a980000005100504d4300000000000000002e49444
c3a7669736967656e69632e636f6d2f676174656b65657065722f416c6961734d616e616765
723a312e3000000000000001149494f5020476174652d4b6565706572000000000000000001000
0002400000000000000010000000100001400000000000000000000000000000010109000000000
```

Slika 2: V niz spremenjena referenca objekta CORBA

vračajo množice podatkovnih objektov pri vsakem proženju. Tukaj je pomembna ekonomika razširjevanja. Skupinsko izvajanje poteka na množici objektov, pri tem pa je potreben le en klic odjemalca. Povpraševanja in iteratorji morajo upoštevati hkratnost dostopa do virov in različne načine sklicevanja objektov. Pri implementaciji iteratorjev je potrebno upoštevati njihovo obnašanje. Če le-ti zahtevajo zaklepanje objektov, lahko pride do zakasnitve, preden dobijo pravico za zaklepanje.

4.5. Komunikacijski model

Zelo pomembno je, da je komunikacijski model posrednika zahtev objektov, ki ga uporabljamo, razširljiv. Nekateri posredniki zahtev objektov uporabljajo za vsakega odjemalca svojo povezavo. Takšni posredniki niso primerni za podporo velikemu številu odjemalcev na strežnik. Če obstaja n odjemalcev in m strežnikov, potem je število povezav (npr. preko protokola TCP) enako $n \times m$. Če naj vaša aplikacija podpira 400 odjemalnih objektov in 30 strežniških objektov, pomeni to 12.000 povezav. Le najmočnejši računalniki prenesejo tolikšno število povezav. Najslabše pa je, da se večina resursov v določenem trenutku ne uporablja. Boljši način je uporaba posrednikov zahtev objektov, ki uporabljajo multipleksiranje. V takšnem primeru bi potrebovali zgolj $m + n$ povezav, torej 430. Tolikšno število povezav pa brez težav podpira vsak zmogljiv računalnik.

Uporabna je tudi asinhrona komunikacija med odjemalci in strežniki v obliki dogodkovnega kanala tipa objavi-naroči. Dogodkovne storitve morajo biti povezane s posrednikom zahtev objektov, tako da je uporabnikom vseeno, katero metodo komunikacije uporabljajo. Podprta naj bi bila pristopa potisni in povleci.

Pomembna zmožnost delovanja je optimizacija komunikacije med odjemalcem in strežnikom, ko sta le-ta locirana na istem računalniku. Takšna optimizacija ponavadi poteka z uporabo deljenega pomnilnika in ne prek omrežnega sklada. Glede na tip aplikacije je taka optimizacija lahko zelo pomembna in vpliva na hitrost izvajanja.

4.6. Nitni modeli

Posredniki zahtev objektov, ki uporabljajo samo eno nit, niso dovolj robustni. Pri taki zgradbi se pojavi problem pri programiranju strežnikov, kjer morajo razvijalci zagotoviti, da se bodo zahteve servisirale dovolj hitro in se ne bodo po nepotrebnem zaustavljale.

Večnitna arhitektura posrednikov zahtev objektov poveča prepustnost in odzivnost sistema. Zahteve se strežejo neodvisno, vsaka v svoji niti. Uporabljena je zmožnost sočasnega izvajanja ukazov sodobnih eno in večprocesorskih sistemov. Za interaktivne aplikacije izboljša odzivni čas sistema. Različne operacije uporabljajo različne niti, zato ne pride do blokiranja zahtev odjemalcev. Večnitna zgradba prav tako omogoča neodvisno izvajanje več strežnikov. Pri tem lahko uporabljamo konvencionalne metode, kot npr. sinhrono dvosmerno proženje zahtev. Večnitnost zahteva tudi manj sistemskih virov, saj je oblikovanje niti ponavadi manj zahtevno od oblikovanja novega procesa.

