Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Igor Čavrak

Prilagodljivi objektno usmjereni raspodijeljeni računalni sustavi

Magistarski rad

Zagreb, 2001.

Magistarski rad je izrađen na Zavodu za automatiku i procesno računarstvo (grupa RASIP), Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu

Mentor: Prof. dr. sc. Mario Žagar

Radnja ima: 174 stranice

Rad br.:

POVJERENSTVO ZA OCJENU U SASTAVU:

1. Prof. dr. sc. Leo Budin
2. Prof. dr. sc. Mario Žagar
3. Prof. dr. sc. Vlatko Čerić

POVJERENSTVO ZA OBRANU U SASTAVU:

1. Prof. dr. sc. Leo Budin
2. Prof. dr. sc. Mario Žagar
3. Prof. dr. sc. Vlatko Čerić

Datum obrane: 5. lipnja 2001.

Zahvala

Kao što ni jednog čovjeka ne možemo promatrati kao samostalnu i izoliranu jedinku, odvajajući ga od okruženja u kojem živi i radi, tako ni njegova djela ne možemo pripisati samo i isključivo njemu. Stoga i ovaj rad nema samo jednog autora.

Nazvati prof. dr. sc. Maria Žagara samo mentorom bilo bi suviše suhoparno. Savjeti, pomoć i vrijeme koje mi je posvetio tijekom pisanja ove radnje, ali i prije, daleko nadmašuju definiciju mentorstva. Zato, puno hvala !

Damir Kovačić – Čmrki i Danko Basch smogli su snage pročitati dijelove teksta i ukazati na pogreške, i njihova je zasluga što je ovaj rad (donekle) čitljiv. Damiru također hvala na vremenu uloženom u rasprave i savjete glede CORBA-e.

Goran Mužak i Denis Trupec morali su otrpjeti moje česte promjene raspoloženja i verbalne eskapade, uvjeravajući me da nije sve tako crno kao što mi se činilo. Iako u vrijeme izrade ove radnje daleko, Armin Stranjak svojim je primjerom znatno olakšao njen nastanak.

Hvala i svim ostalim znanim i neznanim koautorima čiji je doprinos, svjesno ili nesvjesno, ugrađen u ovaj rad.

Naposljetku, hvala mojim roditeljima koji su imali strpljenja i volje ulagati u mene i moje školovanje. Uostalom, bez njih ove radnje sigurno ne bi ni bilo.

Autor

Sadržaj

1. Uvod

Za razliku od prvih generacija elektroničkih računala čiji su predstavnici bili "usamljeni otoci", izolirani od ostatka svijeta u računalnim centrima, strogo namjenski korišteni isključivo za određene poslove, današnja računala tvore potpuno drugačiji svijet. Nagli razvoj tehnologije omogućio je korištenje jeftinih računala u gotovo svim domenama ljudskog života, na gotovo svim mjestima, kako na poslu, tako i u domaćinstvima. Komunikacijska povezanost računala dodala je novu dimenziju njihovu korištenju, kao posljedicu imajući rađanje informacijskog "globalnog sela".

Praćenje razvoja korištenja svojstva umreženosti računala odaje nekoliko faza. U prvoj fazi računala su još uvijek bila tretirana kao odijeljene funkcionalne cjeline, a mogućnost njihove komunikacije korištena je isključivo za osnovni prijenos podataka. Karakteristično za ovu fazu korištenja su aplikacije za udaljeni rad (telnet), prijenos datoteka (ftp) i elektroničku poštu. U drugoj fazi pojavljuju se arhitektura korisnik-poslužitelj, u kojoj se razdvajaju (fizički) potrošači i proizvođači informacije. Sada je moguće obradu podataka vršiti na udaljenom, a rezultate prikazivati na lokalnom računalu. Od informacijskih servisa u ovoj fazi pojavljuju se gopher i WWW, a većinu korisnik – poslužitelj sustava tvore s poslužitelj strane baze podataka, a s korisničke strane jednostavne aplikacije za unos i prikaz podataka. Korisnicima ovih sustava fizička raspodijeljenost dijelova sustava nije primjetna. Treću fazu korištenja karakterizira potpuna raspodijeljenost komponenti sustava. Osim komponenti sustava specifičnih za pojedinu aplikaciju, uvode se komponente široke uporabe, koje implementiraju funkcionalnost često korištenu u širokom spektru različitih sustava, tvoreći tako skup zajedničkih usluga. Time se gubi čvrsta podjela komponenti po uporabi u pojedinoj raspodijeljenoj aplikaciji, tvoreći mrežu usluga istovremeno korištenih od strane više aplikacija.

Svojstveno računalnim mrežama evolucijski je razvoj, uklapanjem novih komponenata (kako sklopovskih, tako i programskih) među postojeće građevne komponente sustava. Kao posljedica evolucijskog rasta javlja se problem raznorodnosti sklopovske opreme (računala, mrežni uređaji) i programskih sustava (programski proizvodi različitih proizvođača). Problem raznorodnosti na razini raspodijeljenih aplikacija očituje se u implementaciji aplikacijskih komunikacijskih protokola, koji ne samo da moraju biti usuglašeni s obzirom na vrstu aplikacije, već moraju voditi računa o mogućoj različitosti u tumačenju podataka kao posljedice raznorodnosti korištenih računalnih platformi i različitih programskih jezika korištenih u implementaciji pojedinih komponenti sustava.

Uvođenjem tehnologije međusloja različitosti uvjetovane korištenjem raznorodnih sklopovskih i programskih komponenti sustava izoliraju se u sloju "ispod" sloja implementacije aplikacije. Međusloj preuzima na sebe zadaće vezane uz usklađivanje komunikacije između raspodijeljenih komponenti sustava, bez obzira na postojeće različitosti kako na sklopovskoj tako i na programskoj razini.

Osim rješavanja problema raznorodnosti, zadaća međusloja je i prikrivanje složenosti izgradnje raspodijeljenih sustava. Programska sučelja prema međusloju imaju zadaću stvoriti takvo okruženje programeru sustava kao da se razvija "obična", neraspodijeljena aplikacija. S obzirom na korištenu paradigmu, razlikuju se stariji mehanizmi međusloja temeljeni na proceduralnoj paradigmi (udaljeni pozivi procedura) i noviji, temeljeni na objektno usmjerenoj paradigmi. Široko rasprostranjen mehanizam srednjeg sloja predviđen za korištenje u aplikacijama temeljenim na objektno orijentiranoj paradigmi je sustav CORBA (eng. *Common Object Request Broker Architecture*).

No mehanizmi srednjeg sloja, pa tako i CORBA, iako su od velike pomoći tijekom razvoja, ne rješavaju osnovne probleme svojstvene raspodijeljenim sustavima, kako u fazi izrade tako i u fazi korištenja sustava. Dok je u fazi izrade sustava naglasak na pronalaženju i ispravljanju pogrešaka (tj. statičkim promjenama sustava), u fazi korištenja za raspodijeljenu aplikaciju bitna je prilagodba promjenama u njenoj radnoj okolini (dinamičkim promjenama sustava).

Mnoštvo postojećih alata za pronalaženje pogrešaka, praćenje rada i analizu razvijanih programa predviđeni su za uporabu na samostalnim programima, te nisu podesni za primjenu na raspodijeljenim sustavima u cjelini. Stoga se kao nužnost javlja razvoj sustava koji bi omogućavao praćenje rada i analizu raspodijeljenog sustava u cjelini, a na osnovu čijih podataka bi se na pojedinim komponentama sustava koristili prije spomenuti alati.

Pod promjenama u okolini podrazumijevaju se promjene u raspoloživosti korištenih resursa, npr. opterećenja računala na kojima se komponente sustava izvode, promjenama u propusnosti računalne mreže, kao i promjene stanja dijela ili aplikacije u cjelini. Tri su osnovna pitanja na koje je nužno posjedovati odgovore u svrhu učinkovitog provođenja prilagodbe raspodijeljenog sustava: kako izvršiti prilagodbu, kada izvršiti prilagodbu, te čemu se prilagoditi. Odgovori na zadnja dva pitanja mogu se dati na osnovu praćenja rada sustava i njegove radne okoline.

Sustav za nadgledanje rada raspodijeljenog sustava mora zadovoljiti dva bitna uvjeta: mora davati vjernu sliku događaja u nadgledanom sustavu, te mora što manje utjecati na rad nadgledanog sustava.

Ovaj rad bavi se izradom sustava nadgledanja rada raspodijeljenih aplikacija temeljenih na sustavu CORBA, te njegovom primjenom u prilagođavanju aplikacija promjenama u radnoj okolini. Kao potvrda navedene koncepcije razvijena je nadgledna usluga sustava CORBA koja, uz minimalne promjene nadgledanih aplikacija, omogućuje praćenje i upravljanje njihovim radom.

2. Raspodijeljeni sustavi

Dva su tehnološka napretka bitno utjecala na razvoj računalne tehnologije, mijenjajući sliku računarstva od njegovih početaka do današnjih dana: razvoj snažnih i jeftinih mikroprocesora počevši od sredine 80-tih godina, te razvoj računalnih komunikacija korištenjem brzih računalnih mreža. Posljedica razvoja tehnologije omogućila je široku dostupnost računalnih sustava velike snage obrade u svim problemskim domenama, na svim razinama organizacije složenih sustava, te njihovu integraciju u svrhu optimalnog korištenja raspoloživih resursa i dijeljenja informacija.

Tannenbaum [35] daje općenitu definiciju raspodijeljenog sustava promatranog sa strane korisnika: "Raspodijeljeni sustav je skup neovisnih računalnih sustava koji se korisniku sustava prikazuju kao jedan računalni sustav". Drugu, često citiranu definiciju raspodijeljenih sustava dao je Leslie Lamport: "Raspodijeljen sustav je onaj sustav na kojemu ne mogu obaviti svoj posao jer je palo jedno računalo za koje nikada nisam čuo". No ove definicije ne spominju niti jedno svojstvo koje raspodijeljene sustave čini bitno različitim od centraliziranih, neraspodijeljenih sustava. Drugi način pokušaja definiranja raspodijeljenih sustava je opisom njihovih posebnosti u odnosu na neraspodijeljene sustave: latencijom, pristupom radnoj memoriji, djelomičnim zatajenjem i paralelizmom izvršavanja [12].

Raspodijeljeni sustavi, s obzirom na Flynnovu podjelu računalnih arhitektura, pripadaju grupi arhitektura MIMD. No točnija podjela [35] s obzirom na jačinu grupiranja između procesora i memorije uvodi unutar arhitekture MIMD podjelu na višeprocesorske sustave (eng. *multiprocessor systems*) (karakterizira snažno grupiranje uslijed korištenja dijeljene memorije) i višeračunalne sustave (eng. *multicomputer systems*) (karakterizira slabo grupiranje uslijed korištenja privatnih memorija svakog od procesora u sustavu). Također, unutar grupe multiračunalnih sustava vrši se podjela s obzirom na vrstu korištenog komunikacijskog podsustava (sabirničkog ili komutiranog). Korištenje sabirnica kao komunikacijskog kanala svojstveno je računalima povezanim računalnim mrežama (LAN, WAN), dok je korištenje komutiranih komunikacijskih kanala svojstveno računalima temeljenim na transputerima. U ovom radu razmatrat će se isključivo raspodijeljeni računalni sustavi temeljeni na neovisnim komunikacijski povezanim računalima.

U doba centraliziranih računalnih sustava Groschov zakon oslikavao je odnos cijena – performanse sustava kao snagu sustava proporcionalnu kvadratu njegove cijene. Stoga je, u to doba, bilo opravdano ulaganje što veće količine novaca u jedno računalo. No, u današnje vrijeme, odnos cijena – performanse drastično se promijenio: snaga pojedinog sustava teži drugom korijenu njegove cijene. Stoga je ekonomski opravdana nabava većeg broja jeftinijih računala, te njihovo komunikacijsko

povezivanje u jednu funkcionalnu cjelinu. Argument u prilog raspodijeljenih sustava je i probijanje gornje granice performansi centraliziranih sustava: povezivanjem proizvoljnog broja računala može se ostvariti računalni sustav vrhunskih performansi, nedostižnih centraliziranim sustavima. Dakako, iskoristivost takvog sustava uvjetovana je vrstom problema (mogućnost njegove raspodjele na više dijelom neovisnih postupaka rješavanja) i načinom implementacije programske podrške. U slučaju pogrešaka pouzdanost raspodijeljenih sustava veća je od pouzdanosti centraliziranih sustava. Zatajenje jedne od komponenti sustava neće (odnosno ne mora, ovisno o rješavanom problemu) za posljedicu imati zatajenje sustava u cjelini. Pravilnom izvedbom utjecaj djelomičnog zastoja može se svesti samo na smanjenje performansi sustava. Također, moguć je iterativan rast sustava dodavanjem novih komponenti, bez utjecaja na postojeće dijelove sustava.

Prilikom opisa nedostataka raspodijeljenih računalnih sustava u prvi plan izbija snažna ovisnost o performansama komunikacijskog kanala, a koja se očituje u brzini i dostupnosti dijelova sustava:

- brzina komunikacije između dvaju računala za otprilike dva reda veličine sporija je od komunikacije unutar sustava (komunikacije između procesa istog računalnog sustava)
- zagušenjem komunikacijskog kanala drastično padaju performanse sustava
- prekidom komunikacijskog kanala dijelovi sustava postaju nedostupni.

Iako raspodijeljeni računalni sustavi olakšavaju rad s dijeljenim resursima, koordinacija njihova korištenja može izazvati probleme, kako u njihovu ispravnom radu, tako i u dostupnosti. Pitanje dozvola korištenja pojedinih dijeljenih resursa i sigurnosti informacija unutar raspodijeljenog računalnog sustava također predstavljaju ozbiljan problem, rješavan korištenjem različitih sigurnosnih mehanizama temeljenih na kriptografiji (šifriranje, potpisivanje, itd.).

No trenutno najveći problem raspodijeljenih sustava čini nedostatak adekvatne programske podrške. Iako se ulažu veliki napori u svrhu pojednostavljenja oblikovanja i izrade, programska podrška raspodijeljenim sustavima kasni za razvojem sklopovlja uslijed postojanja bitnih razlika u oblikovanju i implementaciji programske podrške centraliziranih i raspodijeljenih sustava [12]. Te razlike ne očituju se na razini programskih jezika i alata korištenih za razvoj sustava, već na razini oblikovanja sustava i pojedinih implementacijskih rješenja. Stoga, rješavanje ovog problema zahtijeva dvije bitne stvari:

- visok stupanj znanja i iskustva ljudi uključenih u oblikovanje i implementaciju programske podrške namijenjene raspodijeljenim sustavima

- razvojne alate i radna okruženja koja će smanjiti ili prikriti složenost razvoja i korištenja programske podrške raspodijeljenih sustava.

Osnovne karakteristike raspodijeljenih sustava, koje se njihovom implementacijom žele postići, su transparentnost, prilagodljivost, pouzdanost, učinkovitost i skalabilnost.

Transparentnost raspodijeljenog sustava očituje se u lokacijskoj transparentnosti (eng. *location transparency*) pojedinih komponenti sustava, transparentnosti promjene lokacije resursa (eng. *migration transparency*), transparentnosti višestrukih kopija podataka (eng. *replication transparency*), transparentnosti istovremenog dijeljenja resursa (eng. *concurrency transparency*) i transparentnosti usporednog rada (eng. *parallelism transparency*).

Prilagodljivost raspodijeljenog sustava u većem dijelu ovisi o građi jezgre operacijskog sustava: monolitnoj jezgri (eng. *monolithic kernel*) karakterističnoj za inačice operacijskog sustava UNIX ili minimalnoj jezgri (eng. *microkernel*) karakterističnoj novijim generacijama operacijskih sustava (Amoeba, Mach, itd.).

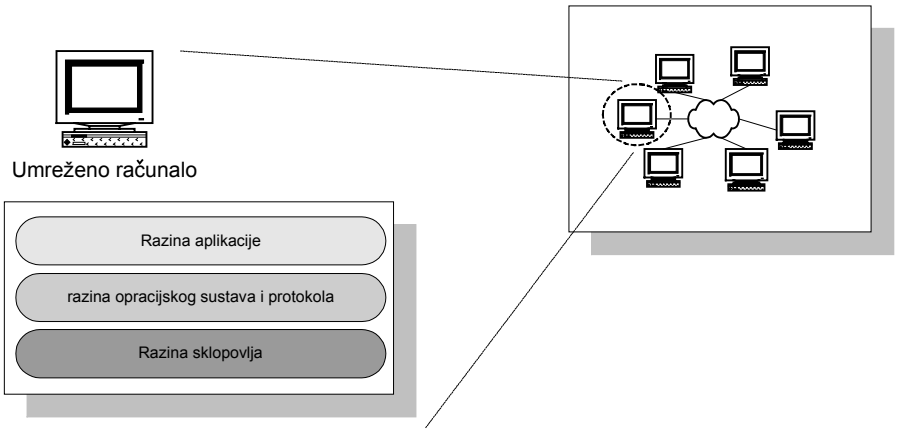
Pouzdanost raspodijeljenog sustava, uz pravilno oblikovanje i izvedbu, značajno je veća u odnosu na centralizirane sustave. Pravilno oblikovanje i izvedba stoga mora podrazumijevati korištenje tehnika otpornosti na pogreške u radu (zatajenje pojedinih komponenti sustava ne smije imati za posljedicu zatajenje čitavog sustava), oporavaka od pogreški (na razini komponente i na razini čitavog sustava), osiguranja redundantnosti komponenti i rješenja pitanja sigurnosti sustava (na razini komponenti i razini podataka).

Učinkovitost raspodijeljenog sustava uglavnom se odnosi na raspodjelu zadataka po komponentama sustava. Zrnatost rješavanih zadataka igra bitnu ulogu u raspodjeli poslova unutar sustava, ponajviše s ciljem smanjenja nužne mrežne komunikacije. Snažno grupirani zadaci male zrnatosti nastoje se izvršavati na istom računalu, dok se zadaci velike zrnatosti nastoje prebaciti na trenutno nekorištene udaljene komponente sustava.

Na skalabilnost sustava utječu centralizacija korištenja komponenti sustava (npr. jedan mrežni datotečni sustav na veliki broj korisnika), centralizirane strukture podataka i centralizirano izvršavanje algoritama. U pravilu se u raspodijeljenim sustavima izvode za tu svrhu oblikovani algoritmi čije su karakteristike:

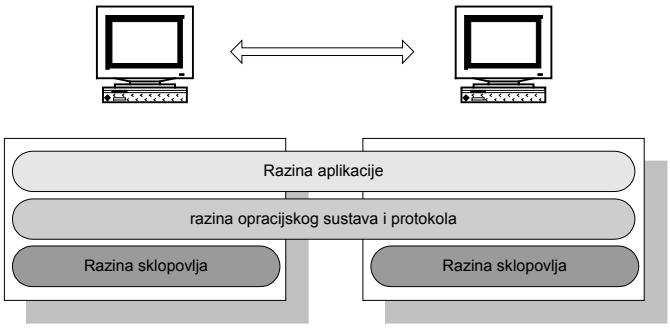
- ni jedna komponenta sustava nema cjelovitu informaciju o stanju čitavog sustava
- odluke se donose na osnovu lokalno dostupnih informacija

- zatajenje jedne od komponenti sustava ne uzrokuje zatajenje izvršavanja cijelog algoritma
- nema implicitne pretpostavke postojanja globalnog sata sustava.



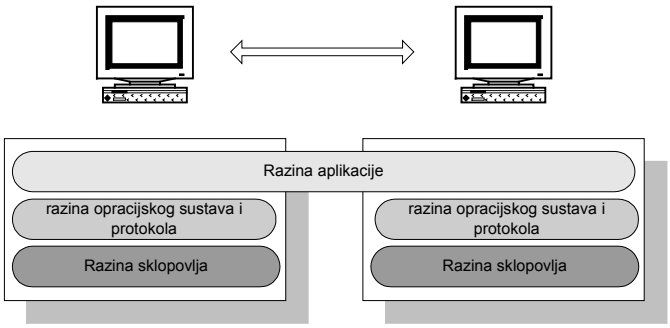
Slika 2.1. Slojevi unutar umreženog računala

Skrivanje udaljenosti resursa unutar raspodijeljenog sustava moguće je na nekoliko razina, ovisno o građi sustava. Za početak, potrebno je identificirati slojeve pojedine građevne komponente raspodijeljenog sustava (slika 2.1). Na najnižoj razini građevne komponente sustava (računala) nalazi se sklopovlje, u koje ubrajamo standardne komponente računalnog sustava (procesor, radnu memoriju, itd.) i komunikacijsku infrastrukturu (mrežne prilagodnike, računalnu mrežu). Građa najnižeg sloja ovisna je o proizvođaču računala, te može značajno varirati s obzirom na osnovne karakteristike računala (širina riječi, značajnost bitova, itd.). Mrežna infrastruktura je, osim u namjenskim izvedbama, standardizirana. Na razini iznad sklopovske nalazi se operacijski sustav računala i implementacije komunikacijskih protokola. Karakteristike operacijskog sustava ovise o njegovu proizvođaču i sklopovskoj platformi za koju su predviđeni. Implementacije komunikacijskih protokola moraju se strogo pridržavati standarda da bi komunikacija između računala različitih operacijskih sustava bila moguća. Na najvišoj razini sustava nalaze se (korisničke) aplikacije.



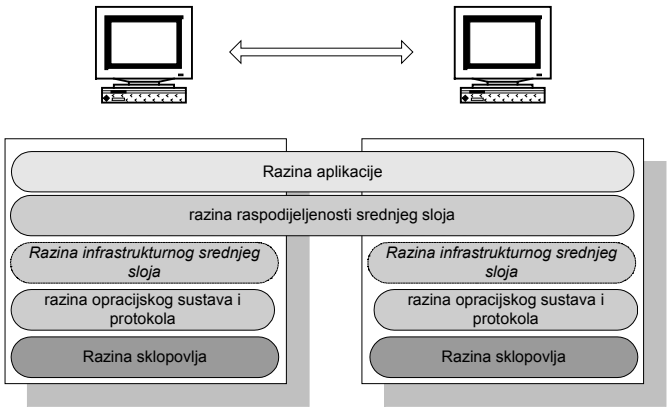
Slika 2.2. Raspodijeljenost na razini operacijskog sustava

Raspodijeljeni operacijski sustavi (slika 2.2) preuzimaju na sebe sav teret koje raspodijeljenost uvjetuje: međuprocesnu komunikaciju, dijeljenje resursa, sigurnost i pouzdanost. Jezgra raspodijeljenog sustava istodobno se izvršava na svim računalima, tvoreći prema višim razinama (razini aplikacije) jedinstveno radno okruženje. No raspodijeljeni operacijski sustavi nisu u raspodijeljenim sustavima zastupljeni u velikom broju. Njihova uporaba uglavnom je ograničena na eksperimentalne ili strogo namijenske sustave. Uvjet njihova korištenja je prisutnost istog operacijskog sustava na svim računalima koje tvore raspodijeljeni sustav. Primjeri raspodijeljenih sustava su Amoeba, Mach, Chorus, itd.



Slika 2.3. Raspodijeljenost na razini aplikacije

Transparentnost uporabe raspodijeljenih sustava u većini slučajeva postiže se na razini implementacije aplikacije (slika 2.3), a korištenjem standardnih operacijskih sustava s ugrađenom podrškom mrežnoj komunikaciji. U ovom slučaju sav teret rješavanja problema koje raspodijeljenost uvjetuje pada na proizvođača sustava (osobe zadužene za oblikovanje i implementaciju sustava). Problemi sežu od heterogenosti računalnih arhitektura na kojima se komponente aplikacije izvršavaju (dužina riječi, značajnost okteta, itd.), različitih operacijskih sustava (implementacija mrežne komunikacije, kodiranje znakova, itd.), implementacijskih jezika komponenti sustava, do implementacije aplikacijskih protokola putem kojih komponente komuniciraju.



Slika 2.4. Raspodijeljenost na razini srednjeg sloja

Na osnovu iskustava u razvoju raspodijeljenih sustava temeljenih na različitim arhitekturama i operacijskim sustavima korištenih računala, zajednički problemi su izdvojeni u dvije zasebne grupe:

- problemi različitosti upravljanja resursima korištenih računala
- problemi vezani uz raspodijeljenost komponenti sustava

Iz navedenih grupa formirane su dvije razine tzv. srednjeg sloja (slika 2.4):

- razina infrastrukturnog srednjeg sloja s ulogom stvaranja jedinstvenog sučelja prema resursima računala neovisno o arhitekturi i operacijskom sustavu

- razina raspodijeljenosti srednjeg sloja s ulogom stvaranja jedinstvenog radnog okruženja prema aplikacijskom sloju

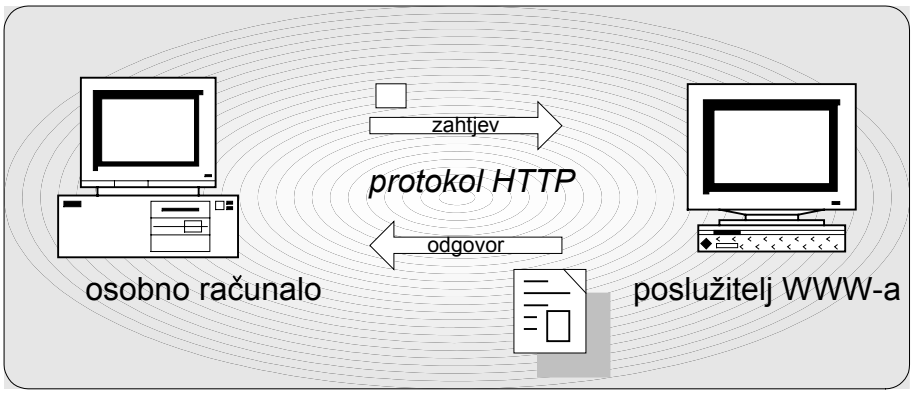
Primjeri izvedbe razine infrastrukturnog srednjeg sloja su DCE (Distributed Computing Environment), ACE (Advanced Communication Environment) [31] i Java. ACE formira jedinstveno programsko sučelje prema mrežnim resursima korištenog računala neovisno o arhitekturi i korištenom operacijskom sustavu. Time je znatno olakšana implementacija komunikacijskog dijela raspodijeljene aplikacije, te prenosivosti i ponovna iskoristivost programskog koda. Java čini ne samo novi objektno usmjereni programski jezik, već i jedinstveno okruženje za rad sa resursima korištenog računala. Postojanje ovog sloja nije preduvjet za korištenje srednjeg sloja raspodijeljenosti.

Razina raspodijeljenosti srednjeg sloja nastoji omogućiti jedinstveno okruženje komponentama raspodijeljene aplikacije, te jednostavno korištenje u programskim

jezicima implementacije. Njezina uloga je stvaranje privida lokalnog radnog okruženja skrivanjem mrežne komunikacije i lokacijskoj transparentnosti korištenih komponenti. Primjeri implementacije razine raspodijeljenosti srednjeg sloja su sustavi CORBA i DCOM.

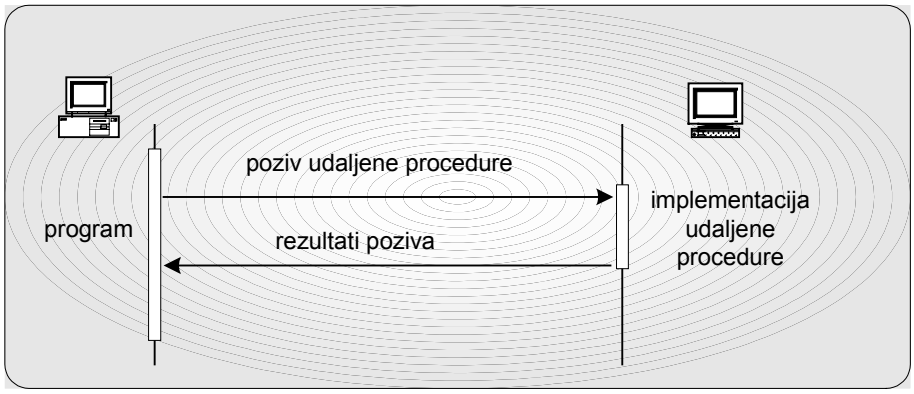
Obje razine srednjeg sloja većinom su implementirane u obliku programskih biblioteka uključenih u izvršni kod aplikacije.

2.1. Modeli komunikacije



Slika 2.5. Model prenošenja poruka u informacijskom servisu WWW

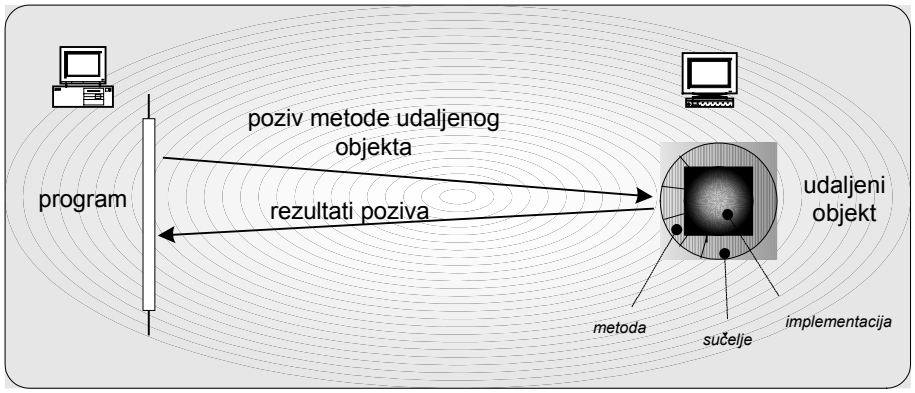
Osnovni model komunikacije u raspodijeljenim sustavima model je prenošenja poruka (eng. *message passing model*) i osnova je klijent – poslužitelj modela organizacije raspodijeljenih sustava. Proces klijent šalje procesu poslužitelju poruku koja sadrži zahtjev za obradom, a poslužitelj vraća poruku s rezultatom zahtijevane obrade. Oblik poruka definiran je protokolom korištene raspodijeljene aplikacije. Tipični predstavnici ovog modela komunikacije različiti su informacijski servisi korišteni na Internetu: WWW, FTP, elektronička pošta, itd. Implementacija komunikacije prenošenjem poruka temelji se na uporabi osnovnih mehanizama ulazno-izlazne komunikacije ugrađene u operacijske sustave – utičnicama (eng. *sockets*) i tokovima (eng. *streams*).



Slika 2.6. Poziv udaljene procedure

Sljedeći korak u razvoju mehanizama mrežne komunikacije uvođenje je poziva udaljenih procedura kao prvog mehanizma temeljenog na srednjem sloju. Poziv procedure čija se implementacija nalazi na udaljenom računalu ne razlikuje se od poziva lokalno implementirane metode procedure. Ovim mehanizmom riješeni su sljedeći problemi:

- mehanizam mrežne komunikacije integriran je u jezik implementacije
- ostvarena je transparentnost poziva lokalnih i udaljenih procedura
- problemi heterogenosti arhitektura i operacijskih sustava riješeni su na razini srednjeg sloja

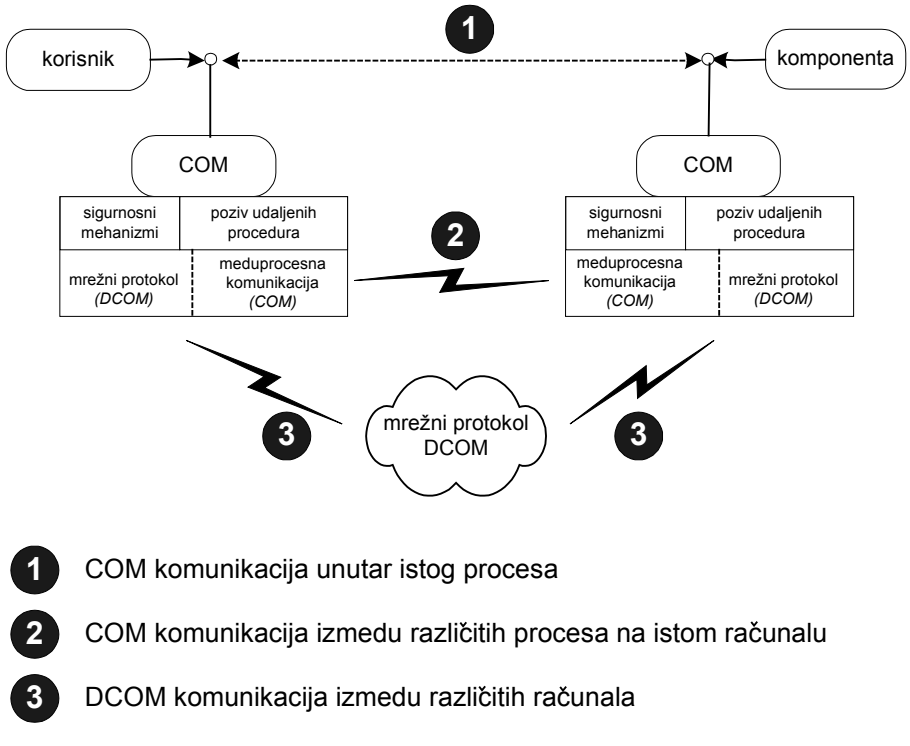


Slika 2.7. Poziv metode udaljenog objekta

Uvođenjem novih metoda razvoja programske podrške i programskih jezika, težište se s proceduralnog načina programiranja prebacilo na objektno usmjerene metode i jezike. Slijedeći ovaj razvoj razvijeni su novi, objektno usmjereni mehanizmi mrežne komunikacije. Umjesto poziva udaljenih procedura, ovi sustavi temelje se na pozivima metoda udaljenih objekata. Sučelja objekata definiraju se opisnim jezicima, čijim se prevođenjem stvaraju kosturi programskog koda kojima se naknadno dodaje funkcionalnost.

2.2. Arhitektura DCOM

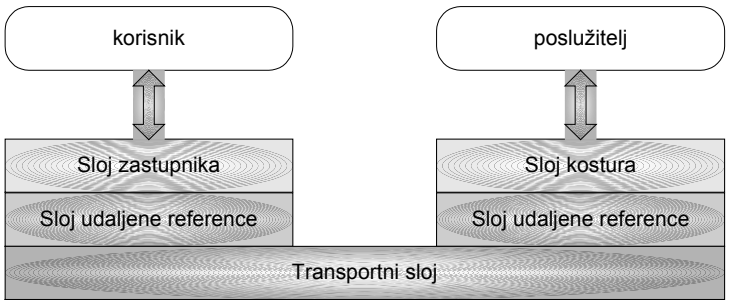
Arhitektura DCOM (eng. *Distributed Component Object Model*) nadogradnja je modela objektnih komponenti (eng. *Component Object Model* – COM) korištenih isključivo u operacijskim sustavima proizvođača Microsoft. DCOM je razvijen u svrhu potpore izgradnji raspodijeljenih aplikacija korištenjem COM programskih komponenti. Njegova uloga je preuzimanje mrežne komunikacije između korisnika komponenti i njihovih implementacija, bez obzira na njihov smještaj u lokalnoj ili globalnoj računalnoj mreži. Model objektnih komponenti oslanja se na standard binarnog zapisa izvršnog koda programske komponente i pripadajućih dodatnih informacija (podržanih sučelja, adresa metoda sučelja u memoriji, itd.). Arhitektura COM omogućuje uporabu komponenti na istom računalu, bez obzira da li se komponenta nalazi u istom ili različitom procesu s obzirom na njezinog korisnika. U slučaju da se korisnik i komponenta nalaze u istom procesu, metode se pozivaju izravno korištenjem virtualnih tablica metoda sučelja, što ovaj mehanizam komunikacije činilo iznimno brzim. Ukoliko se komponenta i korisnik nalaze u različitim procesima nužno je korištenje mehanizama međuprocesne komunikacije, čija je uporaba transparentna u odnosu na korisnika. Mehanizmi komunikacije u arhitekturi DCOM samo proširuju mehanizam međuprocesne komunikacije na procese koji se izvršavaju na različitim računalima (slika 2.8).



Slika 2.8. Arhitektura COM/DCOM

2.3. Arhitektura Java RMI

Cilj arhitekture Java RMI (eng. *Remote Method Invocation*) uvođenje je raspodijeljenog objektnog modela Java integriranog u programski jezik Java i model lokalnih objekata (objekata unutar istog virtualnog stroja Java). Arhitektura RMI temelli se na sljedećem važnom principu: definicija ponašanja i implementacija tog ponašanja su dvije različite stvari. RMI dozvoljava razdvojenost koda koji definira ponašanje i koda koji implementira to ponašanje, te njihovo izvođenje u zasebnim virtualnim strojevima. Ova koncepcija uklapa se u raspodijeljene sustave, u kojima su klijenti zainteresirani samo za definiciju usluge, a poslužitelji za njenu implementaciju i pružanje. Unutar arhitekture RMI, definicija usluge opisana je njenim sučeljem, a implementacija odgovarajućim razredom.



Slika 2.9. Slojevi arhitekture Java RMI

Implementacija arhitekture RMI definira tri sloja: sloj kostura korisnika i kostura poslužitelja, (eng. *Stub and* S*keleton Layer*), sloj udaljene reference (eng. *Remote Reference Layer*) i transportni sloj (eng. *Transport Layer*) (slika 2.9).

Sloj korisnika i poslužitelja presreće pozive metoda udaljenih objekata na strani korisnika i vrši njihovo preusmjeravanje usluzi RMI. Na strani poslužitelja njegova je uloga prihvaćanje poziva od strane usluge RMI i prosljeđivanje implementaciji udaljenog objekta.

Sloj udaljene reference interpretira i upravlja referencama udaljenih objekata i održava logičku vezu s poslužiteljima u kojima se udaljeni objekti nalaze.

Transportni sloj zadužen je za komunikaciju između procesa korisnika i procesa poslužitelja.

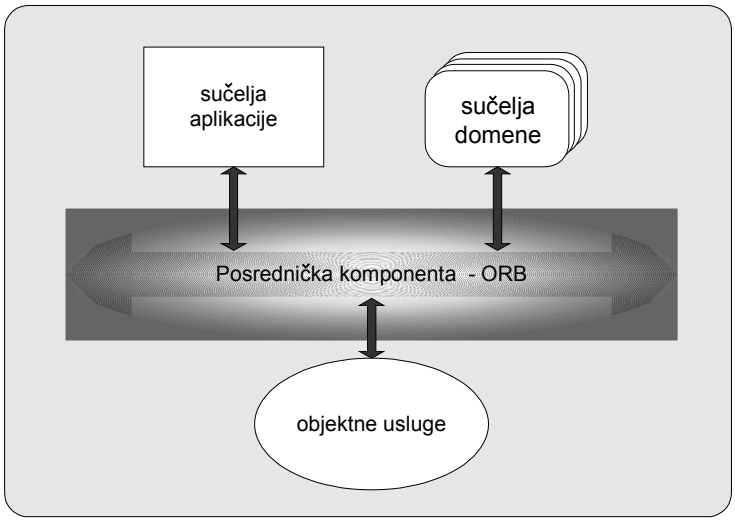
2.4. Arhitektura CORBA

Važna karakteristika računalnih mreža heterogenost je korištenih komponenti – različitost računalnih arhitektura i operacijskih sustava. Heterogenost je posljedica nekoliko bitnih čimbenika:

- postojanje više mogućih rješenja složenih tehničkih problema, a time i korištenih komponenti računalnog sustava na razini organizacijske cjeline (npr. poduzeća)

- učinkovitost korištenih sustava za njihov je odabir važnija od imena njihova proizvođača

- računalnim mrežama svojstvena je evolucija, a ne revolucija, te je od velike važnosti uklapanje novih s već postojećim komponentama sustava.



Slika 2.10. Kategorije OMA sučelja

Organizacija OMG (eng. *Object Management Group*) osnovana je 1989. godine u cilju izrade, usvajanja i promocije standarda za izradu i korištenje aplikacija u raspodijeljenim heterogenim okruženjima. Od tada OMG je prerastao u najveći programski konzorcij na svijetu s više od 700 članova. Arhitektura OMA [36] (eng. *Object Management Architecture*) definira modele raspodijeljenih objekata i njihove interakcije u platformski neovisnim okruženjima. OMA se sastoji od modela objekata (eng. *object model*) i modela referenci (eng. *reference model*). Model objekata propisuje kako objekti raspodijeljeni u heterogenom okruženju mogu biti opisani, dok model referenci opisuje međudjelovanje tih objekata. U modelu OMA objekt posjeduje nepromjenljiv identitet čije usluge mogu biti korištene samo putem čvrsto definiranih sučelja. Korisnici izdaju zahtjeve objektima u svrhu obavljanja usluga.

Implementacija i lokacija pojedinog objekta nisu dostupni klijentu. Slika 2.10 prikazuje komponente OMA modela referenci. Posrednička komponenta (eng. *Object Request Broker* – ORB) zadužena je za komunikaciju između klijenata i objekata. Tri kategorije sučelja objekata koriste posredničku komponentu: sučelja zajedničke usluge, sučelja zajedničkih mogućnosti i sučelja aplikacija.

Objektne usluge (eng. *Object Services*) čine sučelja neovisna o pojedinim domenama problema (horizontalno usmjerena), koriste se u širokom spektru raspodijeljenih objektno usmjerenih aplikacija na razini objekta ili grupe objekata. Kao primjeri najčešće korištenih mogu se navesti imenička usluga (eng. *Naming Service*) i posrednička usluga (eng. *Trading Service*). Ostale bitne usluge definirane od strane organizacije OMG su usluga životnog vijeka objekata, usluga sigurnosti, usluga transakcija, usluga događaja, itd.

Sučelja domene (eng. *Domain Interfaces*) sličnih su karakteristika kao i objektne usluge, no usmjerene su pružanju usluga "više razine", na razini aplikacija. Sučelja su specijalizirana (vertikalno usmjerena) za pojedine domene (telekomunikacije, zdravstvo, itd.). Kao primjer sučelja domena mogu se navesti MASIF (eng. *Mobile Agent System Interoperability Facility*) i PIS (eng. *Person Identification Service*).

Aplikacijska sučelja (eng. *application interfaces*) čine sučelja specifična pojedinoj raspodijeljenoj aplikaciji, te ne podliježu standardizaciji.

2.4.1. Posrednička komponenta

Objekt je osnovni entitet sustava CORBA određen svojim identitetom i sučeljem. Objekt kao takav je virtualni entitet, konkretna implementacija mora biti izvedena korištenjem nekog od ciljnih programskih jezika. Bitno je napomenuti da je životni vijek objekata sustava CORBA neovisan o životnom vijeku njihove konkretne implementacije (npr. objekta programskog jezika Java).

Implementaciju objekta (eng. *servant*) čini entitet programskog jezika kojim se realizira funkcionalnost pripadajućeg objekta ili grupe objekata istog sučelja. Programski entitet implementira sučelje i tijelo (izvršni kod) objekta. Najčešće korišteni jezici implementacije su objektno usmjereni jezici – C++, Java, Smalltalk, no mogu se koristiti i ostali jezici, npr. programski jezik C.

Referenca objekta (eng. *object reference*) jedinstveno opisuje objekt sustava CORBA. Koristi se u svrhu identifikacije, lociranja i adresiranja pojedinog objekta. Korisnici upotrebljavaju referencu na ciljni objekt prilikom poziva njegovih metoda ili kao argument poziva metoda drugih objekata, no ne mogu direktno utjecati na podatke

smještene u njoj. Reference na objekte stvaraju se u trenutku stvaranja objekta na strani poslužitelja.

Korisnik je programski entitet koji poziva metode objekta sustava CORBA. Korištenje objekta transparentno je s obzirom na njegovu trenutnu lokaciju, operacijski sustav računala na kojemu se objekt nalazi i programski jezik implementacije. Korisnik može biti samostalni program (npr. korisnička aplikacija s grafičkim sučeljem) ili drugi objekt sustava CORBA.

Zahtjev je poziv metode objekta. Zahtjevi se pokreću na strani korisnika objekta, korištenjem komunikacijskih mehanizama CORBA-e, na osnovu reference objekta pronalazi se ciljni objekt i na njegovoj implementaciji poziva željena metoda.

Poslužitelj je proces koji čini izvršnu okolinu implementacijama objekata sustava CORBA.

Posrednička komponenta (eng. Object Request Broker - ORB) ključna je komponenta sustava CORBA. Njena uloga je pružanje jedinstvenog sučelja prema višem, aplikacijskom sloju (slika 2.11), u svrhu stvaranja radnog okruženja koje će sakriti raspodijeljenu prirodu čitavog sustava. ORB ima sljedeća svojstva:

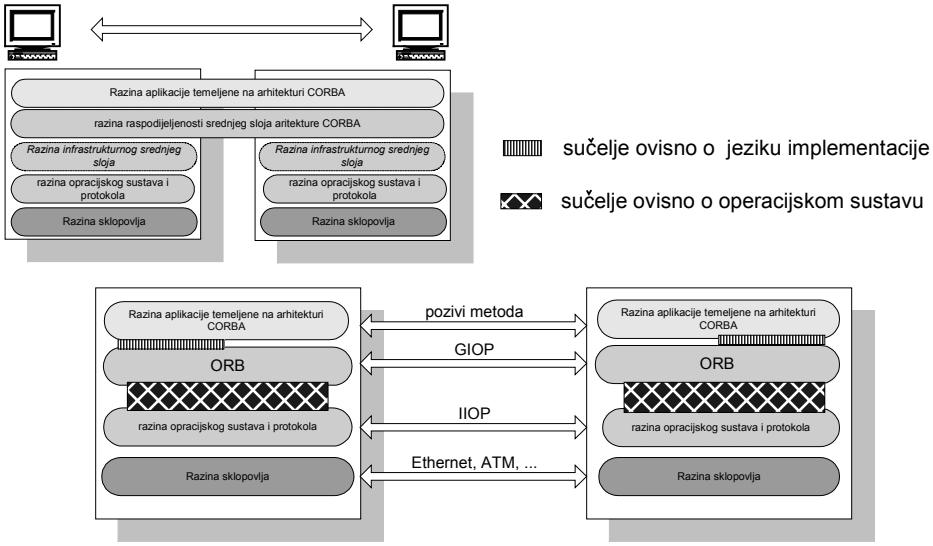
- *Transparentnost lokacije pozivanog objekta*: korisnik objekta nije svjestan trenutne lokacije pozivanog objekta. ORB, poznavajući lokaciju trenutnog objekta, može vršiti optimizacije poziva, no svojstvo transparentnosti se zadržava.

- *Jezična neovisnost*: korisniku nije bitno u kojem je programskom jeziku ciljni objekt implementiran, dovoljno je poznavanje njegove reference i sučelja koje implementira.

- *Neovisnost arhitekture*: korisniku nije bitno na kojoj se platformi izvodi ciljni objekt. Prilagođavanje različitim arhitekturama (različita dužina riječi, itd.) vrši se transparentno unutar posredničke komponente. Isto se odnosi i na različitosti u operacijskim sustavima računala korisnika i ciljnog objekta.

- *Neovisnost o komunikacijskom protokolu*: sva komunikacija između korisnika i objekta svodi se na pozive metoda, ovisno o programskom jeziku implementacije korisnika. Za komunikaciju putem računalne mreže zadužena je posrednička komponenta koja se brine za usklađivanje korištenih protokola između dvije posredničke komponente (na strani korisnika i na strani ciljnog objekta).

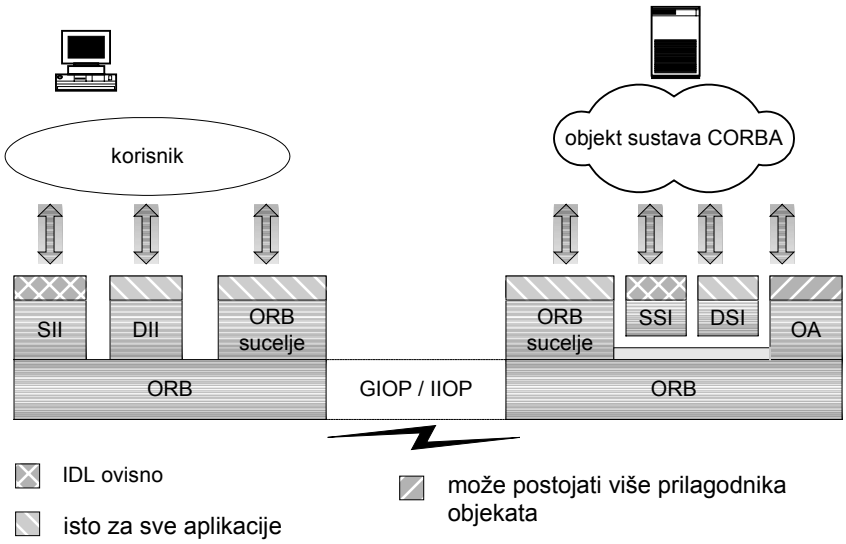
Da bi ostvarila svoju ulogu razine raspodijeljenosti srednjeg sloja, implementacija posredničke komponente mora biti platformno ovisna i koristiti standardizirani protokol komunikacije.



Slika 2.11. Posrednička komponenta kao razina raspodijeljenosti srednjeg sloja

Platformna ovisnost posredničke komponente je nužna, kako bi mogla koristiti sve potrebne resurse računala na kojem se izvodi. Stoga postoje inačice posredničke komponente za sve važnije operacijske sustave – MS Windows, Linux, Solaris, različite inačice Unix-a, itd. Jedina izvedba posredničke komponente koja nije platformno ovisna je ona za rad u okruženju Java. Java kao takva predstavlja razinu infrastrukturnog srednjeg sloja, te čini jedinstveno platformno neovisno sučelje prema resursima lokalnog računala. Posrednička komponenta može biti implementirana u više oblika, kao poseban proces, ili, najčešće, u obliku biblioteka koda i pripadajućih zaglavlja za programske jezike implementacije (C, C++, Java, itd.).

Standardizirani protokol komunikacije nužan je za ispravnu komunikaciju između instanci posredničkih komponenti na različitim računalima. Korišteni protokol naziva se GIOP (eng. *General Inter-ORB Protocol*), te čini viši, apstraktni sloj komunikacijskog protokola. Niži, implementacijski sloj, ovisan je o korištenom mrežnom protokolu, tj. o komunikacijskoj infrastrukturi. Jedini protokol koji standard propisuje, a koji sve posredničke komponente moraju implementirati, je IIOP (eng. *Internet Inter-ORB Protocol*). IIOP se koristi za komunikaciju putem TCP/IP temeljenih računalnih mreža.



Slika 2.12. Građa posredničke komponente

Na slici 2.12 detaljnije je prikazana građa posredničke komponente. Komunikacija između korisnika i ciljnog objekta odvija se na sljedeći način:

- Korisnik posjeduje referencu na udaljeni objekt. Ako je unutar korisničkog dijela aplikacije ugrađena statička komponenta (eng. *static stub*), poziv metode udaljenog objekta vrši se pozivom istoimene metode statičke komponente. Ovaj način komunikacije koristi tzv. statičko sučelje poziva (eng. *Static Invocation Interface* - SII). Ukoliko statička komponenta nije dostupna, ili se želi izvršiti asinkroni poziv (načini pozivanja metoda udaljenih objekata bit će objašnjena kasnije) koristi se tzv. dinamičko sučelje poziva (eng. *Dynamic Invocation Interface* – DII).

- Posrednička komponenta formira poziv metode udaljenog objekta sukladno protokolu GIOP, te se korištenjem protokola nižeg sloja (IIOP) poziv prosljeđuje posredničkoj komponenti na strani udaljenog objekta. Mrežna adresa posredničke komponente i ciljni objekt određuju se na osnovu podataka sadržanih unutar reference ciljnog objekta.

- Posrednička komponenta na strani ciljnog objekta prihvaća zahtjev i prosljeđuje ga ciljnom prilagodniku objekata (eng. *Object Adapter* – OA) u kojem je ciljni objekt prijavljen.

- Prilagodnik objekata identificira pozivani objekt, pronalazi njegovu implementaciju, ukoliko nije aktivna pokreće je, te joj prosljeđuje poziv metode. Slično mehanizmu poziva na strani korisnika, i na strani poslužitelja mogu biti korišteni statički i dinamički mehanizmi poziva implementacije

objekta. Statički poziv implementacije (eng. *Static Skeleton Interface* - SSI) koristi kostur (eng. *skeleton*) implementacije objekta generiran od strane prevodioca opisnog jezika sučelja, dok dinamički poziv implementacije (eng. *Dynamic Skeleton Interface* - DSI) u trenutku dospijeća poziva određuje koja će implementacija objekta biti korištena.

- Po završetku izvođenja poziva metode unutar implementacije objekta, rezultati se vraćaju korisniku.

U sustavu CORBA definirana su tri načina poziva metoda udaljenog objekta:

- Sinkroni pozivi metoda udaljenog objekta: izvršavanje korisnika udaljenog objekta blokirano je do prispijeća odgovora. Ovaj način poziva ekvivalentan je pozivu metoda lokalnih objekata unutar iste niti izvršavanja u objektno usmjerenim jezicima (Java, C++).

- Pozivi jednosmjernih metoda: program korisnik nastavlja s radom nakon što je ORB poslao poziv metode. Jednosmjerne metode ne smiju imati parametre poziva ni vraćati vrijednosti kao rezultate njihova izvršavanja. ORB ne jamči da je poziv metode ciljnog objekta stvarno izvršen.

- Odgođeni sinkroni pozivi metoda (eng. *deferred synchronous*) ne blokiraju rad korisnika. Zadaća korisnika provjeriti je da li je odgovor pozvanog udaljenog objekta pristigao, te preuzeti moguće rezultate poziva. Korištenje odgođenih sinkronih poziva za sada je moguće samo uporabom mehanizma DII.

2.4.2. Prilagodnici objekata

Prilagodnici objekata slijede uzorak oblikovanja prilagodnika, definiranog u [8], čija je uloga prilagodba sučelja objekta sučelju koje njegov korisnik očekuje i koristi. Prilagodnik u sustavu CORBA ispunjava tri ključne uloge:

- Sudjeluje u kreiranju objektnih referenci: svaki objekt sustava CORBA pripada jednom prilagodniku. U trenutku stvaranja reference (ne nužno istovremeno i stvaranju objekta) unutar nje zapisuje se ime pripadajućeg prilagodnika.

- Čuva podatke o registriranim objektima i njihovim implementacijama: svaki objekt mora imati pripadajuću implementaciju u nekom od programskih jezika, no svaka implementacija može pripadati više od jednom objektu.

- Prosljeđuje pozive metode objekata odgovarajućim implementacijama: po primitku ORB poziv prosljeđuje odgovarajućem prilagodniku.

U prvim verzijama standarda CORBA bio je definiran samo osnovni prilagodnik (eng. *Basic Object Adapter* – BOA). No njegovi nedostaci rezultirali su od verzije standarda 2.2. uvođenjem prenosivog prilagodnika objekata (eng. *Portable Object Adapter* – POA). POA sadrži listu svih u njemu registriranih objekata i odgovarajućih implementacija objekata. Prispijećem poziva metode udaljenog objekta ORB na osnovu informacija pohranjenih unutar objektne reference prosljeđuje zahtjev odgovarajućem prilagodniku. Prilagodnik, također na osnovu informacija unutar objektne reference, identificira ime pozvanog objekta i unutar liste registriranih objekata pronalazi pokazivač na implementaciju objekta (eng. *servant*). U slučaju da je implementacija objekta neaktivna, prilagodnik ju aktivira (npr. stvori objekt odgovarajuće klase), te poziva odgovarajuću metodu implementacije. Ako implementacija objekta nije pronađena u listi registriranih objekata, koriste se mehanizmi upravljača implementacija objekata (eng. *Servant Managers*), čija je uloga pronalaženje odgovarajuće implementacije i njeno pokretanje (eventualno i zaustavljanje njena rada).

POA omogućava hijerarhijsku organizaciju prilagodnika (struktura stabla), te dodjeljivanje zajedničkih karakteristika svim objektima na razini prilagodnika (npr. trajni ili privremeni objekti, stupanj sigurnosti, korišteni model pokretanja implementacija, itd.).

Postoje i druge implementacije prilagodnika objekata, npr. RTPOA (inačica 2.4 specifikacije CORBA) ili RTOA (TAO implementacija sustava CORBA) prilagođene specifičnim potrebama aplikacija, u ovom slučaju radu u stvarnom vremenu. U prilagodnicima namijenjenim radu u stvarnom vremenu koriste se optimizacije u svrhu postizanja što veće brzine rada sustava (korištenje bržih mehanizama komunikacije između objekata na istom računalu ili u istom procesu) ili prenošenja informacija o prioritetu izvršavanja niti ili procesa.

2.4.3. Objektne reference

Objektne reference jedinstveno označavaju pripadajući objekt unutar raspodijeljenog sustava temeljenog na CORBA-i. Struktura objektne reference prikazana je na slici 2.13.

Slika 2.13. Struktura objektne reference

Oznaka sučelja jedinstveno označava tip sučelja, a koristi se za dohvat opisa sučelja iz mjesta za pohranu sučelja (eng. *Interface Repository*).

Podatak o adresi procesa, a time i pripadajuće posredničke komponente, nužna je unutar reference kako bi se mogla uspostaviti veza između komponente na strani korisnika objekta i komponente na strani implementacije ciljnog objekta. Ovaj podatak ovisan je o korištenom komunikacijskom protokolu. U većini slučajeva koristi se protokol IIOP, stoga su nužni podaci za uspostavljanje veze IP adresa računala i port na kojem posrednička komponenta osluškuje zahtjeve.

Oba navedena podatka moraju biti dostupna svim implementacijama posredničke komponente kako bi one međusobno mogle surađivati.

Objektni ključ označava pozivani objekt unutar procesa poslužitelja. Objektni ključ sastoji se od imena prilagodnika objekata u kojem je objekt registriran (koristi se od strane posredničke komponente u svrhu prosljeđivanja poziva odgovarajućem prilagodniku) i imena objekta (koristi se od strane prilagodnika u svrhu pozivanja odgovarajuće implementacije objekta). Oblik zapisa objektnog ključa specifičan je implementaciji sustava CORBA, te se koristi samo na strani ciljnog objekta.

Objektne reference mogu biti pohranjene u postojanom obliku (npr. unutar datoteke ili imeničke usluge) radi kasnijeg korištenja. ORB nudi mehanizme pretvorbe objektne reference u niz karaktera i obratno radi lakšeg zapisa i prijenosa reference kao parametra poziva.

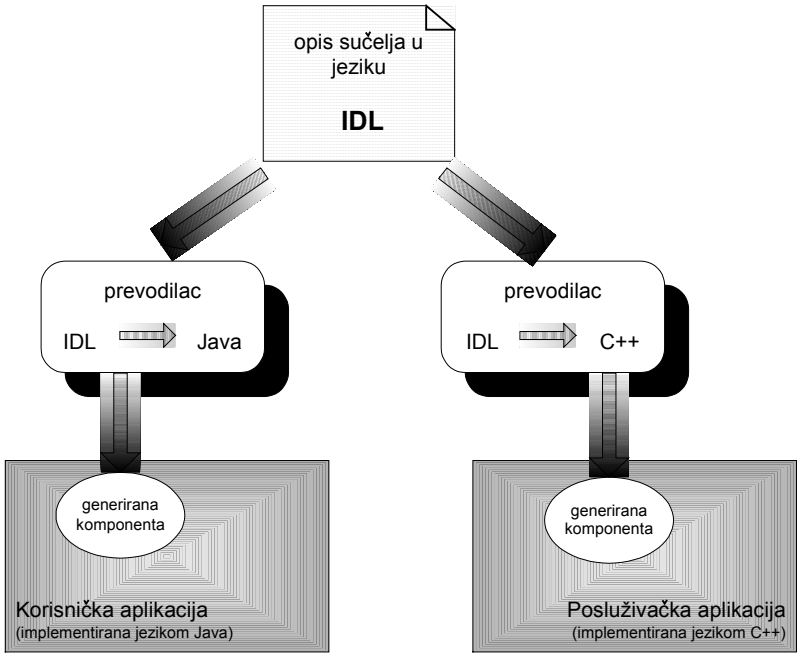
2.4.4. Opisni jezik sučelja

Za opis sučelja objekta sustava CORBA nužno je poznavanje:

- atributa i metoda od kojih se sučelje sastoji
- tipova podataka atributa, parametara metoda i povratne vrijednosti.

Atributi, metode i korišteni tipovi podataka moraju biti opisani neovisno o ciljnom jeziku implementacije i korištenoj platformi, kako bi njihova uporaba mogla biti platformno i jezično neovisna. Stoga je od strane OMG-a definiran opisni jezik sučelja (eng. *Interface Definition Language* – IDL). IDL definira osnovne tipove podataka putem kojih je moguće opisati složenije strukture podataka i metode sučelja sa svim pripadajućim parametrima. Također su u jezik preuzeti neki mehanizmi svojstveni objektno usmjerenim programskim jezicima, npr. nasljeđivanje. No općenitost opisnog jezika i korištenih osnovnih tipova podataka nužno je dovela do određenih kompromisa, kako bi implementacija ovako opisanih sučelja bila moguća u što većoj grupi programskih jezika.

Sučelja opisana jezikom IDL prevode se u ciljni jezik korištenjem odgovarajućeg prevodioca. Trenutno postoje prevodioci za jezike C++, Objektni C, C, Java, Smalltalk, Ada, OLE (za jezike kao što su Visual Basic, Delphi ili sustave kao Power Builder), COBOL, Eiffel, Modula 3,Perl, Tcl, Python, itd.



Slika 2.14. Prevođenje opisnog jezika sučelja u jezik implementacije

Komponente generirane prevođenjem opisa sučelja koriste se kao komponente:

- mehanizma statičkog poziva metoda ciljnog objekta na strani korisnika (SII)
- mehanizma statičkog poziva implementacije na strani implementacije objekta (SDI).

Također, pomoću prevodioca IDL, opisi sučelja pohranjuju se u mjesto za pohranu sučelja (eng. Interface Repository) u svrhu dinamičkog dohvata opisa sučelja i dinamičkog stvaranja poziva metoda ciljnog objekta (DII).

3. Analiza problema raspodijeljenih sustava temeljenih na CORBA-i

Osnovna uloga CORBA-e skrivanje je kompleksnosti mrežne komunikacije između komponenti raspodijeljenog objektno usmjerenog računalnog sustava, prvenstveno problema lokacije udaljenih komponenata, heterogenosti računalnih arhitektura na kojima se komponente izvode i implementacijskih jezika komponenata. Iako se na prvi pogled stječe dojam da CORBA (i srodni sustavi, npr. Java RMI, DCOM) rješavaju pred njih postavljeni problem izrade raspodijeljenih aplikacija, svodeći ih (približno) na problem izrade neraspodijeljenih aplikacija, detaljnija analiza otkriva svu kompleksnost raspodijeljenih sustava, a posljedično i nedostatke u spomenutim rješenjima. U [12] navode se četiri atributa svojstvena raspodijeljenim sustavima, a koji ih bitno razlikuju u odnosu na neraspodijeljene sustave: kašnjenja uzrokovana raspodijeljenošću, uniformnost pristupa udaljenoj memoriji, paralelizam izvršavanja i djelomična nefunkcionalnost sustava. CORBA samo djelomično rješava problem uzrokovan prvim dvama atributima, dok posljednja dva uopće ne dotiče.

Problemom kašnjenja rada raspodijeljenog sustava kao posljedice mrežne komunikacije CORBA se bavi samo na najnižoj razini mrežne komunikacije (GIOP/IIOP). Problemi vezani za kašnjenja uglavnom se svode na pokušaje ponavljanja neuspjelih poziva metoda udaljenih objekata uzrokovanih istekom vremena čekanja na odgovor (eng. *timeout*). Ove pogreške djelomično su transparentne u odnosu na implementacijski kod raspodijeljene aplikacije: greška u komunikaciji bit će dojavljena aplikacijskom kodu tek nakon nekoliko neuspjelih pokušaja poziva metoda udaljenog objekta.

Problem pristupa lokalnoj i udaljenoj memoriji CORBA rješava njihovim striktnim razdvajanjem, a što se izravno očituje u izvornom kodu raspodijeljene aplikacije. Iako se, lokalno, poziv metoda udaljenog objekta implementira kao poziv ekvivalentnih metoda lokalnog objekta zastupnika, a time i pridržavanjem pravila implementacijskog jezika, bitne razlike očituju se u:

- brojanju referenci udaljenog objekta
- rukovanju parametrima metoda udaljenog objekta (s obzirom na tip podatka i tip prosljeđivanja)
- rukovanju iznimkama.

Razlike u radu s lokalnim i udaljenim objektima moraju se uzeti u obzir tijekom faza oblikovanja i implementacije raspodijeljenih aplikacija temeljenih na CORBA-i. U fazi oblikovanja raspodijeljene aplikacije od iznimne je važnosti pravilna

identifikacija skupa udaljenih objekata i oblikovanje njihova sučelja (korištenjem opisnog jezika sučelja IDL). U fazi implementacije bitno je ispravno rukovanje iznimkama, prilagođenost implementacije rukovanju pretpostavljenim brojem objekata unutar sustava i ostalim bitnim implementacijskim pitanjima u raspodijeljenim sustavima (blokiranje rada sustava, usporednost izvršavanja, itd.). Zanemarivanje razlika u radu s udaljenim objektima za posljedicu ima raspodijeljene sustave koji ne odgovaraju početnim zahtjevima (kako funkcionalnim tako i nefunkcionalnim), a pogreške u radu kojih se uočavaju većinom u kasnijim fazama razvoja i u radu sustava. Ispravljanje takvih pogrešaka komplicirano je i zahtijeva puno vremena i znatne troškove, sukladno fazi u kojoj su pogreške uočene.

Sa stanovišta sustava u cjelini prepoznaju se tri osnovne faze životnog vijeka raspodijeljene aplikacije :

- faza razvoja i testiranja
- faza početnog rasporeda komponenti sustava
- faza korištenja i evolucije sustava.

Svakoj od nabrojenih faza životnog vijeka svojstveno je uočavanje pojedinih vrsta pogrešaka (u oblikovanju i implementaciji), prvenstveno kao rezultat promjene izvršne okoline sustava. Većina uočenih pogrešaka kao posljedicu ima statičku promjenu sustava – vraćanje sustava u jednu od prethodnih faza životnog vijeka (izmjene u oblikovanju i implementaciji), dok su dinamičke promjene sustava rjeđe i svojstvene fazi korištenja i evolucije (promjene u sustavu bez potrebe prekidanja njegova korištenja). U daljnjem tekstu bit će opisane pogreške u oblikovanju i implementaciji, s obzirom na fazu životnog vijeka sustava u kojoj se najčešće uočavaju, te njihove posljedice na rad i potrebne izmjene sustava.

3.1. Faza razvoja sustava

U fazi razvoja sustava osnovni je cilj funkcionalna ispravnost rada raspodijeljenog sustava. Osnovna razlika između razvoja "klasičnih" i aplikacija temeljenih na sustavu CORBA korištenje je jezika IDL za opis sučelja objekata. Model razvoja koji se u ovoj fazi oblikovanja sustava koristi najčešće je model vodopada [14] (tj. nastoje se izbjeći dodatne promjene sučelja udaljenih objekata u kasnijoj fazi - kodiranju implementacije sustava), iako je moguća i primjena iterativnog modela. Iterativni model koristi se većinom tijekom same implementacije sustava (kako neraspodijeljenog tako i raspodijeljenog dijela sustava) u smislu postupnog implementiranja funkcionalnosti i testiranja udaljenih objekata (metode udaljenog objekta implementiraju se postupno, uz testiranje njihove implementacije). Alati koji

se koriste u fazi razvoja sustava isti su oni koji se koriste u razvoju "klasičnih" aplikacija:

- integrirana razvojna okruženja (npr. MS Visual Studio, JBuilder, itd.) koja u sebi sadrže prevodilac implementacijskog jezika, alat za traženje i ispravljanje pogrešaka, itd.
- alati za mjerenje performansi programa
- alati za analizu izvornog koda.

Vrste pogrešaka koje se susreću u fazi razvoja, pored klasičnih pogrešaka u korištenom jeziku implementacije, svode se na:

- nedostatke u sučelju udaljenih objekata koji se uočavaju prilikom implementacije i korištenja udaljenih objekata, najčešće se svode na necjelovitost sučelja (npr. u udaljenom objektu koji implementira red poruka nije predviđena metoda za promjenu prioriteta poruke)

- krivom korištenju udaljenih objekata (najčešće posljedica razdvojenosti implementacije i korištenja udaljenog objekta na dva različita programerska tima, nedostatna dokumentacija o korištenju gotovog objekta, nedovoljna komunikacija između različitih timova, itd.)

- klasičnim pogreškama u implementaciji udaljenih objekata (nedovoljno testiranje implementacije objekta, nepredviđene vrijednosti parametara poziva metode, nepredviđen redoslijed poziva metoda, itd.)

- greške uzrokovane neiskustvom u radu sa sustavom CORBA (brojanje referenci, krivo korištenje specifičnih tipova podataka CORBA-e, itd.).

Bolje implementacije CORBA-e imaju mogućnost samostalnog prepoznavanja i prijave pogrešaka tijekom rada (npr. Orbacus4 implementacija CORBA-e prijavljuje pogreške u brojanju referenci i pogrešnim rukovanjem tipovima podataka).

Pogreške svojstvene raspodijeljenim sustavima [12] u fazi razvoja sustava teško je uočiti, prvenstveno zbog "strogo kontroliranog" radnog okruženja. U toku razvoja sustava i korisnik i poslužitelj strana aplikacije (dijela aplikacije) izvršavaju se na istom računalu ili na susjednim računalima priključenim na brzu lokalnu računalnu mrežu, a broj je korisnika (osim prilikom posebnih testiranja sustava) minimalan.

3.2. Početni raspored komponenti sustava

Ovu fazu životnog vijeka aplikacije karakterizira prelazak u potpuno novo radno okruženje u kojemu će aplikacija biti korištena. Tek u ovoj fazi djelomično dolaze do izražaja pogreške u oblikovanju i implementaciji raspodijeljenog sustava, prvenstveno pod utjecajem različitosti raspoloživih resursa. Tri su osnovne grupe problema:

- neprilagođenost novoj radnoj okolini
- problemi u integraciji s postojećim komponentama sustava
- utjecaj kašnjenja na rad aplikacije.

Neprilagođenost novoj radnoj okolini posljedica je nemogućnosti pronalaženja potrebnih resursa za rad sustava. Tipičan primjer ove vrste problema je ugrađivanje lokacije korištenih resursa unutar izvornog koda programa. Prilagodljivost sustava postiže se korištenjem datoteka za podešavanje i usluga za pohranu i dohvat referenci (npr. imeničke usluge).

Integracija s postojećim komponentama sustava naročito je problematična ukoliko rad postojećeg sustava kojeg su one sastavni dio ne smije biti ometan.

Različitost kašnjenja (kao posljedica mrežne komunikacije i različite snage računala na kojima se dijelovi raspodijeljene aplikacije izvršavaju) i početna nestabilnost dijelova sustava (što rezultira zastojima u radu) u pravilu ima sljedeće posljedice:

- neprihvatljivu sporost rada sustava
- nemogućnost komunikacije dijelova sustava uslijed isteka vremena čekanja na odgovor (eng. *timeout*)
- ovisnost o raspoloživosti drugih komponenata sustava i njihovoj međusobnoj povezanosti (npr. redoslijed pokretanja komponenata sustava po prethodno uvedenom redoslijedu nije moguć u novom radnom okruženju)
- nemogućnost oporavka od djelomičnog zatajenja sustava (npr. ponovno pokretanje jedne od komponenti sustava zahtijeva i ponovno pokretanje korisnika te komponente).

Sve navedene pogreške u radu novog raspodijeljenog sustava, uz trenutno neuočene, još snažnije dolaze do izražaja tijekom samog korištenja sustava. Kako troškovi ispravljanja pogrešaka rastu ovisno o fazi životnog vijeka sustava, poželjno je njihovo što ranije otkrivanje. Detaljniji opis prethodno navedenih pogrešaka i njihovih uzroka bit će navedeni u opisu problema tijekom korištenja sustava.

3.3. Korištenje raspodijeljenog sustava

U fazi korištenja raspodijeljenog sustava susreću se dvije vrste problema kojih su uzroci:

- u oblikovanju i implementaciji sustava
- u promjenama raspoloživih resursa radne okoline.

3.3.1. Skalabilnost sustava

Problemi skalabilnosti sustava uočavaju se kao zamjetno smanjenje performansi sustava ili nedostupnost sustava korisniku. Sa strane logičke ispravnosti rada sustava ne mogu se detektirati nikakve pogreške, no sustav ne ispunjava pred njega postavljene funkcionalne zahtjeve. Dva su bitna čimbenika čijim (pojedinačnim ili združenim) utjecajem dolazi do pojave ovog problema :

- veliki istovremeni broj korisnika sustava
- veliki broj aktivnih objekata u sustavu

Uzroci problema rada sustava mogu se raščlaniti na sljedeće:

- ograničenost resursa računalnih sustava na kojima se raspodijeljena aplikacija (ili jedan njen dio) izvodi
- utjecaj kritičnih točaka skalabilnosti u implementaciji sustava CORBA
- utjecaj kritičnih točaka skalabilnosti u implementaciji raspodijeljene aplikacije.

Veliki broj istovremenih korisnika sustava postavlja dodatne zahtjeve na snagu obrade i količinu ugrađene memorije računalnih sustava na kojima se komponente raspodijeljenog sustava izvode. Jačina povezanosti raspodijeljenih komponenata (broj potrebnih međusobnih poziva metoda udaljenih objekata) također postavlja dodatne zahtjeve na propusnost računalne mreže. Utjecaj velikog broja objekata u sustavu uglavnom se ograničava na potrebnu količinu memorije za pohranu njihovih stanja, kako trajne memorije (stanja objekata u pravilu se pohranjuju u bazama podataka), tako i glavne memorije računalnih sustava za rukovanje aktivnim objektima. Najjednostavnije rješenje problema skalabilnosti bilo bi nadogradnja sklopovlja korištenih računalnih sustava i povećanje propusnosti računalnih mreža, no, nažalost, takva rješenja ili iziskuju neprihvatljivo velika sredstva ili se sustavi jednostavno ne mogu dovoljno nadograditi. U pojedinim slučajevima dostatno je promijeniti lokaciju izvršavanja pojedinih komponenti raspodijeljenog sustava kako bi se ostvarile

zadovoljavajuće performanse (premještanje memorijski zahtjevne komponente sustava na računalo s dovoljno glavne memorije, postavljanje dvije snažno komunikacijski povezane komponente na isto računalo, itd.).

Loše oblikovan i (ili) implementiran raspodijeljeni sustav, nažalost, ostati će takav ma koliko resursa dodano računalnim sustavima na kojima se izvodi. Stoga je potrebno razmotriti bitne čimbenike u oblikovanju i implementaciji sustava koji mogu drastično utjecati na kasniji rad sustava (i kasno otkrivanje pogrešaka). Na prvu grupu čimbenika osoba odgovorna za oblikovanje i implementaciju nema izravan utjecaj jer su sadržani u implementaciji sustava CORBA. No moguć je neizravan utjecaj:

- pribavljanjem informacija o načinu implementacije kritičnih točaka i izmjerenih performansi korištene implementacije sustava CORBA od strane njegova proizvođača
- uzimanjem u obzir navedenih kritičnih točaka i smanjenjem njihovog utjecaja na performanse pojedinim rješenjima u implementaciji razvijanog sustava.

Kritične točke u svakoj implementaciji sustava CORBA s obzirom na skalabilnost njima razvijanih sustava su (od inačice standarda CORBA 2.2 na dalje):

- mjesto za pohranu implementacija
- prenosivi prilagodnik objekata.

Zajedničko objema kritičnim točkama je posrednička uloga u pronalaženju određenog objekta. Veliki broj aktivnih objekata, kao i zrnatost njihovog zapisa (na razini prilagodnika objekta kojima pripadaju ili na razini pojedinog objekta) rezultiraju podatkovnim strukturama s velikim brojem elemenata, a time i znatnim vremenom potrebnim za njihovo pretraživanje (naravno, ovisno o korištenoj podatkovnoj strukturi i algoritmima pretraživanja).

Mjesto za pohranu implementacija koristi se prilikom inicijalnog dohvata referenci postojanih objekata (osnovnih objekata dijelova raspodijeljenog sustava), a zadaća mu je čuvanje spomenutih referenci i, po potrebi, pokretanje procesa koji čine njihovu izvršnu okolinu, ovisno o korištenom tipu pokretanja (dijeljeni, po korisniku, po procesu, ...). Ukoliko se procijeni da često dohvaćanje referenci postojanih objekata korištenjem mjesta za pohranu implementacija može imati negativan utjecaj na performanse sustava, moguća su dva rješenja:

- korištenje manjih domena aplikacija (mjesto za pohranu implementacija sadrži manji broj referenci na postojane objekte)

- korištenje drugih načina pohrane i dohvata referenci postojanih objekata (datoteke za prilagodbu sustava, baze podataka, itd.).

Negativne posljedice korištenje alternativnih načina pohrane i dohvata referenci postojanih objekata su:

- problem pristupa objektima koji su promijenili proces – izvršnu okolinu
- nemogućnost transparentnog pokretanja procesa – izvršnih okolina postojanih objekata (procesi – izvršne okoline pokreću se "ručno")
- problem dinamičkog prilagođavanja aplikacije promjenom korištenog udaljenog objekta u toku rada.

Prenosivi prilagodnik objekata zadrži strukturu podataka u kojoj su registrirani svi trenutno aktivni objekti (ime objekta, pokazivač na njegovu implementaciju), a kojoj se pristupa prilikom svakog poziva metode jednog od objekata sadržanog unutar pojedinog prilagodnika. Stoga je nužna kvalitetna implementacija te strukture i algoritma pretraživanja u njoj sadržanih podataka. Kao nadogradnja prethodno opisanog statičkog mehanizma pronalaženja implementacije aktivnog objekta, prenosivi prilagodnik objekata pruža mogućnost i dinamičkog pronalaženja implementacije objekta korištenjem dvaju upravljača implementacija: pokretača implementacije (eng. *Servant Activator*) i lokatora implementacije (eng. *Servant Locator*). Zadaća pokretača implementacije je pronalaženje implementacije trenutno neaktivnog objekta, njeno pokretanje i registracija u strukturi podataka aktivnih objekata prilagodnika. Pokrenuta implementacija ostaje aktivna do daljnjega (do kraja rada procesa – izvršne okoline). Lokator implementacije ima sličnu ulogu kao i pokretač implementacije, bitna razlika je u neregistraciji aktivirane implementacije unutar strukture podataka prilagodnika (lokator implementacije može se promatrati kao prilagodnik unutar prilagodnika). Upravljači pružaju iznimno veliku prilagodljivost rada s implementacijama objekata, no mogu predstavljati i "usko grlo" sustava ukoliko nisu dobro izvedeni. Najčešći problem kod upravljača potrebno je vrijeme pokretanja i zaustavljanja implementacija postojanih objekata čija se stanja trajno pohranjuju (u datoteke, baze podataka, itd.).

Kritične točke skalabilnosti raspodijeljene aplikacije uglavnom se svode na pitanja implementacije usko povezana s potrebnom radnom memorijom i procesorskom snagom računala. Najčešći problemi vezani su s:

- velikim brojem trenutno aktivnih objekata u sustavu
- vremenom potrebnim za pokretanje implementacija neaktivnih objekata
- vremenom potrebnim za izvršavanje poziva pojedinih metoda objekata.

Veliki broj trenutno aktivnih objekata u sustavu neizostavno povlači za sobom i veliku količinu memorije potrebnu za smještaj njihovih stanja. Statička aktivacija svih potrebnih objekata u pojedinim slučajevima čak i nije izvodiva (npr. u sustavima koji služe kao objektno sučelje prema relacijskoj bazi podataka). Stoga se u navedenim slučajevima pribjegava dinamičkom upravljanju aktivnim objektima i njihovim implementacijama. Korištenjem upravitelja implementacija moguća je čvrsta kontrola kako nad brojem trenutno aktivnih objekata, tako i nad identitetom svakog pojedinog aktivnog objekta. Pokretačem implementacije objekata aktivira se implementacija nepokrenutog objekta i, ako je nužno, dobavljaju njegova pohranjena stanja. No nedostatak ove metode je gomilanje aktivnih objekata unutar sustava, što, nakon duljeg rada, rezultira smanjenjem njegovih performansi. Lokator implementacija objekata omogućuje daleko veću kontrolu nad aktivnosti objekta (i pokretanje i zaustavljanje implementacija objekata), no unosi i veće opterećenje sustava (vrijeme potrebno za dohvaćanje i pohranu stanja objekata). Uobičajena strategija korištena u sustavima s velikim brojem objekata je određivanje gornje granice broja trenutno aktivnih implementacija. Pokretanje objekta (ako već nije pokrenut) vrši se na prvi zahtjev za pojedinim objektom (pozivom jedne od njegovih metoda), a ukoliko je gornja granica broja aktivnih metoda dosegnuta, jedna od trenutno aktivnih implementacija se zaustavlja. Bitno je da se učestalost pokretanja i zaustavljana implementacija svede na minimum korištenjem odgovarajućih strategija (zaustavlja se zadnje korištena implementacija, najmanje učestalo korištena implementacija, itd.).

Pojedine metode objekta mogu zahtijevati duže vremensko razdoblje za izvršavanje, pa njihovo učestalo korištenje od strane više korisnika može također predstavljati "usko grlo" sustava. Dva su osnovna načina kojima se smanjuje i (ili) izbjegava navedeni problem: ili, ako je to moguće, korištenjem asinkronih (neblokirajućih) poziva takvih metoda, ili korištenjem više implementacija u raspodijeljenom sustavu (implementacije se izvode na različitim računalima).

3.3.2. Paralelizam u sustavu CORBA

Dva su moguća izvora paralelizma u sustavima CORBA:

- paralelizam kao posljedica različitih modela pokretanja izvršnih okolina
- paralelizam kao posljedica različitih modela dodjele niti izvođenja izvršavanju poziva metoda objekata.

Mjesto za pohranu implementacija (eng. *Implementation Repository*) odgovorno je za pokretanje procesa koji čine izvršnu okolinu korištenih objekata. Osim u slučaju postojanog poslužitelja (u čijem slučaju mjesto za pohranu implementacija nije zaduženo za pokretanje procesa poslužitelja), proces poslužitelj pokreće se :

- kao dijeljeni proces (eng. *shared activation*) - svi pozivi metoda objekata upućuju se istom procesu

- novi proces po svakom pojedinom korisniku objekata (eng. *per-client activation*) - svaki pojedini program korisnik pristupa zasebnom procesu u kojem je sadržan ciljni objekt

- novi proces po svakom pojedinom korisniku sustava (eng. *per-user activation*) – svaki pojedini korisnik sustava pristupa zasebnom procesu u kojem je sadržan ciljni objekt

- novi proces po svakom pojedinom pozivu metode (eng. *per-method activation*) – svaki pojedini poziv metode objekta rezultira pokretanjem novog procesa u kojem je ciljni objekt sadržan.

Unutar samoga procesa koji čini izvršnu okolinu sadržanih objekata postoji više niti izvršavanja, ovisno o pojedinoj implementaciji sustava CORBA. Uobičajene su dvije uslužne niti – jedna za ulaznu a druga za izlaznu komunikaciju s ostalim posredničkim komponentama raspodijeljenog sustava. Broj niti potrebnih za izvršavanje pridošlih poziva metoda sadržanih objekata moguće je kontrolirati uporabom različitih strategija, ovisnih o korištenoj implementaciji. Najčešće implementirane strategije su:

- jedna nit izvršavanja poziva metoda – svi pozivi metoda sadržanih objekata izvode se slijedno, po vremenu prispijeća poziva (i prioritetima poziva)

- nit po programu korisniku – svi pozivi metoda istoga programa korisnika izvode se slijedno

- nit po pozivu metode – svaki poziv pojedine metode objekta izvodi se u zasebnoj niti.

Pokretanje nove niti predstavlja značajno opterećenje sustava, stoga se nastoje spriječiti učestala pokretanja niti u trenutku prihvata poziva. Najčešće implementiran mehanizam dodjeljivanja niti pristiglim pozivima (s obzirom na korištenu strategiju) je tzv. mehanizam bazena niti (eng. *thread pool*). Prilikom pokretanja procesa stvara se predefiniran broj niti, koje se dodjeljuju u trenutku prispijeća poziva metoda, a ostatak vremena su neaktivne, ne trošeći resurse računala na kojem se proces izvršava. Ukoliko u trenutku dolaska poziva metode objekta nema slobodnih niti, poziv se stavlja u red čekanja.

Razvijani sustavi moraju biti oblikovani i implementirani s obzirom na predviđeno korištenje modela pokretanja procesa i strategiju dodjele niti. Posljedice neuzimanja u obzir mogućnosti da istovremeno može biti aktivno više procesa u pravilu za posljedicu ima probleme u radu s dijeljenim resursima (blokiranje rada sustava, utrka za resursima, neispravnost sadržaja datoteka ili zapisa u bazi podataka, itd.). Aktivnost više od jedne niti unutar koda u kojem višenitno izvršavanje nije predviđeno, uz već opisane probleme s dijeljenim resursima, za posljedicu ima ili neodređeno ponašanje procesa ili njegovo rušenje. Kritične točke takvih implementacija u pravilu su strukture podataka, čije nesinkronizirano mijenjanje dovodi do fatalnih pogrešaka u radu procesa i njegovu rušenju.

Također je potrebno obratiti pozornost na moguće probleme prilikom pokretanja i zaustavljanju rada procesa, u kojem trenutku je moguće pojavljivanje utrke za resursima i blokiranja rada kako procesa koji se pokreće tako i procesa koji završava s radom. Najčešći uzrok ovih problema je pristup dijeljenim resursima (npr. datoteka za pohranu trajnih podataka procesa) kod procesa koji koriste mjesto za pohranu implementacija.

3.3.3. Nepovratno trošenje memorije računalnog sustava

Tzv. “curenje memorije” pogreška je svojstvena svim sustavima razvijanim u programskim jezicima koji omogućuju (ili čak potiču) izravan rad s memorijom. No tek u sustavima koji su namijenjeni dugotrajnom i stabilnom radu, a čija stabilnost ovisi o stabilnosti svake pojedine komponente, problemi s nepravilnim oslobađanjem memorije naročito dolaze do izražaja. Čak i gubitak vrlo malih količina slobodne memorije u takvim sustavima može biti kritičan iz dva razloga:

- sama dugotrajnost rada raspodijeljenih sustava rezultira akumulacijom izgubljene memorije

- ukoliko se pogreška nalazi na kritičnom mjestu u sustavu, npr. u destruktoru implementacije često korištenog objekta, ukupni gubitak memorije će se akumulirati u vrlo kratkom vremenu.

Najjednostavniji način rješavanja opisanog problema je periodičko zaustavljanje i ponovno pokretanje sustava, čiji je mehanizam i ugrađen u sustav CORBA putem automatske deaktivacije procesa čiji sadržani objekti nisu korišteni u određenom vremenskiom periodu. Ponovno automatsko pokretanje prilikom prvog poziva metode objekta sadržanog u neaktivnom procesu omogućeno je korištenjem mjesta za pohranu implementacija. Izravan utjecaj nepovratnog trošenja memorije i nije toliko

bitan za ispravan rad sustava, jer su njegove primarne posljedice ograničene na lokalni proces. Bitnije su popratne pojave u raspodijeljenom sustavu, npr. reakcija na nefunkcionalnost jedne od komponenata, te uspješan oporavak sustava od pogreške.

Gubitak radne memorije ne mora biti posljedica pogreške u kodiranju na strani implementacije udaljenog objekta, već na strani njegova korisnika. Pogreške u lokalnom brojanju referenci udaljenog objekta na strani njihovih korisnika mogu za posljedicu imati nepotrebnu aktivnost implementacija objekata na strani poslužitelja. Ukoliko se radi o postojanim objektima, u većini slučajeva ovakve pogreške ne utječu bitno na rad sustava (pod uvjetom da je broj postojanih objekata u sustavu "razumno" velik ili se koristi kontrola broja aktivnih implementacija objekata npr. uporabom upravitelja implementacija). Veći problem nastaje ukoliko se greška dešava u korištenju privremenih objekata, za koje se u velikoj većini slučajeva ne koriste naprednije (a samim time i složenije za implementaciju) metode provjere stvarnog korištenja. Gomilanje aktivnih, a nekorištenih privremenih objekata ima za rad sustava gotovo iste posljedice kao i "curenje" radne memorije.

3.3.4. Djelomično zatajenje sustava

Djelomično zatajenje javlja se u slučaju prestanka rada jedne ili više sastavnih komponenata raspodijeljenog sustava. Ne ulazeći u razloge prestanka rada sastavne komponente sustava, od najveće važnosti je reakcija ostatka sustava na pogrešku u radu. Dobro oblikovani i izvedeni sustavi oporaviti će se od djelomično zatajenja bez bitnih posljedica po cjelovitost podataka i ukupnog stanja raspodijeljenog sustava, po mogućnosti bez potrebe za ponovnim pokretanjem svih (ili većine) gradivnih komponenata. Loše oblikovani i izvedeni sustavi na djelomično zatajenje reagirat će potpunim zatajenjem sustava, gubitkom ili narušavanjem cjelovitosti podataka, te nemogućnosti određivanja globalnog stanja sustava nakon oporavka od pogreške. Bitnu ulogu u izgradnji robusnih sustava imaju i faza oblikovanja (definicija sučelja) i faza implementacije (reakcije na pogreške i metode oporavka), što će na primjerima u daljnjem tekstu biti zorno prikazano.

Oporavak od kritične pogreške (prestanka rada procesa – izvršne okoline objekta) na strani implementacije objekta snažno je ovisan o samoj aplikaciji, te nije od većeg interesa u daljnjem razmatranju. Oporavak od pogreške na strani poslužitelja može biti relativno jednostavno izveden korištenjem posredničke uloge mjesta za pohranu implementacija, koje će u slučaju neaktivnosti procesa inicirati njegovo ponovno pokretanje. Na implementaciji procesa je da detektira razlog prethodnog prestanka rada i pokrene odgovarajući postupak oporavka.

Uočavanje pogreške u radu sustava na strani korisnika udaljenog objekta i pravilno postupanje u cilju smanjenja nastale štete najčešći je nedostatak sustava temeljenih na CORBA-i. Ovi nedostaci posljedica su modela i postupaka programiranja preuzetih iz neraspodijeljenih sustava, gdje pojam djelomičnog zatajenja sustava ne postoji – sustav u cjelini ili radi ili ne radi.

CORBA potiče programera, kroz mehanizme svojstvene pojedinom jeziku implementacije, na uočavanje i obradu dojavljenih pogrešaka u radu sustava, konkretno pogrešaka uočenih tijekom poziva metoda udaljenih objekata. U terminologiji CORBA-e pogreške uočene tijekom poziva metoda udaljenog objekta nazivaju se iznimke. Dvije su osnovne vrste iznimaka: iznimke svojstvene sučelju pozivanog objekta i iznimke sustava. Vrsta iznimke određuje ozbiljnost pogreške i postupke potrebne za oporavak (ako je oporavak moguć).

Iznimke svojstvene pojedinom sučelju definirane su na razini raspodijeljene aplikacije, te, funkcionalno, čine dodatan tip podatka koji može biti vraćen kao odgovor na poziv metode udaljenog objekta. Pojava aplikacijske iznimke označava, gledajući sa strane komunikacijskog dijela sustava, uspješno izvođenje poziva, no pojavu pogreške na razini korištenja udaljenog objekta, ne njegove implementacije. S druge strane, iznimke sustava označavaju pojavu pogreške ili u komunikacijskom podsustavu ili u izvođenju poziva metode udaljenog objekta, što za posljedicu može imati neodređenost globalnog stanja sustava.

Tri su osnovna tipa iznimki sustava:

- `TRANSIENT` – označava nesigurnost sustava u trenutno stanje udaljenog objekta, najčešće kao posljedica nemogućnosti komunikacije s procesom u kojem je objekt sadržan

- `COMM_FAILURE` – u tijeku komunikacije s procesom u kojem je objekt sadržan došlo je do prekida veze. Iznimka se najčešće se javlja kao rezultat kritične pogreške u izvođenju metode i prekidu rada procesa poslužitelja

- `OBJECT_NOT_EXIST` – objekt čija je metoda pozvana više ne postoji (kategorična tvrdnja).

Svaka iznimka posjeduje i dodatnu informaciju o statusu obavljene operacije, a koja je od iznimne važnosti za pravilan oporavak od pogreške. Definirane su: `COMPLETED_YES` (operacija uspješno izvršena unatoč pogrešci), `COMPLETED_NO` (operacija nije izvršena), i `COMPLETED_MAYBE` (ne može se sa sigurnošću tvrditi da li je operacija izvršena ili ne).

Dvije su osnovne grupe propusta u razvoju raspodijeljenih aplikacija koje bitno određuju (ne)mogućnost optimalne reakcije sustava na djelomično zatajenje:

- propusti u implementaciji sustava, svode se na:
  - ignoriranje mogućnosti podizanja iznimke sustava
  - generalizaciji iznimki sustava

- propusti u oblikovanju sučelja.

Potpuno ignoriranje mogućnosti podizanja iznimke sustava prilikom poziva metode udaljenog objekta rezultira prestankom rada procesa u slučaju njene pojave, bez ikakve poruke o vrsti i uzroku pogreške, što je za iole ozbiljan sustav neprihvatljivo. No, u neraspodijeljenim sustavima ovakva praksa je dosta česta jer slijedi logiku da je postupak oporavka od kritične pogreške u lokalnom procesu u bilo kojem slučaju nemoguć.