Različne arhitekture nitenja imajo pomemben vpliv na odzivnost in zmogljivost sistema. V praksi se najboljši rezultati dosegajo z arhitekturami na osnovi nitnega fonda [7]. Te omogočajo hkratno servisiranje velikega števila odjemalcev, hkrati pa omogočajo nadzor nad porabo sistemskih sredstev in preprečujejo prekomerno obremenitev strežniških računalnikov. Posebej bi omenili arhitekturo delovnega nitnega fonda, vodilno sledilnega nitnega fonda in nitnega ogrodja, ki so prikazane na sliki 3.

Slika 3: Delovni, vodilno sledilni nitni fond in nitno ogrodje

4.7. Trajno stanje objektov

Velike aplikacije zahtevajo učinkovit mehanizem zagotavljanja trajnega stanja objektov. Če se ne moremo izogniti preslikavi objektov v relacijsko shemo, bomo prisiljeni opraviti nekaj ročnega kodiranja. Sistem bo potrebno podpreti s predpomnilniško shemo, saj bomo brez te le težko dobili zadovoljive zmogljivosti. Po naših izkušnjah nudijo objektne baze boljše mehanizme predpomnenja in zaradi tega ročno posredovanje ni potrebno [12].

Zelo pomembno je tudi uskladiti politiko aktiviranja objektov med posrednikom zahtev objektov in sistemom za zagotavljanje trajnega stanja. Posebej pri uporabi prenosljivega objektnega pretvornika imajo razvijalci številne možnosti.

4.8. Življenjski cikel objektov

Strežniki, ki oblikujejo veliko število odjemalcev, potrebujejo robustno upravljanje s pomnilnikom. Posebno pozornost je potrebno posvetiti življenjskim ciklom objektov. Skrb za pravočasno brisanje množice objektov, ki jih je začasno potreboval določen odjemalec, lahko uspešno prevzame porazdeljen zbiralec odpadkov, podoben tistemu, ki je implementiran v Javi [1].

Nekateri posredniki zahtev objektov zahtevajo, da so vsi aktivni objekti v pomnilniku. To pomeni zapreko za razširljivost, če pričakujemo potrebo po velikem številu objektov. Za aktiviranje in deaktiviranje lahko uporabimo nalagalnike in prestreznike in tako izkoristimo lastno upravljanje s pomnilnikom.

Premikanje objektov ali pošiljanje objektov po vrednosti je lastnost, ki jo je potrebno uporabljati previdno. Zavedati se je potrebno, da je stanje objekta določeno z množico atributov in povezav med objekti. Fizično spreminjanje lokacije objekta potegne za sabo precej dela, ki zelo vpliva na zmogljivosti [11].

4.9. Zanesljivost in razporeditev obremenitve

V porazdeljenem sistemu se v primerjavi s centraliziranim pojavlja veliko več dejavnikov, ki lahko vplivajo na delovanje sistema in ga onemogočijo. Zato je pri načrtovanju potrebno posebno pozornost posvetiti razreševanju takšnih situacij. Posredniki zahtev objektov nudijo mehanizme za nadzor izjemnih stanj, ki razvijalcem olajšajo delo.

Z uporabo »pametnih« lokatorjev lahko v sistemu dosežemo redundanco strežniških objektov in ob izpadu poskrbimo za avtomatsko preusmeritev zahtev do redundantnih objektov. Zelo enostavno lahko realiziramo tudi ponoven zagon strežniških objektov po izpadu sistema. Pri tem predpostavljamo ustrezno podporo sistema za zagotavljanje trajnega stanja objektov.

Zaradi lokacijske neodvisnosti objektov je razmeroma enostavna tudi uvedba razporeditve obremenitve. Predlagamo rešitev, ki je tesno povezana s storitvami poimenovanja, s katerimi odjemalni objekti dobijo dostop in povezavo do strežniških objektov. Storitve poimenovanja lahko poskrbijo, da se odjemalcem dodeljujejo reference do redundantnih strežniških virov, ne da bi se le-ti tega sploh zavedali. Razporeditev obremenitve lahko aplikaciji dodamo kasneje in razvijalcem ni potrebno spreminjati kode.

Uporabljena v povezavi lahko razporeditev obremenitve prek storitev poimenovanja in uporaba »pametnih« lokatorjev vodita do porazdeljenih sistemov, ki so vsaj tako zanesljivi kot centralizirani. Porazdeljeni sistemi, ki pa takšnih pristopov ne uporabljajo, so po definiciji manj zanesljivi od centraliziranih.

4.10. Razpoložljivost

Razpoložljivost sistema je ponavadi določena z dvema parametroma:

- s povprečnim časom med napakama (MTBF – Mean Time between Failure), ki predstavlja pričakovani čas nastopa odpovedi in
- s povprečnim časom obnove normalnega delovanja (MTTR – Mean Time To Repair).

Izračun poteka po naslednji formuli:

$$\text{Razpoložljivost} = \text{MTBF} / (\text{MTBF} + \text{MTTR})$$

Na MTBF vpliva kakovost programske opreme, ki je funkcija kakovosti razvojnega procesa in razvojne skupine. Kakovost upravljanja sistema izraža MTBR. Popolnih sistemov ni, zato je razpoložljivost vedno manjša od 100%. Pri porazdeljenih sistemih skupno razpoložljivost izražamo kot povprečno vrednost vseh strežnikov v sistemu.

5. ZMOGLJIVOSTI PORAZDELJENIH OBJEKTNIH MODELOV

V tem poglavju prikazujemo rezultate meritev zmogljivosti modelov CORBA/Java in Java RMI. Cilj je bil ugotoviti vpliv modelov pri proženju metod in razširljivost pri povečanju števila hkratnih odjemalcev. Zato smo razvili več scenarijev komunikacije med odjemalnimi in strežniškimi objekti.

5.1. Metoda

Simulirali smo interakcije med odjemalnimi in strežniškimi objekti, tipične za trislojne aplikacije. Definirali smo vmesnike z množico metod. Verzija vmesnika, napisanega v CORBA IDL, je prikazana na izpisu 1. Za osnovo smo vzeli aplikacijo bankomata. Vmesnike smo implementirali v Javi. Posebno pozornost smo posvetili ekvivalenci implementacij za RMI in CORBA. Vse metode so vračale tipizirane rezultate. Za simulacijo prenosa velikih nizov smo uporabili metodo

`Account.getType()`, ki je vrnila nize velikosti 1, 1000, 2000, 3000, 4000, 5000 in 10000 znakov. Uporabljen je bil dvosmerni statični mehanizem proženja. Na strani odjemalca smo oblikovali javanski programček, ki se je povezal na strežniške objekte in prožil metode. Čas smo merili z metodo `System.currentTimeMillis()`, ki vrne čas v milisekundah. Za pridobitev verodostojnejših in zanesljivejših podatkov smo metode prožili tisoč krat. Rezultati, ki jih objavljamo, so povprečne vrednosti petnajstih ponovitev.

```
interface Atm {
...   boolean Working();
long long getAtmNo();
...
};
interface Account {
...   float getBalance();
string getType();
wstring getType();
double getLimit();
...
};
interface Card {
...   long getNumber();
...
};
```

Izpis 1: Vmesnik za meritev zmogljivosti CORBA IDL

Poskuse smo izvedli v treh scenarijih:
(a) strežniški in odjemalni objekt sta se izvajala na istem računalniku,
(b) strežniški in odjemalni objekt sta se izvajala na ločenih računalnikih,
(c) strežniški objekt se je izvajal na enem, odjemalni pa simultano na 2, 3, 4, 5, 6, 7 in 8 računalnikih.

Rezultati iz točk (a) in (b) omogočajo sklepati o vplivu omrežja in primerjava med (b) in (c) o degradaciji zmogljivosti pri množični hkratni strežbi zahtev. Medtem ko je prvi rezultat pomemben za distribucijo, prikazuje drugi tipično večuporabniško interakcijo. Odjemalni javanski programček je prožil metode brez premora. To ni skladno s tipično uporabniško interakcijo in pomeni veliko večje število tipičnih odjemalcev.