Generalizacija pogreški sustava može voditi u potpuno krive postupke oporavka od pogreške. Postoji bitna razlika između `TRANSIENT` i `OBJECT_NOT_EXIST` tipova iznimki sustava: u prvom slučaju potrebno je ponovo pokušati s pozivom metode udaljenog objekta, dok to u drugom slučaju nema nikakvog smisla.

Propusti u oblikovanju sučelja daleko su suptilnije prirode od propusta u obradi iznimki sustava, te su uglavnom vezani na prije opisane popratne informacije iznimki sustava o statusu obavljene operacije. Dok za informacije o statusu poziva metode udaljenog objekta `COMPLETED_YES` ili `COMPLETED_NO` može sa sigurnošću utvrditi stanje sustava nakon pogreške, `COMPLETED_MAYBE` znatno otežava oporavak, a u pojedinim slučajevima i u potpunosti onemogućava integraciju dijelova sustava u stabilnu i sigurnu raspodijeljenu aplikaciju. Stoga je potrebno uvođenje dodatnih informacija o stanju sustava, a koje je potrebno razmjenjivati između funkcionalno vezanih komponenti. Uvođenje dodatnih informacija o stanju sustava, najčešće u obliku dodatnih paramatara poziva metoda udaljenih objekata, zahtjeva i promjene postojećih definicija sučelja. Što je nužnost promjene definicije sučelja otkrivena kasnije u razvoju (ili radu) sustava, to su vrijeme i sredstva za izvođenje promjene veća.

3.3.5. Prenošenje podataka neodređene dužine

Problem prenošenja podataka neodređene dužine u sinkronim mrežnim sustavima svodi se na problem količine raspoložive radne memorije za prihvat i pohranu tih podataka. U krajnjem slučaju, ukoliko je količina podataka koja se dohvaća jednim

pozivom metode udaljenog objekta dovoljno velika, za posljedicu će imati popunjavanje sve raspoložive memorije i pojavu kritične pogreške sustava (rušenje procesa).

U svrhu izbjegavanja nekontroliranog popunjavanja velike količine memorije uvode se tzv. objekti iteratori, korištenjem kojih se podaci dohvaćaju postupno, u dužini segmenta navedenoj kao parametar poziva udaljene metode. Naravno, mogućnost dohvata podataka čija je dužina neodređena treba biti predviđena već u fazi oblikovanja sučelja. Objekti iteratori u pravilu su privremeni objekti, stoga je potreban oprez kod njihove uporabe kako ne bi došlo do njihova gomilanja na strani poslužitelja i gubitka raspoložive radne memorije.

Postoje dvije vrste objekata iteratora:

- iteratori temeljeni na ”guranju” podataka (eng. *push iterators*)
- iteratori temeljeni na ”povlačenju” podataka (eng. *pull iterators*).

Najčešća se koristi implementacija iteratora temeljenih na “povlačenju“ podataka od strane korisnika udaljenog objekta. Privremeni objekti iteratori stvaraju se na strani poslužitelja, a korisnik uzastopnim pozivima metode “dovlači“ podatke u segmentima određene dužine. Implementacija iteratora temeljenih na “guranju“ podataka rjeđa je jer zahtijeva kreiranje objekata iteratora na strani korisnika, a što u većem broju slučajeva uzrokuje komplikacije u implementaciji korisničke strane raspodijeljene aplikacije.

3.3.6. Višestruka preusmjeravanja poziva

CORBA omogućuje transparentno preusmjeravanje poziva metoda jednog udaljenog objekta na drugi. U protokolu GIOP u tu svrhu definirane su dvije poruke:

- `LOCATION_FORWARD`
- `LOCATION_FORWARD_PERMANENT`

koje poslužitelj zajedno s referencom na novi objekt može vratiti procesu korisniku kao odgovor na poziv jedne od metoda. Posrednička komponenta (ORB) u slučaju primanja navedenih poruka ponavlja poziv metode transparentno s obzirom na kod aplikacije (kod aplikacije ne dobiva nikakvu poruku o preusmjeravanju poziva), ali korištenjem vraćene reference na drugi objekt. Razlika između dviju navedenih poruka je sljedeća:

- Preusmjeravanje poziva korištenjem poruke `LOCATION_FORWARD` je jednokratno. Svaki slijedeći poziv metode istog objekta obavit će se korištenjem originalne reference ciljnog objekta.

- Preusmjeravanje poziva korištenjem poruke `LOCATION_FORWARD_PERMANENT` vrijedi do kraja životnog vijeka procesa korisnika. Svaki slijedeći poziv metode istog objekta obavit će se korištenjem reference na objekt na koji je poziv preusmjeren.

Problem prilikom preusmjeravanja poziva je u tome što svaki zahtjev za pozivom metode udaljenog objekta nosi sa sobom i sve `in` i `inout` parametre poziva. Ako je duljina prenošenih podataka velika, dolazi do nepotrebnog opterećenja mreže i gubitka vremena za formiranje poziva. U boljim implementacijama sustava CORBA posrednička komponenta prilikom prvog poziva jedne od metoda ciljnog objekta šalje GIOP poruku `LOCATE_REQUEST`, u kojoj nisu uključeni parametri poziva. Nakon dobivenog odgovora s pridruženom referencom ciljnog objekta (originalnog ili objekta na kojeg se poziv preusmjerava) šalje se zahtjev za pozivom metode s uključenim parametrima.

Preusmjeravanje poziva postaje problem u raspodijeljenom sustavu u sljedećim slučajevima:

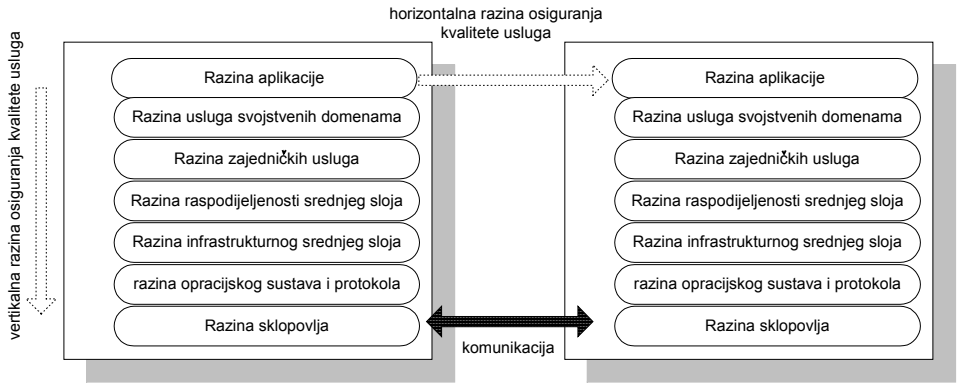
- čestim preusmjeravanjima poziva metoda s velikom duljinom `in` i `inout` parametara
- trajnim preusmjeravanjem poziva korištenjem `LOCATION_FORWARD` poruka umjesto `LOCATION_FORWARD_PERMANENT`
- ulančavanjem preusmjeravanja poziva (čest slučaj kod poziva metoda objekata koji su nekoliko puta promijenili domenu aplikacije).

3.3.7. Zrnatost sučelja objekata

Oblikovanje sučelja udaljenih objekata na istim principima kao i lokalnih objekata (npr. objekata u jeziku C++) rezultira zrnatošću metoda neprimjerenom korištenju u raspodijeljenim sustavima. Potrebno vrijeme za izvršavanje poziva metode udaljenog objekta duže je otprilike za dva reda veličine u odnosu na izvršenje poziva metode lokalnog objekta. Ovaj omjer ima tendenciju povećanja s obzirom na rast procesorske snage i propusnosti računalnih mreža. Stoga, nadomještanjem velikog broja poziva jednostavnih operacija udaljenog objekta s manjim brojem složenijih operacija postižu se znatne uštede u vremenu izvođenja.

3.4. Dinamičke promjene u okolini raspodijeljenog sustava

CORBA, zaključno s inačicom standarda 2.3, predstavlja stabilno okruženje za razvoj i implementaciju objektno usmjerenih raspodijeljenih sustava. No ono što manjka ugrađeni su mehanizmi transparentnog osiguranja kvalitete pojedinih usluga (eng. *Quality of Service*) i, naglašeno, mehanizmi prilagodljivosti na promjene u radnom okruženju. Glavni zahtjevi na sustav koje nova generacija raspodijeljenih aplikacija zahtijeva predvidljive su performanse, sigurna komunikacija i otpornost na pogreške u radu. Današnja rješenja navedenih problema ad hoc su rješenja, svojstvena pojedinim implementacijama arhitekture i izgrađenim aplikacijama. Stoga ona ne zadovoljavaju uvjete prenosivosti i standardiziranosti kao najznačajnijih odlika CORBA-e. Tek od inačice standarda CORBA 2.4 uvode se mehanizmi kvalitete usluga, prvenstveno zastupljeni u obliku mehanizma asinkrone komunikacije i podrške radu u stvarnom vremenu.



Slika 3.1. Razine raspodijeljene aplikacije sustava CORBA i razine osiguranja kvalitete usluga

Sustav temeljen na CORBA-i može se prikazati u sedam osnovnih slojeva (slika 3.1) [28]. U svrhu osiguranja željene kvalitete usluge na razini aplikacije potrebno je osigurati odgovarajuću razinu kvaliteta usluga na svim navedenim razinama CORBA temeljenog sustava. Stoga razlikujemo dvije razine osiguranja željenih kvaliteta usluga:

- horizontalnu razinu osiguranja kvalitete usluga
- vertikalnu razinu osiguranja kvalitete usluga.

Horizontalna razina osiguranja usluga odnosi se na razinu aplikacije, odnosno svih korištenih komponenata raspodijeljenog sustava u svrhu obavljanja zadaće s potrebnom razinom usluge. Parametri kvalitete usluga vezani su uz funkcionalnost aplikacije, a mogu biti zajednički na razini CORBA-e (prioriteti izvršavanja poziva ili

poruka, garancije izvršavanja, itd.) ili aplikacijsko specifični parametri. Kvalitetu usluge temeljene na parametrima zajedničkim na razini CORBA-e dužno je osigurati korištenjem vertikalnih mehanizama unutar implementacije arhitekture (unutar posredničke komponente sustava), a transparentno s obzirom na programski kod aplikacije. Osiguranje aplikacijsko specifičnih parametara kvalitete usluge zadaća je pojedinih komponenti raspodijeljene aplikacije.

Svojstveno horizontalnoj razini osiguranja kvalitete usluge od točke do točke (eng. *point-to-point*) korištenje je standardnih mehanizama komunikacije u sustavima CORBA-e. Osiguranje potrebne kvalitete usluge u većini se slučajeva svodi na:

- pronalaženje objekta koji ispunjava postavljene kriterije kvalitete usluge
- pregovaranje s objektom o potrebnoj razini kvalitete usluge
- deklariranjem željene kvalitete usluge prilikom poziva metode udaljenog objekta.

Pronalaženje objekta koji ispunjava postavljene kriterije kvalitete usluge obavlja se korištenjem posredničkih komponenti unutar raspodijeljenog sustava. Implementacija posredničke komponente može biti ili aplikacijski specifična ili je moguće korištenje gotovih komponenti, npr. trgovačke usluge. Na osnovu upita u kojem su navedene željene vrijednosti parametara kvaliteta usluga, trgovačka usluga vraća listu referenci na postojeće objekte koji te kriterije zadovoljavaju. S druge strane, objekti nudeći svoje usluge prijavljuju se trgovačkoj usluzi navodeći vrijednosti parametara kvalitete usluga koje mogu zadovoljiti. Alternativan način korištenje je mehanizma stvaranja objekata - instanci usluga sa željenim vrijednostima parametara kvalitete. Uobičajen mehanizam stvaranja objekata i dobavljanje njihove reference je uzorak tvornice objekata (eng. *factory pattern*) [8]. U parametrima poziva metode objekta tvornice navode se, među ostalim podacima, željene vrijednosti parametara kvalitete usluga. Ukoliko objekt tvornica može stvoriti objekt koji zadovoljava navedene parametre, kao rezultat poziva metode vraća se referenca na novostvoreni objekt. Primjer korištenja ovog mehanizma dobavljanje je objekata zastupnika u usluzi poruka (eng. *Notification Service*) [30].

Pregovori s objektom o razini kvalitete usluga odvijaju se većinom jednostrano, na strani korisnika objekta. Svi podaci o mogućnostima kvalitete pružene usluge sadržani su u referenci udaljenog objekta, a temelje se na politikama (eng. *policies*) definiranim na razini grupe objekata pridruženih zajedničkom prilagodniku objekata. Korisnik može mijenjati politike svojstvene pojedinoj referenci na tri razine: razini same reference, razini izvršne niti (objekt razreda *Current* pridružen svakoj izvršnoj niti) i razini posredničke komponente. Podložne promjeni su samo one politike s utjecajem isključivo na strani korisnika objekta, promjene se ne prenose na stranu korištenog objekta. Primjer uporabe politika u definiranju željenih kvaliteta usluga su

poruke CORBA-e (eng. *CORBA Messaging*) i sustavi CORBA prilagođeni za rad u stvarnom vremenu (eng. *Real-Time CORBA*) [1].

Posljednji često korišten način deklariranja željene kvalitete usluge navođenje je razine kvalitete kao parametra ili dijela parametra poziva metode. Kao primjer može se navesti deklariranje prioriteta strukturirane poruke u usluzi poruka kao sastavnog dijela same poruke. Ovako definirani zahtjevi u pravilu se odnose na aplikacijski specifične kvalitete usluge, neovisne o mehanizmima nižih slojeva (mrežne komunikacije, kvalitete usluga na razini operacijskog sustava, itd.).

Treba primijetiti da horizontalna kvaliteta usluge ne mora biti eksplicitno navedena, npr. u obliku parametra poziva metode udaljenog objekta, niti njeno korištenje pokrenuto (implicitno ili eksplicitno) od strane korisnika usluge. Kao primjer ove tvrdnje može se navesti mehanizam raspodjele opterećenja između više računala poslužitelja (eng. *load balancing*). U ovom slučaju osiguranje određene razine kvalitete usluge osigurava se isključivo na strani poslužitelja, korištenjem mehanizama koji ne moraju nužno činiti sastavni dio implementacije raspodijeljene aplikacije. Zadaću preusmjeravanja poziva metoda udaljenih objekata mogu obavljati specijalizirane implementacije posredničkih komponenti ili mjesta za pohranu implementacija. Parametri kvalitete usluge ne navode se po pozivu metode ili korištenju objekta u toku njegova životnog vijeka, već se određuju na razini performansi dijela sustava ili sustava u cjelini. Oblik izražavanja parametara kvalitete usluga ne mora biti deklarativan (putem varijabli), već definiran npr. skupom pravila u obliku programskog koda komponente raspodijeljenog sustava koja nadgleda i usmjerava njegov rad.

Vertikalna razina osiguranja kvalitete usluga odnosi se na korištenje dostupnih mehanizama osiguranja kvalitete usluge na pojedinom čvoru raspodijeljenog sustava (računalu na kojem se komponenta raspodijeljene aplikacije izvršava). Zadaća je posredničke komponente ili komponente aplikacije, ukoliko je u pitanju aplikacijsko specifična kvaliteta usluge, osigurati kvalitetu usluge definiranu na razini kvalitete usluge aplikacije, korištenjem mehanizama osiguranja kvalitete usluga dostupnih resursa. Dostupni mehanizmi osiguranja razine usluge mogu biti zajednički pojedinoj vrsti resursa ili ovisni o proizvođaču opreme (kako programske podrške tako i sklopovlja). Kao primjer različitih sučelja prema istoj vrsti resursa mogu se navesti aplikacijska programska sučelja za rad s nitima. Razlikujemo rad s nitima na različitim inačicama UNIX operacijskih sustava (standard IEEE POSIX 1003.1-1996) i Microsoft Windows operacijskom sustavu. Razlike u radu s resursima uočljive na razini operacijskog sustava moguće je sakriti na razini infrastrukturnog srednjeg sloja stavljanjem na raspolaganje višim slojevima jedinstvenog sučelja prema resursima koje ciljna platforma nudi. Dva su često korištena infrastrukturna srednja sloja u raspodijeljenim aplikacijama: Java platforma i sustav ACE (eng. *Advanced*

Communication Environment). Java osigurava jedinstveno sučelje prema resursima ciljnog računala bez obzira na njegovu arhitekturu i korišteni operacijski sustav. ACE [31] nudi platformno neovisno sučelje za mrežnu komunikaciju korištenjem sustava objekata i uzoraka. Naravno, za korištenje infrastrukturnog srednjeg sloja potrebno je platiti određenu cijenu, većinom u performansama izvođenja aplikacije i odustajanju od korištenja naprednijih mogućnosti svojstvenih pojedinom resursu ili operacijskom sustavu, a koje niži sloj nudi.

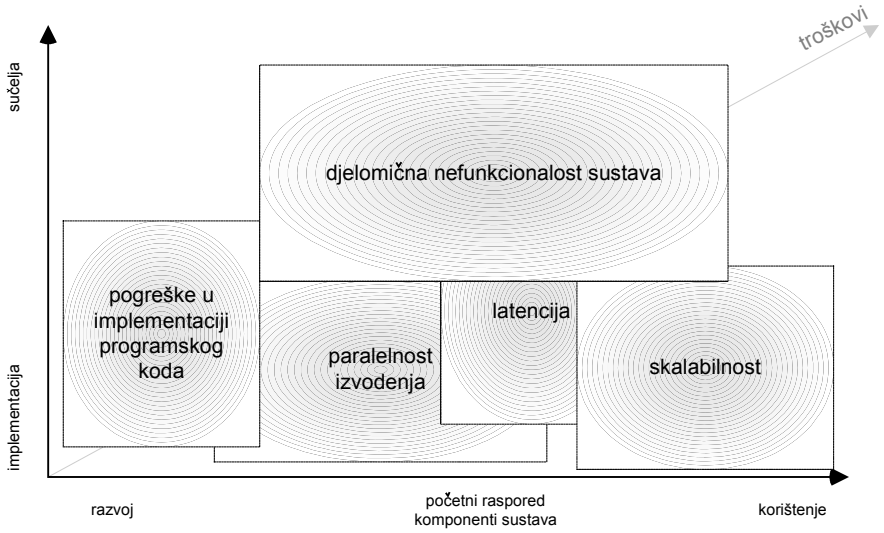
Prilagodljiv raspodijeljeni sustav mora na raspolaganju imati, osim mehanizama osiguranja kvalitete usluga dostupnih resursa, i mehanizme reakcije na promjene u kvaliteti usluga, tj. na promjene u njegovu radnu okruženju. Informacije o promjenama u okolini mogu potjecati iz nekog od slojeva raspodijeljenog sustava (npr. obavijest o smanjenju propusnosti mreže poslana od strane infrastrukturnog srednjeg sloja) ili od entiteta van raspodijeljene aplikacije (npr. nadglednog sustava). Izazvane promjene u raspodijeljenom sustavu s obzirom na transparentnost promjene dijele se na:

- izravne promjene u radu sustava
- neizravne promjene u radu sustava.

Izravne promjene očituju se na aplikacijskoj razini raspodijeljenog sustava, stoga moraju biti predviđene u fazama oblikovanja i implementacije. Mehanizmi izravnih promjena najčešće su implementirani u obliku poziva metoda objekata raspodijeljenog sustava, putem kojih se prenose informacije o novom stanju ili dostupnosti kvalitete usluga. U ovisnosti o predočenim informacijama, raspodijeljeni sustav vrši prilagodbu novonastalom stanju u njegovoj okolini. Mehanizam izravnih promjena snažno je vezan uz konkretne implementacije sustava, te je njegova izvedba strogo namjenska, onemogućujući njegovu široku uporabu i ponovnu iskoristivost. Kao primjer uporabe izravnih mehanizama prilagodbe rada raspodijeljenog sustava možemo razmotriti sustav za prijenos toka video podataka u stvarnom vremenu. Promjene u propusnosti mreže između korisnika i poslužitelja (izvora toka) moraju za posljedicu imati smanjenje količine prenošenih podataka. Količina prenošenih podataka može se smanjiti korekcijom kvalitete prenošene slike, npr. broja okvira, broja boja, razlučivosti slike, itd. Korekcija ovih parametara kvalitete usluge izravno se očituje u radu algoritma za kodiranje slike, tj. u entitetu definiranom na aplikacijskoj razini.

Neizravne promjene nemaju kao posljedicu promjene u radu entiteta sustava na aplikacijskoj razini. Potrebne prilagodbe vrše se transparentno, na nižim razinama. Karakteristično je za neizravne promjene u radu sustava da su u većini slučajeva općenite, nevezane za specifičnu aplikaciju. Kao primjer neizravne promjene može se navesti preusmjeravanje poziva metode na drugi objekt, npr. u raspodjeli opterećenja

sustava. Pozivajući entitet ne može uočiti različitost objekata čija se metoda poziva jer se preusmjeravanje poziva obavlja unutar posredničke komponente, bez učešća viših razina CORBA-e. Zanimljiv način neizravne implementacije raspodijeljenog sustava promjena je strukture posredničke komponente u toku rada aplikacije. Ovo svojstvo specifičnost je pojedine implementacije, najpoznatija inačica sa svojstvom promjene strukture je dynamicTAO [13] implementacija sustava CORBA.



Slika 3.2. Grupe pogrešaka u raspodijeljenim aplikacijama sustava CORBA

Slika 3.2 shematski prikazuje osnovne grupe pogrešaka u raspodijeljenim aplikacijama sustava CORBA s obzirom na prirodu (greška u implementaciji ili oblikovanju), fazu razvoja aplikacije u kojoj dolaze do izražaja i troškovima potrebnim za njihovo ispravljanje.

4. Nadgledanje i upravljanje raspodijeljenim računalnim sustavima

Programski proizvodi dijele se [14] na proizvode opće i posebne namjene.

Programski proizvodi opće namjene namijenjeni su širokom krugu korisnika u svrhu obavljanja općenitih zadataka iz širokog spektra domena. Pod proizvodima opće namjene ne podrazumijevaju se samo cjeloviti programski proizvodi (npr. programi za obradu teksta), već i gotove programske komponente namijenjene ugradnji u druge programske proizvode (kako opće tako i namjenske). Najčešći oblici gotovih programskih komponenti programske su biblioteke i objektne komponente uključene u programske proizvode u fazi povezivanja (bilo statičkog u tijeku faze razvoja, bilo dinamičkog u tijeku izvršavanja samog programa).

Ukoliko posebnosti domene problema ne omogućuju, uslijed funkcionalnih ili nefunkcionalnih zahtjeva, uporabu programskih rješenja opće namjene, nužan je razvoj namjenskih programskih proizvoda. Namjenski proizvodi oblikovani su sukladno zahtjevima domene problema, onemogućavajući time njihovu širu primjenu.

Snažan rast potražnje za programskim rješenjima u prethodnim godinama, te nemogućnost programske industrije da zadovolji potražnju doveo je do tzv. “krize programske podrške” (eng. *software crisis*) [2]. Odgovor programske industrije u cilju povećanja produktivnosti rada izrade programskih proizvoda zbio se na četiri plana:

- metodologijama oblikovanja programskih sustava
- korištenju novih programskih jezika
- korištenju alata i okruženja za brz razvoj programskih proizvoda
- višestrukom korištenju jednom razvijenog koda.

Proces izrade programskog proizvoda sastoji se od četiri osnovne aktivnosti zajedničke svim procesima [14]:

- opis proizvoda
- izrada proizvoda
- ispitivanje proizvoda
- evolucija gotovog proizvoda.

Aktivnosti vezane uz izradu (kodiranje) i ispitivanje proizvoda (osim završnog ispitivanja prilikom preuzimanja proizvoda od strane naručitelja) snažno su

isprepletene, slijedeći evolucijski model razvoja. Po istraživanjima, faza ispitivanja (uključujući i fazu izrade u kojoj se ispravljaju uočene pogreške) sudjeluje u ukupnom vremenu razvoja s, okvirno, 50% utrošenog vremena. Kod zahtjevnijih sustava, s posebnim zahtjevima na sustav ili s otežanim prepoznavanjem i određivanjem uzroka pogrešnog rada, faza ispitivanja sudjeluje u ukupnom vremenu razvoja čak do 70%. Primjeri sustava u kojima postoji veliki stupanj nedeterminizma ili je otežano ispitivanje su raspodijeljeni sustavi, sustavi za rad u stvarnom vremenu, te, naročito, raspodijeljeni sustavi za rad u stvarnom vremenu [15].

Faza ispitivanja programskog proizvoda konvencionalne arhitekture (centraliziranih aplikacija) podržana je nizom alata za pronalaženje i ispravljanje pogrešaka, te statičku (analizatori izvornog koda) i dinamičku analizu (ponašanja tijekom rada). Centraliziranost svih sastavnih komponenti razvijane aplikacije (programskog koda i korištenih gotovih komponenti), korištenih resursa lokalnog računala i predvidivost izvršavanja programa ovu fazu razvoja sustava čine bitno jednostavnijom u usporedbi s raspodijeljenim sustavima.

Posljedica sljedećih karakteristika svojstvenih raspodijeljenim sustavima čini njihovo ispitivanje znatno složenijim [15]:

- kontinuirano izvršavanje
- asinkrona komunikacija između raspodijeljenih procesa
- nepredvidljivost kašnjenja u komunikaciji i utrke za dijeljenim resursima
- neodređenost sustava i neponovljivost izvršavanja
- problem globalnog sata raspodijeljenog sustava
- višestruke niti interakcije među raspodijeljenim procesima.

Konvencionalni alati za nalaženje pogrešaka i analizu rada korišteni u razvoju neraspodijeljenih aplikacija ne predstavljaju cjelovito rješenje u fazi analize i ispravljanja raspodijeljenih sustava. Njihova funkcionalnost ograničena je na pojedine komponente sustava, te je njihova uporaba moguća (i poželjna) tek nakon što su uočene komponente raspodijeljenog sustava koje ne zadovoljavaju pred njih postavljene funkcionalne ili nefunkcionalne uvjete.

Za analizu rada raspodijeljenog sustava u cjelini, u svrhu dobivanja cjelovite slike o sustavu i pronalaženje pogrešaka u radu, nužno je korištenje specijaliziranih nadglednih alata. Nadgledni sustavi moraju alatima (ili korisniku) pružiti dovoljno informacija prvenstveno o međudjelovanju entiteta raspodijeljenog sustava (učestalost poziva, vrijeme izvođenje poziva, podaci razmjenjivani u pozivima, itd.).

Za razliku od faze izrade, u kojoj je najveća pozornost usmjerena na logičku ispravnost rada, u fazama početnog rasporeda komponenti (eng. *deployment*) i korištenja pozornost je usmjerena na performanse, dostupnost korisnicima i razna pitanja sigurnosti sustava. Ispravnost rada raspodijeljene aplikacije ne ovisi samo o ispravnosti rada svake pojedine komponente sustava zasebno, već i o ispravnosti njihova međudjelovanja. Utjecaj izvršne okoline, koja se u raspodijeljenom sustavu može razlikovati za svaku pojedinu komponentu, jedan je od najvažnijih čimbenika za ispravnost rada.

Različitost radnih okolina između faze razvoja i faze korištenja aplikacije najveći je izvor pogrešaka uočenih tek u fazi početnog rasporeda komponenata raspodijeljene aplikacije. Dva su načina rješavanja uočenih problema u fazi početnog rasporeda:

- intervencije u implementaciji sustava
- prilagodljivost sustava u fazi početnog rasporeda.

Intervencija u implementaciji sustava (promjena organizacije sustava i ispravljanje programskog koda) najlošija je varijanta, te se primjenjuje samo ukoliko je nužno. Daleko je prihvatljivije uključivanje mehanizama prilagodbe sustava radnom okruženju već od faze oblikovanja i implementacije. Implementacije mehanizma mogu varirati, od zadavanja parametara podešavanja u trenutku pokretanja dijelova sustava, korištenjem datoteka za podešavanje, do samostalnog prilagođavanja aplikacije na osnovu prikupljenih podataka o radnoj okolini i probnim izvođenjima.

Dinamičke promjene radne okoline (prvenstveno raspoloživosti resursa, npr. opterećenje procesora, raspoloživa količina radne memorije, raspoloživog diskovnog prostora, propusnost računalne mreže, itd.) također mogu bitno utjecati na rad raspodijeljenog sustava. Reakcija raspodijeljene aplikacije na dinamičke promjene u radnoj okolini mora biti moguća bez potrebe za prekidanjem rada sustava. Stoga mehanizmi prilagodljivosti moraju biti uključeni u sustav već od faze njegova oblikovanja.

Mehanizam prilagodljivosti raspodijeljenog sustava (bez obzira na način njegove implementacije) mora odgovoriti na tri osnovna pitanja:

- čemu se prilagoditi ?
- kako se prilagoditi ?
- kada se prilagoditi ?

Kako se prilagoditi rješava implementirani mehanizam prilagodbe, svojstven arhitekturi i namjeni raspodijeljenog sustava. Čemu se i kada prilagoditi sustav mora zaključiti na osnovu informacija o sebi (o svim ili samo podskupu entiteta koji čine

raspodijeljeni sustav) i radnoj okolini komponenata. Zadaća je nadglednog sustava učinkovito osigurati potrebne informacije o radu raspodijeljenog sustava, bez obzira na fizičku raspodijeljenost njegovih sastavnih komponenti.

Osnovne izvedbe nadgledanja raspodijeljenih sustava dijele se na:

- sklopovske (specijalizirani sklopovski moduli ugrađeni unutar raspodijeljenog sustava)
- programske (programski moduli ili dijelovi koda dodani sustavu)
- hibridne (kombinacija posebnog sklopovlja i programske podrške nadgledanju).

Sklopovske i hibridne izvedbe nadgledanja koriste se gotovo isključivo u nadgledanju specijaliziranih raspodijeljenih sustava građenih od za tu namjenu posebno razvijenog sklopovlja i programske podrške (različiti ugrađeni raspodijeljeni sustavi za rad u stvarnom vremenu). Spomenuti sustavi oslanjaju se na pasivno promatranje nadgledanog sustava (nadgledanjem komunikacijskih kanala: sabirnica, signalnih linija, itd.) te ne utječu na njegov rad.

Za razliku od prije opisanih izvedaba, programska rješenja čine dio promatranog sustava (dio su programskog koda nadgledanog sustava, koriste iste resurse kao i nadgledani sustav), a time neminovno utječu na njegov rad. Programske izvedbe nadgledanja uglavnom se koriste na računalnim sustavima opće namjene (osobna računala, radne stanice) za nadgledanje programskih sustava kod kojih ne postoje čvrsti zahtjevi na performanse sustava (za razliku od npr. sustava za rad u stvarnom vremenu). Programska rješenja nadgledanja jedina su od interesa u području nadgledanja raspodijeljenih aplikacija.

Ciljevi postupaka nadgledanja programskih raspodijeljenih sustava su:

- prikupiti što više informacija o sustavu
- što manje utjecati na rad nadgledanog sustava.

Utjecaj postupka nadgledanja na sustav bit će to manja ukoliko se prilikom oblikovanja sustava nadgledanja obratilo pozornost na četiri bitna čimbenika:

- transparentnost nadgledanja
- smanjenje i predviđanje utjecaja na rad sustava
- zahtjeve na potrebne resurse nadglednog sustava
- smanjenje međuovisnosti nadglednog i nadgledanog sustava.

Transparentnost nadgledanja podrazumijeva nepostojanje ili vrlo malu potrebu za intervencijama u nadgledanom sustavu od strane korisnika nadglednog sustava. Programski kod potreban za rad nadglednog sustava (pretpostavljamo da se radi o programskoj izvedbi sustava nadgledanja) mora se moći dodati u nadgledani sustav automatski, uz pomoć gotovih alata, a bez potrebe za intervencijom programera kako bi se što više smanjila mogućnost unošenja pogrešaka, većinom nastalih ljudskim faktorom. Podešavanje i upravljanje nadgledanjem potrebno je izvesti korištenjem alata, preporučljivo s prijateljskim i intuitivnim korisničkim sučeljem.

Kao posljedica utjecaja nadglednog sustava na ciljni nadgledani raspodijeljeni sustav može se javiti promjena ponašanja nadgledanog sustava. Promjena ponašanja može sezati od promjena koje ne moraju imati veliki utjecaj na ispravnost rada sustava (uvođenje malog kašnjenja u rad sustava) do promjena koje potpuno onemogućuju ispravan rad sustava (promjena redoslijeda događaja u sustavu, blokiranje rada sustava, pojavu "izgladnjivanja" procesa u sustavu). Stoga je potrebno iznaći načine implementacije nadglednog sustava koji će, ako već nije moguće izbjeći njegov utjecaj, imati stalan i (ili) predvidljiv utjecaj, te da taj utjecaj neće bitno mijenjati ponašanje nadgledanog sustava.

Čak i ako su prva dva uvjeta zadovoljena, nadgledni sustav može se pokazati neuporabljivim ukoliko nije riješeno pitanje zadovoljenja potreba za resursima nužnim za njegov nesmetan rad. Kao primjer možemo navesti potrebnu količinu radne memorije (i sekundarne, trajne memorije) potrebne za pohranu podataka kao rezultata nadgledanja.

Zadnje navedeni čimbenik na tragu je koncepta ponovne iskoristivosti koda: ukoliko je moguće, preporučljivo je izraditi nadgledni sustav takav da je moguća njegova iskoristivost u različitim programskim proizvodima. Time se dobiva na smanjenju potrebnog vremena za traženje i ispravljanje pogrešaka, te na mogućnosti razvoja i uporabe standardiziranih alata za analizu i predočavanje podataka prikupljenih nadglednim sustavima.

Tri su osnovna načina implementacije programskih sustava za nadgledanje raspodijeljenih računalnih sustava:

- sustavi temeljeni na prozivanju nadgledanih entiteta raspodijeljenog sustava
- sustavi temeljeni na prikazu podataka o nadgledanim entitetima korištenjem varijabli stanja i logičkih spremišta podataka
- sustavi temeljeni na promatranju događaja unutar raspodijeljenog sustava.

Sustavi temeljeni na prozivanju nadgledanih entiteta najjednostavniji su oblik implementacije nadglednih sustava. Prikupljanje podataka o trenutnom stanju sustava

vrši se izravnom komunikacijom nadglednog sustava sa svim entitetima (ili podskupom entiteta) koji čine raspodijeljeni sustav. Mehanizam komunikacije korišten za komunikaciju nije od velike važnosti, uglavnom se koristi onaj mehanizam koji je u uporabi za komunikaciju među entitetima raspodijeljenog sustava (poruke, pozivi udaljenih procedura, pozivi metoda udaljenih objekata, itd.). Promotrimo kako sustavi temeljeni na prozivanju zadovoljavaju uvjete postavljene pred nadgledne sustave:

- *Transparentnost nadgledanja*: nužna je izmjena koda promatranog entiteta u svrhu prikupljanje podataka o stanjima unutar entiteta i omogućavanje komunikacije s nadglednim sustavom. Time se izravno mijenja programski kod sustava, a što nije naročito preporučljivo.

- *Smanjenje i predviđanje utjecaja na rad sustava*: asinkronost svojstvena raspodijeljenim sustavima onemogućava predviđanje utjecaja nadgledanja kod opisane implementacije nadglednih sustava. Zahtjev za informacijama od strane nadglednog sustava može pristići u bilo kojem trenutku, imajući za posljedicu utjecaj na rad promatranog sustava (trošenje vremena procesora za obradu zahtjeva, itd.). Problem se naglašava u slučaju postojanja više nadglednih sustava zainteresiranih za informacije o pojedinom nadgledanom entitetu.

- *Zahtjevi na potrebne resurse nadglednog sustava*: nisu od velikog značaja.

- *Međuovisnost sustava za nadgledanje i nadgledanog sustava*: međuovisnost implementacija je naglašena, nadgledni sustav mora poznavati:

    - lokacije nadgledanih entiteta (mrežne adrese)
    - tipove podataka kojima se opisuju stanja
    - internu strukturu entiteta u svrhu dohvata i tumačenja stanja entiteta.

Jedina prednost sustava temeljenih na prozivanju jednostavnost je implementacije. Njihovo korištenje preporučuje se samo u jednostavnijim slučajevima nadgledanja u kojima snažna međuovisnost između implementacija nadgledanog i nadglednog sustava ne predstavlja problem, kao ni nepredvidljiv utjecaj na rad nadgledanog sustava.

Sustavi temeljeni na prikazu podataka o nadgledanim entitetima korištenjem varijabli stanja i logičkog spremišta podataka u domenu nadgledanja raspodijeljenih aplikacija djelomično su preuzeti iz domene nadgledanja i upravljanja resursima računalnih

mreža. Struktura ovih sustava vrlo je slična sustavima nadgledanja temeljenim na protokolu SNMP (eng. *Simple Network Management Protocol*):

- stanje nadgledanog entiteta predstavlja se u obliku skupa varijabli
- varijable su određene njihovim imenom, tipom podataka koji sadrže (jednostavnim ili složenim) i tipom varijable (dozvoljeno samo čitanje ili čitanje i pisanje)
- varijable se pohranjuju u bazu podataka upravljanja (eng. *Management Information Base – MIB*) koja služi kao posrednička komponenta prema zainteresiranim korisnicima informacija o nadgledanim entitetima raspodijeljenog sustava
- zainteresirane aplikacije dohvaćaju vrijednosti varijabli iz baze podataka upravljanja i (ili) postavljaju njihove vrijednosti u svrhu upravljanja radom aplikacije, moguća je i pretplata na obavijesti o promjeni vrijednosti varijable unutar baze podataka upravljanja
- komunikacija s bazom podataka upravljanja odvija se korištenjem standardnog i široko korištenog protokola SNMP.

Transparentnost nadgledanja u ovoj izvedbi nadgledanja ne postoji uslijed potrebe da nadgledani entitet eksplicitno oglašava vrijednosti svojih varijabli u bazi podataka upravljanja. Oglašavanje vrijednosti varijabli i njihovo dohvaćanje u većini se slučajeva iz nadgledanog entiteta izvodi pozivima funkcija `get()` i `set()` sadržanih u biblioteci koda povezanoj s kodom entiteta u fazi razvoja sustava. No netransparentnost nadgledanja u većini slučajeva korištenja prethodno opisanog načina nadgledanja i nije veliki nedostatak jer se takvi sustavi oblikuju i izvode podrazumijevajući utjecaj sustava za nadgledanje i upravljanje. Nadgledni sustav čini dio raspodijeljenog sustava od trenutka oblikovanja pa nadalje, a ne naknadno dodani dio sustava. Samim time ovakvi sustavi zadovoljavaju uvjet smanjenja i predviđanja utjecaja nadglednog dijela sustava na rad cjelokupnog raspodijeljenog sustava.

Nadgledani entitet i ostali dijelovi nadglednog sustava odvojeni su bazom podataka upravljanja. Time je nepovoljan odnos između nadgledanog entiteta i zainteresiranih nadglednih aplikacija, koji je u sustavima temeljenim na prozivanju bio 1:n, prebačen na stranu baze podataka upravljanja. Odnos baze podataka upravljanja i nadgledanog entiteta postaje 1:1, što utjecaj nadgledanja (tj. komunikacije nadgledanog entiteta s bazom) čini predvidljivim. Odvojenost mjesta za pohranu vrijednosti varijabli čini jedinu bitnu razliku između standardnih implementacija SNMP sustava za

nadgledanje entiteta računalnih mreža i nadgledanja programskih raspodijeljenih sustava.

Pokretač komunikacije nadgledanog entiteta i baze podataka upravljanja (općenito gledajući - sustava nadgledanja) je, u najvećem broju slučajeva, nadgledani entitet, što omogućuje kontrolu utjecaja nadgledanja na rad sustava. Sustav nadgledanja kao inicijator komunikacije pojavljuje se samo u slučaju slanja obavijesti o promjeni vrijednosti varijabli u bazi podataka upravljanja.

Količina podataka potrebna za opis stanja svih entiteta sustava, naročito ako je broj entiteta od kojih se raspodijeljeni sustav sastoji vrlo velik, može predstavljati problem. No glavnina tog problema rješava se kroz odgovarajuću implementaciju baze podataka upravljanja, a ne kroz implementaciju nadgledanih entiteta.

Međuovisnost nadgledanog sustava i nadglednih sustava očituje se samo u:

- tumačenju informacija pohranjenih u bazi podataka upravljanja
- lokacijskoj ovisnosti o bazi podataka upravljanja.

Navedeni problemi zadovoljavajuće su riješeni u mnogobrojnim postojećim izvedbama SNMP sustava. Velika prednost ovako izvedenih nadglednih sustava jednostavno je korištenje postojećih alata za analizu i predočavanje stanja sustava, te integraciju nadgledanja raspodijeljenih aplikacija unutar postojećih sustava za nadgledanje računalnih mreža, čime je moguće dobiti cjelovitu sliku stanja raspodijeljenih sustava i korištenih resursa na više razina (eng. *multi-layer monitoring*).

Sustavi temeljeni na promatranju događaja unutar raspodijeljenog sustava temelje se na prepoznavanju pojedinih prethodno definiranih događaja unutar entiteta raspodijeljenog sustava i slanju odgovarajućih poruka nadglednom sustavu. Treba primijetiti da u ovoj vrsti sustava ne postoji eksplicitno definirana posrednička komponenta između nadgledanog entiteta i nadglednog sustava. Nadgledani entitet (izvor događaja) pribavlja podskup događaja za čije nadgledanje je zainteresiran putem sustava pretplate na događaje najčešće definiranih korištenjem specijaliziranih jezika, ovisnih o implementaciji sustava prenošenja događaja u raspodijeljenom sustavu. Trenutne pretplate na događaje od strane nadglednih sustava mogu imati utjecaj na slanje poruka od strane nadgledanih entiteta, tj. nadgledani entiteti mogu imati informacije o pretplatama na događaje čiji su oni izvor.

Kao i u dva prethodno opisana mehanizma izvedbe nadglednih sustava, i u izvedbi temeljenoj na događajima nužno je unutar nadgledanih entiteta dodavanje programskog koda, čija je zadaća prepoznavanje promatranih događaja i slanje

obavijesti nadglednom sustavu. No, u pojedinim izvedbama raspodijeljenih aplikacija moguće je dodavanje koda bez mijenjanja postojećeg koda aplikacije (dodavanje koda na meta razini sustava), što mehanizam nadgledanja čini neovisnim o promatranom entitetu.

Komunikacija između nadgledanog entiteta i nadglednog sustava (konkretno, mehanizma prijenosa događaja) je, logički promatrano, potpuno asinkrona i u najvećem broju slučajeva inicirana od strane promatranog entiteta, što za posljedicu ima:

- potpunu odvojenost nadglednog sustava od nadgledanog entiteta
- minimalno kašnjenje u radu nadgledanog entiteta kao posljedica komunikacije s nadglednim sustavom
- neovisnost o performansama nadglednog sustava.

Zahtjevi na potrebne resurse kod sustava temeljenih na promatranju događaja mogu predstavljati značajan problem. Prilikom implementacije ovog tipa nadgledanja posebnu pozornost potrebno je obratiti na vjerojatan veliki broj generiranih događaja a koji može značajno utjecati na uporabljivost sustava. Broj događaja koje sustav treba obraditi ovisi o:

- broju i vrsti promatranih događaja
- učestalosti pojavljivanja događaja
- broju promatranih entiteta.

Posljedice velikog broja događaja očituju se i na strani promatranog entiteta (usporenje rada) i na strani nadglednog sustava (usporenje rada, zagušenost komunikacijskih kanala, potrebna količina memorije za prihvaćanje, obradu i trajnu pohranu podataka o događajima). Problem se rješava na sljedeće načine:

- promatranjem samo događaja od interesa korištenjem mehanizma pretplate
- promatranjem samo podskupa entiteta od interesa u raspodijeljenom sustavu
- generiranjem samo onih događaja unutar entiteta na koje postoji pretplata
- generiranjem složenih događaja kao rezultata pojavljivanja više međuovisnih jednostavnih događaja
- hijerarhijskom organizacijom komunikacijskih kanala
- načinima izvedbe entiteta uključenih u rad sustava.

Međuovisnost opisanih sustava očituje se u:

- značenju informacija sadržanih u poruci

- formatu poruka kojima se šalju obavijesti
- implementaciji komunikacijskih kanala.

Događaji u raspodijeljenom sustavu mogu se podijeliti u dvije grupe:

- događaji svojstveni arhitekturi promatranog raspodijeljenog sustava (npr. događaji svojstveni arhitekturama DCOM, JavaRMI, CORBA)
- događaji svojstveni pojedinom raspodijeljenom sustavu.

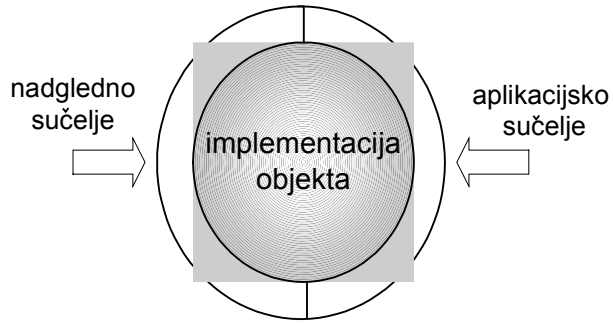
Promatranje događaja svojstvenih pojedinog arhitekturi moguće je korištenjem općenitih alata namijenjenih za tu arhitekturu (npr. interakcije entiteta unutar raspodijeljenog sustava, vremena trajanja obavljanja udaljenih poziva, itd.), dok je za promatranje događaja svojstvenih pojedinoj aplikaciji potrebno ili u tu svrhu izraditi namjenske alate ili izraditi dodatne module za postojeće općenite alate. Jedna od mogućnosti je i izrada prilagodnika na druge izvedbe sustava nadgledanja (npr. prilagodnik za sustave temeljene na SNMP protokolu) kako bi se dobavljene informacije o sustavu mogle uključiti u već postojeće nadgledne sustave.

Sustavi temeljeni na promatranju događaja u sustavu, za razliku od sustava temeljenih na korištenju varijabli stanja i baze podataka upravljanja, ne rješavaju istovremeno i mehanizam upravljanja radom raspodijeljenih sustava.

4.1. Pregled postojećih tehnologija

CORBA-Assistant [16] uvodi pojam upravljanog objekta u standardni sustav za nadgledanje resursa računalnih mreža, težeći ostvarenju zajedničkog upravljanja svim resursima raspodijeljenih računalnih sustava. Nadgledani objekti opisani su njihovim atributima, operacijama i događajima koje mogu generirati, što se uklapa u definiciju sučelja objekata sustava CORBA [1]. Svaki upravljani objekt implementira upravljačko sučelje čijim putem sustavu za upravljanje stavlja na raspolaganje svoje atribute, operacije i događaje. Time se postiže dvojnost pristupa objektu: jedno sučelje koristi se na razini aplikacije, a drugo na razini upravljanja sustavom (slika 4.1). Informacije koje pojedina vrsta upravljanog objekta putem svog upravljačkog sučelja čini dostupnim upravljačkim aplikacijama opisane su u tzv. CORBA MIB-u korištenjem jezika IDL. Upravljačke i posredničke komponente nadglednog sustava (upravljači, agenti, itd.), zajedno s dodatnim komponentama sustava (usluga registracije aktivnih objekata, itd.) također su realizirane korištenjem sustava CORBA, što znatno pojednostavljuje implementaciju i korištenje. Moguća je i implementacija prilagodnika sustava drugim protokolima za upravljanje mrežnim resursima, kao što su SNMP i CMIP. Konkretna implementacija sustava oslanja se na

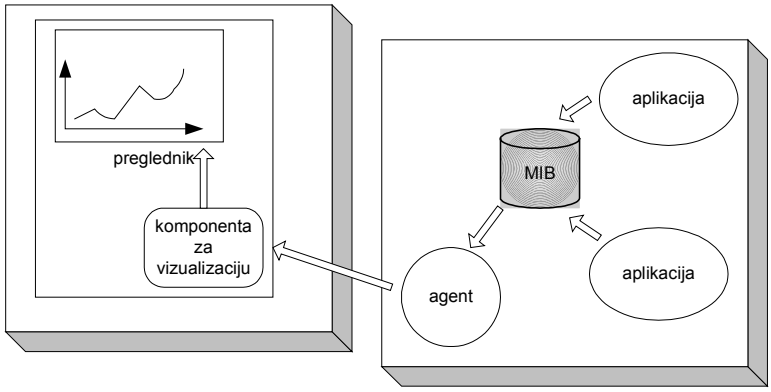
programski jezik C++ i Orbix implementaciju sustava CORBA (Iona Technologies), te koristi neke od njenih specifičnosti, što onemogućava prenosivost sustava.



Slika 4.1. Upravljani objekt sustava CORBA

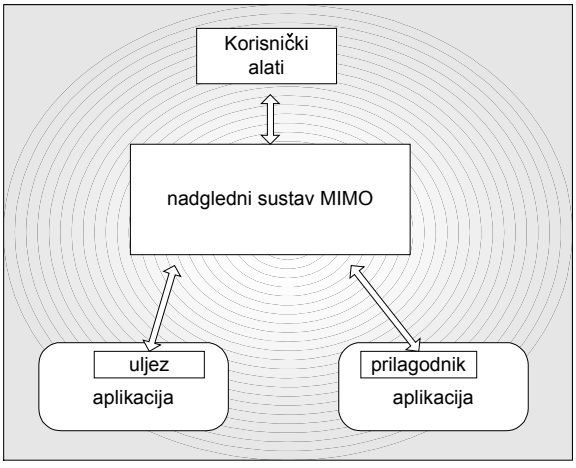
Projekt MAScOTTE [17] (eng. *Management Services for Object Oriented Distributed Systems*) nadogradnja je prije opisanog sustava CORBA-Assistant. Osnovni principi rada u oba su sustava isti, no MAScOTTE daje općeniti prikaz problema nadgledanja i građe sustava, bez naglaska na njegovoj konkretnoj implementaciji.

The Distributed Object Visualization Environment (DOVE) [20] nastoji stvoriti jedinstveno radno okruženje za nadgledanje i upravljanje raspodijeljenim aplikacijama, te za jednostavno kreiranje nadglednih i upravljačkih aplikacija korištenjem gotovih programskih komponenti JavaBeans. Sustav DOVE sastoji se od pet komponenti (slika 4.2): DOVE omogućene aplikacije koja svoje stanje korištenjem definiranog programskog sučelja pohranjuje u DOVE MIB, DOVE MIB kao komponentu koja sadrži stanja nadgledanih aplikacija, preglednika DOVE korištenjem kojega je omogućeno promatranje sadržaja DOVE MIB-a uz pomoć komponenata za vizualizaciju, komponente za vizualizaciju koja služi za komunikaciju i tumačenje informacija sadržanih u DOVE MIB-u, te agent DOVE koji je zadužen za oglašavanje usluga, slanje obavijesti o promjenama vrijednosti u DOVE MIB-u, obradu jednostavnih događaja i podešavanje vizualizacijskih komponenti. Sustav DOVE implementiran je korištenjem programskog jezika Java.



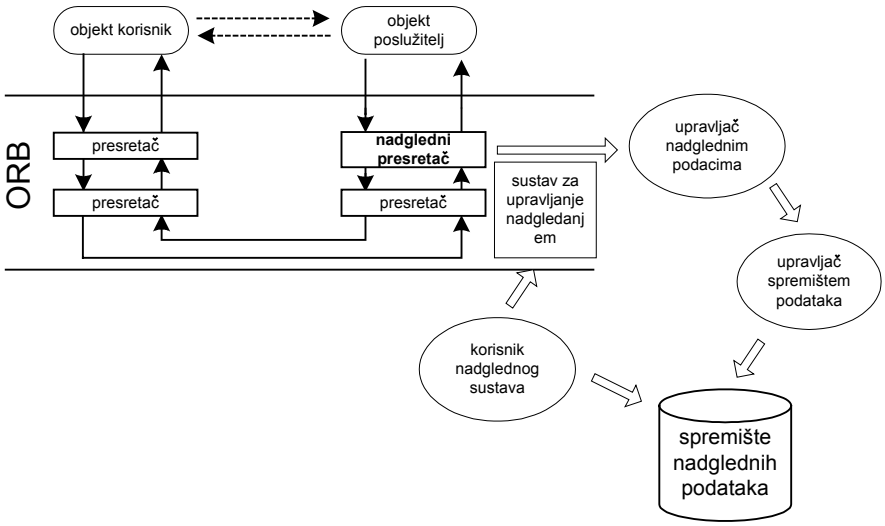
Slika 4.2. Organizacija okruženja DOVE

Sustav MIMO [18] [19] uvodi nadgledanje na svim razinama apstrakcije raspodijeljenog sustava (sklopovlje, izvršna okolina, implementacijska razina, razina raspodijeljenih objekata, razina sučelja i razina aplikacije). Nadgledanje pojedine razine implementirano je odgovarajućim programskim komponentama umetnutim na svakoj pojedinoj razini, a koje omogućavaju nadgledanje resursa i komunikaciju s nadglednim sustavom. Umetnute komponente za nadgledanje dijele se na "uljeze" i "prilagodnike". Prvo spomenuta grupa komponenata transparentno se umeće unutar nadgledane aplikacije, bez promjene njena izvornog koda, dok se druga grupa komponenata ugrađuje unutar aplikacije, čineći njen sastavni dio. Podatke prikupljene nadglednim komponentama prihvaća i pohranjuje komponenta srednjeg sloja sustava –nadgledni sustav MIMO. Alati zainteresirani za podatke prikupljene nadgledanjem pristupaju isključivo komponenti srednjeg sloja sustava, postavljajući upite o vrijednostima nadglednih varijabli registrirajući se na događaje unutar nadgledanih aplikacija. Ovakvom organizacijom nadglednog sustava (slika 4.3) bitno se smanjuje utjecaj korisnika nadglednog sustava na rad nadgledanih aplikacija. Sve komponente sustava nadgledanja implementirane su korištenjem arhitekture CORBA i programskog jezika C.



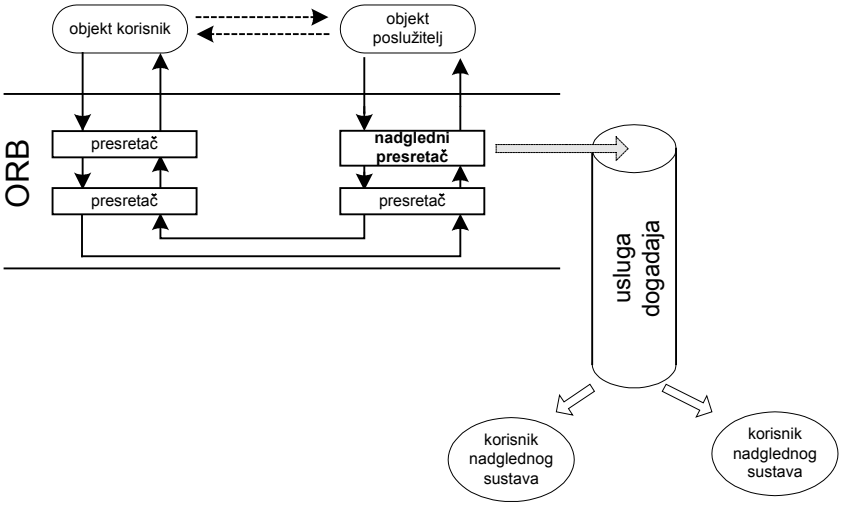
Slika 4.3. Organizacija nadglednog sustava MIMO

U magistarskom radu “Monitoring and Analyzing Method Invocations in the 2k Operating System” [21] opisana je izvedba sustava nadgledanja temeljena na transparentnom pribavljanju podataka i analizi poziva metoda udaljenih objekata u operacijskom sustavu 2k korištenjem mehanizma presretača poziva (slika 4.4). Sva mrežna komunikacija u spomenutom sustavu temelji se na sustavu CORBA, konkretno njezinoj implementaciji dynamicTAO. Značajka dynamicTAO [22] implementacije ORB-a je njegova modularnost i prilagodljivost (prilikom pokretanja i u toku rada aplikacije).



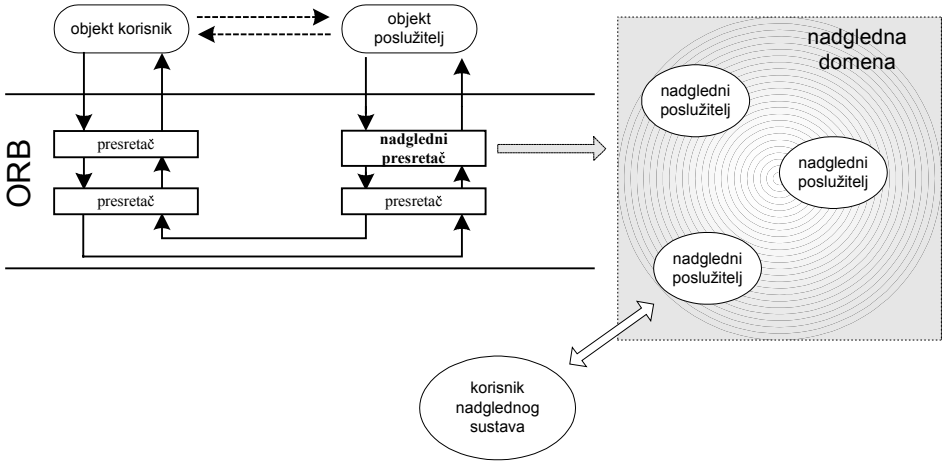
Slika 4.4. Organizacija nadgledanja poziva objekata u sustavu 2k

Pozivi metoda objekata presreću se na strani poslužitelja, bilježe i, po svršetku poziva, šalju sustavu za pohranu. Korisnik pristupa nadglednim podacima korištenjem sučelja spremišta nadglednih podataka (vrste jednostavne objektno usmjerene baze podataka). Korisnik ima mogućnost upravljanja nadglednim sustavom, odnosno prikupljanjem podataka postavljanjem dozvola/zabrana prikupljanja na razini imena objekata, te omogućavanjem ili onemogućavanjem rada presretača poziva u cijelosti (promjenom strukture ORB-a u toku rada sustava).



Slika 4.5. Organizacija nadgledanja međudjelovanja komponenti raspodijeljenog sustava korištenjem usluge događaja

U [23] i [24] predložen je mehanizam nadgledanja rada raspodijeljenih sustava temeljenih na komponentama izvedenih korištenjem sustava CORBA. Nadgledanje se temelji na presretanju komunikacije između komponenata sustava korištenjem presretača poziva, te stvaranja, prosljeđivanja, dohvaćanja i analize poruka o komunikaciji. Prilagodljivost sustava temelji se na filtriranju i pretplati na poruke od interesa korisnicima nadglednog sustava. Sustav je implementiran u programskom jeziku Java.



Slika 4.6. organizacija nadgledanja u sustavu GoodeWatch

GoodeWatch sustav [25] (slika 4.6) ima za cilj praćenje rada raspodijeljene aplikacije na razinama procesa, objekata, poziva i aktivnosti, a koristi se mehanizmom presretača poziva i promjenama unutar posredničke komponente (koristi se Orbacus 3 nadograđena implementacija arhitekture CORBA). Prikupljeni podaci emitiraju se u

lokalni računalnu mrežu, te ih zainteresirani nadgledni poslužitelji prikupljaju i pohranjuju. Alati za analizu i predočavanje pristupaju nadglednim poslužiteljima korištenjem njihovih javnih sučelja definiranih opisnim jezikom IDL. Događaji u sustavu podijeljeni su u dvije osnovne grupe: događaji sustava (pokretanje procesa, zaustavljanje procesa, pokretanje aktivnosti, zaustavljanje aktivnosti, pokretanje implementacije objekata, zaustavljanje implementacije objekta, slanje zahtjeva, primanje zahtjeva, slanje odgovora, primanje odgovora) i korisnički definirane događaje.

Java Management Framework [33] dodaci čine standardiziran pristup nadgledanju i upravljanju raspodijeljenih aplikacija temeljenih na komponentama JavaBeans. Upravljive komponente nasljeđuju dio funkcionalnosti od temeljne MBean upravljive komponente, te se prijavljuju poslužitelju MBean unutar aplikacije. Uloga poslužitelja MBean održavanje je liste aktivnih upravljivih komponenti, te ostvarivanje komunikacije s ostalim komponentama nadglednog sustava. Za komunikaciju s nadglednim poslužiteljem mogu biti korišteni različiti protokoli (Java RMI, HTTP, SNMP, itd.) putem odgovarajućih prilagodnika protokola.

Borland AppCenter [34] osigurava centralizirano nadgledanje i upravljanje raspodijeljenih aplikacija korištenih na različitim sklopovskim i programskim platformama. Sustav se temelji na modeliranju raspodijeljene aplikacije, nadgledanju njezina rada (kako na razini komponente sustava tako i sustava u cjelini) i upravljanju (od strane operatera ili automatskog upravljanja na osnovu definiranih politika). Moguće je upravljanje aplikacijama čije su komponente temeljene na različitim arhitekturama – Microsoft DCOM, Sun Enterprise JavaBeans, CORBA, itd. Nadgledanje samih resursa vrši se uporabom specijaliziranih agenata koji razmjenjuju nadgledne i upravljačke informacije sa srednjim slojem nadglednog sustava. Srednji sloj sustava sastoji se od posredničkih komponenata i upravljačke logike. Treći, korisnički sloj nadglednog sustava različite su inačice sučelja za predočavanje i kontrolu, a mogu biti izvedene kao zasebne aplikacije ili kao dio već postojećih nadglednih sustava (npr. sustav za nadgledanje mrežnih resursa temeljen na protokolu SNMP). Sustav AppCenter je namijenjen upravljanju sustava u cjelini, te nije dovoljno prilagodljiv za rad na niskoj razini praćenja rada implementacije pojedinih komponenti sustava.

5. Računalna refleksija

Etimološki, refleksija se u svom izvornom značenju odnosi na optički fenomen odražavanja slike objekta od ravne površine. Sljedeće značenje koju je riječ vremenom poprimila označavalo je primjenu (nečega) na samoga sebe. Od težišta na fizičkoj slici reflektiranog objekta, značenje riječi početkom 17. stoljeća pomiče se na mentalnu sliku, označavajući preispitivanje samoga sebe, te na (krajem 17. stoljeća) označavanje rasuđivanja, sveobuhvatne misli. Zanimljivo je da se u engleskom jeziku rabe dva oblika pisanja riječi refleksija: "reflexion", čijoj se uporabi daje prednost u označavanju optičkog fenomena, te "reflection" za ostala značenja.

U računarstvu pojam refleksije odnosi se na domenu programa koji (stvarno ili potencijalno) opisuju i mijenjaju sebe ili srodne programe. Ideja uporabe refleksije nije potpuno nova, korištena je od strane osnivača formalne logike i računarske znanosti u njihovim ranim teoremima o snazi i granicama rasuđivanja i računanja (Cantor, Gödel, Turing, itd.).

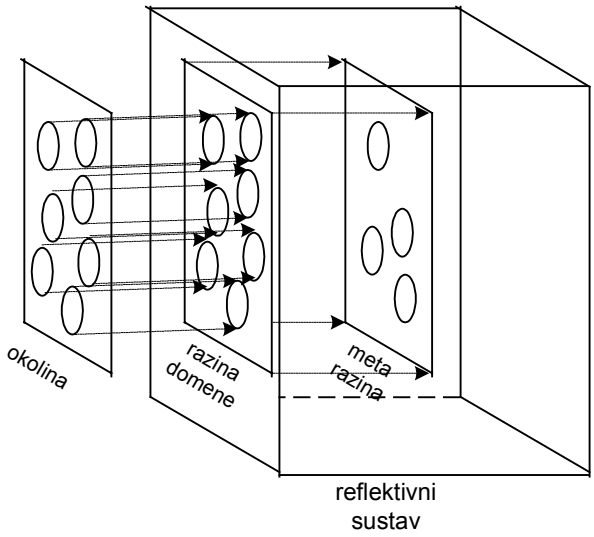
Pojam refleksije u moderno računarstvo uveo je B.C. Smith u svojoj doktorskoj disertaciji "Proceduralna refleksija u programskim jezicima" ("Procedural Reflection in Programming Languages", PhD Thesis, MIT) [4] u kojoj tvrdi da "ako je moguće izgraditi proces računanja čija je namjena rasuđivanje o vanjskom svijetu, a čiji je sastojak proces (interpreter) koji rukuje s formalnim prikazom tog svijeta, onda je moguće izgraditi i takav proces računanja koji će rasuđivati o samome sebi, sadržavajući drugi proces (interpreter) koji rukuje s formalnim prikazom vlastitih operacija i struktura".

Računalnu refleksiju (eng. *computational reflection*) Pattie Maes [4] opisuje kao "ponašanje predočeno od strane reflektivnog sustava, gdje je reflektivan sustav onaj računalni sustav koji je u posljedičnoj vezi sa samim sobom". Zadaća svakog računalnog sustava rješavanje je problema iz određene domene, tj. kažemo da je sustav o domeni. Domena problema prikazana je (točnije rečeno modelirana) unutar računalnog sustava korištenjem struktura u kojima su smješteni podaci o elementima domene (entiteti, veze) i programi kao skup pravila i postupaka rukovanja podacima u svrhu nalaženja rješenja problema i/ili poduzimanja akcija unutar domene. Posljedična veza (eng. *causal connection*) računalnog sustava i domene ogleda se u "jačini" povezanosti entiteta unutar domene i ekvivalentnih entiteta unutar računalnog sustava. Ako se promjena entiteta unutar jednog sustava (gotovo) istovremeno očituje promjenom ekvivalentnog entiteta unutar drugog sustava kažemo da su sustavi posljedično povezani. Slikovit primjer posljedične povezanosti naveden je u [4] primjerom robotske ruke i upravljačkog računalnog sustava. Refleksivan sustav,

slijedeći prije navedene definicije, posjeduje strukture o samome sebi, te njihovom promjenom mijenja i svoje izvršavanje. Reflektivna arhitektura sustava mora jamčiti:

- otkrivanje (reifikaciju) svoje strukture (ili dijela strukture)
- uzročnu povezanost strukture sa samim sustavom.

Potrebno je uočiti dvije razine reflektivnog sustava (slika 5.1) :



Slika 5.1. Dvije razine reflektivnog sustava

- razinu problemske domene o kojoj sustav je (strukture o problemskoj domeni)
- razinu domene o samome sustavu (strukture o samom sustavu).

Posljedično, prva razina naziva se razina domene (eng. *domain-level*), a druga meta razina (eng. *meta-level*). Isključivo razina domene rješava probleme i poduzima akcije unutar vanjske domene, dok meta razina pridonosi ukupnoj organizaciji sustava (tj. rješava probleme i poduzima akcije o rješavanju problema i poduzimanju akcija unutar vanjske domene). Svaka meta razina može imati svoju meta razinu, tvoreći rekurzivnu strukturu sustava (s beskonačno puno meta razina).

Postoje dvije vrste reflektivnih arhitektura, razlikovane u ovisnosti o načinu na koji sustav prikazuje samoga sebe:

- proceduralne reflektivne arhitekture
- deklarativne reflektivne arhitekture.

Proceduralna refleksija temelji se na otkrivanju implementacije sustava (programskog koda), čija promjena kao posljedicu ima i promjenu izvršavanja sustava (uvjet

uzročne povezanosti je zadovoljen). Deklarativna arhitektura otkriva sustav (ili dio sustava) korištenjem tvrdnji o sustavu. Tvrdnje su najčešće izražene u obliku ograničenja na ponašanje sustava. Zadovoljavanje uvjeta uzročne povezanosti u deklarativnim refleksivnim arhitekturama nešto je složenije stoga što sustav mora pravilno tumačiti deklarirane tvrdnje te ih "pretočiti" u stvarnost prilikom samog izvršavanja.

Pojam reifikacija (eng. *reification*) označava operaciju kojom se dotad implicitan, "nevidljiv" dio sustava eksplicitno formulira, tj. postaje "vidljiv " i podložan korištenju (mijenjanju). Kao primjer reifikacije u refleksivnim programskim jezicima može se navesti činjenje dostupnim dijelova interpretera izvršavanom programu (neovisno o kojoj refleksivnoj arhitekturi je riječ.

Meta programiranje označava disciplinu razvoja programa za pisanje, čitanje i/ili mijenjanje drugih programa.

5.1. Refleksija u raspodijeljenim sustavima CORBA

Osnovna uloga srednjeg sloja (eng. *middleware*) u raspodijeljenim sustavima štićenje je programera od kompleksnosti mrežne komunikacije i heterogenosti korištenih računalnih arhitektura, te definiranje uniformnog modela programiranja. CORBA nabrojena svojstva postiže korištenjem objektnog modela, u kojem glavnu ulogu imaju sučelja i objekti (osnovni entiteti implementacije i raspodijeljenosti), te reference na objekte (čine jedini mehanizam komunikacije među entitetima sustava pritom skrivajući kompleksnost komunikacije i posebnosti implementacija). No, osim osnovnih operacija nad posredničkom komponentom, te metoda zajedničkih svim objektima sustava CORBA, ne postoje dodatni mehanizmi kontrole i podešavanja rada srednjeg sloja. Iako je u tijeku razvoja dostupan dio sustava u obliku izvornog koda (kao rezultat prevođenja definicija sučelja objekata), funkcionalnost srednjeg sloja zasniva se na modelu "crne kutije" (eng. *black box*).

Raspodijeljeni sustav temeljen na CORBA-i skup je međusobno komunikacijski povezanih objekata. Objekti unutar sustava modelirani su na temelju prepoznatih entiteta unutar domene problema, te čine razinu domene aplikacije. Prilikom oblikovanja sustava koriste se standardne metode objektno usmjerene analize i oblikovanja, uz dodatne posebnosti neizbježne u oblikovanja raspodijeljenih sustava [12].

Korištenjem CORBA-e u sve širem spektru raspodijeljenih sustava, model "crne kutije" pojavljuje se kao faktor ograničavanja upotrebljivosti takvih sustava.

Raspodijeljeni sustavi novijih generacija kao jedan od bitnih zahtjeva nameću prilagodljivost okolini u kojoj djeluju. Neki od primjera prilagodljivih raspodijeljenih sustava su raspodijeljeni sustavi za rad u stvarnom vremenu, multimedijalni raspodijeljeni sustavi, ugrađeni sustavi, itd.

S obzirom na dinamičnost zahtijeva na prilagodljivost možemo prepoznati:

- statički prilagodljive sustave
- dinamički prilagodljive sustave.

Raspodijeljeni sustavi za rad u stvarnom vremenu, kao i ugrađeni sustavi zahtijevaju implementaciju sustava CORBA što manjeg zauzeća memorije i predvidljivog trajanja izvršavanja operacija. Malo zauzeće memorije postiže se korištenjem samo onih dijelova CORBA-e nužnih za rad konkretnog sustava, a ispuštanjem nepotrebnih dijelova (većinom implementacije naprednijih svojstava). Predvidljivo trajanje izvršavanja operacija postiže se optimiranjem implementacije, izbjegavanjem preslikavanja struktura podataka, onemogućavanjem inverzije prioriteta niti, itd.

Moguće su dvije vrste implementacija sustava CORBA za rad u stvarnom vremenu i ugrađene sustave:

- monolitne implementacije
- modularne implementacije.

Dinamika zahtjeva za prilagodbom sustava u nabrojanim slučajevima relativno je mala, tako da sam trenutak prilagođavanja možemo ograničiti na trenutak pokretanja sustava. Prilagođavanje sustava određuje se datotekama za prilagodbu i (ili) argumentima naredbenog retka. Za razliku od ovih sustava, multimedijalni raspodijeljeni sustavi (i ostali sustavi sličnih zahtjeva) moraju odgovoriti na promjene u okolini tijekom rada (propusnost mreže, opterećenje sustava, itd.). Stoga je prilagodba ograničena na trenutak pokretanja sustava nedovoljna. Izvedba mehanizama prilagodbe u dinamički prilagodljivim sustavima mora biti omogućena:

- “iznutra“ - definiranjem sučelja srednjeg sloja korištenjem kojeg sustav može izvršiti prilagodbu samog sebe

- “izvana“ - definiranjem sučelja srednjeg sloja korištenjem kojih se može izvršiti prilagodba sustava od strane entiteta koji nisu sastavni dio sustava.

S obzirom na rješenje problema prilagodljivosti, statički prilagodljive sustave možemo promatrati kao podskup dinamički prilagodljivih sustava, te za obje vrste sustava koristiti iste (dinamičke) mehanizme prilagođavanja.

Prilagodljivost sustava ne smije počivati isključivo na prilagodljivosti entiteta članova nivoa domene iz sljedećih bitnih razloga:

- u entitete razine domene prilagodljivost sustava uvela bi neprihvatljivu dozu grupiranja funkcionalnosti, a time i bitno uvećala složenost implementacije (i mogućnost pogrešaka)

- nužnost implementacije prilagodljivosti u svim entitetima sustava dovela bi do povećanja zahtjeva za potrebnim resursima, te, vjerojatno, problema u skalabilnosti sustava

- promjena strategija prilagodljivosti nužno bi za sobom povukla i promjenu implementacije entiteta razine domene

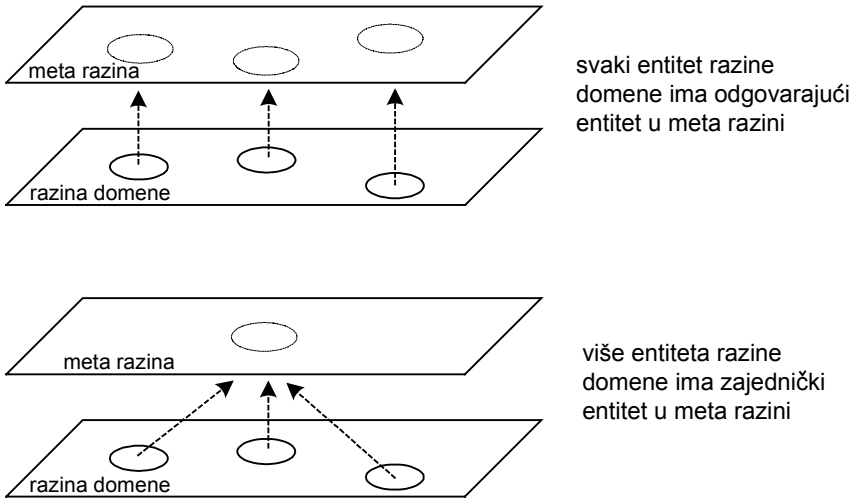
- onemogućila bi se ponovna uporabljivost koda u implementacijama drugih sustava.

Jedino cjelovito rješenje problema implementacije mehanizama dinamičke prilagodbe nalazi se u meta razini srednjeg sloja. Pod prilagodbom srednjeg sloja podrazumijeva se:

- prilagodba skupa korištenih komponenti srednjeg sloja
- prilagodba funkcionalnosti srednjeg sloja.

Prilagodbu skupa korištenih komponenti srednjeg sloja kao i korištenje različitih parametara očekivane kvalitete usluga (eng. *Quality of Service*) možemo ubrojiti u mehanizme deklarativne refleksije (definira se *što* se od sustava očekuje). Prilagodba funkcionalnosti sustava putem zamjene ili dodavanja koda unutar sustava ubraja se u mehanizme proceduralne refleksije (definira se *kako* se sustav treba ponašati).

Entiteti unutar meta razine prilagođavaju semantiku rada entiteta unutar razine domene. Kardinalnost entiteta meta razine definira odnos među entitetima obiju razina (slika 5.2):



Slika 5.2. Odnos entiteta razine domene i meta razine

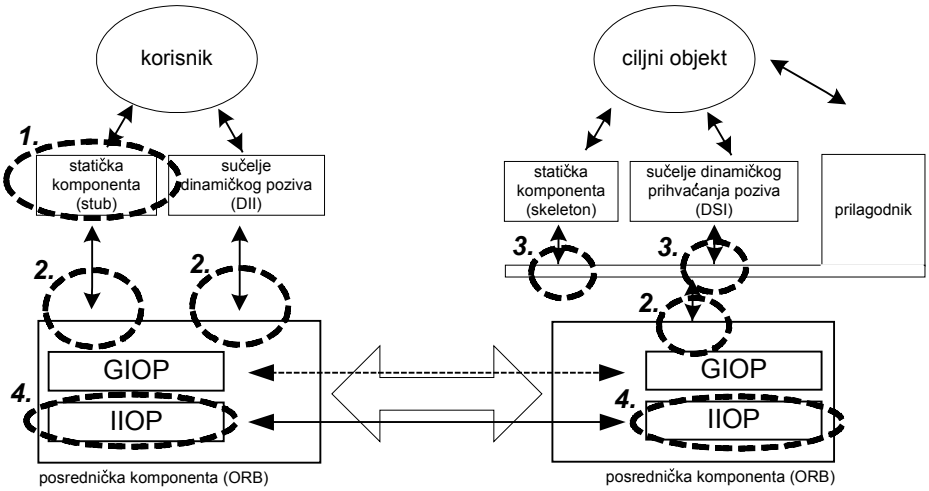
- Kardinalnost 1 označava odnos 1:1 entiteta meta razine i entiteta razine domene. Svakom entitetu razine domene pridružen je jedan entitet meta razine. Ovaj odnos entiteta omogućava "finiju" kontrolu rada sustava, ali zahtijeva veće resurse.

- Kardinalnost n označava odnos 1:n entiteta meta razine i entiteta razine domene. Više entiteta razine domene pridruženo je jednom entitetu meta razine. Zahtjevi na resurse su manji, ali je i kontrola rada sustava "grublja". Mogu se javiti problemi skalabilnosti ako je jednom entitetu meta razine pridružen velik broj entiteta razine domene.

Negativni učinci korištenja mehanizama meta programiranja u sustavima srednjeg sloja većinom se javljaju kao posljedica u izvedbama sustava s funkcionalnom reflektivnom arhitekturom:

- smanjenje brzine rada sustava (kod sustava izvodi se na dvije razine: razini domene i meta razini, problemi u skalabilnosti, itd.)
- integritet sustava (kod na meta razini čini dio funkcionalnosti srednjeg sloja).

Točke reifikacije sustava CORBA određene su arhitekturom objektnog modela (eng. *Object Management Architecture - OMA*) [36] definiranog unutar specifikacije arhitekture. Zanimljivo je da nije moguće, iako bi se to kod objektno usmjerenih sustava očekivalo, reificirati točke sustava bitne za životni vijek objekata, već samo točke bitne za njihovu međusobnu komunikaciju. Na slici 5.3 označene su reificirane točke unutar komunikacijskog puta općenitog sustava temeljenog na CORBA-i:

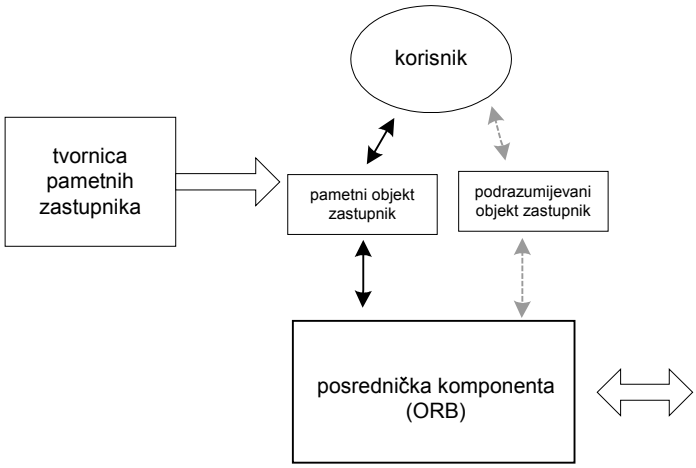


Slika 5.3. Točke reifikacije arhitekture CORBA

1. objekt zastupnik (eng. *proxy object, stub*)
2. presretači (eng. *interceptors*)
3. upravljači implementacija objekata (eng. s*ervant managers*)
4. ugradivi protokoli (eng. *pluggable protocols*).

5.1.1. Objekt zastupnik

Svaki poziv metode udaljenog objekta prolazi kroz dvije komponente sustava CORBA specifične svakom pojedinom objektu. Programske komponente nastale prevođenjem opisa sučelja u jeziku IDL ugrađuju se na strani korisnika objekta (eng. *stub*) i na stranu implementacije udaljenog objekta (eng. *skeleton*). Njihova je uloga komplementarna, a služe kao posrednici između programskog koda implementacije (na strani korisnika udaljenog objekta i na strani implementacije udaljenog objekta) i posredničke komponente sustava CORBA. Moguće je, na strani korisnika udaljenog objekta, zamijeniti standardnu implementaciju objekta posrednika s implementacijom specifičnom za pojedinu aplikaciju (slika 5.4), transparentno s obzirom na programski kod aplikacije (ne mijenja se sučelje objekta zastupnika, samo implementacija). Aplikacijski specifična implementacija objekta zastupnika naziva se "pametni zastupnik" (eng. *smart proxy*). Implementacija mehanizma zamjene standardne implementacije aplikacijski specifičnom temelji se na obrascu tvornice (eng. *factory pattern*) [8].



Slika 5.4. Pametni objekt zastupnik

Analizirajući mehanizam prilagodbe sustava korištenjem "pametnih zastupnika" možemo zaključiti:

- "pametni zastupnik" je meta objekt definiran na razini sučelja
- "pametni zastupnik" je primjer proceduralnog mehanizma refleksije.

Osnovne karakteristike "pametnih zastupnika" su sljedeće:

- unutar implementacije "pametnih zastupnika" mogu se mijenjati i vrijednosti i broj parametara poziva, povratna vrijednost, te vrsta povratne vrijednosti (uspješan poziv, iznimka, prosljeđivanje poziva)

- mehanizam je asimetričan stoga što ne postoji ekvivalentan mehanizam meta programiranja na strani implementacije udaljenog objekta

- mehanizam sam po sebi nužno ne uvodi dodatno kašnjenje u rad sustava.

Najčešća uporaba "pametnih zastupnika" dodavanje je parametara pozivima metoda, bilježenja komunikacije na strani korisnika udaljenog objekta, optimiranja komunikacije s udaljenim objektom, podrška specifičnim mehanizmima osiguravanja kvalitete usluga (*QoS*), osiguranje sigurnosti komunikacije, itd.

Implementacija "pametnih zastupnika" nije određena specifikacijom, već je ovisna o implementaciji sustava. Od poznatijih implementacija sustava CORBA "pametni zastupnici" mogu se naći u sustavima TAO i Orbix.

5.1.2. Prenosivi presretači

Za razliku od mehanizma "pametnih zastupnika", koji su usko povezani sa svakim pojedinim tipom (sučeljem) udaljenog objekta, prenosivi presretači [11] općenit su mehanizam meta programiranja putem kojega je moguće pratiti sve faze komunikacije unutar raspodijeljenog sustava temeljenog na CORBA-i. Temeljna uloga prenosivih presretača podrška je implementaciji dodatnih usluga (eng. *CORBA Services*)

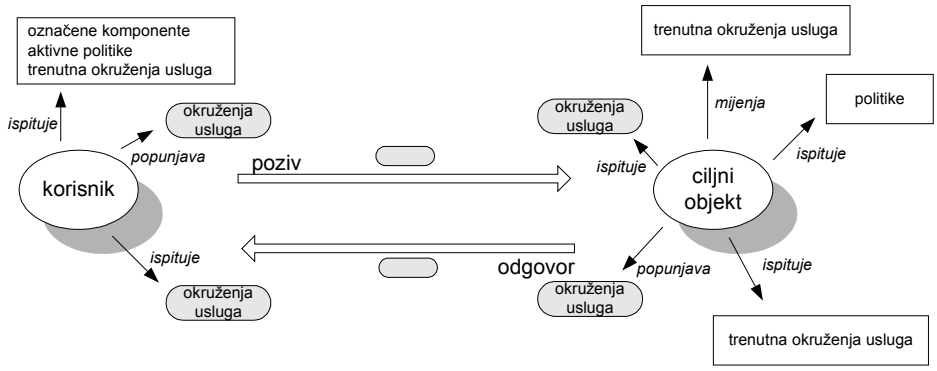
Prenosivi presretači dijele se s obzirom na ulogu unutar sustava na dvije grupe:

- presretači poziva metoda udaljenog objekta (eng. *request interceptors*)
- presretači stvaranja objektnih referenci (eng. *IOR interceptors*).

Osnovna uloga presretača prenošenje je informacija o okruženju usluga trenutnog poziva udaljenog objekta. Informacije o okruženju svake od aktivnih usluga nalazi se u objektima (tzv. "Current" objektima) pridjeljenim svakoj od aktivnih niti izvođenja CORBA temeljene aplikacije. Okruženje usluge može sadržavati bitne informacije od utjecaja na obradu poziva i (ili) tumačenje rezultata poziva metoda udaljenog objekta. Stoga je važno prenijeti sve informacije o okruženju usluge kao dijela svakog pojedinog poziva:

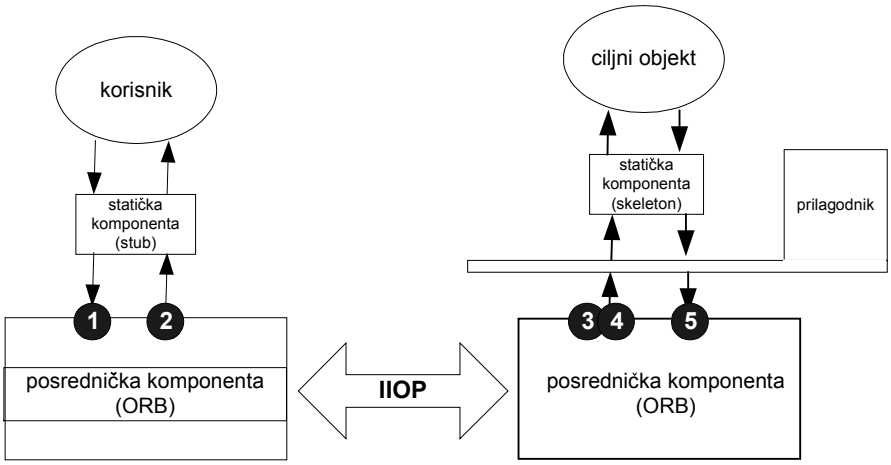
- sa strane korisnika na stranu implementacije udaljenog objekta
- sa strane implementacije udaljenog objekta na stranu korisnika udaljenog objekta.

Prenošenje i korištenje okruženja usluge prikazano je na slici 5.5:



Slika 5.5. Prenošenje okruženja usluga tijekom poziva metode ciljnog objekta

Točke presretanja poziva metoda udaljenog objekta definirane su (slika 5.6) :



Slika 5.6. Točke presretanja poziva metoda udaljenog objekta

- na strani korisnika udaljenog objekta (eng. *Client Interceptors*):

  1. slanje poziva udaljenog objekta
  2. primanje odgovora udaljenog objekta.

- na strani implementacije udaljenog objekta (eng. *Server Interceptors*):

  1. primanje okruženja usluga poziva od strane udaljenog objekta
  2. primanje poziva od strane implementacije udaljenog objekta
  3. slanje rezultata izvođenja metode udaljenog objekta.

Presretači su implementirani u obliku objekta prijavljenog unutar posredničke komponente. Prijava (jednog ili više) presretača mora se obaviti prije inicijalizacije posredničke komponente. Nakon inicijalizacije posredničke komponente prijavljeni presretači postaju njen sastavni dio, a naknadna odjava nije moguća. Objekt presretač mora implementirati standardni skup metoda pozivanih od strane posredničke komponente. S obzirom na tip poziva metode udaljenog objekta i rezultata poziva, točke presretanja razlikuju sljedeće događaje, a što za posljedicu ima pozive odgovarajućih metoda:

- Presretač na strani korisnika udaljenog objekta:
  - slanje poziva:
    - slanje poziva: `send_request( )`
    - slanje upita o završenoj obradi (asinkroni pozivi): `send_poll( )`

  - primanje odgovora:

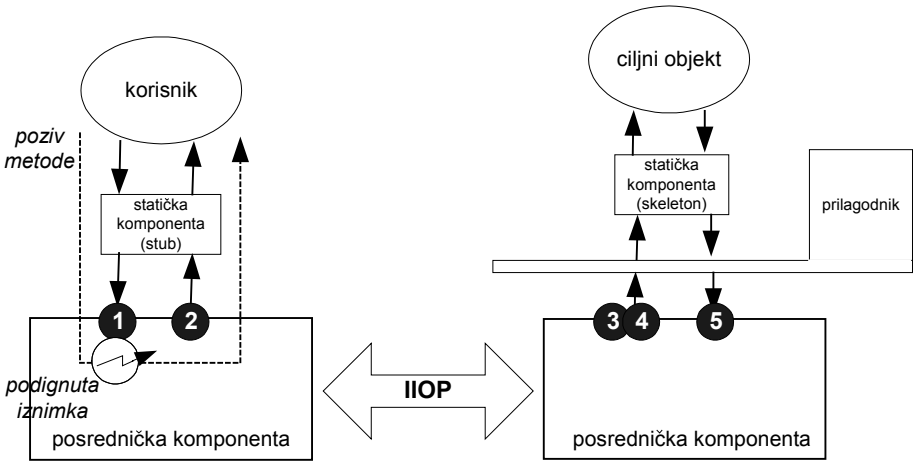
- poziv uspješno obavljen: `receive_response( )`
- greška u obradi poziva: `receive_exception( )`
- obrada poziva rezultirala je posebnim oblikom odgovora (asinkroni poziv, prosljeđivanje poziva drugom objektu, itd.): `receive_other( )`

- Presretač na strani implementacije udaljenog objekta:
  - primanje okruženja usluge poziva
    - prihvat okruženja usluge: `receive_request_service_context( )`

  - primanje poziva:
    - prihvat poziva: `receive_request( )`

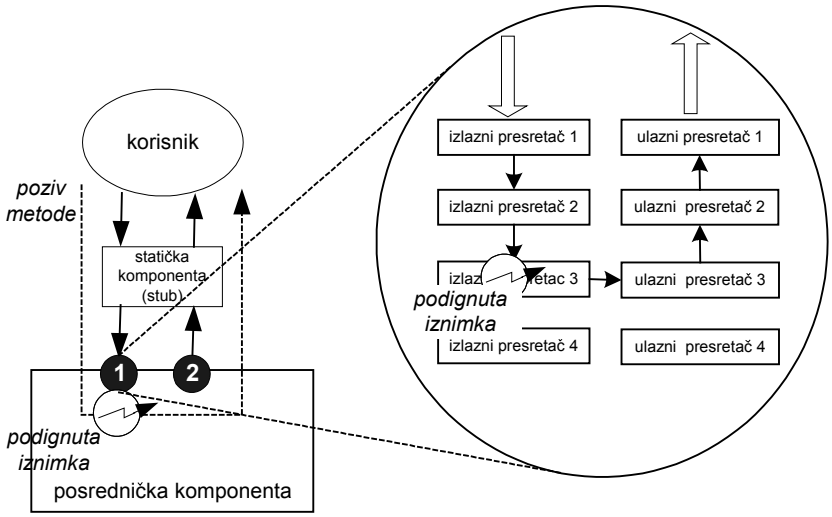
  - slanje rezultata:
    - odgovor uspješno obavljenog poziva: `send_response( )`
    - greška u obradi poziva: `send_exception( )`
    - obrada poziva rezultirala je posebnim oblikom odgovora (asinkroni poziv, prosljeđivanje poziva drugom objektu, itd.): `send_other()`

Presretači poziva imaju pristup svim argumentima i povratnim vrijednostima poziva, no za razliku od "pametnih zastupnika" ne mogu ih izravno mijenjati; moguća je jedino promjena tipa rezultata poziva. Ukoliko se tip rezultata promijeni unutar izlaznog puta poziva (metode `send_request` ili `send_poll`), poziv neće ni dostići implementaciju udaljenog objekta, a time neće biti pozvane ni metode presretača na strani implementacije udaljenog objekta (slika 5.7).



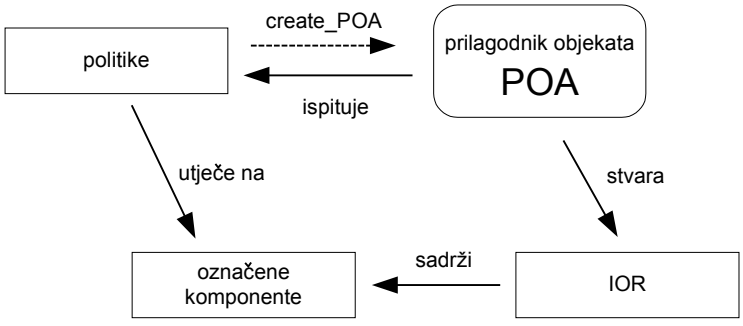
Slika 5.7. Podizanje iznimke u izlaznom presretaču na strani korisnika

U slučaju da je unutar posredničke komponente sustava registrirano više presretača specifikacija ne definira redoslijed u kojem će presretači biti pozivani, jedino jamči da će za sve presretače čija je početna metoda pozvana, bit će pozvana i završna metoda (npr. kod presretača na strani implementacije udaljenog objekta čija je metoda `receive_request` pozvana, bit će pozvana i jedna od metoda iz završnog skupa (`send_response`, `send_exception` ili `send_other`). Ukoliko je unutar jednog od presretača prekinut ulazni put podizanjem iznimke ili prosljeđivanjem poziva drugom objektu (ili izlazni put, ovisi o kojem tipu presretača je riječ), svi presretači kojima jedna od ulaznih metoda nije pozvana neće za taj poziv udaljenog objekta ni biti korišteni.