Javino izvorno kodo smo prevedli z JavaSoft Java Development Kit 1.1.4 (JDK), ki je referenčna platforma za razvoj [6]. Za eksperimente z modelom CORBA smo uporabili Visigenic Visibroker for Java 3.0. Visibroker je eden najpopularnejših posrednikov zahtev objektov in je vgrajen v Netscape Navigator. Zato je posrednik, ki ga najpogosteje najdemo na računalnikih. V nekaterih meritvah zmogljivosti z jezikom C++ [8] je bil hitrejši od Iona Orbix. Kot profiler smo uporabili JProbe Profiler 1.1 podjetja KL Group. Vsi računalniki so uporabljali Microsoft Windows NT 4.0 Workstation. Strežnik je bil Pentium II 233 MHz s 64 MB RAM in odjemalni računalniki so bili Pentium 200 MHz tudi z 64 MB RAM. V današnjih aplikacijah prek interneta je pasovna širina ključnega pomena. Da smo simulirali dejanske razmere, smo se odločili za povezavo računalnikov v omrežje 10 MBps Ethernet. Omrežje je bilo brez drugega prometa.

5.2. Lokalni in porazdeljeni objekti

Lokalno proženje metode v programskem jeziku Java je na omenjeni strojni osnovi pri uporabi opisanih testnih vzorcev trajalo okrog 400 ns (nanosekund). Proženje enake metode preko modela RMI zahteva približno 1.5 ms (milisekunde), preko modela CORBA pa okrog 2 ms. Sloj porazdeljenega procesiranja ima velik vpliv na zmogljivosti. Omogoča komunikacijo med procesi in med računalniki. Rezultati nazorno kažejo na manjšo kompleksnost modela RMI, ki podpira le programski jezik Java. CORBA podpira različne programske jezike, operacijske sisteme in platforme [3].

5.3. Distribucija v omrežje

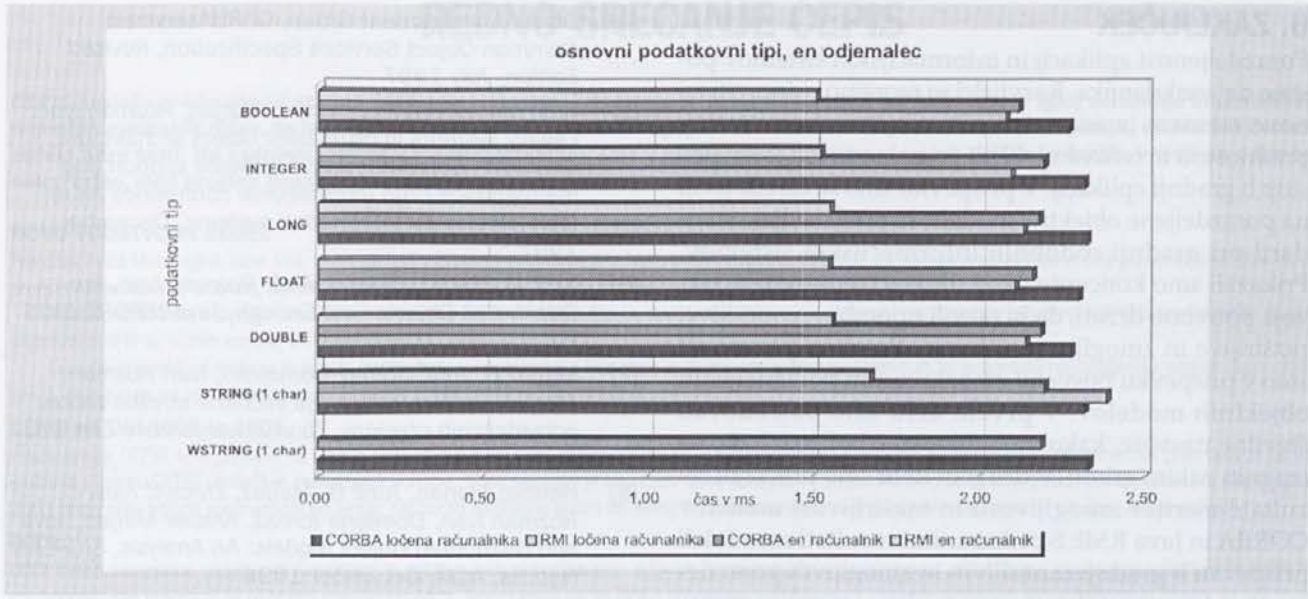
Slika 4 prikazuje razlike v časih proženja metod z različnimi podatkovnimi tipi za modela CORBA in RMI v scenariju z enim odjemalcem in strežnikom na enem in dveh ločenih računalnikih, povezanih v omrežje. Zmogljivosti modelov CORBA in RMI sta pri porazdelitvi v omrežje precej izenačeni.

Slika 5 prikazuje zmožnost upravljanja velikih nizov podatkov, ki jih pogosto srečamo v sodobnih informacijskih sistemih. Očitna je prednost modela CORBA pri navadnih nizih in manjša degradacija zmogljivosti pri distribuciji v omrežje.

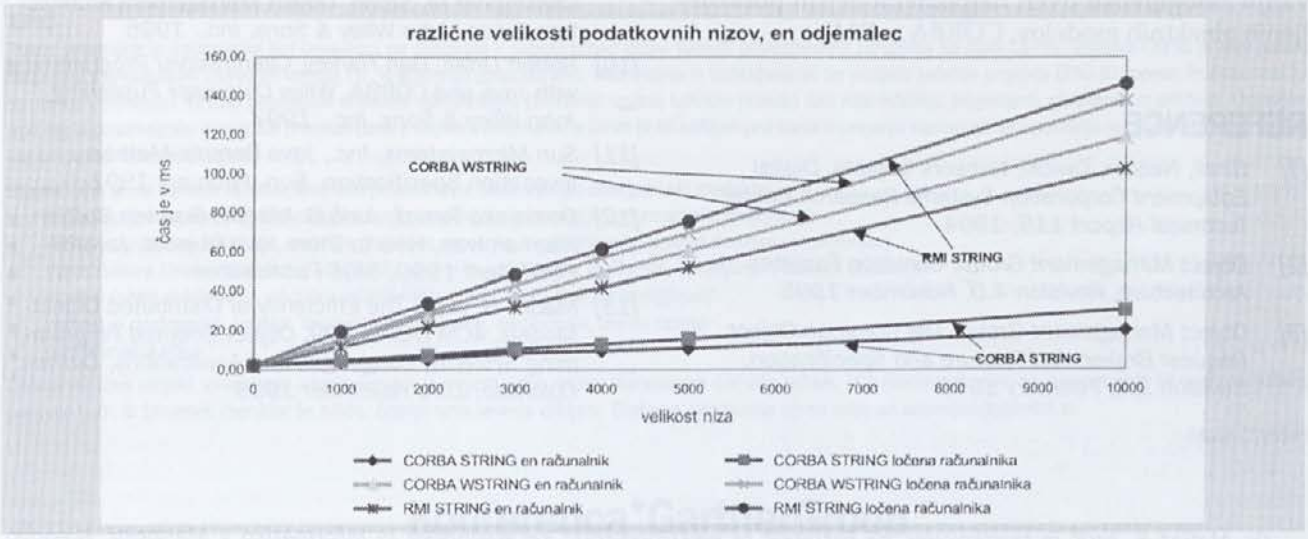
5.4. Razširljivost in množična hkratna strežba odjemalcev

Slika 6 prikazuje degradacijo zmogljivosti pri množični hkratni strežbi odjemalcev. Do osem hkratnih odjemalcev je prožilo metode na strežniških objektih brez prekinitve. Zato rezultati odražajo stanje za desetkrat večje število realnih odjemalcev [8]. Model CORBA izkazuje veliko boljše rezultate.