Slika 5.8. Redoslijed poziva presretača na strani korisnika u slučaju podizanja iznimke u izlaznom putu

Presretači stvaranja interoperabilnih (IOR) referenci imaju ulogu dodavanja informacija unutar reference objekta. Dodatne informacije u obliku označenih komponenti unutar reference (eng. *tagged components*) koriste se za pohranu informacija specifičnih za pojedini objekt, npr. informacije o kvaliteti usluga, sigurnosti, prioritetima obrade, itd. Bitno je napomenuti da specifikacija ne definira točan trenutak poziva ovog presretača, stoga on ne mora biti pozvan kod stvaranja svakog objekta (tj. reference) pojedinačno, već je to ovisno o implementaciji.



Slika 5.9. Presretač stvaranja objektnih referenci

Prenosivi presretači su proceduralan mehanizam refleksije, temeljeni na dodavanju koda unutar meta razine sustava. Implementacija dodatne funkcionalnosti je simetrična, s mogućnošću dodavanja na jednoj ili obje strane komunikacije (na strani korisnika udaljenog objekta i (ili) na strani implementacije udaljenog objekta). Meta objekt (presretač) definiran je na grupi objekata (tj. na svim objektima korisnicima posredničke komponente kod koje je presretač registriran).

Zbog visoke učestalosti poziva metoda objekta presretača (prosječno pet poziva po jednom pozivu implementacije udaljenog objekta ako je sustav simetričan), ovaj mehanizam meta programiranja može postati "usko grlo" sustava, stoga je njegova pravilna implementacija od velike važnosti.

Specifikacija prenosivih presretača dio je (trenutno nedovršenog) standarda CORBA 3. Implementacije ORB-a koji u sadašnjem trenutku podržavaju prenosive presretače su TAO, Orbacus 4 i Orbix 2000.

5.1.3. Ugradivi protokoli

Standard propisuje da svaka implementacija sustava CORBA mora podržavati IIOP protokol kao osnovni protokol komunikacije između posredničkih komponenti. Implementacija i korištenje ostalih protokola u standardu se ne spominje. Kao rezultat nedefiniranosti standarda, proizvođači implementacija sustava CORBA razvili su vlastita, neudruživa rješenja. Navođenje korištenja pojedinog protokola u tim rješenjima moguća je, većinom, samo prilikom pokretanja sustava. Promjena protokola vrijedi za sve objekte unutar sustava i transparentna je glede njihove implementacije. Dinamička promjena korištenog protokola zanimljiva je u području multimedijalnih aplikacija te, naročito, u domeni pokretnog računarstva.

Implementacije sustava CORBA Orbacus 4 i TAO podržavaju promjenu korištenog protokola prije pokretanja sustava promjenom komponente sustava zadužene za mrežnu komunikaciju. DynamicTAO [13] implementacija sustava CORBA temelji se na sustavu građenom isključivo od komponenti koje je moguće mijenjati i u tijeku izvođenja.

5.1.4. Upravitelji implementacije objekata

Upravitelji implementacije objekata (eng. servant managers) unutar prilagodnika objekata POA imaju zadaću upravljanja životnim vijekom implementacija objekata. Dvije su vrste upravitelja implementacija:

- pokretači implementacija objekata (eng. *servant activators*)

- lokatori implementacije objekata (eng. *servant locators*).

Pokretači implementacija objekata pronalaze i pokreću programske entitete koji sadrže kod implementacije neaktivnih objekata, aktiviraju ih i registriraju unutar prilagodnika objekata POA. Ovim mehanizmom upravlja se samo početnim stadijem života implementacije objekata.

Lokatori implementacije objekata upravljaju čitavim životnim vijekom implementacije objekata, od njihova pronalaženja, pokretanja, izvođenja pozvane metode, te zaustavljanja rada. Njihova funkcionalnost slična je funkcionalnosti prenosivih presretača, ali s ograničenjem na objekte pridružene prilagodniku objekata u kojem je upravitelj registriran.

Upravitelji implementacije objekata ne mogu se definirati kao općeniti mehanizam meta programiranja zbog njihove bliske povezanosti s implementacijama udaljenih objekata. Iako same implementacije ne moraju biti svjesne upravitelja i njihovih akcija u određivanju životnog vijeka, sami upravitelji moraju posjedovati dovoljno znanja o upravljanim implementacijama.

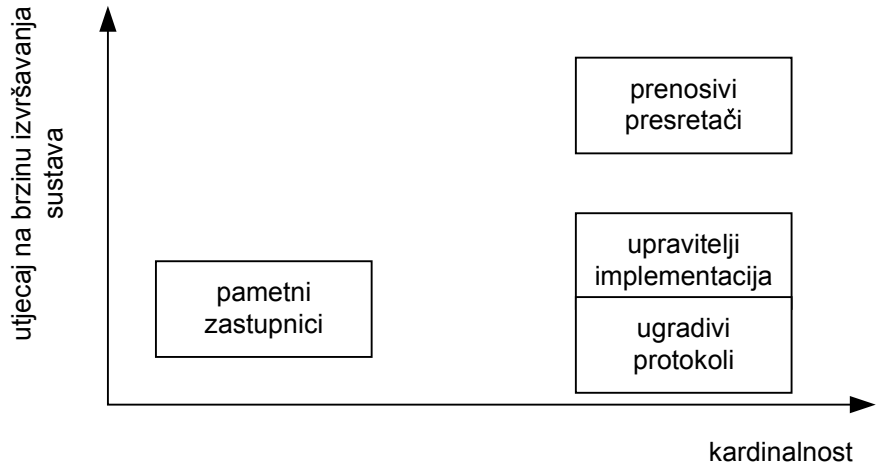
Upravitelji objekata prisutni su u svim implementacijama sustava CORBA od inačice standarda 2.2.

5.1.5. Usporedba mehanizama meta programiranja

mehanizam	**kardinalnost**	**utjecaj na brzinu izvršavanja**	**dostupnost argumenata poziva**	**mogućnost promjene argumenata poziva**	**promjena vrste odgovora**	**utjecaj na skalabilnost**	**standardiziranost**
pametni zastupnici	po udaljenom objektu	mali	da	da	da	ne	ne
prenosivi presretači	svi objekti ORB-a	uočljiv	da	ne	da	da	da (standard CORBA 3)
upravitelji implementacija	svi objekti prilagodnika objekata POA	mali	ne	ne	da	da	da (prilagodnik objekata POA)
ugradivi protokoli	svi objekti ORB-a	ne	ne	ne	ne	ne	ne

Tablica 5.1: usporedba svojstava mehanizama meta programiranja u sustavu CORBA

Usporedba pokazuje da općeniti mehanizmi meta programiranja u sustavu CORBA (a to su mehanizmi pametnih zastupnika i prenosivih presretača) nude različita, komplementarna svojstva. Uvođenje mehanizma pametnih zastupnika nema gotovo nikakav utjecaj na brzinu izvršavanja, pa čak (uvođenjem optimizacija poziva metoda) može doći i do poboljšanja performansi sustava. Nedostatak spomenutog mehanizma je što je njegov utjecaj ograničen samo na korisničku stranu komunikacije. Prenosivi presretači omogućuju utjecaj na izvođenje sustava i na korisničkoj strani i na strani implementacije udaljenog objekta, ali imaju uočljiv negativan utjecaj na performanse sustava. Pozitivna značajka prenosivih presretača također je i u njihovoj standardiziranosti, za razliku od pametnih zastupnika čija implementacija ovisi o proizvođaču implementacije sustava CORBA.



Slika 5.10. Svojstva mehanizama meta programiranja u sustavu CORBA

Slika 5.10 prikazuje svojstva pojedinih mehanizama meta programiranja prisutnih u implementacijama sustava CORBA s obzirom na kardinalnost i utjecaj na brzinu izvođenja sustava koje njihova uporaba uvodi [6].

6. Nadgledna usluga

Namjena sustava za nadgledanje rada raspodijeljenih aplikacija temeljenih na CORBA-i, razvijenog u okviru grupe RASIP, Zavoda za automatiku i procesno računarstvo, Fakulteta elektrotehnike i računarstva u Zagrebu, praćenje je rada i prikupljanje informacija o sustavu u svrhu:

- pronalaženja pogrešaka u toku razvoja raspodijeljene aplikacije
- vizualizaciji rada raspodijeljene aplikacije
- prilagodbe rada nadgledanih raspodijeljenih sustava.

U njegovu oblikovanju i implementaciji postavljeni su sljedeći zahtjevi:

- minimalan utjecaj nadglednog sustava na nadgledani sustav
- prenosivost sustava na sve (ili što veći broj) postojećih implementacija sustava CORBA
- prilagođenost nadglednog sustava modelu OMA
- ponovna iskoristivost dijelova sustava.

U poglavlju o nadgledanju raspodijeljenih sustava već su nabrojena četiri bitna čimbenika o kojima treba voditi računa prilikom oblikovanja nadglednog sustava: transparentnost nadgledanja, smanjenje i predviđanje utjecaja na rad nadgledanog sustava, smanjenje resursa potrebnih za rad nadglednog sustava i smanjenje međuovisnosti nadgledanog i nadglednog sustava.

Prenosivost sustava uvjetovana je neovisnošću nadglednog sustava o promatranom sustavu i korištenjem standardnih mehanizama CORBA-e za njegovu implementaciju. Time se otvara mogućnost korištenja nadglednog sustava u svim raspodijeljenim aplikacijama temeljenim na CORBA-i, bez obzira na to da li je prilikom oblikovanja i implementacije aplikacije uzeta u obzir mogućnost nadziranja njenog rada.

Prilagođenost nadglednog sustava modelu podrazumijeva nadgledanje svih bitnih parametara svojstvenih raspodijeljenim aplikacijama temeljenim na CORBA-i.

Oblikovanje i implementacija svih komponenti potrebnih za sustav nadgledanja protivila bi se jednoj od osnovnih postavki modela OMA – građenju raspodijeljenih sustava od univerzalnih gotovih komponenti (tzv. usluga) i komponenti ovisnih o pojedinoj aplikaciji. Stoga je jedan od ciljeva izgradnje nadglednog sustava korištenje što većeg broja gotovih komponenti, a implementacija samo onih dijelova sustava koji su specifični sustavu nadgledanja.

U osnovi, postojale su samo dva načina ostvarenja nadglednog sustava – korištenjem varijabli stanja i logičkog spremišta (MIB) ili nadgledanja temeljenog na događajima u sustavu. U daljnjem tekstu bit će opisane spomenute realizacije s obzirom na tražena svojstva nadglednog sustava.

Minimalan utjecaj na nadgledani sustav: podaci predstavljeni varijablama stanja sustava u pravilu su stanja tzv. upravljanih objekata, te je nužna ugrađena podrška nadgledanju već od faze oblikovanja sustava, i to u razini entiteta aplikacije – sučelja i implementacije objekata (upravljani objekti [16] [17] [20]). Time se mogućnost nadgledanja ograničava samo na one sustave kojih je nadgledanje sastavni dio. Također, nadglednom sustavu moraju biti poznati svi entiteti aplikacije koji imaju mogućnost nadgledanja, kako bi im mogao pristupiti i dohvatiti informacije o njihovu stanju. Podaci o entitetima s mogućnosti nadgledanja mogu biti sastavni dio nadglednog sustava (što sustav čini strogo namjenski) ili biti pohranjeni u obliku datoteke za podešavanje, što omogućava nešto veću prilagodljivost, ali nedostatnu sustavima karakteriziranim brzim promjenama. Utjecaj prikupljanja podataka od strane nadglednog sustava je predvidljiv zbog ugrađenosti samog mehanizma u aplikaciju. Prikupljanje se u većini slučajeva vrši periodički, te se može kontrolirati. S druge strane, sustavi temeljeni na događajima prikupljanim na meta razini aplikacije potpuno su neovisni o aplikacijskom sloju sustava, te primjenljivi na sve aplikacije temeljene na CORBA-i. Jedini problem prenošenje je velikog broja poruka o događajima ostalim komponentama nadglednog sustava, a koji zahtjeva određene resurse računalnog sustava na kojemu se nadgledani sustav izvodi. Veliki broj događaja i njihova nepredvidljivost pojave kod loše oblikovanog i izvedenog nadglednog sustava mogu za posljedicu imati nezanemariv utjecaj na rad nadgledanog sustava.

Prenosivost sustava: kao što je u prethodnom tekstu spomenuto, sustave temeljene na varijablama stanja karakterizira visok stupanj grupiranja s nadgledanim sustavom. Implementacija baza podataka upravljanja (MIB) i ostale moguće komponente sustava neovisne su o nadgledanom sustavu, no dio sustava za prikupljanje podataka pokazuje visok stupanj ovisnosti. Sustav temeljen na događajima oslanja se na mehanizme ugrađene u CORBA-u, a samim time prenosive između različitih platformi i programskih jezika.

Prilagođenost nadglednog sustava modelu OMA: sustavi temeljeni na varijablama stanja pogodni su za periodičko praćenje stanja pojedinih entiteta. Praćenje učestalih i nepredvidljivih promjena ovim mehanizmom nije moguće riješiti na zadovoljavajući način. Stoga se ovi sustavi većinom koriste kao sustavi za nadgledanje opće namjene. Sustavi temeljeni na događajima pokazuju svojstvo dinamičnosti, brze reakcije na događaje čije je vrijeme pojave nepredvidljivo, a usporedo imaju i mogućnost

periodičnog izvješćivanja o stanjima sustava. Stoga su ovi sustavi daleko pogodniji za praćenje rada sustava temeljenih na CORBA-i, na "niskoj" razini sustava.

Ponovna iskoristivost komponenti sustava: implementacija sustava temeljenog na varijablama stanja zahtijevala bi razvoj svih komponenti sustava, od dijela za prikupljanje podataka ugrađenog u nadgledani sustav, preko baze podataka upravljanja do alata za nadgledanje rada i pohranu podataka. U sustavu temeljenom na događajima za većinu komponenti sustava moguće je koristiti već raspoložive usluge CORBA-e, a potrebno je razviti dio sustava ugrađenog u promatranu aplikaciju i pojedine alate za obradu, analizu, pohranu i prikaz prikupljenih podataka.

Nedostatak sustava temeljenih na događajima je što, za razliku od sustava temeljenih na varijablama stanja, zahtijevaju kompleksniju analizu prikupljenih podataka u svrhu dobivanja složenih parametara sustava. Sustavi temeljeni na varijablama stanja eksplicitno prikazuju stanje sustava, bilo da se radi o jednostavnim (broj poziva metoda pojedinog objekta) ili složenim parametrima (kvalitete usluga). Sustavi temeljeni na događajima, u većini slučajeva, te podatke moraju dobiti analizom nekoliko osnovnih tipova događaja.

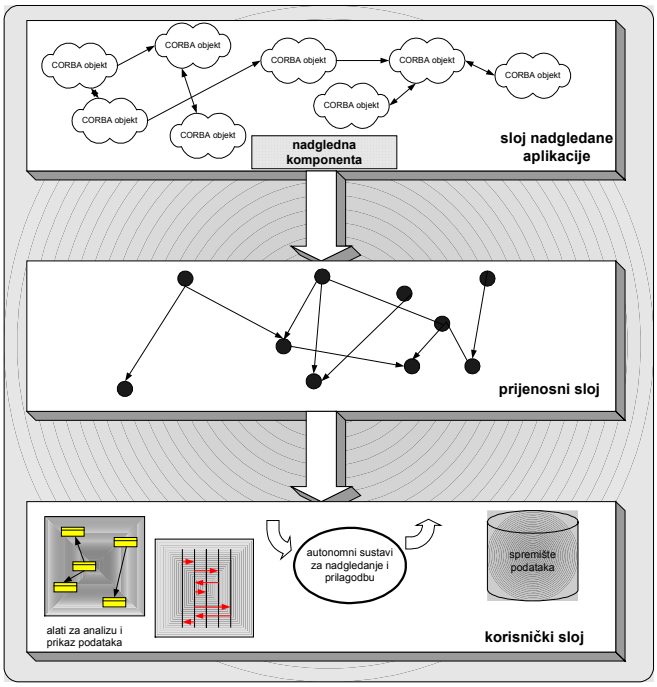
Provedena analiza jasno je pokazala prednosti i nedostatke obaju sustava, te je odlučeno da nadgledni sustav koristi mehanizam prikupljanja događaja. Spomenuti mehanizam daleko se bolje uklapa u model OMA, te je daleko prilagodljiviji od sustava temeljenih na varijablama stanja. Velika prilagodljivost nadglednog sustava naročito dolazi do izražaja u sustavima s velikom dinamikom uključenih komponenti, npr. u sustavima koji koriste autonomne pokretne objekte.

6.1. Organizacija nadglednog sustava

Nadgledni sustav temelji se na troslojnoj arhitekturi (slika 6.1):

- prvi sloj predstavlja sloj nadgledane aplikacije
- drugi sloj predstavlja prijenosni sloj sustava
- treći sloj predstavlja korisnički sloj sustava.

Nadgledni sloj čine entiteti nadgledane raspodijeljene aplikacije – objekti sustava CORBA – i komponenta nadglednog sustava zadužena za prepoznavanje događaja i prosljeđivanje odgovarajućih poruka prijenosnom sloju nadglednog sustava.



Slika 6.1. Organizacija nadglednog sustava

Poruke nadglednog sustava dijele se s obzirom na vrstu događaja na sljedeće grupe:

- događaje svojstvene sustavu CORBA
- događaje svojstvene nadgledanoj aplikaciji.

Svaka poruka nadglednog sustava sastoji se od zaglavlja, zajedničkog bez obzira na vrstu pridruženog događaja, i tijela poruke s podacima o pojedinom događaju. Tijelo poruke pridružene događaju svojstvenom sustavu strogo je definirano, dok je tijelo poruke pridruženo događaju svojstvenom aplikaciji definirano na razini svake aplikacije zasebno.

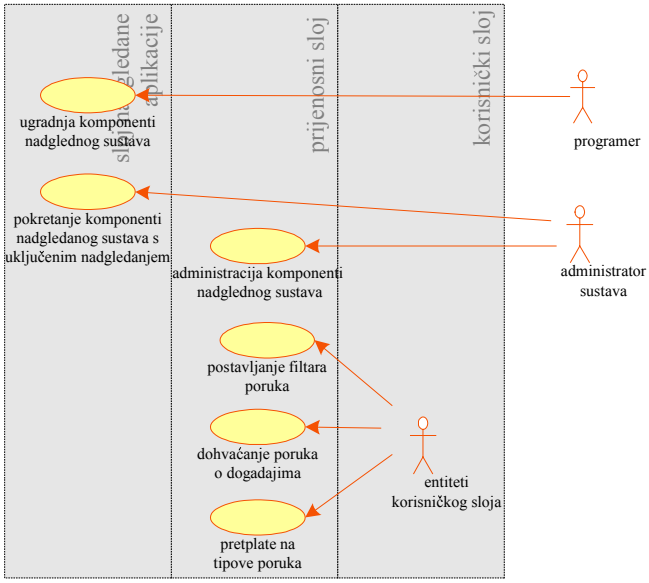
Zadaća prijenosnog sloja prenošenje je poruka o događajima od strane komponente nadglednog sustava iz sloja nadgledane aplikacije do entiteta u korisničkom sloju. Poruke se prenose sustavom (virtualnih) kanala organiziranih u obliku usmjerenog acikličkog grafa. Poruke se usmjeravaju korištenjem filtara definiranih na početnim i završnim krajevima kanala. Kanali također prenose informacije o pretplatama na

poruke od entiteta korisničkog sloja do komponenti nadglednog sustava. Na osnovu informacija o pretplatama komponente sustava nadglednog sloja mogu vršiti selektivno slanje poruka samo o onim događajima za koje postoji zainteresiranost u korisničkom sloju.

Entiteti korisničkog sloja prikupljaju poruke o događajima u sustavu za koje su zainteresirani i vrše daljnju obradu. Svoju zainteresiranost za pojedinim vrstama poruka iskazuju postavljanjem odgovarajućih filtara na završnim krajevima kanala i korištenjem mehanizama pretplate.

Na slici 6.2 prikazane su uloge unutar nadglednog sustava, te u kojim slojevima sustava se one očituju. Entiteti korisničkog sloja vrše svoje uloge korištenjem mehanizama ugrađenih u prijenosni sloj, ne utječući izravno na komponente sustava u nadgledanom sloju. Nepostojanje grupiranja entiteta korisničkog i nadgledanog sloja daje ovakvoj organizaciji nadglednog sustava neovisnost o promatranim entitetima. Sučelje nadglednog sustava promatrano sa strane entiteta korisničkog sloja svodi se na sučelje korištene usluge poruka CORBA-e.

Osim entiteta korisničkog sloja nadglednog sustava, pojavljuju se dva dodatna entiteta (osobe) u interakciji sa sustavom. Programer je zadužen za umetanje komponente nadglednog sustava u nadgledanu aplikaciju. Umetanje komponente svodi se na dodavanje nekoliko redaka unutar programskog koda komponenata nadgledane aplikacije i povezivanja s bibliotekom koda koja implementira funkcionalnost komponente nadglednog sustava. Administrator sustava pokreće komponente nadgledanog sustava s dodatnim parametrima koji kontroliraju rad nadgledne komponente, te administrira komponente prijenosnog sloja nadglednog sustava.



Slika 6.2. Uloge u nadglednom sustavu

6.2. Sloj nadgledanog sustava

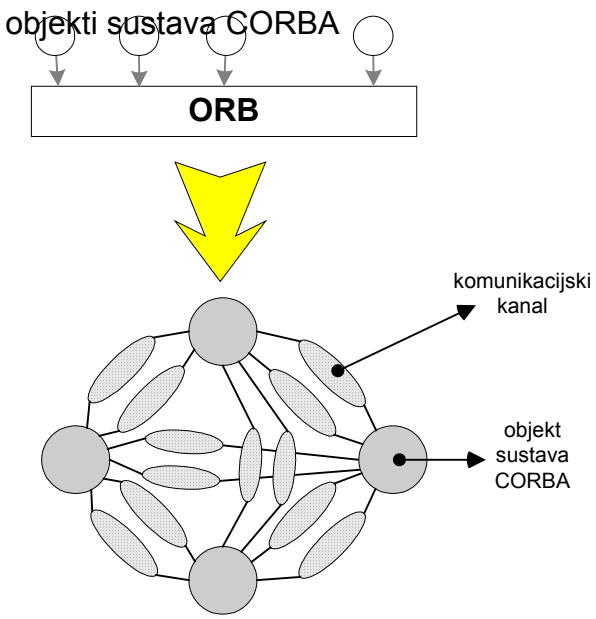
Opis raspodijeljenih računalnih sustava temeljenih na CORBA-i [1] [9] [10] uglavnom se svodi na skup međusobno komunikacijski povezanih računala na kojima se izvršavaju objekti u okviru izvršnih okolina (računalnih procesa). No za detaljniju analizu opisanog sustava potrebno je usvojiti općenitiji pristup. Korištenjem modela asinkronog mrežnog sustava [7] moguće je postići dovoljnu općenitost i formalnost prikaza svakog raspodijeljenog sustava temeljenog na asinkronoj komunikaciji između entiteta sustava, pa tako i raspodijeljenog sustava temeljenog na CORBA-i.

Osnovni problem u nadgledanju rada raspodijeljenih sustava dobivanje je cjelovite i ispravne slike rada sustava temeljene na promatranju karakterističnih događaja za arhitekturu (npr. komunikacije između komponenti sustava) i za pojedinu aplikaciju (npr. životni vijek entiteta). Za stvaranje ispravne slike rada sustava bitan je poredak promatranih događaja, čije osiguranje, s obzirom na problem usklađenosti vremena na različitim računalima na kojima se komponente sustava izvode, ne predstavlja trivijalan problem.

U daljnjem tekstu opisana je metoda dobivanja poretka događaja u raspodijeljenom sustavu temeljenom na CORBA-i implementacijom Lamportovog algoritma lokalnog logičkog vremena. Rješenje temeljeno na mehanizmu prenosivih presretača [11] omogućuje dodavanje implementacije algoritma bez nužnosti mijenjanja koda nadgledane aplikacije, te posjeduje svojstva transparentnosti i prenosivosti.

6.2.1. Model sustava CORBA

Modelom asinkrone mreže, zahvaljujući njegovoj općenitosti, moguće je opisati i raspodijeljene mrežne sustave temeljene na CORBA-i. No prevelika općenitost u opisu konkretnog sustava može kao posljedicu imati veliku (i nepotrebnu) kompleksnost modela, izazivajući time probleme u analizi i oblikovanju predstavljenih sustava. Stoga je nužno prepoznati osnovne elemente asinkronog mrežnog modela koji su dostatni za opis modela raspodijeljenih sustava temeljenih na CORBA-i.



Slika 6.3. Sustav CORBA prikazan modelom asinkrone mreže

Temeljne razlike između raspodijeljenih sustava opisanih modelom asinkronog mrežnog sustava i raspodijeljenih sustava temeljenih na CORBA-i posljedica su različitih pogleda na opisane sustave. Dok se u opisu raspodijeljenih algoritama najveća pozornost posvećuje korektnosti definicije matematičkog modela algoritma i dokazivanje ispravnosti ("nizak" pogled na sustav), u opisu sustava temeljenih na CORBA-i naglasak je na općenitosti i čvrstom opisu sučelja komponenata ("visok" pogled na sustav). Osnovna postavka od koje je potrebno krenuti u analizi primjene asinkronog modela je da CORBA implementira objektno usmjeren pristup u raspodijeljenom računalnom sustavu. Jedno od temeljnih svojstava objektno usmjerenih sustava je enkapsulacija – skrivanje unutarnje strukture i implementacije objekta, te mogućnost pristupa (komunikacije s objektom) samo putem čvrsto definiranog sučelja. Stoga će u daljnjoj analizi naglasak biti na pristupu elementima modela (objektima ili procesima) kao "crnim kutijama" (eng. *black box*), većinom zanemarujući elemente koji definiraju "unutrašnjost" komponenata modela.

Osnovna sličnost između modela asinkrone mreže i raspodijeljenog sustava temeljenog na CORBA-i, koja se odmah uočava, je veza između skupa čvorova V grafa $G = (V, E)$ i skupa objekata opisivanog sustava. U sustavima opisanim modelom asinkronog mrežnog sustava svaki proces P_i (predstavljen čvorom V_i grafa G) obavlja strogo definiranu ulogu unutar sustava, opisanu korištenjem modela ulazno-izlaznog automata. Proces (ulazno-izlazni automat) [7], definiran je s pet elemenata: potpisom, skupom stanja, skupom početnih stanja, skupom tranzicija i zadaćama. Vanjski potpis procesa (automata) definiran je skupom ulaznih i skupom izlaznih akcija.

Sučelje objekta opisano je korištenjem opisnog jezika sučelja IDL, u kojem se definiraju:

- skup ulaznih akcija objekta *in(S)*
- gramatike kojih su argumenti članovi ulaznih akcija.

U terminologiji CORBA-e jezik IDL opisuje sučelje objekta definiranjem metoda i tipova podataka korištenih u metodama. Uočava se nedostatak izravne definicije izlaznih akcija objekata. Izlazne akcije objekta određene su ulaznim akcijama od njega korištenih objekata. Za točniju definiciju ove tvrdnje potrebno je razmotriti dva moguća načina poziva metoda udaljenog objekta:

- statički mehanizam poziva metoda udaljenog objekta (eng. *static invocation interface – SII*)
- dinamički mehanizam poziva metoda udaljenog objekta (eng. *dynamic invocation interface – DII*).

Statički mehanizam poziva metoda udaljenog objekta oslanja se na korištenje lokalnog objekta zastupnika (eng. *proxy object*) udaljenog objekta za pripremu i slanje poziva putem posredničke komponente. Gledano sa strane posredničke komponente, pozivajući objekt, korištenjem objekta zastupnika, izvodi akciju istoimenu akciji definiranoj u sklopu sučelja pozivanog objekta (pojam akcija koristi se kao sinonim poziva metode). Stoga se, gledajući sa strane posredničke komponente (komunikacijskog kanala) može zaključiti da je skup izlaznih akcija nekog objekta unija skupova ulaznih akcija dostupnih udaljenih objekata (udaljenih objekata čiji objekti zastupnici postoje u istom adresnom prostoru kao i promatrani objekt, a koje promatrani objekt ima namjeru koristiti).

Za korištenje dinamičkog mehanizma poziva metoda udaljenog objekta nije potreban lokalni objekt zastupnik, što određivanje skupa izlaznih akcija (teoretski) čini nemogućim. Uz potrebno znanje o sučelju udaljenog objekta, a koje je moguće pribaviti i u tijeku rada programa korištenjem mjesta za pohranu sučelja (eng. *Interface Repository*) [1], može se izvršiti poziv bilo koje metode sučelja udaljenog objekta čija je objektna referenca dostupna promatranom objektu.

Svaka komponenta asinkronog mrežnog sustava može igrati ulogu aktivne komponente sustava, tj. može biti neovisni pokretač aktivnosti sustava kao posljedice obavljanja zadaća definiranih unutar ulazno-izlaznog automata. Pod izrazom neovisni pokretač aktivnosti sustava podrazumijeva se ona komponenta sustava koja ima barem jednu zadaću čije je izvršavanje neovisno o prethodnom izvršavanju sustava (tj. nije posljedica ulaznih akcija komponente tijekom prethodnog izvršavanja sustava).

Kao primjer pasivne komponente sustava može se navesti komunikacijski kanal, čije su izlazne akcije uvjetovane ulaznim akcijama (prosljeđivanje poruke uvjetovano je njenim primitkom). Kao primjer aktivne komponente može se navesti komponenta koja igra ulogu sata, čija je zadaća slanje poruka svake sekunde. Periodičko slanje poruke u ovom slučaju nije (i ne smije biti) uvjetovano primitkom poruke od strane bilo koje druge komponente sustava. Komponenta može primati poruke od drugih komponenti sustava u svrhu implementacije dodatne funkcionalnosti, ali one ne smiju utjecati na osnovnu funkciju periodičkog slanja poruka.

Aktivnosti objekata raspodijeljenih sustava temeljenih na CORBA-i većinom su posljedica reakcije na događaje unutar sustava (poglavito primitak poziva metode) a manje na unutarnje događaje samog objekta. Izvršavanje objekta (implementacije objekta) vrši se unutar osnovne izvršne okoline računalnog sustava u kojoj je objekt implementiran. Valja primijetiti da velik broj objekata može dijeliti istu izvršnu okolinu, a što je u implementacijama sustava temeljenih na CORBA-i najčešće i slučaj.

Komunikacija elemenata sustava, kako smjer komunikacije (orijentacija grana), tako i grupe elemenata koje međusobno komuniciraju (susjedni čvorovi čvora V_i), određeni su algoritmom kojeg asinkroni mrežni sustav implementira (programskim dijelom sustava) i mogućnostima komunikacijskih kanala sustava (sklopovskim dijelom sustava).

Komunikacije između posredničkih komponenti sustava CORBA temeljena je na protokolu IIOP, implementaciji aplikacijskog protokola nad protokolom TCP. Uz poznata svojstva protokola TCP možemo ustvrditi da posrednička komponenta predstavlja ekvivalent pouzdanih FIFO komunikacijskih kanala u asinkronom mrežnom modelu. Također, posrednička komponenta omogućava transparentni komunikacijski kanal sa svim dostupnim objektima (pod pojmom dostupnosti podrazumijeva se dostupnost reference ciljnog objekta). Stoga se može ustvrditi da je graf $G = (V, E)$, koji predstavlja raspodijeljeni sustav temeljen na CORBA-i, snažno povezan neusmjeren graf. Izuzetak od ovog pravila čine, uvedeni od inačice standarda CORBA 3, lokalni objekti, s kojima je moguća komunikacija samo unutar iste izvršne okoline.

6.2.2. Praćenje rada raspodijeljenog sustava

Pristup objektu sustava CORBA kao “crnoj kutiji” onemogućava opažanje unutarnjih događaja i stanja komponenata, a time i korištenje izvršavanja, odnosno odsječaka izvršavanja, u opisu rada sustava. No praćenje izvršavanja moguće je realizirati

korištenjem postupaka temeljenih na tragovima izvršavanja kompozicije asinkronog mrežnog sustava.

Trag izvršavanja β sustava A čini niz vanjskih događaja sustava (elemenata skupa *ext(A)*) poredan po njihovu kronološkom slijedu pojavljivanja. Također, $\beta|A_i$ čini trag izvršavanja elementa A_i sustava A, tj. lokalni trag izvršavanja sustava. Vrijedi i suprotno: pridržavanjem pravila međuovisnosti događaja moguće je iz lokalnih tragova izvršavanja rekonstruirati globalni trag, nerazlučiv od stvarnog kronološki temeljenog traga izvršavanja.

Mogućnost rekonstrukcije globalnog traga izvršavanja sustava na osnovu lokalnih tragova pojedinih elemenata iznimno je bitna u raspodijeljenim sustavima temeljenim na CORBA-i iz dva razloga:

- detekcija pojave događaja (poziva metode udaljenog objekta) moguća je isključivo unutar izvršne okoline objekta
- skup elemenata koji čine sustav ne mora biti stalan niti s prethodno poznatom topologijom (izgledom grafa G sustava).

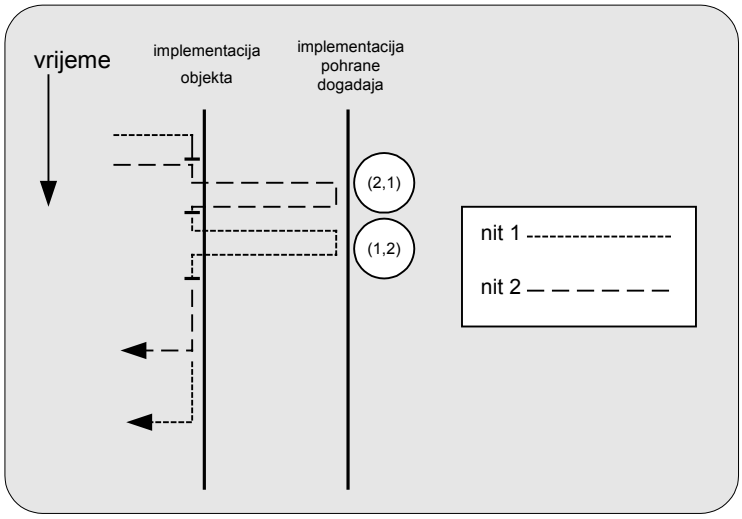
Lokalno dobavljanje tragova izvršavanja, njihovo dohvaćanje i rekonstrukcija (po potrebi na jednom ili na više mjesta u sustavu) daju praćenju izvršavanja potrebnu prilagodljivost. Najveći problem praćenja traga izvršavanja sustava kompozicijom lokalnih tragova izvršavanja predstavlja parcijalna uređenost događaja unutar traga, kako unutar lokalnih tragova, tako i prilikom njihove kompozicije u globalni trag izvršavanja.

Na prvi pogled kronološka uređenost slijeda lokalnih događaja elementa sustava temeljenog na CORBA-i ne bi trebala predstavljati problem. Svaki pojedini događaj u trenutku pojave dodavao bi se na kraj (hipotetskog) reda događaja, formirajući ispravan trag lokalnog izvršavanja sustava. No potrebno je uzeti u obzir i različite modele izvršavanja objekata s obzirom na pristigle pozive metoda, u kojima je moguća aktivnost više niti unutar pojedine implementacije objekta. U određenom trenutku, kao posljedica isključivanja rada jedne i nastavljanje rada druge niti, moguća je zamjena redoslijeda dodavanja događaja u red događaja (slika 6.4).

Relativno jednostavno rješenje problema zamjene reda događaja moguće je na dva načina:

- uvođenjem sinkronizacije rada niti u kritičnom odsječku dodavanja u red događaja
- uvođenjem vremenske komponente događaja.

Sinkronizacija niti prilikom dodavanja događaja u red događaja bilo bi univerzalno rješenje za proizvoljan broj aktivnih niti, no kao posljedicu imalo bi značajno usporenje izvršavanja programa kod iole većeg broja aktivnih niti.



Slika 6.4. Zamjena redoslijeda događaja

Uvođenjem novog elementa kao člana reda događaja (*e* , *t*) u kojem *e* označava događaj, a *t* apsolutno vrijeme trenutka u kojem se događaj pojavio, izbjegla bi se nužnost uporabe sinkronizacije niti, a time i utjecaja na izvršavanje programa. No kod uporabe apsolutnog vremena javljaju se problemi jedinstvenog prikaza vremena na više računalnih platformi i operacijskih sustava. U raspodijeljenim sustavima temeljenim na CORBA-i ovaj problem riješen je uvođenjem usluge vremena (eng. *Time Service*) koja u okruženju CORBA standardizira prikaz i operacije s vremenom. Dodatni problem uvodi razlučivost lokalne implementacije sata sustava. Na standardnim osobnim računalima (arhitektura Intel) rezolucija lokalnog sata je 1 ms, što je za ozbiljniju uporabu nedostatno.

Korištenje apsolutnog vremena rezultiralo bi grupama događaja s istim pridijeljenim vremenom, za koje bi bilo moguće odrediti samo odnos prema drugim grupama događaja, različitih apsolutnih vremena.

U rekonstrukciji globalnog traga izvršavanja sustava, osim problema rezolucije apsolutnog vremena, javio bi se i problem usklađivanja lokalnih satova. Iako postoje protokoli za usklađivanje lokalnih satova umreženih računala, ne garantira se usklađenost dostatna za rješavanje opisanog problema.

Kao posljedica navedenih problema u označavanju redoslijeda događaja, apsolutno vrijeme koristi se samo za određivanje lokalnog odnosa pojave događaja, ali ne i apsolutnog poretka događaja na razini raspodijeljenog sustava.

Primjena Lamportovog algoritma (odnosno transformacije) logičkog vremena predstavlja rješenje problema označavanja redoslijeda događaja, kako lokalno, tako i globalno, na razini čitavog sustava. Po definiciji Lamportove transformacije, da bi asinkroni mrežni sustav *A* podržavao Lamportovo logičko vrijeme potrebno je izvršiti transformaciju *L(A)* složenog sustava, što ima za posljedicu transformaciju $L(A_i)$ svakog pojedinog elementa A_i složenog sustava.

Primjena Lamportove transformacije na raspodijeljeni računalni sustav temeljen na CORBA-i podrazumijeva transformaciju svakog pojedinog objekta raspodijeljenog sustava tako da njegova implementacija:

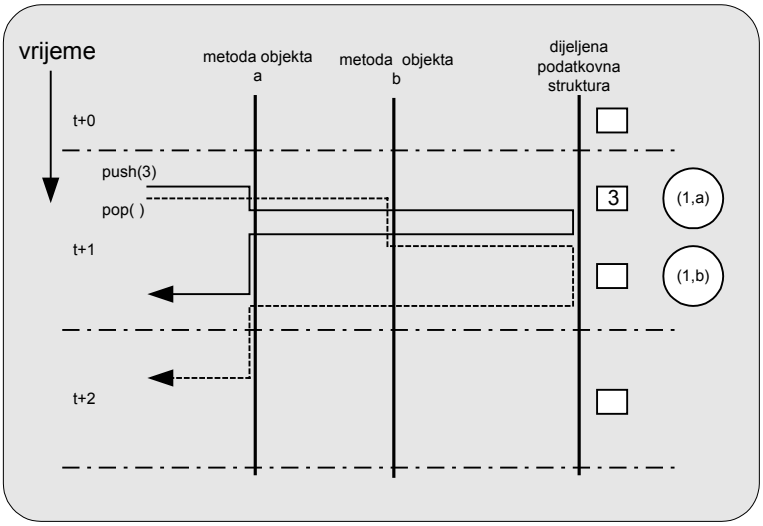
- sadrži lokalni logički sat
- prilikom komunikacije izmjenjuje podatke o lokalnom logičkom vremenu s ostalim objektima članovima sustava.

Implementacija lokalnog logičkog sata za svaki objekt unutar raspodijeljenog sustava uvodi dodatne probleme koji nisu bili vidljivi tijekom prethodne analize mogućih rješenja praćenja rada sustava temeljenih na CORBA-i: količinu resursa koja je potrebna za implementaciju lokalnog logičkog vremena i svojstvo grupiranja objekata sustava.

Broj objekata tijekom životnog vijeka sustava nije ograničen. Trenutni broj objekata može biti ograničen raspoloživim resursima sustava (npr. broj zapisa u bazi podataka). Gledano kroz dulje vremensko razdoblje i životni vijek objekata, u proizvoljno dugom izvršavanju sustava broj objekata može biti neograničeno velik. Ova činjenica bitno utječe na zapis lokalnog logičkog vremena.

Svaki objekt unutar sustava tijekom svojeg životnog vijeka troši prostor za pohranu vrijednosti logičkog sata (*c, i*). Količina potrebne memorije za pohranu lokalnih logičkih vremena u sustavu raste linearno s brojem objekata u sustavu. Gornja granica varijable *i* (jedinstvenog identifikatora objekta) nije ograničena. U stvarnoj implementaciji sustava gornja granica određena je veličinom predviđenog memorijskog prostora za pohranu vrijednosti, a koji mora biti dovoljan za pohranu očekivanog (velikog) broja objekata u sustavu.

Svojstvo grupiranja objekata unutar izvršne okoline (procesa) predstavlja drugi značajan problem kod implementacije lokalnih logičkih satova. Promotrimo sljedeći primjer (slika 6.5):



Slika 6.5. Grupiranje objekata unutar istog procesa s logičkim satom na razini objekta

Objekti a i b se nalaze unutar procesa P koji čini njihovu izvršnu okolinu na nekom umreženom računalu. Oba objekta dijele strukturu podataka u obliku stoga.. Neka je nad objektom a pozvana metoda push(3), a nad objektom b metoda pop(). Neka su se oba događaja pojavila unutar vremenskog razmaka manjeg od rezolucije lokalnog sata. Koje je stanje dijeljene strukture podataka promatrano izvana, korištenjem traga izvršavanja sustava?

Postoje dva međusobno nerazlučiva traga izvršavanja:

1) (push, (1, a)), (pop, (1,b))

2) (pop, (1,b)), (push, (1,a))

ali koji daju različite odgovore na postavljeno pitanje:

1) (stog je prazan)

2) 3

U slučaju objekata unutar istih izvršnih okolina, implementacija lokalnog logičkog vremena na razini objekta ne može prikazati apsolutni redoslijed događaja na razini izvršne okoline. Nedostatak nije izražen ako nema međuovisnosti između objekata unutar izvršne okoline, no pristup ne zadovoljava kriterij univerzalnosti metode.

Kao posljedica neograničenosti broja objekata u sustavu i moguće međuovisnosti objekata unutar izvršne okoline, model raspodijeljenog računalnog sustava temeljenog na CORBA-i zahtijeva promjene definicije čvorova grafa *G* raspodijeljenog sustava:

- čvorovi V_i grafa *G* su izvršne okoline (procesa) objekata CORBA temeljenog sustava

- skup ulaznih događaja izvršne okoline (procesa) $in(A_i)$ čini unija skupova ulaznih događaja svih objekata sadržanih izvršnom okolinom

- skup izlaznih događaja izvršne okoline (procesa) $out(A_i)$ čini unija skupova izlaznih događaja svih objekata sadržanih izvršnom okolinom.

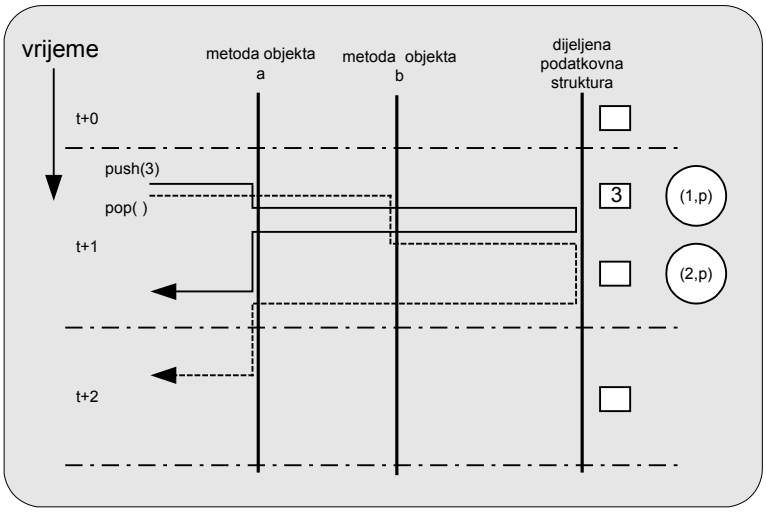
Posljedice ovako definiranog modela na implementaciju Lamportovog algoritma logičkog vremena sustava su:

- postoji samo jedan lokalni logički sat za sve objekte unutar izvršne okoline

- lokalno logičko vrijeme pokazuje apsolutni redoslijed događaja svih objekata unutar izvršne okoline.

Postojanje samo jednog lokalnog logičkog sata za neograničeni broj objekata koji se za svog životnog vijeka mogu nalaziti unutar izvršne okoline znatno umanjuje potrebne resurse sustava. Interakcije između objekata unutar iste izvršne okoline ne mogu se izravno prikazati u tragu izvršavanja sustava (stoga što ne koriste standardne komunikacijske kanale), no može se doći do određenih zaključaka s obzirom na apsolutni poredak događaja unutar izvršne okoline kao cjeline.

Kao rezultat izvršavanja prethodnog primjera (slika 6.6) s logičkim satom implementiranim na razini izvršne okoline tragovi izvršavanja bili bi:



Slika 6.6. Grupiranje objekata unutar istog procesa s logičkim satom na razini procesa

1) (push, (1, p)), (pop, (2,p)) ili

2) (pop, (1,p)), (push, (2,p))

nakon čega bi bilo jednostavno odrediti trenutno stanje dijeljene podatkovne strukture.

Drugi element potreban za implementaciju Lamportovog algoritma logičkog vremena prosljeđivanje je lokalnog vremena tijekom komunikacije dvaju entiteta sustava. U prethodnim tekstu izbjegnut je opis konkretnog načina implementacije izloženih ideja, većinom stoga što njihov izbor ne uvjetuje građu čitavog sustava. No u razmatranju načina transparentnog prenošenja vrijednosti logičkog vremena izbor mehanizma implementacije od presudne je važnosti za implementaciju i uporabivost čitavog sustava.

Sinkroni poziv metode udaljenog objekta sastoji se od dva koraka:

- poziva metode i prenošenja ulaznih i ulazno-izlaznih parametara
- vraćanja povratne vrijednosti metode, izlaznih i ulazno-izlaznih parametara poziva.

U svakom od ta dva koraka Lamportov algoritam zahtijeva i prosljeđivanje lokalnog logičkog vremena: kod poziva metode logičkog vremena objekta pozivatelja, a kod vraćanja povratne vrijednosti lokalnog logičkog vremena pozvanog objekta. Tri su načina prosljeđivanja dodatnih podataka moguća u sustavu CORBA:

- dodavanje parametara svim metodama svih sučelja unutar sustava (na razini opisa sučelja u jeziku IDL) s jednim ulazno-izlaznim parametrom
- prilagodba *stub* i *skeleton* objekata
- uporaba prenosivih presretača (eng. P*ortable Interceptors*).

Promjena definicije sučelja (naročito kod korištenih sustava) izrazito je nepoželjna jer zahtjeva promjene i na strani poslužitelja i na strani korisnika.

Prilagodba *stub* i *skeleton* objekata moguća je, ali uz sljedeća ograničenja:

- mora biti dostupan njihov izvorni kod
- nakon svake promjene definicije sučelja mijenja se pripadajući izvorni kod stub i skeleton razreda, te se prilagodba koda mora vršiti nakon svakog prevođenja IDL definicije sučelja
- izvorni kod stub i skeleton razreda snažno je ovisan o korištenoj implementaciji CORBA-e
- obje strane komunikacije moraju podržavati dodatne parametre poziva.

Ovaj pristup zahtijevao bi razvoj dodatnih programskih alata za svaku implementaciju sustava CORBA posebno, te bi unio dodatni korak u razvoj aplikacije. Prilagodljivost praćenja izvršavanja tako implementiranog sustava ne bi bila zadovoljavajuća.

Treći, najpogodniji, način prenošenja dodatnih parametara poziva uporaba je jednog od mehanizama meta programiranja u sustavu CORBA [6] - prenosivim presretačima [11]. Prenosivi presretači omogućuju slanje proizvoljnih podataka kao dodatka svakom pozivu metode u vidu okruženja usluge (eng. *service context*). Uporaba okruženja usluge za prijenos vrijednosti lokalnog logičkog sata ima sljedeća svojstva:

- nije potrebna nikakva promjena definicija sučelja ili izvornog koda razreda stvorenog od strane IDL prevodioca
- moguće je jednostavno dodavanje već postojećim sustavima
- nije nužno da su sve komponente raspodijeljenog sustava prošle Lamportovu transformaciju.

Model sustava temeljenog na CORBA-i uporabom prenosivih presretača, uz neka ograničenja, odgovara modelu asinkronog mrežnog sustava u kojemu su čvorovi V_i grafa G izvršne okoline objekata:

- prenosivi presretači definiraju se na razini posredničke komponente (pretpostavimo jednu posredničku komponentu po izvršnoj okolini)

- točke presretanja zajedničke su svim objektima korisnicima iste posredničke komponente.

Prenosivi presretači definiraju se na razini posredničke komponente. U tipičnoj raspodijeljenoj aplikaciji temeljenoj na sustavu CORBA, unutar izvršne okoline aktivna je samo jedna posrednička komponenta. Prenosivi presretač zadužen je za presretanje poziva metoda samo onih objekata dodijeljenih posredničkoj komponenti kojoj je spomenuti presretač pridjeljen. Moguće je koristiti više od jedne posredničke komponente unutar pojedinog procesa, te svakoj od njih dodijeliti poseban presretač. Kao posljedica korištenja više od jedne posredničke komponente (a time i presretača) u istom procesu može se javiti problem apsolutnog poretka događaja unutar izvršne okoline. Stoga, ukoliko je unutar istog procesa aktivno više posredničkih komponenti, potrebno je osigurati da objekti korisnici različitih komponenti nemaju skrivenih međuovisnosti (dijeljenih struktura podataka i sl.).

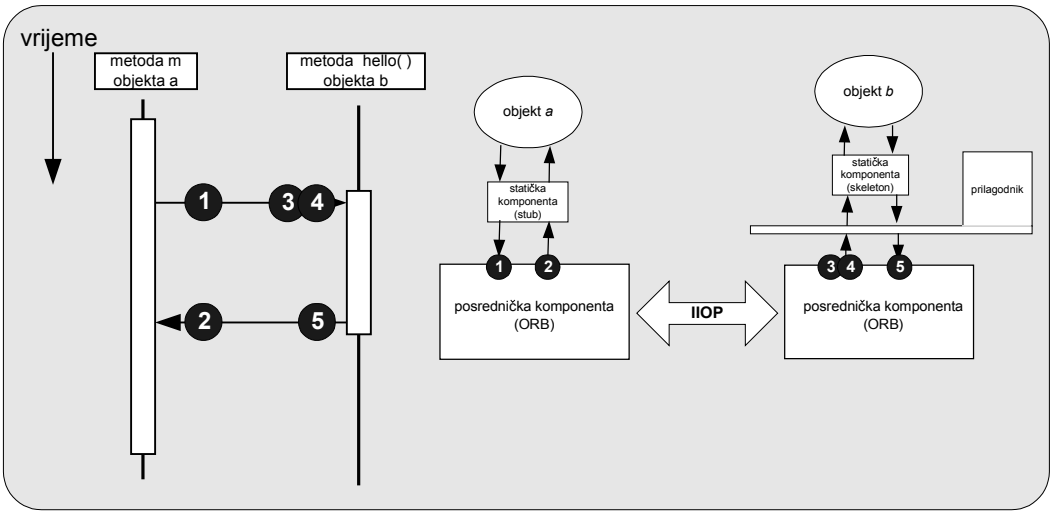
Prenosivi presretači implementiraju dva skupa meta operacija nad svim objektima V_i korisnicima ORB-a kojem je presretač dodijeljen:

- skup izlaznih događaja $out(V_i)$ (send_request, send_poll, send_reply, send_exception, send_other)

- skup ulaznih događaja $in(V_i)$ (receive_reply, receive_exception, receive_other, receive_request, receive_poll).

Na strani posredničke komponente objekta inicijatora komunikacije prenosivi presretač jamči (uz određene iznimke) poziv jedne od meta operacija iz skupa izlaznih događaja prije samog slanja poziva udaljenom objektu, te poziv jedne od meta operacija neposredno prije prosljeđivanja odgovora objektu pozivatelju. Na strani posredničke komponente pozvanog objekta prenosivi presretač jamči poziv jedne od meta operacija iz skupa ulaznih događaja prije poziva ciljne metode, te poziv jedne od meta operacija iz skupa izlaznih događaja neposredno nakon slanja odgovora objektu pozivatelju (također uz određene iznimke).

Neka se raspodijeljeni računalni sustav A temeljen na CORBA-i sastoji od dva objekta, *a* i *b*, smještena na različitim računalima u lokalnoj mreži. Neka objekt *a* poziva metodu hello( ) objekta *b*. Primjer traga izvršavanja sustava je:



Slika 6.7. Dobivanje traga izvršavanja sustava korištenjem presretača

(1) send_request(a, hello),
(4) receive_request(b, hello),
(5) send_reply(b, hello),
(2) receive_reply(a, hello)

Implementacija Lamportove transformacije sustava A uporabom prenosivih presretača temelji se na dodavanju programskog koda u svakoj točki presretanja sa zadaćom (slika 6.7):

- skup točaka presretanja izlaznih događaja:

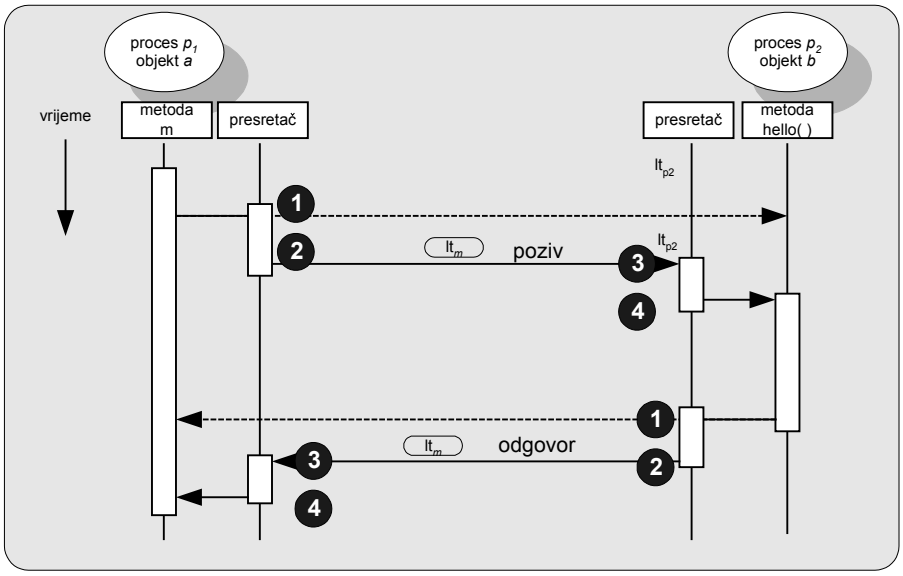
  (1) povećanja vrijednosti lokalnog logičkog sata za jedan

  (2) pohrane trenutne vrijednosti lokalnog logičkog sata u okruženje usluge poziva

- skup točaka presretanja ulaznih događaja:

  (3) dobavljanja vrijednosti logičkog vremena pridjeljenog poruci

  (4) povećanja vrijednosti lokalnog logičkog sata (u ovisnosti o logičkom vremenu pridjeljenog poruci.



Slika 6.8. Implementacija Lamportovog algoritma korištenjem presretača

Neka su vrijednosti lokalnih logičkih satova procesa p_1 i p_2 neposredno prije poziva metode lt_{p1} i lt_{p2}. Vrijednost logičkog vremena proslijeđena pozivom označena je s lt_m. Vrijednosti u točkama prikazanim na slici mijenjaju se:

(1) $lt_{p1} = lt_{p1} + 1$

(2) $lt_m = lt_{p1}$

(3) $lt_{p2} > lt_m$ (nema akcije)

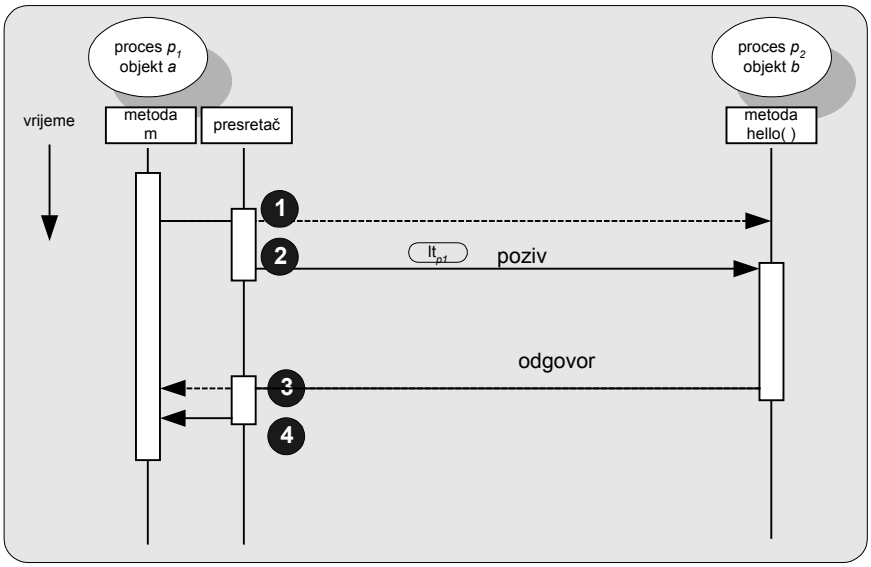
$lt_{p2} =$

$lt_{p2} <= lt_m$: $lt_{p2} = lt_m$

(4) $lt_{p2} = lt_{p2} + 1$

Ista pravila vrijede i prilikom slanja odgovora objekta *b* objektu pozivatelju *a*.

Ukoliko na jednoj od strana komunikacije nije implementiran mehanizam logičkih satova (slika 6.9), u točki (3) presretanja poziva presretač na strani objekta pozivatelja neće primiti povratno okruženje usluge, te će vrijednost lokalnog logičkog sata u njoj ostati nepromijenjena.



Slika 6.9. Asimetrično implementiran Lamportov algoritam

Implicitne operacije nad objektima sustava CORBA:

- get_interface( )
- is_a( )
- non_existent( )
- get_domain_managers( )
- get_component( )

mogu i ne moraju za posljedicu imati komunikaciju među komponentama sustava CORBA, a time i rezultirati pozivima operacija prenosivih presretača. Odluka da li navedene akcije promatrati kao i svaki drugi poziv metode udaljenog objekta ili ne prepušta se konkretnoj implementaciji Lamportovog algoritma.

Bitno svojstvo presretača je mogućnost promjene vrste ishoda udaljenog poziva. Presretač ima mogućnost :

- valjan ishod poziva pretvoriti u poziv čiji ishod rezultira iznimkom
- promijeniti vrstu iznimke
- proslijediti poziv drugom objektu transparentno s obzirom na objekt pozivatelj.

ORB predviđa registraciju više presretača, no ne definira njihov redoslijed pozivanja. Ukoliko je, osim presretača u kojem je implementiran Lamportov algoritam, registriran još neki presretač, mora se u obzir uzeti njegov mogući utjecaj na rad

sustava. Poziv udaljenog objekta neće biti detektiran od strane presretača koji implementira Lamportov algoritam u slučaju:

- podizanje iznimke u izlaznom putu poziva udaljenog objekta na strani objekta pozivatelja, u presretaču čije izvršavanje prethodi izvršavanju presretača u kojem je implementiran Lamportov algoritam

- podizanje iznimke u ulaznom putu poziva udaljenog objekta na strani pozvanog objekta, u presretaču čije izvršavanje prethodi izvršavanju presretača u kojem je implementiran Lamportov algoritam

- preusmjeravanju poziva u ulaznom putu poziva udaljenog objekta na strani pozvanog objekta, u presretaču čije izvršavanje prethodi izvršavanju presretača u kojem je implementiran Lamportov algoritam.

U prvom slučaju poziv udaljenog objekta neće biti tretiran kao događaj, te vrijednost lokalnog logičkog sata neće biti promijenjena. Utjecaj ove anomalije nije koban za rekonstrukciju izvršnog traga sustava, ali on neće u potpunosti odgovarati stvarnom izvršavanju sustava.

U drugom i trećem slučaju poziv udaljenog objekta na strani objekta pozivatelja bit će pravilno detektiran, s iznimkom kao rezultatom poziva. Na strani pozvanog objekta poziv neće biti detektiran, što je i ispravan ishod, jer poziv nikad i nije dosegao implementaciju pozvanog objekta.

6.2.3. Ostali promatrani događaji u sustavu

Uz komunikaciju između objekata, u sustavima CORBA sljedeći su događaji, neovisno o pojedinoj aplikaciji, zanimljivi za promatranje:

- događaji vezani za životni vijek objekata (stvaranje i uništavanje objekata)
- događaji vezani za životni vijek implementacija objekata (stvaranje i uništavanje implementacija objekata)
- događaji vezani za životni vijek procesa (početak i kraj rada izvršne okoline objekata).

Osnovno je pitanje da li se ti događaji mogu pratiti transparentno s obzirom na programski kod sustava, tj. da li se promjene nužne za praćenje ovih događaja mogu lokalizirati na meta razini sustava. Dva su osnovna preduvjeta koja kvalificiraju nabrojene događaje za njihovo transparentno praćenje, ubrajajući ih u događaje svojstvene CORBA-i, a ne svojstvene pojedinoj aplikaciji:

- determinizam njihova pojavljivanja i povezanosti
- postojanje mehanizama njihove detekcije u meta razini sustava.

6.2.4. Životni vijek entiteta sustava

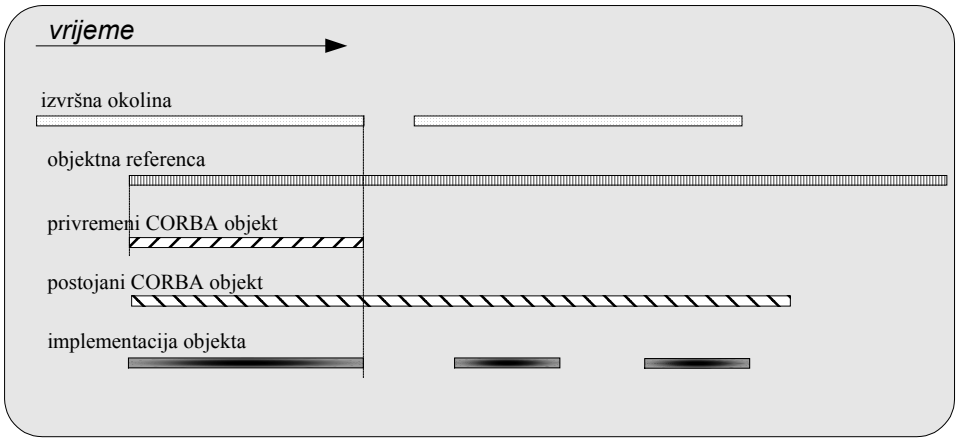
Za analizu životnog vijeka objekata sustava CORBA potrebno je promatrati tri funkcionalno usko povezana entiteta: objektne reference, objekte i implementacije objekata. Objektne reference i objekti pripadaju virtualnom sloju, dok implementacije objekata pripadaju sloju implementacije sustava. Životni vijek tri navedena entiteta može, ali ne mora nužno biti povezan.

Gledano u virtualnom sloju, objektna referenca, kao i u programskim jezicima, pokazuje na objekt koji se želi koristiti (pozivati njegove metode). Referenca može pokazivati na valjani objekt, na nepostojeći objekt ili ne pokazivati ni na jedan objekt (tzv. *nul* referenca). Gledano u sloju implementacije, objektna referenca služi prilagodniku objekata za pronalaženje implementacije objekta na koji pokazuje. Trenutak stvaranja objektne reference u većini slučajeva vezan je s tipom objekta kojemu je pridružena. Reference na postojane objekte, u većini slučajeva, stvaraju se potpuno neovisno o stvaranju implementacije objekta. Zadatak je prilagodnika objekata da u trenutku poziva metode ciljnog objekta, ukoliko nije pokrenuta, pokrene implementaciju i proslijedi joj primljeni poziv. Prilagodnik objekata ili implementacija objekta odlučuju da li objekt na koji referenca pokazuje postoji ili ne. Postojanje implementacije objekta ne podrazumijeva i postojanje samog objekta – jedna implementacija može opsluživati više objekata. Reference na privremene objekte čvršće su vezane za životni vijek implementacije. Stvaraju se, u većini slučajeva, u trenutku stvaranja implementacije privremenog objekta, te prestaju biti važeće u trenutku prestanka postojanja implementacije – ili u trenutku kada se implementacija eksplicitno deaktivira ili u trenutku prestanka rada izvršne okoline u kojoj se implementacija nalazi (završetak rada procesa).

Stoga, općenito gledano, ne postoji veza između životnog vijeka entiteta unutar virtualnog sloja sustava CORBA. Životni vijek reference proteže se od trenutka kad je stvorena do trenutka kad su svi njeni zapisi (u bilo kojem obliku postojali) izgubljeni. Životni vijek objekta, na kojeg objektna referenca pokazuje, počinje ne ranije od trenutka stvaranja pridružene reference. Privremeni objekti završavaju svoj životni vijek relativno brzo, ne kasnije od kraja životnog vijeka njihove izvršne okoline. Postojani objekti, za razliku od privremenih, imaju životni vijek potpuno neovisan o životnom vijeku njihove izvršne okoline, te je njihov prestanak postojanja uglavnom rezultat eksplicitne akcije. No sam trenutak prestanka postojanja postojanog objekta ne mora biti posljedica akcije unutar sustava. Na primjer, ako su stanja objekta

pohranjena u obliku zapisa u bazi podataka, a zapis bude izbrisan ili akcijom administratora ili pogreškom, objekt će prestati postojati, ali trenutak njegova nestanka neće biti moguće detektirati unutar promatranog sustava.

U sloju implementacije očituje se još veća dinamika životnog vijeka entiteta. Životni vijek implementacije privremenih objekata određuje, mada ne potpuno, životni vijek privremenih objekata. Treba primijetiti da se implementacije privremenih objekata mogu aktivirati i deaktivirati (proći cijeli životni ciklus) proizvoljan broj puta tijekom jednog životnog ciklusa izvršne okoline. Isto vrijedi i za implementacije postojanih objekta, no njihove aktivacije i deaktivacije nisu vezane uz jedan životni ciklus izvršne okoline.



Slika 6.10. Prikaz (uobičajenog) životnog vijeka entiteta u sustavu CORBA

Mehanizmi meta razine na raspolaganju za detekciju nabrojenih događaja skromni su. Presretač stvaranja interoperabilnih (IOR) referenci nudi mogućnost presretanja događaja stvaranja reference, a time i (uvjetno rečeno) objekta. No problem koji unosi nesigurnost u korištenje ovog mehanizma je neodređenost standarda u pogledu točnog trenutka poziva ovog presretača, što za posljedicu ima ovisnost o pojedinoj implementaciji. Time je ovaj mehanizam isključen iz daljnjeg razmatranja. Trenutak nestanka objekta potpuno je ovisan o implementaciji sustava ili je posljedica akcija izvan sustava, te je zaključeno da njegova sigurna i pravilna detekcija nekim općenitim mehanizmom nije moguća.

Za presretanje događaja stvaranja i uništavanja ne postoji ni jedan standardan mehanizam. Mogući su zahvati na dvije razine:

- na, uvjetno rečeno, meta razini – upravljačima implementacija
- na razini programskog koda implementacije objekata.

Kao što je već navedeno prilikom opisa mehanizama meta razine u sustavu CORBA, upravljači objekata mogu se svrstati u meta razinu, no oni su ipak vezani na pojedinu implementaciju raspodijeljenog sustava. Stoga, korištenje ovog mehanizma nije cjelovito rješenje problema.

Slični problemi pojavljuju se kod promatranja životnog vijeka izvršne okoline objekata – procesa u računalnom sustavu. Životni vijek posredničke komponente je, u većini slučajeva, čvrsto vezan s životnim vijekom procesa u kojem se koristi. No njihovo poistovjećivanje priječe dvije činjenice:

- posrednička komponenta ne mora nužno biti stvorena na početku rada procesa, te ne mora nužno biti uništena neposredno prije kraja rada procesa
- može postojati više posredničkih komponenti unutar jednog procesa.

Kao zaključak izvodi se da je zaključivanje o životnom vijeku procesa na osnovi promatranje životnog vijeka posredničke komponente moguće samo uz poznavanje nadgledanog procesa, te je ovisno o aplikaciji.

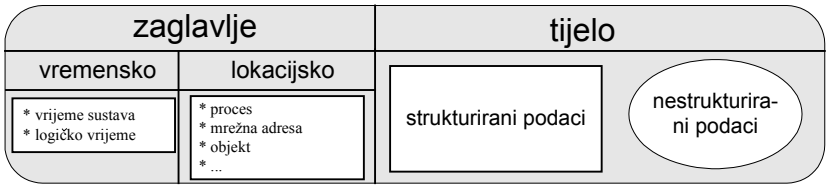
Jedini pravilan način promatranja svih dosad nabrojenih događaja je kao događaja svojstvenih promatranoj aplikaciji. Stoga je za njihovo promatranje potrebno:

- poznavati građu promatrane aplikacije
- mijenjati programski kod aplikacije u svrhu detekcije promatranih događaja.

6.2.5. Poruke o događajima

Događaj unutar nadgledanog sustava virtualni je entitet. Zadaća nadglednog sustava je detekcija i stvaranje odgovarajućeg opisa događaja - poruka, a kojeg je moguće prenijeti njegovim potencijalnim korisnicima. Poruke su određene formatom i podacima koje prenose. U daljnjem tekstu pozornost će biti usmjerena na prenošene podatke, a sam format poruka, ovisan o komunikacijskom mehanizmu, bit će detaljno objašnjen u poglavlju o implementaciji nadglednog sustava.

Događaji unutar sustava određeni su trenutkom i mjestom njihove pojave. Stoga ove dvije grupe podataka čine podatke zajedničke svim događajima u sustavu, bez obzira na njihov tip. Ostali podatci o događaju svojstveni su svakom pojedinom tipu događaja. Poruke, sukladno njihovoj ulozi, slijede prethodno navedena svojstva događaja. Podatci u poruci čine dva odvojena dijela. Zajednički podatci čine zaglavlje poruke, a podatci svojstvene tipu događaja tvore tijelo poruke. Zaglavlje poruke prenosi dvije grupe podataka: podatke o vremenu nastanka događaja i podatke o okruženju u kojem se događaj pojavio (slika 6.11).



Slika 6.11. Struktura poruke

Dva su podatka bitna za određivanje nastanka događaja, a opisana su u prethodnom tekstu o praćenju komunikacije između komponenti raspodijeljenog sustava:

- *vrijeme sustava* – apsolutna vrijednost fizičkog sata unutar sustava (računala) u trenutku pojave događaja
- *logičko vrijeme* – relativna vrijednost lokalnog logičkog sata (procesa) u trenutku pojave događaja.

Podatak o logičkom vremenu trenutka pojave događaja služi za rekonstrukciju parcijalnog niza događaja kako u lokalnom procesu tako i u raspodijeljenom sustavu u cjelini. Vrijeme sustava određuje stvarni trenutak pojave događaja u vremenu, tako i za izvođenje različitih parametara kvalitete usluga (npr. trajanje poziva metode udaljenog objekta, vremena potrebnog za aktiviranje neaktivne implementacije objekta, kašnjenja uzrokovana mrežnom komunikacijom, itd.).

Za točno određivanje lokacije pojave događaja unutar raspodijeljenog sustava potrebno je više hijerarhijski organiziranih podataka :

- *ime objekta* za čijeg izvršavanja se pojavio događaj
- *ime prilagodnika objekata* kojemu objekt pripada
- *oznaka niti* za čijeg izvršavanja se događaj pojavio
- *oznaka posredničke komponente* kojoj prilagodnik objekata pripada
- *oznaka procesa* u kojemu se događaj pojavio
- *oznaka aplikacije ili komponente* u kojoj se događaj pojavio
- *mrežna adresa računala* na kojoj se izvršava proces.

Ime objekta i ime prilagodnika objekata kojem objekt pripada ne mora biti dostupno stoga što korisnik objekta ne mora biti objekt, već i "obična" aplikacija. Stoga je uveden i podatak o aplikaciji (ili komponenti) izvoru događaja. Podatak o niti izvršavanja može poslužiti u promatranju raspodijele aktivnih niti (prilikom dodjele niti iz fiksnog broja niti predviđene za obradu poziva – eng. *thread pool*) po aktivnim pozivima. Rad s nitima u većini je slučajeva pod čvrstom kontrolom posredničke komponente, ali djelomično može biti pod kontrolom implementacije komponenti

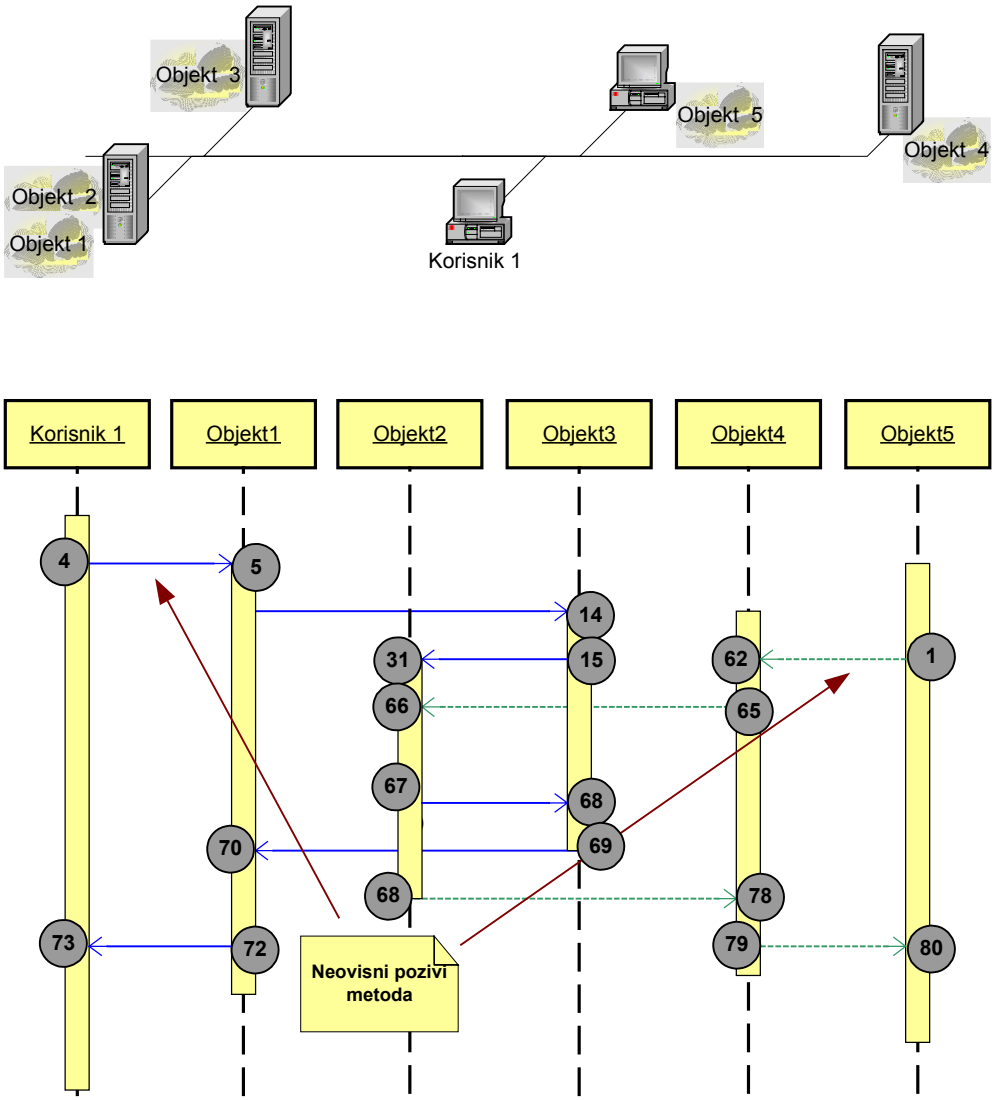
raspodijeljenog sustava). Uobičajeno je, ali nije nužno, da postoji samo jedna posrednička komponenta po procesu. Oznaka posredničke komponente korisna je u slučaju više posredničkih komponenti aktivnih unutar jednog procesa. Oznaka procesa u kojem se događaj pojavio jedinstveno određuje proces unutar računalnog sustava u kojem se događaj pojavio. Označavanje procesa nužno je u slučaju da su na istom računalu pokrenute dvije ili više istih aplikacija. Mrežna adresa računala jedinstveno određuje računalo na kojem se proces izvodi. Treba primijetiti da adresa računala ovisi o korištenom mrežnom protokolu – u slučaju korištenja protokola IIOP, tj. TCP/IP temeljene komunikacije, adresa računala je ili simbolička (npr. igor.rasip.fer.hr) ili brojčana (npr. 161.53.67.42).

Podatak koji je također nužno uključiti u svaku poruku je tip poruke, odnosno tip događaja koje poruka opisuje. O tipu poruke ovisi interpretacija podataka smještenih u tijelu poruke. Ostali podaci koje je moguće, ali nije nužno, uključiti su ovisni o samoj implementaciji nadgledne usluge, npr. inačica implementacije poruke u svrhu udruživosti novih i starijih implementacija usluge.

Tijelo poruke prenosi podatke svojstvene pojedinom tipu događaja, te ovdje neće biti opisivano. Podaci u tijelu poruke dijele se na strukturirane (parovi ime – vrijednost) i nestrukturirane (pohranjene u obliku bloka podataka). Za pojedine predefinirane tipove poruka (npr. poruke o komunikacijskim događajima, pokretanju posredničke komponente, itd.) podaci u tijelu poruke strogo su definirani (u ovisnosti o inačici), dok je za poruke svojstvene pojedinoj aplikaciji broj i vrsta podataka ostavljena na volju korisnicima usluge.

6.2.6. Praćenje niti izvršavanja unutar raspodijeljenog sustava

Korištenjem logičkog vremena za označavanje događaja unutar raspodijeljenog sustava omogućeno je promatranje njihovog stvarnog redoslijeda na razini raspodijeljenog sustava u cjelini. No u određenim slučajevima potrebno je promatrati samo određenu grupu uzročno-posljedično povezanih događaja nastalih kao posljedica poziva metode udaljenog objekta. Uređen niz događaja nastalih unutar komponenti raspodijeljenog sustava kao posljedice *neovisnog poziva metode* udaljenog objekta u daljnjem tekstu bit će označavan kao *nit izvršavanja raspodijeljenog sustava*. *Neovisan poziv metode* udaljenog objekta označava poziv metode ciljnog objekta, a čiji uzrok nije neovisna pobuda sustava. Kao primjere neovisnih poziva metoda ubrajamo pozive uzrokovane istekom vremenskog perioda, pozive uzrokovane akcijom na korisničkom sučelju programa korisnika, itd. Na slici 6.12 prikazane su dvije niti izvršavanja unutar raspodijeljenog sustava s pripadnim komunikacijskim događajima i njihovim logičkim vremenima. Jednu nit pokreće korisnička aplikacija (nije objekt), a drugu Objekt5.

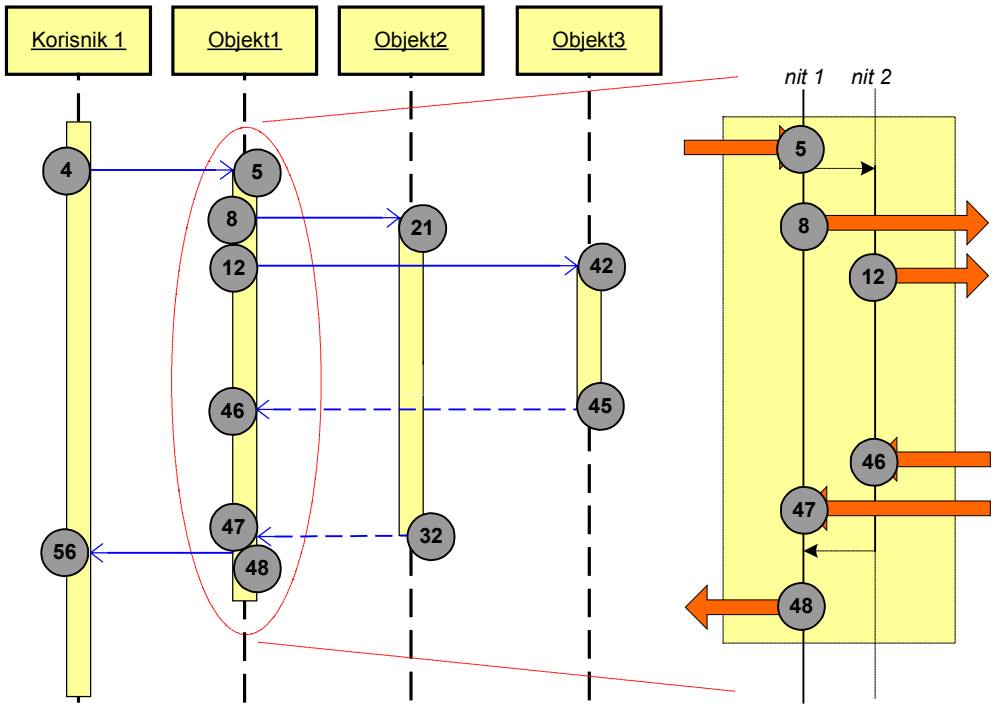


Slika 6.12. Niti izvršavanja raspodijeljenog sustava

Nit izvršavanja može uključivati jednu ili više komponenti raspodijeljenog sustava. U slučaju praćenja niti izvršavanja raspodijeljenog sustava javljaju se dva problema, koja je nemoguće ili je vrlo teško riješiti korištenjem dosad opisanih podataka sadržanih u porukama o događajima. Prvi problem predstavlja kompleksnost postupka određivanja pripadnosti pojedinog događaja promatranom nizu događaja, a drugi problem nastaje internom komunikacijom između niti unutar komponente raspodijeljenog sustava.

Promotrimo komunikacijske događaje nastale unutar komponente sustava “Objekt2“ kao rezultat istovremene obrade dviju poziva metoda ciljnog objekta. Usporedno izvršavanje omogućeno je korištenjem dviju niti izvršavanja unutar same

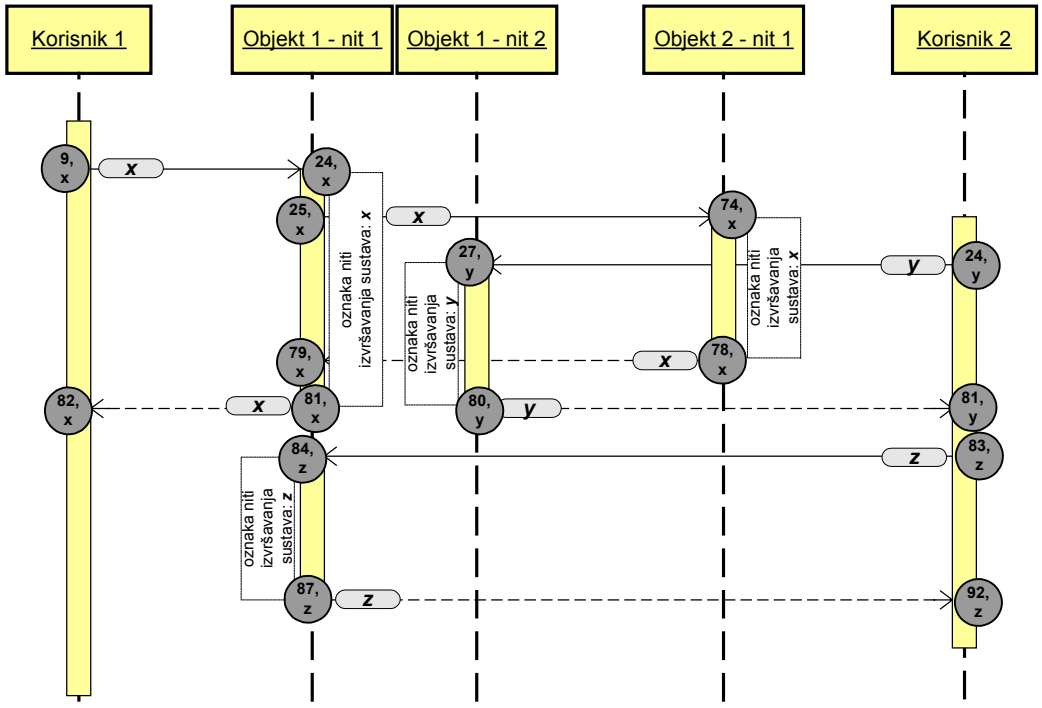
komponente, a za što je zadužena posrednička komponenta. Vrijednosti lokalnog logičkog sata pridjeljene događajima omogućuju jednostavno određivanje redoslijeda događaja, kako na razini komponente (31, 66, 67, 68), tako i na razini cijelog sustava. Problem se javlja u trenutku određivanja pripadnosti događaja pojedinoj niti izvršavanja raspodijeljenog sustava. U slučaju da dva međuovisna događaja (slanje poziva i prihvaćanje poziva) imaju dodijeljena susjedna logička vremena (npr. Korisnik1 i Objekt1: 4⇒5) unutar sustava kao cjeline, određivanje pripadnosti je trivijalno. No, ukoliko logička vremena međuovisnih događaja nisu susjedna, gotovo je nemoguće utvrditi pripadnost pojedinoj niti izvršavanja. Objekt2 prihvaća gotovo istodobne pozive od objekata Objekt3 i Objekt4. Jednostavno je utvrditi da događaj s logičkim vremenom 31 ne pripada niti izvršavanja pokrenutom od objekta Objekt5. No događaj prihvaćanja poziva objekta Objekt3 na strani objekta Objekt2 može imati vrijednosti ili 31 ili 66, što onemogućava pouzdano određivanje niti izvršavanja u sustavu – da li su međuovisni događaji 15⇒31 ili 15⇒66?



Slika 6.13. Dodatna nit izvršavanja u komponenti raspodijeljenog sustava

Drugi problem najbolje se oslikava primjerom na slici 6.13. Nit obrade poziva metode objekta stvara dodatnu nit, koja u svom životnom vijeku vrši pozive metoda drugih (ili istog) objekata. Čak i ako je moguće povezati nit izvođenja unutar komponente sustava sa niti izvođenja raspodijeljenog sustava, u ovom slučaju to postaje nemoguće s novostvorenom niti.

Problem prepoznavanja niti izvršavanja raspodijeljenog sustava u nadglednoj usluzi riješen je uporabom oznaka niti izvršavanja. Oznaka niti jedinstven je niz brojeva na globalnoj razini (eng. *universally unique identifier* - UUID). Ideja praćenja niti izvršavanja raspodijeljenog sustava počiva na dodjeljivanju identifikatora svakoj lokalnoj niti obrade poziva metode ciljnog objekta. Svim porukama o događajima unutar sustava, a koji su se pojavili tijekom izvršavanja lokalne niti obrade poziva metode, dodjeljuje se trenutno važeća oznaka. Između lokalnih niti obrade poziva metode oznaka se prosljeđuje kao dio poziva metode korištenjem mehanizma okruženja usluge, istog onog mehanizma korištenog u prosljeđivanju vrijednosti logičkog sata. Na slici 6.14 detaljnije je prikazan postupak prenošenja oznaka niti s pozivom metode, te njezino prosljeđivanje tijekom daljnjih poziva metoda drugih objekata u sustavu. Prosljeđivanje oznaka potpuno je transparentno s obzirom na nadgledanu aplikaciju.

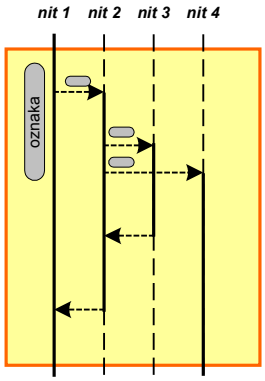


Slika 6.14. Prenošenje oznake niti izvršavanja raspodijeljenog sustava

Predviđena su dva moguća trenutka stvaranja nove niti izvršavanja raspodijeljenog sustava, tj. stvaranja nove oznake:

- u trenutku slanja poziva, ako u trenutnoj lokalnoj niti izvršavanja ne postoji oznaka niti raspodijeljenog sustava

- po primitku poziva metode ciljnog objekta ako u okruženju usluge dospjelog poziva ne postoji oznaka niti raspodijeljenog sustava



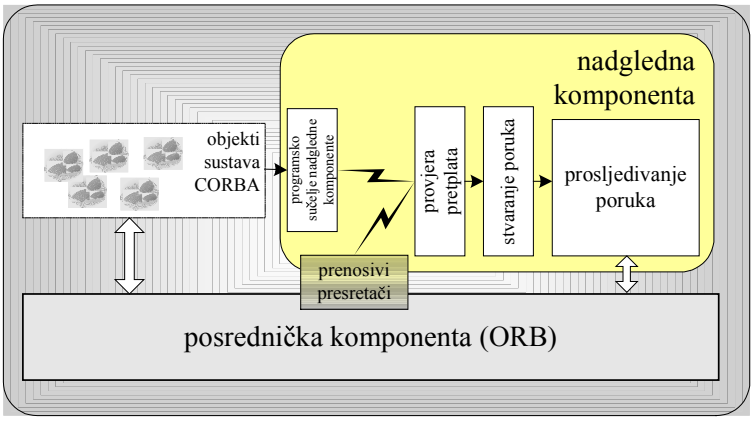
Slika 6.15. Nasljeđivanje oznake prilikom stvaranja novih niti

Problem stvaranja nove niti unutar komponente rješava se nasljeđivanjem okruženja usluge niti-djece od niti-roditelja (slika 6.15). U sustavu CORBA ne postoji mehanizam nasljeđivanja podataka pridruženih niti stoga što je to pitanje implementacije a ne uporabe. Nasljeđivanje tih podataka potrebno je implementirati mehanizmima ovisnim o platformi za koju se aplikacija (odnosno komponenta raspodijeljene aplikacije) razvija.

6.2.7. Nadgledna komponenta

Nadgledna komponenta "ubačena" u nadgledanu komponentu raspodijeljenog sustava zadužena je za:

- transparentno prepoznavanje događaja unutar nadgledane komponente
- stvaranje korisnički definiranih događaja
- održavanje podataka o pretplatama na promatrane događaje
- stvaranje poruka o događajima i njihovo popunjavanje odgovarajućim podacima
- prosljeđivanje poruka prijenosnom sloju.



Slika 6.16. Građa nadgledne komponente

Shematski prikaz građe nadgledne komponente dan je na slici 6.16. Korištenje prenosivih presretača kao mehanizma prikupljanja događaja vezanih uz komunikaciju između komponenti raspodijeljenog sustava, a koji su usko povezani s posredničkom komponentom, također rezultira uskom povezanošću nadgledne komponente s posredničkom komponentom. Predviđeno je postojanje samo jedne nadgledne komponente po izvršnoj okolini. U pravilu, nadgledna komponenta registrira se na razini posredničke komponente, a ne izvršne okoline promatranih entiteta. Svi entiteti koji koriste posredničku komponentu s pridijeljenom nadzornom komponentom podložni su promatranju. Ukoliko unutar izvršnog okruženja postoji više posredničkih komponenti, nadglednu je komponentu moguće istovremeno registrirati unutar više posredničkih komponenti.

Osim transparentnog načina prikupljanja događaja nadgledna usluga, putem sučelja definiranog u opisnom jeziku IDL, omogućava definiranje i stvaranje događaja iz programskog koda nadgledane izvršne okoline. U daljnjem postupanju s aplikacijski definiranim događajima koristi se isti mehanizam kao i sa transparentno prikupljenim događajima vezanim uz komunikaciju.

Na osnovi prepoznatih događaja nadgledna komponenta stvara poruke odgovarajućeg tipa i prosljeđuje ih prijenosnom sloju nadglednog sustava. Velik broj nadgledanih entiteta - objekata sustava CORBA koji koriste nadgledanu posredničku komponentu, a time i velik broj stvorenih poruka može rezultirati velikim opterećenjem na nadgledne komponente, s posljedicom smanjenja performansi. Stoga se koriste dvije tehnike smanjena opterećenja:

- kontrolirano slanje poruka u samo jedan (virtualni) kanal prijenosnog sloja
- stvaranje i prosljeđivanje samo onih poruka o događajima za koje postoje zainteresirani entiteti u korisničkoj razini nadglednog sustava.