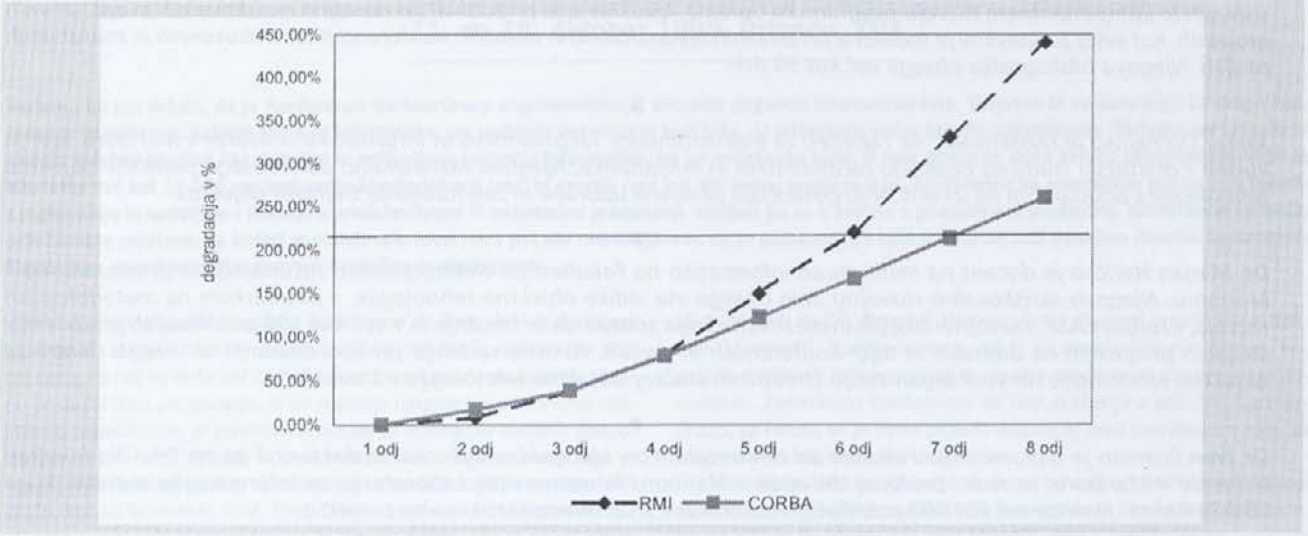
Rezultati meritev zmogljivosti sovpadajo z namenom modelov RMI in CORBA. Za enostavnejše aplikacije, kjer ne pričakujemo velikega števila hkratnih odjemalcev in prenosa velikih količin podatkov prek proženj oddaljenih metod, izkazuje RMI zmogljivosti, ki so celo za odtenek boljše od modela CORBA. Pri zahtevnih operacijah in predvsem pri meritvi degradacije zmogljivosti pri hkratni strežbi velikega števila odjemalcev, pa je model CORBA nedvomno ustreznejša rešitev.



Slika 4: Vpliv porazdeljenih objektnih modelov pri proženju metod z različnimi podatkovnimi tipi in vpliv porazdelitve v omrežje



Slika 5: Upravljanje velikih nizov podatkov



Slika 6: Degradacija zmogljivosti pri množični hkratni strežbi

6. ZAKLJUČEK

Porazdeljenost aplikacij in informacijskih sistemov postaja del vsakdanjika. Razvijalci se moramo pripraviti na nove razmere in se soočiti s številnimi novimi izzivi, prednostmi in težavami, ki jih prinaša porazdeljeni pristop h gradnji aplikacij. V prispevku smo se osredotočili na porazdeljene objektne modele, ki predstavljajo standard pri gradnji sodobnih informacijskih sistemov. Prikazali smo koncepte, ki se jih je v trenutni fazi razvoja potrebno držati, da bi razvili uporabne, zanesljive, razširljive in zmogljive aplikacije. Posebno pozornost smo v prispevku posvetili zmogljivostim porazdeljenih objektnih modelov. V prvem delu smo posredovali številne nasvete, kako zmogljivosti izboljšati, kako se izogniti ozkim grlom. V drugem delu smo prikazali rezultate meritev zmogljivosti in razširljivost modelov CORBA in Java RMI. S prikazanimi rezultati smo želeli prispevati h gradnji zanesljivih in zmogljivih porazdeljenih informacijskih sistemov in doprinesti k razumevanju zmogljivosti dveh najpomembnejših porazdeljenih objektnih modelov, CORBA in RMI.