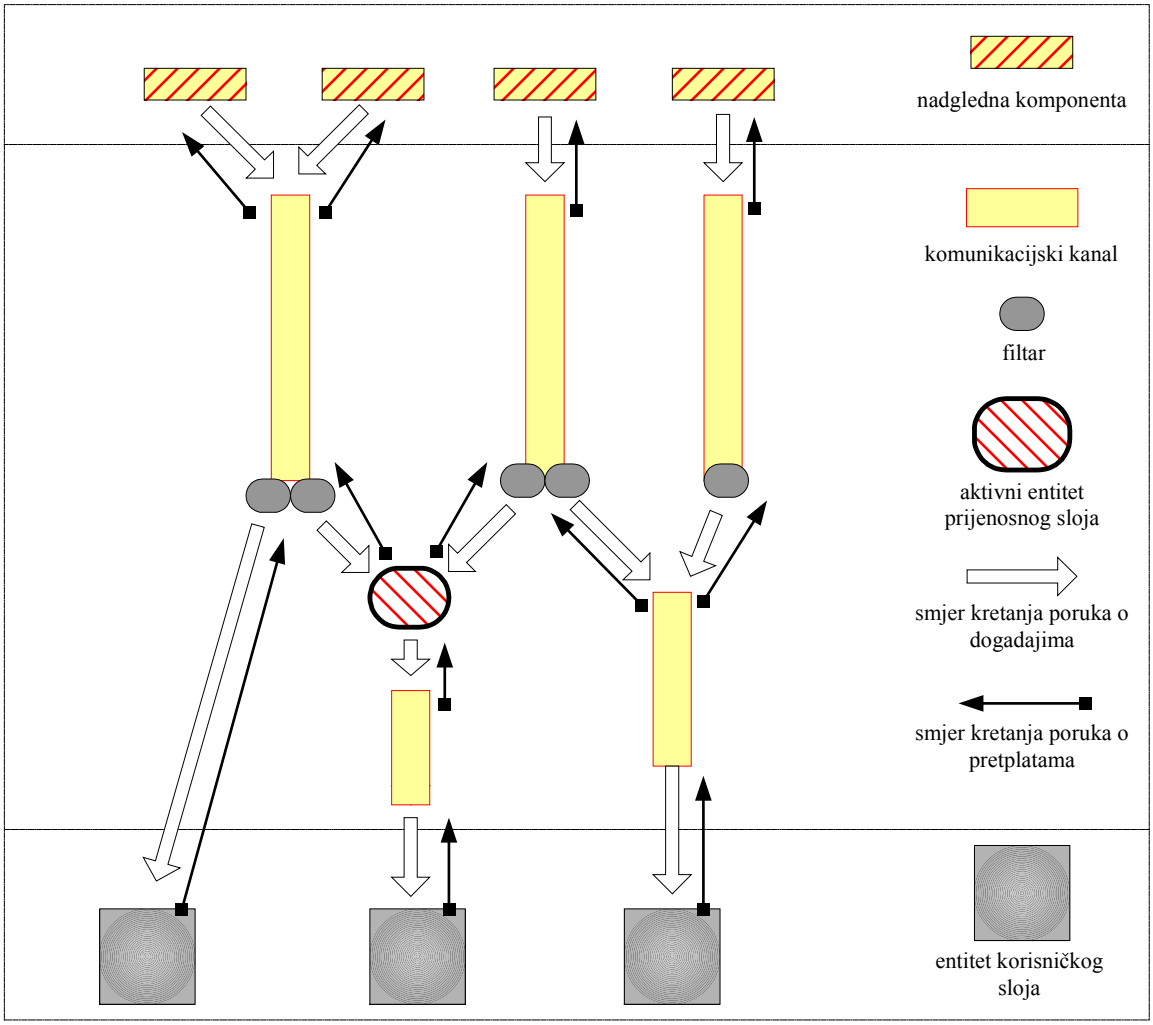
Kontrolirano slanje poruka u kanal prijenosnog sloja bit će naknadno opisano u poglavlju o implementaciji nadglednog sustava, stoga što je usko povezano s implementacijom, a ne građom nadglednog sustava.

Informacije o postojećim pretplatama nadgledna komponenta dobiva od prijenosnog sloja. Osnovno pitanje je na kojoj razini omogućiti definiranje pretplate na događaje. Mogućnost detaljnijeg opisa događaja za koje je korisnički sloj zainteresiran omogućava veće smanjenje opterećenja nadgledne komponente i prijenosnog sloja. Broj stvorenih i proslijeđenih poruka bit će manji nego prilikom "grubljeg" opisa događaja. No s "finijim" opisom događaja javlja se problem memorijskog prostora za pohranu informacija i vremena potrebnog za pretraživanje podataka o pretplatama.

6.3. Prijenosni sloj

Osnovna uloga prijenosnog sloja je prenošenje poruka o događajima proslijeđenih od strane nadglednih komponenti entitetima korisničkog sloja. Korištenjem prijenosnog sloja kao posrednika između izvora i korisnika poruka smanjuje se utjecaj broja aktivnih korisnika na performanse nadgledanog sustava. No to nije i jedina funkcija prijenosnog sloja. Poruke proslijeđene u (virtualne) kanale prijenosnog sloja mogu, korištenjem filtara, biti odbačene ili proslijeđene na odgovarajuće izlaze kanala. Osim korisnika poruka smještenih unutar korisničkog sloja, na izlaze kanala mogu biti priključeni drugi kanali i ostali entiteti smješteni unutar prijenosnog sloja, tvoreći hijerarhijsku organizaciju kanala predstavljenu acikličkim usmjerenim grafom. Smjer prosljeđivanja podataka o pretplatama na događaje suprotnog je smjera od prenošenja poruka, a vodi od korisničkog sloja k nadglednom sloju.

Primjer organizacije prijenosnog sloja dan je slikom 6.17. Nadgledna komponenta pridružena je samo jednom komunikacijskom kanalu u koji prosljeđuje poruke i prima informacije o trenutnim pretplatama. Informacije o pretplatama dobivene od strane komunikacijskog kanala predstavljaju skup svih pretplata entiteta kako iz korisničke sloja, tako i iz prijenosnog sloja. Na krajevima svakog komunikacijskog kanala nalaze se filtri kojima je zadaća propuštanje samo onih poruka koje zadovoljavaju postavljene uvjete. U prijenosnom sloju mogu postojati aktivni entiteti (programi) koji imaju ulogu usmjernika (prosljeđuju poruke na osnovu njihova sadržaja drugim kanalima) ili vrše ulogu složenijih filtara (na osnovu slijeda poruka iz više kanala generiraju nove, složenije poruke i prosljeđuju ih drugim kanalima). Entiteti korisničkog sloja postavljaju filtre na kraj komunikacijskog kanala na koji su priključeni ("finije" filtriranje poruka) i objavljuju zainteresiranost za pojedine vrste poruka ("grublje" filtriranje poruka).



Slika 6.17. Primjer organizacije prijenosnog sloja

Za realizaciju prijenosnog sloja koristi se usluga poruka (eng. *Notification Service*) [30], koja u sebi sadržava sve mehanizme potrebne za ostvarenje opisane funkcionalnosti. Svaki poslužitelj usluge poruka može sadržavati veliki broj (virtualnih) komunikacijskih kanala, sa pridruženim parametrima kvalitete usluga. Svakom korisniku pojedinog kanala dodijeljen je objekt zastupnik kanala, putem kojeg je moguće postavljanje filtra. Time je omogućeno filtriranje poruka i na ulazu i na izlazu kanala. Postavljanje filtara, osim za svakog korisnika zasebno, moguće je i na razini svih korisnika kanala, također i na ulazu i na izlazu komunikacijskog kanala. Filtri na krajevima komunikacijskih kanala definiraju se korištenjem specijaliziranih jezika, najčešće jezikom E-TCL (eng. *Extended Trader Constraint Language*), kojeg svaka implementacija usluge poruka, sukladno standardu, mora podržavati. Sučelje prijenosnog sloja na strani entiteta korisničkog sloja ekvivalentno je standardnom sučelju usluge poruka, definirano opisnim jezikom IDL. Time je omogućena implementacija entiteta korisničkog sloja na svim platformama i programskim jezicima za koje postoji implementacija sustava CORBA.

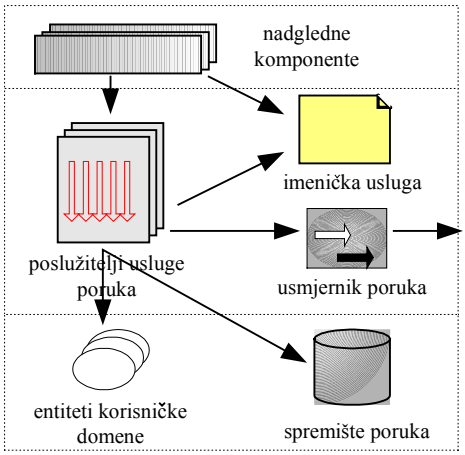
Veliki broj podržanih kanala po poslužitelju, te moguće korištenje više od jednog poslužitelja omogućava različite kombinacije prilikom organizacije prijenosnog sloja. Nadgledana komponenta raspodijeljenog sustava i poslužitelj usluge poruka mogu biti na istom ili na različitim računalima sa sljedećim učincima:

- prosljeđivanje poruka između komunikacijskih kanala istog poslužitelja vrlo je brzo, ali dodatno opterećuje računalo na kojem se poslužitelj izvršava
- prosljeđivanje poruka između kanala dvaju poslužitelja smještenih na različitim računalima sporije je, ali je opterećenje pojedinih računala manje.

Organizacija prijenosnog sloja nadglednog sustava ovisi o organizaciji nadgledanih raspodijeljenih sustava (fizičkoj raspodjeli komponenti sustava po računalima), svojstvima nadgledanog sustava i svojstvima računalne infrastrukture (brzine računala, propusnosti mreže, itd.). Zadaća je administratora nadglednog sustava pravilno organiziranje i podešavanje nadglednog sustava, a što se u većini slučajeva svodi na pravilnu organizaciju prijenosnog sloja.

6.3.1. Nadgledne domene

Razvojem raspodijeljenih sustava velikih razmjera (eng. *large-scale distributed systems*) promatranje sustava u cjelini postalo je neprovedivo zbog vrlo velikog broja objekata u sustavu, te velikog broja poruka o događajima kao posljedice nadgledanja. Vrlo veliki broj poruka o događajima tijekom nadgledanja rada sustava rezultirao bi zagušenjem prijenosnog sloja. Filtri korišteni za usmjeravanje poruka bili bi zbog velikog broja različitih poruka složeni. Zbog velikog broja korisnika broj aktivnih filtara u poslužitelju bio bi proporcionalno velik. Velik broj složenih filtara i poruka unutar poslužitelja drastično bi utjecao na performanse rada poslužitelja i računala na kojem se poslužitelj izvodi. Brzina komunikacije između slojeva nadglednog sustava, kao i između komponenti prijenosnog sloja, također bi bila drastično smanjena uslijed zagušenja računalne mreže. Stoga je uvedena osnovna organizacijska jedinica nadgledanja – nadgledna domena – čiji je cilj podjela raspodijeljenog sustava u manje cjeline radi smanjenja opterećenja i kompleksnosti administracije nadglednog sustava.



Slika 6.18. Organizacija nadgledne domene

Na slici 6.18 prikazana je organizacija nadgledne domene. Nadgledna domena sastoji se od:

- nadglednih komponenata
- imeničke usluge
- komunikacijskih kanala prijenosnog sloja
- usmjernika poruka (nije nužan)
- entiteta korisničkog sloja
- usluge pohrane poruka (nije nužna).

Nadglednoj domeni pripadaju svi entiteti izvršne okoline čija nadgledna komponenta koristi jedan od imenovanih kanala domene. Nije moguće dodijeliti dio entiteta komponente raspodijeljenog sustava jednoj, a preostali dio drugoj nadglednoj domeni.

Unutar imeničke usluge (eng. *Naming Service*) pohranjene su reference na sve bitne komponente nadgledne usluge u domeni nadgledanja. Među ostalim referencama, pohranjene su i reference na imenovane kanale prijenosnog sloja. Unutar nadgledne domene postoji samo jedno mjesto na kojem su ove informacije pohranjene.

Komunikacijski kanali domene nadgledanja implementirani su jednim ili više poslužitelja usluge poruka (eng. *Notification Service*). Imenovani kanali registrirani su unutar imeničke usluge pod određenim imenom, te se mogu koristiti od strane nadglednih komponenti i entiteta korisničkog sloja. Neimenovani kanali nisu registrirani unutar imeničke usluge, te je njihovo korištenje predviđeno samo unutar prijenosnog sloja.

Entiteti korisničkog sloja mogu koristiti dva izvora poruka:

- izvor poruka u stvarnom vremenu prihvaćanjem poruka izravno iz jednog ili više kanala prijenosnog sloja
- izvor poruka pohranjenih u spremištu poruka.

Za formiranje domena nadgledanja ne postoje čvrsta pravila, no od koristi mogu biti sljedeće smjernice:

- *lokacijsko grupiranje nadgledanih entiteta*: u jednu nadglednu domenu grupirati entitete locirane na računalima povezanima u lokalnu računalnu mrežu
- *funkcionalno grupiranje nadgledanih entiteta*: u jednu nadglednu domenu grupirati entitete koji čine jedinstvenu funkcionalnost (uslugu) ili više usko povezanih funkcionalnosti (srodne usluge)
- *grupiranje s obzirom na predviđeno opterećenje nadglednog sustava*: u jednu nadglednu domenu grupirati entitete čije karakteristike neće bitno utjecati na proces nadgledanja, a time i na performanse nadgledanog sustava.

Podjela nadgledanog sustava u više nadglednih domena znatno olakšava administraciju nadglednog sustava i smanjuje njegovo opterećenje, no javlja se problem cjelovitog praćenja rada nadgledanog sustava. Poruke o događajima unutar nadgledanih entiteta prosljeđuju se isključivo entitetima korisničkog sloja iste nadgledne domene. Kao cilj nadgledne usluge postavljeno je cjelovito i prilagodljivo nadgledanje rada raspodijeljenog sustava, stoga ograničenje nadgledanja samo unutar pojedinih nadglednih domena nije prihvatljivo. Tipična raspodijeljena aplikacija temeljena na sustavu CORBA oslanja se na koncepciju usluga koje pružaju grupe objekata, locirane unutar iste i udaljenih računalnih mreža, međusobno povezanih komunikacijskim kanalima slabije propusnosti (s obzirom na propusnost lokalne mreže). Dva se problema javljaju kod nadgledanja snažno raspodijeljenih sustava:

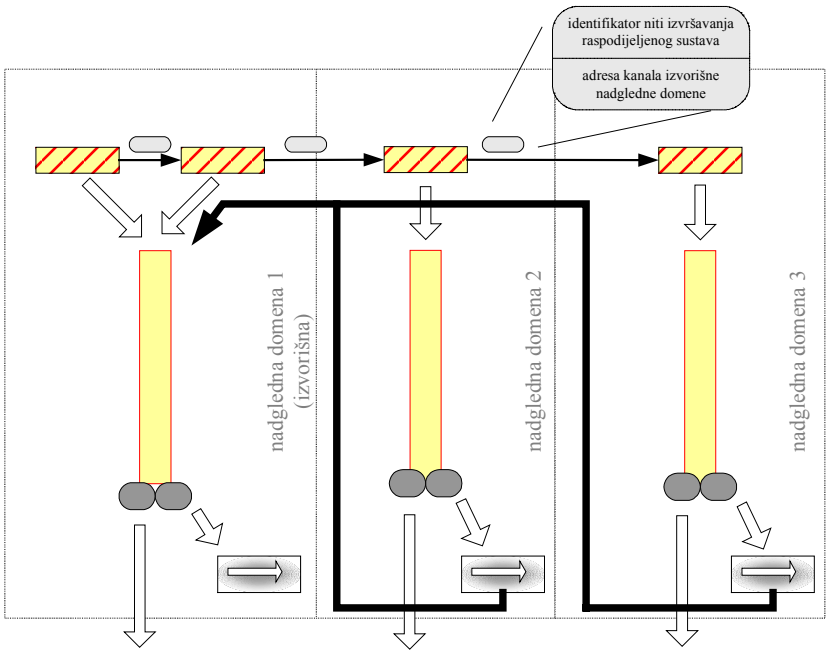
- uporaba komunikacijskih kanala slabije propusnosti i veće cijene po jedinici prenošene informacije
- nemogućnost statičkog određivanja nadglednih domena svih korištenih entiteta.

Uz pretpostavku da su poznate sve domene nadgledanja čiji se entiteti koriste tijekom izvršavanja nadgledane raspodijeljene aplikacije, javlja se problem pribavljanja svih poruka o događajima na jednom mjestu (unutar jedne nadgledne domene) u svrhu njihove analize i stvaranja slike o izvršavanju sustava. Prosljeđivanje svih poruka iz svih domena korištenih entiteta nije prihvatljivo – količina podataka proslijeđenih komunikacijskim kanalima slabije propusnosti bila bi suviše velika, te bi negativno utjecala na rad cjelokupnog računalnog sustava, a ne samo nadgledane aplikacije. Kao

moguće rješenje javlja se korištenje jednog od entiteta korisničkog sloja, čija bi zadaća bila dohvaćanje samo poruka povezanih s nadgledanim sustavom i njihovo prosljeđivanje nadglednoj domeni u kojoj se vrši analiza rada sustava. Takav pristup zahtijevao bi rješenja specifična pojedinom nadgledanom sustavu, te poznavanje svih korištenih nadglednih domena unaprijed. U jednostavnijim sustavima moguće je odrediti sve entitete korištene tijekom izvršavanja sustava, no u složenijim aplikacijama, u kojima se koriste dinamički mehanizmi dohvata referenci korištenih objekata, to ne mora biti moguće. Ovaj problem još je izraženiji prilikom korištenja autonomnih pokretnih objekata temeljenih na sustavu CORBA.

Kao rješenje prethodno opisanih problema, unutar nadgledne domene uvedena je nova komponenta – usmjernik poruka. Zadaća usmjernika poruka prosljeđivanje je svih poruka nastalih unutar nadgledne domene jednom od kanala one nadgledne domene (izvorišne nadgledne domene), čiji je entitet pokretač niti izvršavanja raspodijeljenog sustava. Za usmjeravanje se koristi adresa kanala izvorišne nadgledne domene proslijeđene zajedno s oznakom niti izvršavanja raspodijeljenog sustava (slika 6.19).

Na prethodno opisan način realiziran je općenit mehanizam nadgledanja niti izvršavanja nadgledanog sustava, bez potrebe za razvojem specifičnih entiteta korisničkog sloja, te bez potrebe poznavanja svih uključenih entiteta nadglednih domena. Uvjete za ispravan rad ovog mehanizma čine nadgledanje svih entiteta uključenih u rad sustava i postojanje usmjernika poruka u svim korištenim nadglednim domenama.



Slika 6.19. Prosljeđivanje poruka o događajima pripadajuće niti izvršavanja raspodijeljenog sustava izvorišnoj nadglednoj domeni

6.3.2. Odnos nadglednih domena i lokacijskih domena

Lokacijske domene [9] u sustavima CORBA definirane su korištenjem mjesta za pohranu implementacija (eng. *Implementation Repository*). S obzirom na broj registriranih objekata po mjestu za pohranu implementacija (tj. broju izvršnih okolina i prilagodnika objekata) lokacijske domene dijelimo na velike i male. Velike lokacijske domene omogućuju jednostavnu migraciju objekata iz jedne izvršne okoline u drugu, bez potrebe za izmjenom koda komponenti koje ih koriste. Istovremeno, ovisnost velikog broja komponenti sustava o jednom poslužitelju (referenci na objekte smještene unutar lokacijske domene) predstavlja veliki rizik za prestanak rada sustava u cjelini u slučaju pogreške u radu tog poslužitelja. S druge strane, male lokacijske domene smajuju rizik nefunkcioniranja čitavog sustava, ali i smanjuju broj poslužitelja na koje objekti mogu biti premješteni.

Domene nadgledanja nisu vezane za lokacijske domene, no smjernice za njihovo oblikovanje upućuju na sličan način organizacije. Prilikom nadgledanja malih lokacijskih domena sadržani objekti u većini slučajeva bit će i članovi jedne nadgledne domene, dok će u slučaju velikih lokacijskih domena nadgledani entiteti biti raspoređeni unutar više nadglednih domena, preporučljivo po načelu funkcionalnog grupiranja nadgledanih entiteta.

Migracija objekata unutar male lokacijske domene, ako se njene granice poklapaju s nadglednom domenom, neće imati nikakav utjecaj na nadgledni sustav. No migracija objekata izvan granica pripadne nadgledne domene može stvoriti određene probleme u nadgledanju. Ako promatrani objekt nije pokretač niti izvršavanja raspodijeljenog sustava, poruke o nastalim događajima bit će putem usmjernika poruka proslijeđene u izvorišnu nadglednu domenu. No, ako je promatrani objekt pokretač niti, promijenit će se i izvorišna nadgledna domena. U slučaju da takav razvoj događaja nije poželjan, moraju se koristiti različite hijerarhijske organizacije kanala prijenosnog sloja. U tom slučaju potrebno je unaprijed poznavati skup svih nadglednih domena u koje objekt u toku svog životnog vijeka može prijeći.

Nadgledanje rada autonomnih pokretnih objekata temeljenih na sustavu CORBA [38] svodi se na prethodno opisane slučajeve migracije objekata, s izraženijom nepredvidljivosti i učestalosti kretanja objekta po sustavu.

6.4. Korisnički sloj

Sučelje spram prijenosnog sloja čini jedini element unutar sloja korisnika definiran od strane nadgledne usluge. Korištenje usluge poruka (eng. *Notification Service*) za implementaciju prijenosnog sloja istovremeno uvjetuje i korištenje njenog sučelja kao sučelja nadgledne usluge. Implementacija entiteta korisničkog sloja u potpunosti je prepuštena je korisnicima nadgledne usluge.

Međudjelovanje entiteta korisničkog sloja i prijenosnog sloja korištenjem sučelja prijenosnog sloja svodi se na:

- oglašavanje pretplata na događaje
- postavljanje filtara na događaje
- prihvat poruka.

S obzirom na njihovu ulogu, entiteti korisničkog sloja dijele se na:

- alate za prikupljanje, analizu i prikaz podataka o nadgledanim entitetima u stvarnom vremenu ili iz spremišta podataka

- programe za prikupljanje, analizu i upravljanje nadgledanim entitetima u stvarnom vremenu

- spremišta prikupljenih poruka.

Alati za prikupljanje, analizu i prikaz podataka dohvaćaju podatke iz (jednog ili više) kanala prijenosnog sloja metodom "povlačenja" poruka iz kanala u alat (alat je aktivni sudionik komunikacije) ili metodom "guranja" poruka iz kanala u alat (kanal je aktivni sudionik komunikacije). Tijekom prikupljanja poruka vrši se njihova korelacija – grupiranje poruka s obzirom na u njima sadržane podatke (npr. grupiranje poruka s obzirom na njihov izvor, s obzirom na nit izvršavanja raspodijeljenog sustava, itd.). Kao primjeri ove grupe entiteta korisničkog sloja mogu se navesti alati za analizu opterećenja pojedinih komponenti sustava, alati za analizu rada sustava (dijela ili cjeline), alati za analizu komunikacije u sustavu i formalnu analizu (npr. usporedba s dijagramom tijeka izvođenja definiranim u jeziku UML), itd.

Programi za prikupljanje, analizu i upravljanje radom nadgledanog sustava implementiraju i povratnu vezu prema nadgledanim entitetima raspodijeljenog sustava, prilagođavajući njihov rad na osnovu prikupljenih podataka o radu sustava, kao i ostalih podataka neizravno vezanih za sam promatran sustav (opterećenje na računalima na kojima se entiteti izvršavaju, opterećenje računalne mreže, itd.).

U trenutku prikupljanja poruka spremište ima istu ulogu kao i svaki drugi entitet korisničkog sloja. Tek u trenutku prispijeća zahtjeva za skupom događaja, spremište se pretvara u izvor poruka. Sama izvedba spremišta poruka nije od velike važnosti, no preporučuje se uporaba dnevničke usluge (eng. *Log Service*) [37], koja je prilagođena zahtjevima nadgledne usluge na spremište poruka.

7. Implementacija nadgledne usluge

Nadgledna usluga implementirana je u skladu s osnovnim postavkama sustava CORBA – neovisnosti o platformama i jezicima implementacije komponenti raspodijeljenog sustava. Stoga je moguća njena uporaba u nadgledanju širokog spektra raspodijeljenih sustava temeljenih na CORBA-i, no s pojedinim ograničenjima:

- nadgledna komponenta ovisna je o jeziku implementacije nadgledane komponente i korištenoj implementaciji sustava CORBA
- korištena implementacija sustava CORBA mora podržavati mehanizam prenosivih presretača.

Ovisnost nadgledne komponente o jeziku implementacije nadgledane komponente i korištenoj implementaciji CORBA-e nije moguće izbjeći - standard ne propisuje binarnu, već komunikacijsku udruživost komponenti. Stoga je potrebno posjedovati inačicu nadgledne komponente za pojedini jezik implementacije, te biblioteku koda prevedenu na ciljnoj platformi i s ciljnom implementacijom sustava CORBA. Horizontalnu prenosivost – prenosivost između različitih platformi i implementacija arhitekture korištenjem jednog programskog jezika s nadglednim sustavom nije teško postići. Izvorni kod nadgledne komponente sustava u najvećem dijelu pisan je platformno neovisno, s izdvojenim platformo ovisnim segmentima. Njegovo prevođenje na različitim platformama i s različitim implementacijama ne predstavlja problem. Jedini zahtjev na implementaciju arhitekture ugrađena je podrška prenosivim presretačima. Sve novije implementacije podržavaju ili će u najskorije vrijeme podržavati ovaj mehanizam. Uporaba nadgledne usluge na implementacijama arhitekture koje ne podržavaju prenosive presretača nije (u ovom obliku) moguća. Vertikalna prenosivost – prenosivost između različitih jezika implementacije – nije moguća. Potrebno je razviti programski kod nadgledne komponente za svaki ciljni programski jezik posebno.

Podjelom nadglednog sustava u slojeve problem prenosivosti ograničen je samo na problem prenosivost nadgledne komponente, što znatno smanjuje vrijeme potrebno za uporabu sustava na različitim implementacijama arhitekture i različitim jezicima implementacije.

Nadgledna usluga trenutno je predviđena za korištenje samo s Orbacus 4 implementacijom arhitekture CORBA, razvijene od strane tvrtke Object-Oriented Concepts (www.ooc.com). Jezik implementacije (a time i jezik uporabe) je C++, a platforma Microsoft Windows U razvoju sustava korišteni su sljedeći alati i komponente:

Naziv	Inačica	Proizvođač	Opis
Visual C++	6.0	Microsoft	Prevodilac jezika C++ i radno okruženje
Orbacus	4.0.2	Object-Oriented Concepts	Implementacija sustava CORBA 2.3
JThreads/C++	1.0.12	Object-Oriented Concepts	Biblioteka za podršku rada s nitima
NotificationService	1.0	Object-Oriented Concepts	Implementacija usluge poruka
Telecom Log Service	1.0 beta 1	Object-Oriented Concepts	Implementacija usluge dnevnika

Tablica 7.1. Korištene biblioteke i aplikacije

Trenutno je predviđeno prenošenje postojeće implementacije usluge na operacijski sustav Linux, te implementacija usluge u programskom jeziku Java.

U slijedećem tekstu bit će opisana implementacija i korištenje nadgledne usluge po slojevima sustava.

7.1. Nadgledni sloj

Korištenje nadgledne usluge za nadgledanje pojedinih komponenti raspodijeljenog sustava zahtjeva minimalne promjene u njihovoj implementaciji:

- dodavanje programskog koda nadgledne komponente u implementaciju izvršne okoline nadgledanih entiteta

- pokretanje izvršne okoline nadgledanih entiteta s dodatnim argumentima naredbenog retka.

Dodavanje programskog koda izvršnoj okolini nadgledanih entiteta svodi se na:

- dodavanje nekoliko redaka programskog koda prilikom inicijalizacije posredničke komponente

- prevođenje programskog koda komponente raspodijeljenog sustava i povezivanje s bibliotekom koda nadgledne usluge.

Dodavanje programskog koda lokalizirano je unutar osnovne funkcije programa implementiranih u programskom jeziku C++ - funkciji `main()`. Na sljedećem primjeru dan je programski kod najčešće korištene implementacije funkcije `main()` u komponentama raspodijeljenog sustava (jezik implementacije C++). Osnovna zadaća funkcije je inicijalizacija posredničke komponente, pozivanje funkcije `run()` koja implementira funkcionalnost komponente (npr. inicijalizacija sadržanih entiteta – objekata sustava CORBA i posluživanje poziva metode ciljnih objekata), te uništavanje posredničke komponente prilikom završetka rada. Redci programskog koda dodani u svrhu ubacivanja nadgledne komponente prikazani su podebljano.

Kao prvi korak, u korištene datoteke zaglavlja potrebno je dodati datoteku `MonitoringService.h`, koja sadrži deklaracije u nadglednoj usluzi korištenih nepromjenljivih, tipova podataka, funkcija i razreda. U redcima 4 i 5 stvara se promjenljiva tipa ORB i definira promjenljiva koja označava stanje sustava. U redcima 7 i 8 stvara se objekt razreda `MSInitializer`, tj. nadgledna komponenta. Redci 10 i 11 registriraju objekt `MSInitializer` kao objekt koji utječe na posredničku komponentu prilikom njene inicijalizacije. U retku 15 obavlja se inicijalizacija posredničke komponente. Novostvorena nadgledna komponenta će, tijekom inicijalizacije posredničke komponente, registrirati presretače poruka. Redak 16 sadrži poziv metode `attachedToORB()` nadgledne komponente, kojoj se ovim pozivom daje na znanje koju će posredničku komponentu nadgledati. U retku 18 poziva se funkcija `run()` koja implementira stvarnu funkcionalnost komponente raspodijeljenog sustava. Prilikom završetka rada komponente (izvršne okoline) u retku 25 poziva se metoda `flush()` nadgledne komponente, čija je zadaća sve do tog trenutka prikupljene poruke o događajima proslijediti prijenosnom sloju.

```
   ...
   #include <MonitoringService.h>
   ...


1  int main(int argc, char* argv[])
2  {
3
4     CORBA::ORB_var orb;
5     int status = EXIT_SUCCESS;
6
```

```
 7    MonitoringService::MSInitializer* msInitializer = new
 8       MonitoringService::MSInitializer( argc, argv );
 9
10    PortableInterceptor::register_orb_initializer(
11     msInitializer );
12
13    try {
14         orb = CORBA::ORB_init( argc, argv );
15
16         msInitializer -> attachedToORB( orb, argc, argv );
17
18         status = run( orb );
19    }
20    catch ( CORBA::SystemException& )
21    {
22         status = EXIT_FAILURE;
23    }
24
25    msInitializer -> flush( );
26
27    if( ! CORBA::is_nil( orb ) )
28    {
29         try {
30              orb->destroy( );
31         }
32         catch ( CORBA::SystemException& )
33         {
34              status = EXIT_FAILURE;
35         }
36    }
37
38    return status;
39 }
```

Nakon provedenih izmjena u programskom kodu nadgledane komponente potrebno je kod prevesti, te povezati s bibliotekom funkcija `MonitoringService.lib`. Kao rezultat promjena dobiva se komponenta raspodijeljenog sustava s nadograđena s nadglednom uslugom.

Za korištenje nadgledne usluge potrebno je komponentu raspodijeljenog sustava pokrenuti s dodatnim argumentima naredbenog retka. U slučaju da potrebni argumenti nisu navedeni, nadgledna komponenta neće biti pokrenuta. Svi argumenti nadgledne usluge imaju zajednički prefiks `–MS`.

Slijedi pregled svih argumenata naredbenog retka nadgledne usluge:

Argument	Objašnjenje
`-MSActivateClient`	Pokreće nadglednu komponentu i prati komunikacijske događaje vezane uz korisničku stranu komunikacije
`-MSActivateServer`	Pokreće nadglednu komponentu i prati komunikacijske događaje vezane uz poslužiteljsku stranu komunikacije
`-MSActivate`	Pokreće nadglednu komponentu i prati sve komunikacijske događaje
`-MSAppName <ime>`	Određuje ime nadgledane komponente (ako nije naveden podrazumijeva se ime izvršne datoteke)
`-MSDomainChannel <ime>`	Određuje logički kanal u koji se šalju poruke o događajima (ako nije naveden podrazumijeva se DefaultNotificationChannel logički kanal)
`-MSVirtualChannel <ime>`	Određuje logički kanal izvorišne domene korišten ako je nadgledani entitet pokretač neovisne niti izvršavanja raspodijeljenog sustava (nema podrazumijevane vrijednosti)
`-MSFlushPeriod <period>`	Vremenski period (u milisekundama) slanja svih poruka u međuspremniku (podrazumijevana vrijednost 5000)
`-MSBufferSize <broj>`	Broj poruka u međuspremniku iznad kojeg se sadržaj međuspremnika šalje prijenosnom sloju (podrazumijevana vrijednost 100)

`-MSArguments`	Omogućuje uključivanje vrijednosti argumenata pozvanih metoda i povratnih vrijednosti u poruke o komunikacijskim događajima
`-MSSubscriptions`	Omogućuje korištenje pretplata na vrste događaja

Tablica 7.2. Argumenti naredbenog retka nadgledne usluge

Argumenti `-MSActivateClient`, `-MSActivateServer` i `-MSActivate` osnovni su argumenti nadgledne usluge. Ako niti jedan od njih nije naveden, usluga nadgledanja neće biti pokrenuta.

Uz prethodno nabrojene argumente, nadglednoj komponenti moraju biti na raspolaganju reference na imeničku uslugu i uslugu poruka putem mehanizma dohvata tzv. inicijalnih referenci (poziv metode `resolve_initial_references()` na objektu posredničke komponente). Reference na navedene usluge mogu biti navedene kao argumenti naredbenog retka ili zapisane u datoteci za podešavanje, ovisno o korištenoj implementaciji sustava CORBA.

Navođenjem `-MSArguments` argumenta naredbenog retka omogućuje se praćenje argumenata poziva i povratnih vrijednosti tijekom komunikacije. No dodavanje tih podataka u poruke o događajima ima negativan utjecaj na nadgledani sustav iz dva razloga:

- veličina poruke znatno se povećava, što ima utjecaj na propusnost mreže
- potrebna je veća količina memorije prilikom lokalne pohrane poruka u međuspremniku
- preslikavanje argumenata u poruku (naročito ako se radi o većoj količini podataka) usporava rad nadglednog sustava, a time i usporava izvođenje nadgledanog sustava.

Mehanizam pretplata na događaje može se pokazati korisnim u smanjenju broja stvorenih poruka, no nije ga moguće primijeniti u svim prilikama. Problem naročito dolazi do izražaja u nadgledanju korisničke aplikacije, koja u sebi ne mora sadržavati objekte. Mehanizam pretplata zahtijeva od komponente koja ga koristi poseban objekt, s posebnom niti izvođenja, koji će prikupljati informacije o pretplatama proslijeđene od strane prijenosnog sloja (tj. usluge poruka). Implementacije programa

korisnika u pojedinim slučajevima ne omogućuju podršku objektima i (ili) višenitnosti. Ustrajavanje na korištenju mehanizma pretplata djelomično bi ograničilo korisnost nadgledne usluge, stoga je njegovo korištenje dano kao dodatna mogućnost, koju je potrebno izravno omogućiti.

Nadgledna komponenta prepoznaje sve događaje vezane uz komunikaciju. No događaji svojstveni pojedinom raspodijeljenom sustavu moraju biti stvoreni eksplicitno, unutar koda implementacije entiteta sustava. Nadgledna usluga stavlja programeru na raspolaganje sučelje korištenjem kojega se definiraju i stvaraju događaji. Prepoznavanje i prosljeđivanje poruka o događajima i dalje ostaje zadatak posredničke komponente.

Novi događaji definiraju se i stvaraju korištenjem objekta tvornice, čije je sučelje opisano slijedećom definicijom korištenjem opisnog jezika pseudo-IDL:

```
interface EventObject {

      void addValue( in string name, in any value );
      void setUnstructuredData( in any data );
};

interface MonitoringService {

      EventObject getEvent( in string event_type,
                            in string event_name );
      void sendEvent( in EventObject event );
};
```

Na sljedećem primjeru prikazano je stvaranje korisnički definiranog događaja unutar implementacije metode nadgledanog CORBA objekta:

```
...
//promjenljiva tipa any za prihvat argumenta događaja
CORBA::Any_var any = new CORBA::Any( );

//stvranje objekta događaja
MonitoringService::EventObject* obj = MonitoringService::
  getEvent("PersistentObject", "UpdateFailed");

//id (nekog) korištenog objekta u promjenljivu tipa any
any <<= persistentObject -> id( );

//dodavanje informacije o objektu u događaj
event -> addValue( CORBA::string_dup("OBJECT_ID"), any );

//slanje događaja
MonitoringService::sendEvent( obj );

...
```

Uporaba usluge poruka za implementaciju prijenosnog sloja daje mogućnost korištenja više oblika poruka, čije prenošenje usluga podržava. Za potrebe nadgledne usluge koriste samo tzv. strukturirane poruke iz dva bitna razloga:

- oblik strukturiranih poruka pogodan je za opis događaja
- primjena filtara unutar usluge poruka najbolje je prilagođena baš strukturiranim porukama.

Strukturirana poruka sastoji se od zaglavlja i tijela poruke. Zaglavlje poruke sastoji se od čvrsto definiranog dijela (eng. *fixed header*) i promjenljivog dijela (eng. *variable header*) – parova imena i vrijednosti. Tijelo poruke u potpunosti je promjenljivo, a sastoji se od strukturiranog dijela (parova imena i vrijednosti) i nestrukturiranog dijela (vrijednost tipa CORBA::Any).

Poruka nadglednog sustava sastoji se od zaglavlja, koje sadrži podatke zajedničke svakoj poruci, te tijela poruke koje sadrži podatke svojstvene pojedinom događaju. Zaglavlje poruke čvrsto je definirano, te korisnik nadglednog sustava (pod pojmom korisnik podrazumijevamo programera koji koristi prethodno opisano sučelje za definiranje i stvaranje novih događaja) na njega nema utjecaj.

MonitoringService	
[tip događaja]	
[ime događaja]	
SystemTime	
LogicalTime	
ExecutionTraceUUID	
objectID*	
adapterID*	
applicationName	
virtualChannelIOR*	
[*strukturirano tijelo poruke*]	
[*nestrukturirano tijelo poruke*]	

U sljedećoj tablici dana su objašnjena polja zaglavlja poruke. Zvjezdica (*) pored imena polja označava da polje, ovisno o okolnostima, ne mora imati definiranu vrijednost.

Ime polja zaglavlja	Opis
`MonitoringService`	Prvo polje zaglavlja poruke mora, po specifikaciji usluge poruka, označavati domenu primjene. Stoga se u njemu nalazi tekst "MonitoringService".
`[tip događaja]`	Drugo polje zaglavlja označava tip događaja. Komunikacijski događaji označeni su s `CLIENT_COMM` i `SERVER_COMM`. Prilikom definicije korisničkog događaja u ovo polje smješta se prvi argument poziva metode `MonitoringService::getEvent( )`.
`[ime događaja]`	Treće polje označava ime događaja. Vrijednosti ovog polja za komunikacijske događae bit će opisana u daljnjem tekstu. Prilikom definicije korisničkog događaja u ovo polje smješta se drugi argument poziva metode `MonitoringService::getEvent( )`.
`SystemTime`	Sadrži vrijednost fizičkog sata računala na kojem se događaj zbio.
`LogicalTime`	Sadrži vrijednost logičkog sata raspodijeljenog sustava.
`ExecutionTraceUUID`	Sadrži oznaku niti izvršavanja raspodijeljenog sustava.
`objectID*`	Sadrži ime objekta u kojemu se događaj zbio. Polje ne mora sadržavati vrijednost ako se događaj nije zbio unutar objekta.
`adapterID*`	Sadrži ime prilagodnika objekata objekta u kojemu se događaj zbio. Polje ne mora sadržavati vrijednost ako se događaj nije zbio unutar objekta.
`applicationName`	Sadrži ime aplikacije definirano s argumentom `–MSAppName` ili imenom izvršne datoteke.
`virtualChannelIOR*`	Sadrži referencu na objekt logičkog kanala unutar izvorišne domene. Ne mora biti definiran ako je izvršna okolina koja sadrži entitet - pokretač niti komunikacije pokrenuta bez navođenja argumenta `–MSVirtualChannel`.

Tablica 7.3. Zaglavlje poruke

Komunikacijski događaji jedini su događaji s čvrsto definiranim oblikom tijela poruke. Dijele se na dva osnovna tipa:

- događaje povezane s korisničkom stranom komunikacije (`CLIENT_COMM`)
- događaje povezane s poslužiteljskom stranom komunikacije (`SERVER_COMM`).

Prilikom definiranja korisničkog događaja, korisnik može utjecati na dijelove zaglavlja (tip i ime događaja), te na tijelo poruke. Prilikom stvaranja objekta događaja definiramo tip i ime događaja:

```
MonitoringService::EventObject* obj = MonitoringService::
  getEvent("PersistentObject", "UpdateFailed");
```

S metodom `addValue( )` dodajemo parove ime-vrijednost unutar tijela poruke:

```
event -> addValue( CORBA::string_dup("OBJECT_ID"), any );
```

S metodom `setUnstructuredData` postavljamo vrijednost unutar nestrukturiranog dijela tijela poruke.

```
event -> setUnstructuredData( bulkData );
```

Novostvorena poruka izgleda ovako:

7.1.1. Performanse nadgledne komponente

Nadgledna usluga nužno uvodi kašnjenje u rad nadgledanog sustava, koje je potrebno svesti na najmanju moguću mjeru. Osim uvođenja najmanjeg mogućeg kašnjenja, nadglednu komponentu potrebno je implementirati tako da uvedeno kašnjenje bude predvidljivo, u svrhu pravilne procjene utjecaja na nadgledani sustav. Stoga je u daljnjem tekstu opisana analiza rada nadgledanog sustava i komentirani njome dobiveni rezultati.

Nadgledani sustav čini jednostavna raspodijeljena aplikacija, sastavljena od dvije komponente: korisničkog programa i CORBA objekta. Objekt implementira sljedeće sučelje (opisano jezikom IDL):

```
module InterceptorTest {
      interface SampleApp {
            void sayHello( );
      };
};
```

Korisnički program uzastopno poziva metodu `sayHello( )` ciljnog objekta tisuću puta, te mjeri vrijeme potrebno za izvršavanje poziva. Implementacija metode ne posjeduje nikakvu funkcionalnost. Korištenje ovako jednostavnog primjera uvjetovano je željom za mjerenjem performansi sustava u odnosu na "golu" funkcionalnost sustava temeljenog na CORBA-i, bez dodatnih kašnjenja koje bi uvelo prosljeđivanje parametara metodi udaljenog objekta. Ovaj primjer ujedno čini i najgori slučaj uvođenja kašnjenja u nadgledani sustav od strane nadgledne komponente, promatrano sa stanovišta relativno uvedenog kašnjenja izvođenja poziva metode udaljenog objekta.

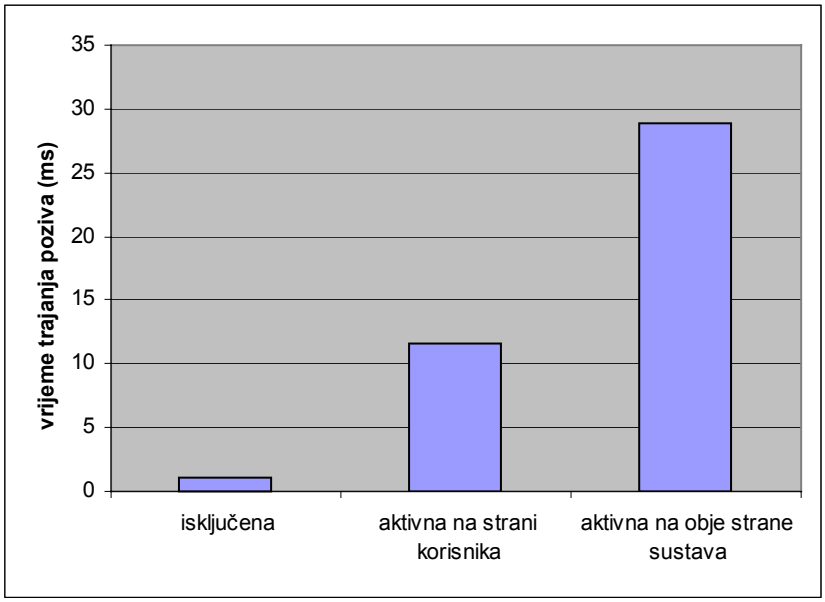
Mjerenje trajanja poziva udaljenog objekta obavljeno je na dvije konfiguracije raspodijeljenog sustava, s obzirom na smještaj korisničkog programa i ciljnog objekta. U prvom slučaju i korisnik i objekt nalazili su se na istom računalu, dok su se u drugom slučaju nalazili na različitim računalima. Imenička usluga i usluga poruka bile su u oba slučaja pokrenute na trećem računalu. Sva tri korištena računala bila su priključena na lokalnu računalnu mrežu propusnosti 100Mbit/s, s procesorom Intel Celeron 466MHz i 128MB memorije. Korišten je operacijski sustav Microsoft Windows 2000.

U prvom krugu mjerena su prosječna vremena jednog poziva metode udaljenog objekta bez uključene nadgledne usluge, s nadglednom uslugom pokrenutom na strani

korisnika, te na obje komponente raspodijeljenog sustava (tablica 7.4, slika 7.1). Obje komponente bile su smještene na istom računalu, a korištene su podrazumijevane veličine međuspremnika poruka (100 poruka) i perioda slanja poruka (5 sekundi) nadgledne komponente. Praćenje vrijednosti argumenata poziva nije korišteno.

Status nadgledne usluge	Prosječno trajanje jednog poziva metode
isključena	1,17 ms
aktivna na strani korisnika	11,65 ms
aktivna na obje strane	28,88 ms

Tablica 7.4. Prosječno trajanje poziva metode udaljenog objekta



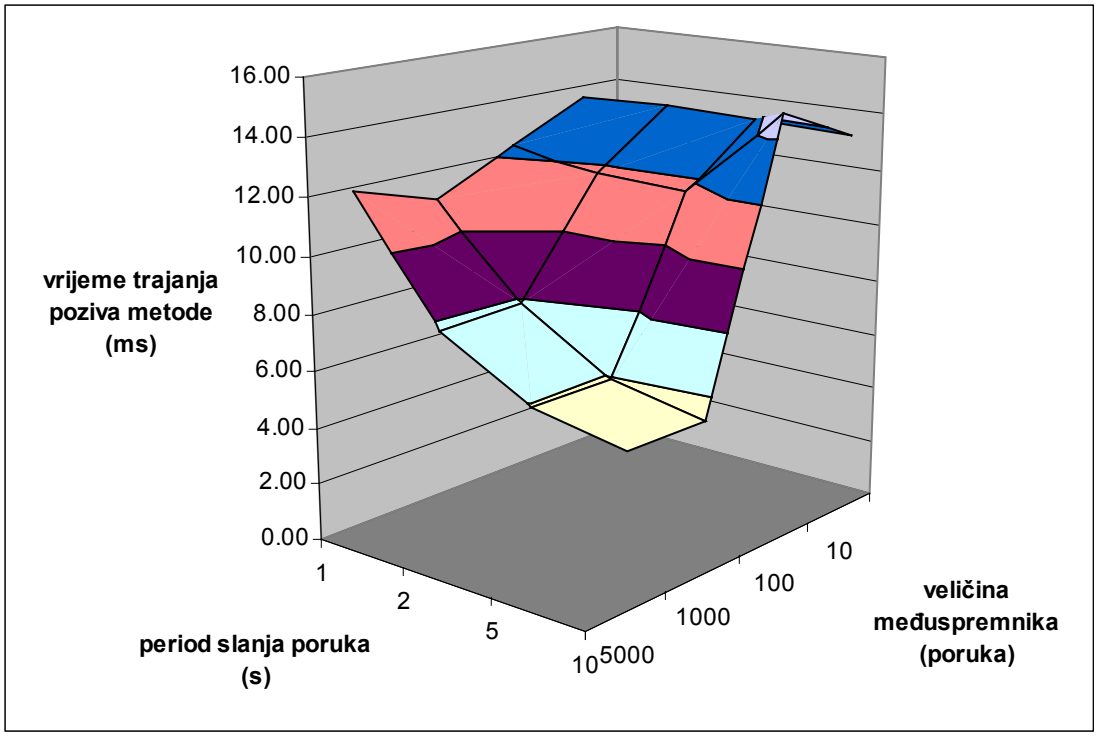
Slika 7.1. Prosječno trajanje poziva metode udaljenog objekta

Detaljnija analiza trajanja izvršavanja pojedinih funkcionalnih cjelina nadgledne komponente otkriva da se, prosječno, troši oko 4 ms po pojedinom prosljeđivanju poruka u komunikacijski kanal prijenosnog sloja, a preostalo vrijeme na izvršavanje koda nadgledne komponente. Zanimljivo da samo korištenje mehanizma presretača, bez ikakve implementirane funkcionalnosti, troši otprilike 0,2 ms po pozivu metode presretača.