REFERENCE

[1] *Birell, Nelson, Owicki,* Network Objects, *Digital Equipment Corporation Systems Research Center Technical Report 115, 1994*

[2] *Object Management Group,* Common Facilities Architecture, *Revision 4.0, November 1995*

[3] *Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998*

[4] *Object Management Group,* CORBAservices: Common Object Services Specification, *Revised Edition, July 1997*

[5] *Jurič Matjaž Branko, Heričko Marjan, Rozman Ivan:* Legacy systems in distributed object architecture, *CASSAM - Computer aided software support and maintenance / 5th International conference on re-technologies for information systems, December 1997*

[6] *Jurič B. Matjaž, Živkovič Aleš, Rozman Ivan,* Are Distributed Objects Fast Enough, *JavaREPORT, SIGS Publications, May 1998*

[7] *Matjaž B. Jurič, Tomaž Domajnko, Ivan Rozman, Marjan Heričko,* Evaluacija večnitne strežbe zahtev porazdeljenih objektov, *Zbornik konference ERK'98, Portorož 1998*

[8] *Heričko Marjan, Jurič B. Matjaž, Živkovič Aleš, Rozman Ivan, Domajno Tomaž, Krisper Marjan,* Java and Distributed Object Models: An Analysis, *SIGPLAN Notices, ACM, December 1998*

[9] *Object Management Group, Richard Mark Soley, Christopher M. Stone,* Object Management Architecture Guide, *John Wiley & Sons, Inc., 1995*

[10] *Robert Orfali, Dan Harkey,* Client/Server Programming with Java and CORBA, *Wiley Computer Publishing, John Wiley & Sons, Inc., 1997*

[11] *Sun Microsystems, Inc.,* Java Remote Method Invocation Specification, *Sun, February 1997*

[12] *Domajnko Tomaž, Jurič B. Matjaž, Brumen Boštjan, Rozman Ivan,* How to Store Java Objects, *JavaREPORT, April 1999, SIGS Publications*

[13] *Matjaž B. Jurič,* The Efficiency of Distributed Object Models, *ACM OOPSLA'99, Object Oriented Programming Systems, Languages and Applications, Denver, Colorado, ZDA, november 1999*

◆

Dr. Matjaž B. Jurič je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru. Njegovo raziskovalno razvojno delo pokriva vse vidike objektne tehnologije s posebnim poudarkom na porazdeljenih objektnih sistemih in komponentnem razvoju programske opreme. Izkušnje si je pridobival pri razvojno raziskovalnih in aplikativnih projektih. Kot avtor prispevkov je sodeloval pri številnih mednarodnih in domačih konferencah ter v strokovnih in znanstvenih revijah. Njegova bibliografija obsega več kot 90 del.

◆

Tomaž Domajnko je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, kjer je vpisan v doktorski študij na področju računalništva in informatike. Njegovo raziskovalno delo obsega področje objektne tehnologije s poudarkom na izkoriščanju potenciala ponovne uporabe in zagotavljanju trajnosti objektov.

◆

Dr. Marjan Heričko je docent na Inštitutu za informatiko na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru. Njegovo raziskovalno-razvojno delo obsega vse vidike objektne tehnologije, s poudarkom na metodologijah razvoja, orodjih CASE, razvojnih okoljih in metrikah. Svoja spoznanja in izkušnje je v več kot 100 publikacijah predstavil v številnih prispevkih na domačih in tujih konferencah in revijah. Aktivno sodeluje pri koordiniranju aktivnosti Centra za objektno tehnologijo ter vodi organizacijo strokovnih srečanj Objektna tehnologija v Sloveniji.

◆

Dr. Ivan Rozman je diplomiral na Fakulteti za elektrotehniko v Ljubljani, magistriral in doktoriral pa na Tehniški fakulteti Univerze v Mariboru. Je redni profesor Univerze v Mariboru in ustanovitelj Laboratorija za informacijske sisteme, ki ga vodi še danes. Je avtor več kot 400 publikacij, vodi številne znanstveno raziskovalne projekte.

◆