Drugi krug mjerenja predstavljalo je ispitivanje utjecaja veličine međuspremnika poruka i perioda slanja poruka (tablica 7.5, slika 7.2) na prosječno vrijeme trajanja poziva metode udaljenog objekta. Nadgledna usluga bila je aktivna samo na strani korisničke aplikacije.

Period slanja poruka (s)	Veličina međuspremnika (poruka)			
	10	100	1000	5000
1	13,58 ms	12,26 ms	10,92 ms	11,89 ms
2	13,66 ms	11,75 ms	7,87 ms	7,75 ms
5	13,63 ms	11,65 ms	5,90 ms	5,92 ms
10	13,52 ms	14,86 ms	5,28 ms	5,30 ms

Tablica 7.5. Prosječno trajanje poziva metode u odnosu na veličinu međuspremnika i period slanja poruka



Slika 7.2. Prosječno trajanje poziva metode u odnosu na veličinu međuspremnika i period slanja poruka

Iz rezultata je vidljivo da je kašnjenje tim manje što je manji broj slanja poruka u prijenosni sloj. Broj slanja poruka ovisan je i o veličini međuspremnika, ali i o zadanom vremenskom periodu slanja. Broj poruka u nadgledanom primjeru je oko 2.000 (jedan poziv rezultira s dvije poruke). Stoga su najbolji rezultati postignuti kada je veličina međuspremnika bila dovoljno velika da primi sve stvorene poruke, a vrijeme osvježavanja veće od vremena potrebnog za obavljanje svih poziva. U ovom slučaju jasno se uočava da, oduzmemo li od ukupnog prosječnog vremena potrebnog za obavljanje poziva s aktivnom nadglednom uslugom vrijeme potrebno za slanje poruka u komunikacijski kanal i stvarno vrijeme izvršavanja poziva metode bez aktivne nadgledne usluge, izvršavanje koda implementacije nadgledne komponente unosi kašnjenje od, otprilike, 1 do 1,5 ms.

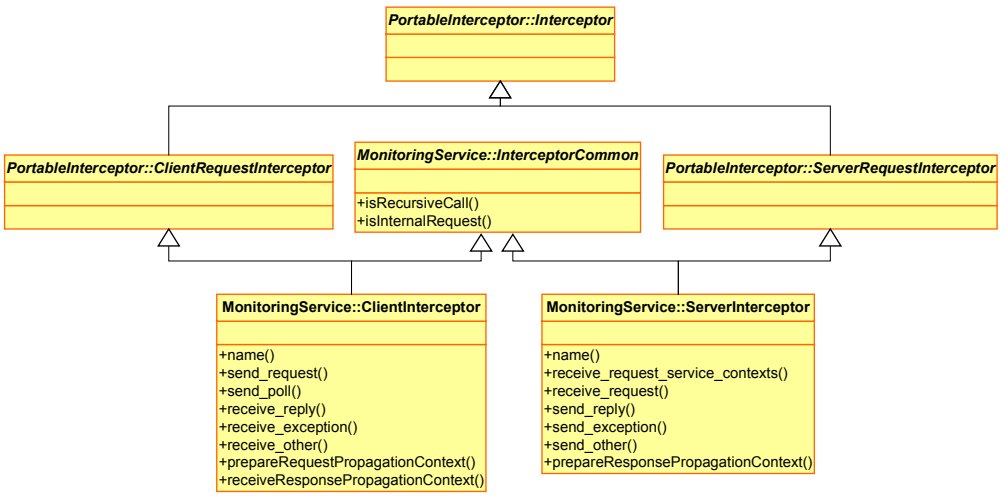
Vrijednosti korištenih argumenata veličine međuspremnika i perioda slanja poruka nadglednog sustava ovisit će o nadgledanom sustavu:

- u nadgledanju sustava kod kojih je bitno brzo prepoznavanje stanja sustava bit će korišteni kraći periodi slanja poruka
- u nadgledanju sustava s velikim brojem međusobnih poziva metoda objekata, bit će korišteni veći međuspremnici poruka.

Rezultati mjerenja trajanja poziva u konfiguraciji s komponentama smještenim na različitim računalima bili su gotovo istovjetni. Korištena implementacija sustava CORBA (Orbacus 4) nema ugrađene optimizacije komunikacije komponenti na istom računalu (npr. korištenjem dijeljene memorije), već se koristi standardnim mehanizmom mrežne komunikacije. To, kao i brzina lokane računalne mreže, rezultiralo je gotovo istovjetnim vremenima poziva metoda u oba ispitivana primjera. No, općenito uzevši, komunikacija između komponenti smještenih na dva odvojena računala ne može ni u kom slučaju biti brža od dobivenih rezultata, stoga postotni udio uvedenog kašnjenja može biti samo manji. To se naročito odnosi na pozive metoda s velikim brojem parametara.

7.1.2. Implementacija nadgledne komponente

Komunikacijski događaji presreću se korištenjem mehanizma prenosivih presretača. S obzirom na ulogu komponente u pojedinoj komunikaciji (korisnik ili poslužitelj) poziva se objekt odgovarajućeg razreda presretača prijavljen posredničkoj komponenti. Na slici 7.3 prikazana je organizacija razreda prenosivih presretača.



Slika 7.3. Razredi prenosivih presretača

Prenosivi presretač nadgledne usluge na strani korisnika (razreda `MonitoringService::ClientInterceptor`) implementira metode za presretanje komunikacije, a koje se mogu podijeliti na dvije grupe:

- grupa metoda pozivanih prilikom slanja poziva (`send_request`, `send_poll`)
- grupa metode pozivanih prilikom prihvata odgovora (`receive_reply`, `receive_exception`, `receive_other`).

Zajedničko implementacijama svih metoda presretača na strani korisnika je sljedeće:

- zapisivanje vremena sustava u trenutku u kojem je pozvana metoda presretača:

```
//record the current time
TimeBase::UtcT receivedTime = m_pCommonInterceptorData ->
getSystemTime( ) -> getTime( );
```

- provjera da li je usluga nadgledanja pravilno inicijalizirana:

```
//check if the service has been initialized !
if( ! m_pCommonInterceptorData -> m_bMSInitialized )
     return;
```

- provjera da li je presretač pozvan kao posljedica komunikacije pokrenute od nadgledne usluge:

```
//check if this is a recursive call (made by
//Monitoring Service)
if( isInternalRequest( info ) )
     return;
```

Implementacija nadgledne usluge komunicira s objektima prijenosnog sloja u trenutku pronalaženja kanala i prosljeđivanja poruka o događajima. Svaki poziv metode udaljenog objekta rezultira pozivom metode presretača, što u slučaju implementacije nadgledne usluge može za posljedicu imati blokiranje rada sustava uslijed beskonačne rekurzije. Stoga je potrebno ignorirati pozive presretača kao posljedice poziva metoda udaljenih objekata tijekom izvršavanja presretača. To se postiže korištenjem zastavice koja označava tekuće izvršavanje koda presretača implementirane kao jednog od utora (eng. *slot*) trenutnog okruženja usluga presretača – objekta `PICurrent`. Metoda `InterceptorCommon::isRecursiveCall` nastoji pronaći zastavicu okruženja usluge presretača (objekt `PICurrent`) niti iz koje je izvršen poziv metode objekta, proslijeđenu od strane posredničke komponente. Ukoliko je zastavica postavljena, trenutni poziv presretača rezultat je komunikacije unutar koda presretača, te poziv

treba biti ignoriran. Ukoliko zastavica nije postavljena, treba ju postaviti (metoda `InterceptorCommon::isInternalRequest`) i nastaviti s izvršavanjem.

Metode `send_request` i `send_poll` nalaze se u izlaznom putu poziva metode udaljenog objekta, stoga su one odgovorne za kreiranje nove oznake niti izvršavanja raspodijeljenog sustava, te prosljeđivanje logičkog vremena i reference kanala izvorišne domene. Nabrojeni podaci prenose se korištenjem okruženja usluge definirane sljedećim opisom sučelja u jeziku IDL:

```
module MonitoringService {

      typedef octet octet8[8];

      struct IDL_UUID {

            unsigned long    Data1;
            unsigned short   Data2;
            unsigned short   Data3;
            octet8           Data4;
      };

      struct MSPropagationContext {

            //local logical time
            unsigned long logicalTime;

            //execution trace UUID
            IDL_UUID executionTraceUUID;

            //virtual monitoring domain reference
            string virtualDomainIOR;
      };
};
```

Nova oznaka niti izvršavanja raspodijeljenog sustava potrebno je stvoriti samo u slučaju da je presretnuti poziv tzv. neovisni poziv. Neovisni poziv karakterizira nepostojanje okruženja usluge u za nju predviđenom utoru objekta okruženja usluge niti iz koje je poziv izvršen (objekt `PICurrent`). Stoga je potrebno stvoriti novo okruženje usluge i popuniti ga odgovarajućim podacima. Metoda `ClientInterceptor::prepareRequestPropagationContext` implementira prethodno opisanu funkcionalnost:

- ako unutar utora postoji okruženje usluge, u polje `logicalTime` zapisuje se trenutna vrijednost lokalnog logičkog sata

- ako unutar utora ne postoji okruženje usluge
    - u polje `logicalTime` zapisuje se trenutna vrijednost lokalnog logičkog sata

- u polje `executionTraceUUID` zapisuje se nova oznaka niti izvršavanja raspodijeljenog sustava
- u polje `virtualDomainIOR` zapisuje se referenca na kanal izvorišne domene.

Potom se okruženje usluge kodira i pridodaje pozivu metode udaljenog objekta.

Metode `receive_reply`, `receive_exception` i `receive_other` pozivaju se u trenutku prispijeća odgovora na poziv metode udaljenog objekta. Njihova zadaća je usklađivanje vrijednosti logičkih satova na strani korisnika i ciljnog objekta. Ova funkcionalnost implementirana je unutar metode `ClientInterceptor::receiveResponsePropagationContext`.

Prenosivi presretač nadgledne usluge na strani poslužitelja (razreda `MonitoringService::ServerInterceptor`) implementira metode za presretanje poziva metoda objekata zajedničke posredničke komponente. Metode presretača mogu se podijeliti u dvije grupe:

- grupa metoda pozivanih prilikom prihvata poziva (`receive_request_service_contexts`, `recive_request`)
- grupa metode pozivanih prilikom slanja odgovora (`send_reply`, `send_exception`, `send_other`).

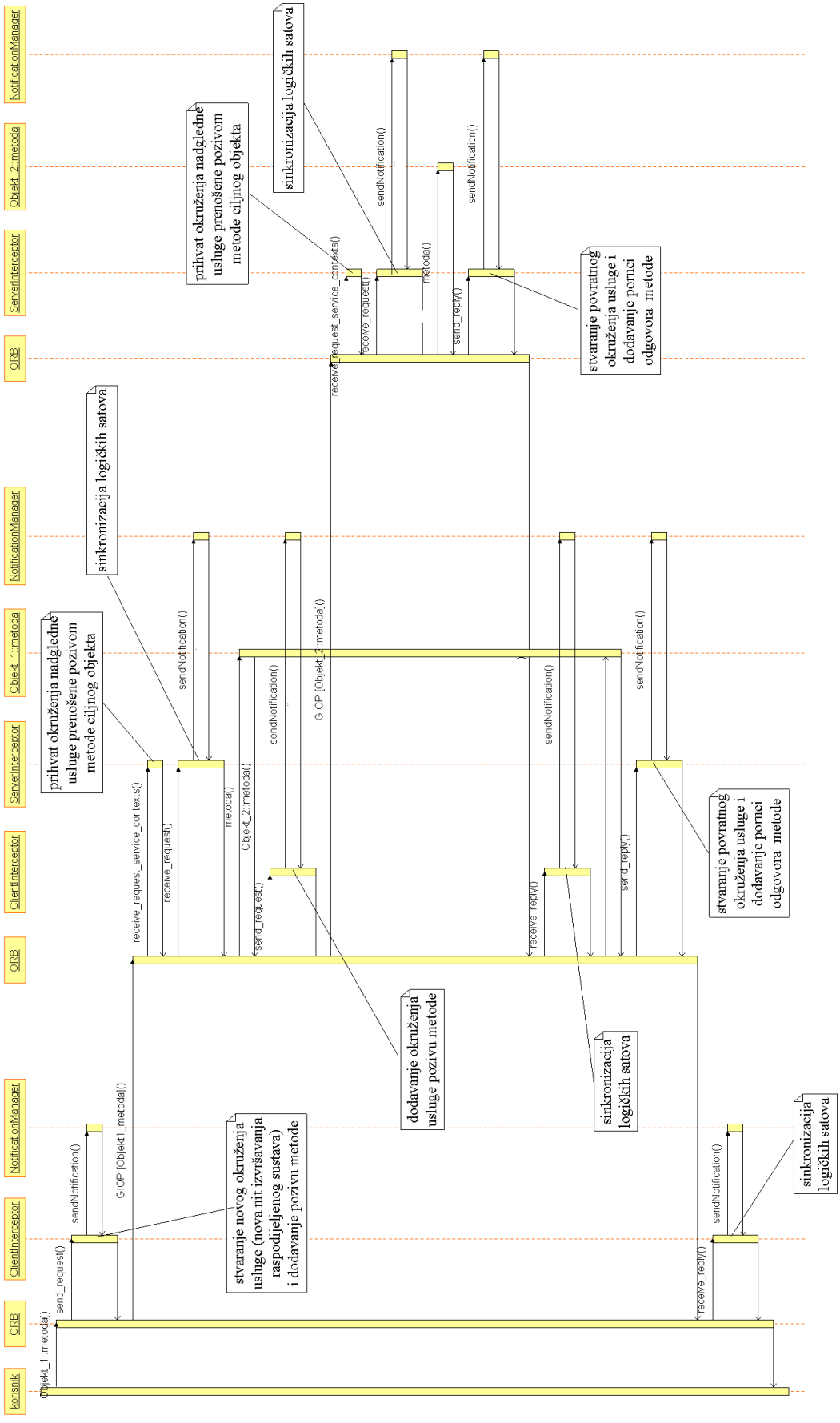
Uloga metode `receive_request_service_contexts` prihvaćanje je i pohrana okruženja nadgledne usluge u odgovarajući utor okruženja usluge niti korištene za obradu pristiglog poziva (objekt PICurrent).

Metoda `recieve_request` zadužena je, kao i metode unutar izlaznog puta poziva na strani korisnika, za bilježenje vremena sustava trenutka dolaska poziva metode, provjeru stanja nadgledne usluge i onemogućavanje rekurzivnih poziva presretača. U skladu s vrijednosti logičkog sata korisnika, sadržanog u prenesenom okruženju usluge, mijenja se i vrijednost lokalnog logičkog sata. Ako unutar prispjelog poziva okruženje usluge nije pronađeno, stvara se novo okruženje s odgovarajućim vrijednostima lokalnog logičkog sata, novom oznakom niti izvršavanja raspodijeljenog sustava i (ako je naveden kao argument prilikom pokretanja izvršne okoline) referencom kanala izvorišne domene nadgledanja. Novo okruženje usluge pohranjuje se u za to predviđen utor okruženja usluge niti unutar koje će poziv metode ciljnog objekta biti obrađen.

Nakon završene obrade poziva u tijelu pozvane metode, ovisno o rezultatu obrade, poziva se jedna od metoda presretača u izlaznom putu. Osim uobičajenih zadaća

(bilježenje vremena, provjera stanja usluge, onemogućavanje rekurzivnih poziva), metode izlaznog puta pohranjuju vrijednost lokalnog logičkog sata u postojeće okruženje usluge, kodira okruženje i pridodaju ga odgovoru na poziv metode ciljnog objekta.

Na slici 7.4 prikazan je pojednostavljen dijagram slijeda (eng. *Sequence Diagram*) tijeka niti izvršavanja raspodijeljenog sustava. Program korisnik (ne implementira niti jedan objekt) poziva metodu `metoda` objekta `Objekt_1`, a implementacija metode poziva metodu `metoda` objekta `Objekt_2`. Tijekom presretanja poziva metode unutar programa korisnika stvara se novo okruženje usluge (ne postoji prethodno definirano okruženje usluge), te se zajedno s pozivom prosljeđuje izvršnom okruženju (tj. posredničkoj komponenti) u kojem se nalazi objekt `Objekt_1`. Također, stvara se poruka o događaju i prosljeđuje upravitelju nadgledanja na daljnju obradu. Posrednička komponenta unutar izvršnog okruženja ciljnog objekta presreće poziv, te koristi odgovarajuće metode presretača u ulaznom putu poziva (objekt razreda `ServerInterceptor`) za dohvat prispjelog okruženja usluge (`receive_request_service_contexts`) i sinkronizaciju logičkih satova korisnika i poslužitelja (`recive_request`). Metoda `recive_request` stvara poruku o događaju i prosljeđuje je upravitelju nadgledanja. Nakon obrade poziva u presretaču, poziva se implementacija metode ciljnog objekta. Unutar implementacije metode `metoda` poziva se metoda `metoda` objekta `Objekt_2`. Poziv navedene metode presreće se u njenom izlaznom putu od strane objekta razreda `ClientInterceptor`. Pozivu metode dodaje se okruženje nadgledne usluge (ne stvara se novo okruženje jer ono već postoji za ovu nit poziva u raspodijeljenom sustavu) s vrijednosti lokalnog logičkog sata, te se poziv prosljeđuje posredničkoj komponenti izvršnog okruženja ciljnog objekta. Daljnji tijek događaja prati redoslijed naveden u dosadašnjem tekstu.



Slika 7.4. Dijagram slijeda izvršavanja niti raspodijeljenog sustava

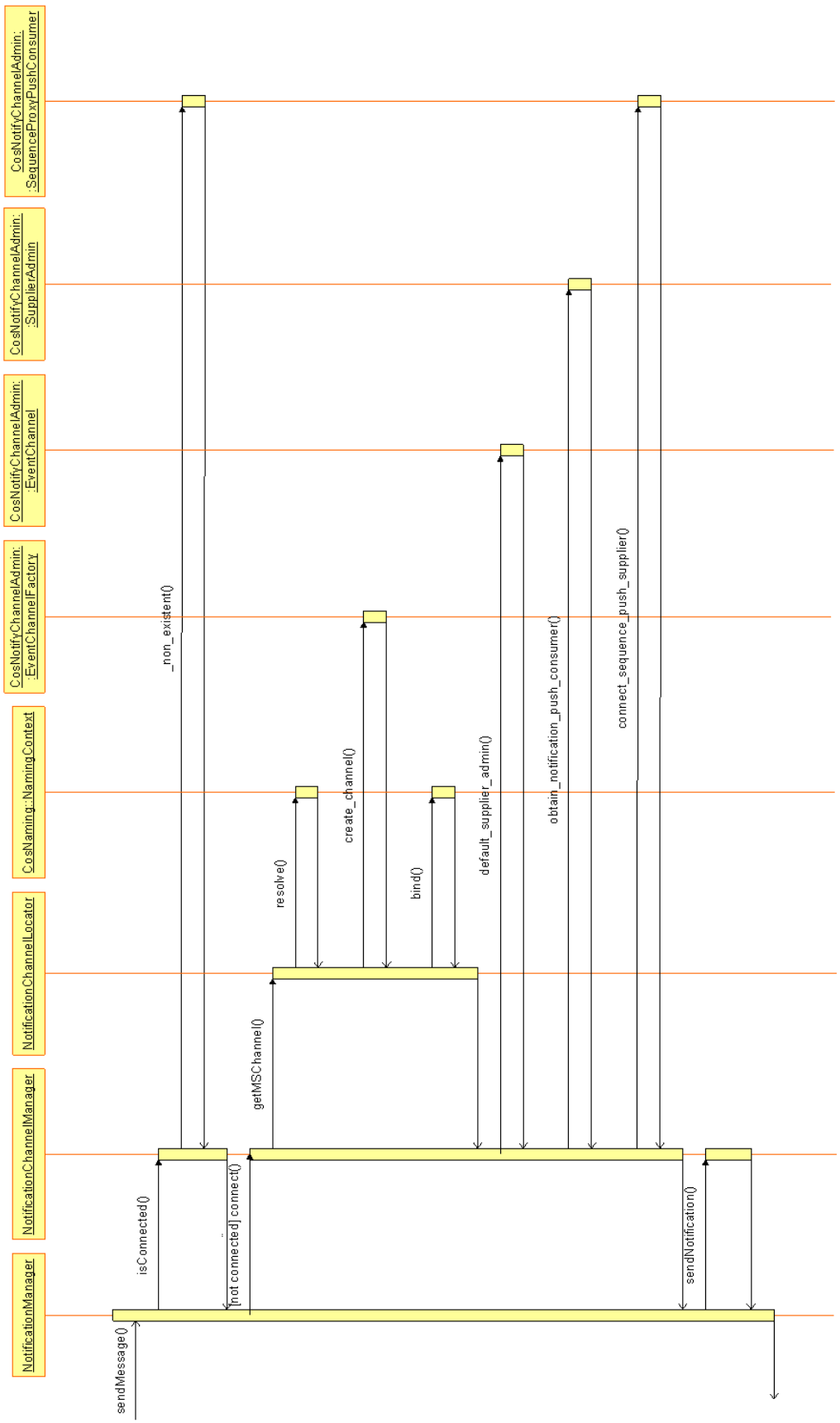
Stvaranje poruke o događajima vrši se na kraju implementacije svih metoda presretača i u ulaznom i u izlaznom putu poziva metode udaljenog objekta s iznimkom metode `receive_request_service_contexts` na strani poslužitelja. Na sljedećem primjeru prikazana je izvedba stvaranja poruke o događaju i njeno prosljeđivanje objektu upravitelju nadgledanja u metodi `ServerInterceptor::receive_request`:

```
//creating a notification message
EventObject* event = new EventObject( "SERVER_COMM", "REQUEST",
      receivedTime, m_pCommonInterceptorData -> getLogicalTimer( ) ->
      getCurrentLT( ), request_uuid, objectID, adapterID,
      m_pCommonInterceptorData -> m_applicationName,
      virtualChannelIOR );

//interception point specific data - operation name
any <<= CORBA::string_dup( info -> operation( ) );
event -> addValue( CORBA::string_dup("OPERATION"), any );

//sending the message
m_pCommonInterceptorData -> getNotificationManager( ) ->
      sendNotification( event );
```

Objekt razreda `NotificationManager` zadužen je za obradu poruka o događajima i njihovo slanje u komunikacijski kanal posredničkog sloja, a čije je ime zadano kao argument prilikom pokretanja izvršne okoline (`-MSDomainChannel` argument). Implementacija metode `sendNotification` provjerava korištenjem objekta razreda `NotificationChannelManager` da li je kanal u koji treba proslijediti poruku aktivan. Aktivnost kanala ispituje se pozivanjem metode `_non_existent` nad objektom koji predstavlja komunikacijski kanal, a nalazi se na poslužitelju usluge poruka. Ako je kanal aktivan (objekt postoji) poruka se prosljeđuje objektu razreda `NotificationChannelManager` na daljnju obradu.

Ukoliko kanal nije aktivan poziva se metoda `connect` objekta razreda `NotificationChannelManager`. Navedena metoda korištenjem objekta razreda `NotificationChannelLocator` pribavlja novi kanal, te stvara novi zastupnički objekt strukturiranog kanala poruka. Također može, u ovisnosti o ostalim parametrima sustava, registrirati lokalni objekt za obradu pretplata na poruke.

Objekt razreda `NotificationChannelLocator` dobavlja referencu na novi kanal unutar poslužitelja usluge poruka (eng. *Notification Service*) i zapisuje ju na predviđeno mjesto unutar imeničke usluge (eng. *Naming Service*). Slijed poziva metoda objekata (kako lokalnih objekata implementiranih jezikom C++, tako i udaljenih objekata) prikazan je na slici 7.5.



Slika 7.5. Dijagram slijeda poziva metoda prilikom stvaranja komunikacijskog kanala

Slanje poruka u komunikacijski kanal u trenutku njihova stvaranja za posljedicu bi imalo primjetno usporenje rada programa tijekom izvođenja koda presretača, a time i znatan utjecaj na rad nadgledanog sustava. Usporenje bi najvećim dijelom bilo posljedica učestale komunikacije s udaljenim objektom koji predstavlja komunikacijski kanal. Stoga je kao jedan od zahtjeva postavljen pred implementaciju nadgledne komponente bilo smanjenje komunikacije s prijenosnim slojem sustava. Dva u tu svrhu korištena mehanizma su:

- slanje više poruka u okviru jednog poziva objekta komunikacijskog kanala
- izdvajanje slanja poruka u zasebnu nit što nižeg prioriteta izvršavanja.

Poruke proslijeđene objektu razreda `NotificationChannelManager` pozivom metode `sendNotification` smještaju se u polje poruka spremnih za slanje u komunikacijski kanal. Ako je, nakon smještanja poruke u polje poruka, polje ispunjeno, obavlja se njihovo slanje u komunikacijski kanal:

- ako se koristi dodatna nit, šalje se poruka niti o popunjenosti polja (nit poziva metodu `_flush` koja implementira istovremeno slanje niza poruka u komunikacijski kanal

- ako se ne koristi dodatna nit, poziva se metoda `_flush` u tekućoj niti izvršavanja.

U početnom dijelu implementacije metode koristi se binarni semafor kako bi se spriječilo istovremeno pristupanje dviju niti polju poruka, te kao posljedicu, nepredviđeno ponašanje sustava:

```
//synchronization
JTCSynchronized synchronized(*this);
```

Zatim se provjerava postojanje niti zadužene za slanje poruka u komunikacijski kanal (ako se zasebna nit koristi), te se, ako ne postoji, stvara:

```
//check if the thread is running ( THREADED policy )
if( ( m_threadPolicy == THREADED ) && (! m_thread ->
  isAlive()) )
{
      m_thread = new NotificationChannelThread
       (this,m_flushInterval);
      m_thread -> setPriority( JTCThread::JTC_MIN_PRIORITY );

      //reset the buffer
      m_currentLength = 0;
      m_eventBuffer -> length( 0 );

      //start the thread
      m_thread -> start( );
}
```

Nit zadužena za slanje poruka predstavljena je objektom razreda `NotificationChannelThread`, a prilikom njegova stvaranja drugi parametar konstruktora označava vremenski period (u milisekundama) nakon čijeg isteka će nit periodički slati pristigle poruke bez obzira da li je polje popunjeno ili ne. Na ovaj način se osigurava da će i prilikom vrlo malog broja događaja u promatranom sustavu oni u razumnom roku biti dostavljeni entitetima korisničkog sloja.

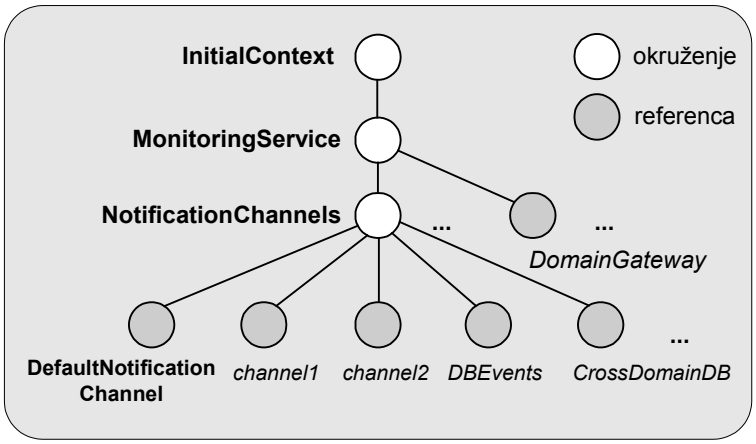
Korištenjem različitih vrijednosti za određivanje veličine polja za pohranu pristiglih poruka i vremenskog perioda pražnjenja polja može se rad nadglednog sustava prilagoditi specifičnom uvjetima nadgledanog sustava.

7.2. Prijenosni sloj

U 6. poglavlju dan je općenit opis prijenosnog sloja i definirana struktura nadgledne domene. Standardni članovi nadgledne domene su usluga poruka (eng. *Notification Service*), imenička usluga (eng. *Naming Service*) i usmjernik poruka domene (eng. *Domain Gateway*). Usluga poruka implementira komunikacijske kanale domene s mogućnošću postavljanja filtara i pretplata na poruke. Nadgledna usluga pretpostavlja postojanje gotovih poslužitelja usluge poruka i imeničke usluge, te njihova implementacija neće biti opisana. U ovom poglavlju bit će opisani organizacija pohrane referenci unutar imeničke usluge, implementacija usmjernika poruka i korištenje pomoćnih alata za administraciju nadgledne domene i usmjernika poruka.

7.2.1. Imeničkа usluga

Imenička usluga predstavlja centralizirano mjesto za pohranu i dohvat referenci na bitne objekte (dijela) raspodijeljenog sustava. Unutar pojedine nadgledne domene samo jedan poslužitelj imeničke usluge sadrži sve reference bitne za njen rad.



Slika 7.6. Organizacija imeničke usluge

Organizacija imeničke usluge slična je organizaciji stabla mapa datotečnog sustava: svaka mapa može sadržavati reference (listovi) i nove mape (grane). Unutar početnog okruženja (eng. *Initial Context*) imeničke usluge nalazi se okruženje pod imenom `MonitoringService` (slika 7.6). Na slici su podebljano prikazani oni elementi koji čine obveznu strukturu zapisa podataka o nadglednoj usluzi, dok su ukošeno prikazani opcionalni elementi. Unutar okruženja `MonitoringService` pohranjuju se sve reference bitnih objekata nadglednog sustava. Izravno u okruženju pohranjuju se reference na entitete prijenosnog sloja. Na slici je prikazana referenca na usmjernik poruka domene, no pored nje može postojati proizvoljan broj pohranjenih referenci na ostale entitete specifične pojedinoj nadglednoj domeni. U okruženju `NotificationChannels` nalaze se reference na sve logičke kanala nadgledne domene. Logički kanal definiran je parom ime-referenca, pri čemu ime predstavlja ime pod kojim je referenca pohranjena unutar okruženja `NotificationChannels`, a referenca pokazuje na stvarnu implementaciju komunikacijskog kanala unutar poslužitelja usluge poruka. Logički kanal `DefaultNotificationChannel` prisutan je kao podrazumijevani kanal unutar svake nadgledne domene (koristi se od strane nadgledne komponente ako prilikom pokretanja komponente raspodijeljenog sustava nije naveden parametar `-MSDomainChannel`). Nadgledne komponente i entiteti korisničkog sloja korištenjem ovih referenci priključuju se na kanale, te u njih šalju ili iz njih primaju poruke. Moguće je i stvaranje dodatnih okruženja unutar okruženja `MonitoringService`, u svrhu pohrane više srodnih referenci nekog entiteta iz

prijenosnog ili korisničkog sloja. Treba naglasiti da se koristi samo *id* komponenta imena okruženja i referenci, dok je *kind* komponenta ostavljena prazna.

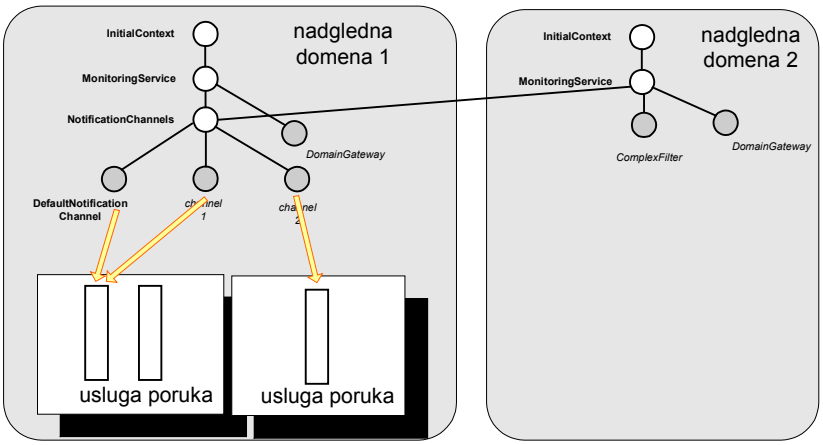
Registracija referenci na komunikacijske kanale unutar okruženja `NotificationChannels` može se vršiti na dva načina:

- korištenjem alata priloženih uz implementaciju imeničke usluge
- korištenjem alata `mdadmin` priloženog implementaciji nadgledne usluge.

U najvećem broju slučajeva, jedan logički (imenovani) kanal čija je referenca pohranjena unutar imeničke usluge označavat će jedan komunikacijski kanal unutar usluge poruka, no moguće su i drugačije organizacije imenovanja kanala:

- više logičkih kanala označava jedan te isti komunikacijski kanal
- komunikacijski kanali mogu biti raspodijeljeni na više poslužitelja usluge poruka, na više računala
- veza između logičkih i komunikacijskih kanala može biti promijenjena u bilo kojem trenutku
- mogu se koristiti dijeljene komponente između dvaju ili više nadglednih domena korištenjem tzv. federacije imeničkih usluga.

Na slici 7.7 logički kanali `DefaultNotificationChannel` i `channel1` označavaju isti komunikacijski kanal. Logički kanal `channel2` koristi komunikacijski kanal na različitom poslužitelju usluge poruka u odnosu na prethodna dva kanala. Druga prikazana nadgledna domena ne posjeduje svoje kanale, već se koristi istom infrastrukturom kao i prva domena.



Slika 7.7. Odnos logičkih i komunikacijskih kanala

7.2.2. Alat MDADMIN

Programski alat `mdadmin` stvoren je u svrhu olakšavanja administracije nadgledne domene. Implementiran je kao konzolna aplikacija, kako bi se mogao koristiti i unutar različitih programskih skripata. Alat podržava sljedeće akcije:

Naredba	Opis
`mdadmin –list`	Ispisuje sve komunikacijske kanale unutar usluge poruka, sve logičke kanale unutar nadgledne domene, te veze komunikacijskih i logičkih kanala.
`mdadmin –create`	Stvara novi komunikacijski kanal unutar usluge poruka.
`mdadmin –destroy <broj_komunikacijskog_kanala>`	Uništava kanal unutar usluge poruka.
`mdadmin –link <ime_logičkog_kanala> <broj_komunikacijskog_kanala>`	Stvara logički kanal i povezuje ga s komunikacijskim kanalom.
`mdadmin –unlink <ime_logičkog_kanala>`	Uništava logički kanal.

Tablica 7.6. Argumenti naredbenog retka alata MDADMIN

7.2.3. Usmjernik poruka nadgledne domene

Usmjernik poruka implementiran je u obliku programa poslužitelja. Njegova uloga opisana je u poglavlju o nadglednim domenama (6.3.1), stoga će u daljnjem tekstu biti opisan način njegove upotrebe i mehanizmi korišteni u implementaciji.

Program poslužitelj usmjernika poruka sadrži u sebi dva objekta, čije se definicije sučelja nalaze u datoteci `DomainGatewayInterfaces.idl`:

```
#include "CosNotifyComm.idl"

module DomainGatewayInterface {

typedef sequence<string> ChannelList;

enum ChannelState { ACTIVE, SUSPENDED, DANGLING, DELETED,
NO_CHANNEL };
```

```
struct ChannelInfo {

      string name;
      string proxyIOR;
      ChannelState state;
};

typedef sequence<ChannelInfo> ChannelInfoList;

exception ChannelNotExist {

};

exception ChannelConnectionException {

      string reason;
};

exception ChannelFilterException {

};

exception ChannelSuspended {

};

exception ChannelNotSuspended {

};

exception AlreadyConnected {

};


interface DomainGatewayPushConsumer :
CosNotifyComm::StructuredPushConsumer {
};

interface DomainGatewayAdministrator {

      //register domain gateway at the <channelName> domain
event
      //channel
      void add( in string channelName ) raises (
ChannelNotExist,
        ChannelConnectionException, AlreadyConnected );

      //remove <channelName> domain event channel from the list
      //of channels
      void remove( in string channelName ) raises (
        ChannelNotExist );

      //listen for event propagations
```

```
      void resume( in string channelName ) raises (
        ChannelNotExist );

      //stop listening for events on the <channelName> domain
      //event channel
      void suspend( in string channelName ) raises (
        ChannelNotExist );

      //list all registered channels
      ChannelList channel_list( );

      //info on channel monitoring status
      ChannelInfo channel_info( in string channelName )
        raises (ChannelNotExist );

      //get filter expression for the channel <channelName>
      string getFilter( in string channelName ) raises(
        ChannelNotExist, ChannelConnectionException );

      //set the filter expression for the channel <channelName>
      void setFilter( in string channelName, in string filter )
        raises( ChannelNotExist, ChannelConnectionException,
        ChannelFilterException );

      //stop the execution of the domain gateway daemon
      void exit( );
};

};
```

Objekt razreda `DomainGatewayAdministrator` tvori upravljačko sučelje usmjernika poruka, a koje se koristi od strane alata `dgadmin` za upravljanje njegovim radom. Objekt razreda `DomainGatewayPushConsumer` implementira funkcionalnost primanja poruka iz jednog ili više komunikacijskih kanala lokalne nadgledne domene i njihova prosljeđivanja u komunikacijski kanal izvorišne nadgledne domene.

Sve nadgledne komponente lokalne nadgledne domene prosljeđuju poruke samo u onaj logički kanal koji je naveden prilikom pokretanja nadgledane komponente raspodijeljenog sustava korištenjem argumenta `-MSDomainChannel` (ako ovaj argument nije naveden, sve poruke prosljeđuju se u logički kanal `DefaultNotificationChannel`). Usmjernik poruka stoga ima mogućnost praćenja većeg broja kanala istovremeno, izdvajanja poruka s definiranim poljem `virtualChannelIOR`, te prosljeđivanja u navedeni kanal izvorišne nadgledne domene.

Dva mehanizma koriste se za smanjenje opterećenja usmjernika poruka:

- filtriranje poruka prikupljenih iz (jednog ili više) kanala
- aktivna politika održavanja veza s kanalima izvorišne nadgledne domene.

U trenutku spajanja usmjernika na komunikacijski kanal (implementacija metode `add` sučelja `DomainGatewayAdministrator`) postavlja se sljedeći filtar (definiran u jeziku Extended-TCL):

```
$virtualChannelIOR != ''
```

Ovaj filtar onemogućuje prosljeđivanje usmjerniku onih poruka koje nemaju definirano polje `virtualChannelIOR`, prebacujući time opterećenje izdvajanja ove grupe poruka na stranu usluge poruka i smanjujući nepotrebnu mrežnu komunikaciju između usluge i usmjernika.

Prosljeđivanje poruke u naznačeni kanal izvorišne nadgledne domene zahtjeva dva koraka:

- uspostavljanje veze s ciljnim kanalom
- slanje poruke.

Uspostavljanje veze s ciljnim kanalom u pravilu je daleko sporije od samog slanja poruke, stoga se u implementaciji usmjernika posebna pozornost posvećena smanjenju broja uspostava veza. Kako svaka komunikacija u nadglednom sustavu rezultira s najmanje dva događaja (a time i dvije poruke) u kratkom vremenskom razmaku, implementiran je mehanizam održavanja veze s kanalom udaljene nadgledne domene u određenom vremenskom periodu nakon što je u njega proslijeđena zadnje pristigla poruka. Taj vremenski period određuje se prilikom pokretanja poslužitelja argumentom `-timeout <milisekundi>`, a ako nije naznačen, podrazumijevana vrijednost je pet sekundi. Podaci o aktivnim vezama s kanalima drugih domena pohranjuje se u tablici veza. U jednom trenutku moguće je imati najviše pedeset aktivnih veza (ako nije drugačije naznačeno argumentom `-channels <najveći broj aktivnih veza>`). Moguća je samo jedna aktivna instanca usmjernika poruka po nadglednoj domeni.

7.2.4. Alat DGADMIN

Program `dgadmin` služi za upravljanje radom usmjernka poruka. Implementiran je kao konzolna aplikacija, a podržava sljedeće naredbe:

Naredba	Opis
`dgadmin –list`	Ispisuje sve kanale sa kojih usmjernik prosljeđuje poruke.
`dgadmin –add <ime_kanala>`	Dodaje kanal u listu kanala s kojih usmjernik prosljeđuje poruke.
`dgadmin –remove <ime_kanala>`	Briše kanal s liste kanala.
`dgadmin –resume <ime_kanala>`	Nastavlja s obustavljenom dostavom poruka s navedenog kanala.
`dgadmin –suspend <ime_kanala>`	Privremeno obustavlja dostavu poruka s navedenog kanala.
`dgadmin –info <ime_kanala>`	Ispisuje stanje kanala i dodatne informacije.
`dgadmin –exit`	Prekida rad usmjernika.

Tablica 7.7. Argumenti naredbenog retka alata DGADMIN

7.3. Korisnički sloj

Sučelje nadglednog sustava prema entitetima korisničkog sloja istovjetno je sučelju usluge poruka. U daljnjem tekstu bit će opisana uporaba nadgledne usluge od strane entiteta korisničkog sloja korištenjem potrebnih sučelja imeničke usluge i usluge poruka. Svi primjeri dani su u jeziku C++ za implementacije usluga tvrtke Object Oriented Concepts.

Za prihvat poruka iz prijenosnog sloja nadglednog sustava dva su osnovna koraka koje entitet korisničkog sloja mora poduzeti:

- pronalaženje reference na logički kanal (ili više kanala) iz kojeg će poruke biti prihvaćane
- spajanje na logički kanal i prihvaćanje poruka.

Dodatne akcije koje entitet nadglednog sustava može poduzeti su:

- prihvaćanje samo onih poruka za koje je entitet zainteresiran definiranjem filtara na kanalu
- definiranje pretplata na tipove poruka za koje je entitet zainteresiran.

Kao prvi korak spajanja na logički kanal potrebno je pronaći referencu na objekt koji predstavlja komunikacijski kanal (logički kanal označava komunikacijski kanal s pridjeljenim imenom). Referenca na kanal pohranjena je u imeničkoj usluzi (struktura imeničke usluge opisana je u tekstu o implementaciji prijenosnog sloja). Sljedeći primjer programskog koda u jeziku C++ dohvaća referencu na objekt komunikacijskog kanala:

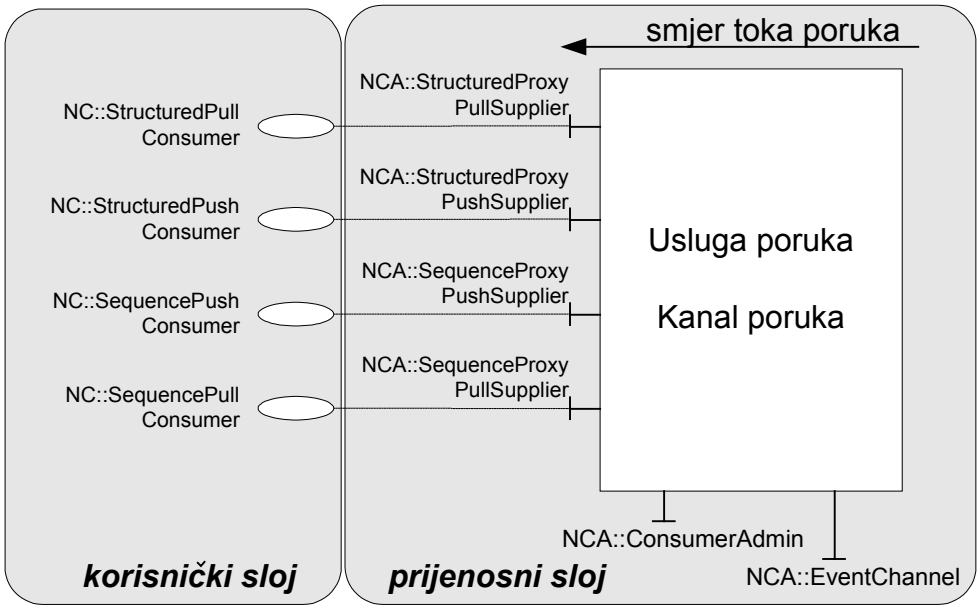
```
...

//StructuredEventChannel
CosNotifyChannelAdmin::EventChannel_var eventChannel;

//handle location of the notification channel using name service

//get the name service IOR
CORBA::Object_var obj;
try
{
      obj = orb -> resolve_initial_references( "NameService" );
}
catch( CORBA::ORB::InvalidName& )
{
      //NameService not found in initial references
      return EXIT_FAILURE;
}

//narrow the object reference
CosNaming::NamingContext_var initialNamingContext;
try
{
      initialNamingContext = CosNaming::NamingContext::_narrow(obj);
}
catch( CORBA::Exception& )
{
      //cannot narrow the NamingService reference
      return EXIT_FAILURE;
}


//create the path to the context containing the channel references
CosNaming::Name name;
name.length(2);
name[0].id = CORBA::string_dup( "MonitoringService" );
name[1].id = CORBA::string_dup( "NotificationChannels" );

try
```

```
{
      obj = initialNamingContext -> resolve( name );
}
catch( CORBA::Exception& )
{
      //cannot resolve NotificationChannel reference
      return EXIT_FAILURE;
}

//narrow the context reference
CosNaming::NamingContext_var channelNamingContext;
try
{
      channelNamingContext = CosNaming::NamingContext::_narrow(obj);
}
catch( CORBA::Exception& )
{
      //cannot narrow the reference
      return EXIT_FAILURE;
}

//look for the NC reference in the channelNamingContext context
CosNaming::Name channelName;
channelName.length(1);

//get the reference name
channelName[0].id = CORBA::string_dup( channel );

try
{
      obj = channelNamingContext -> resolve( channelName );
}
catch( CosNaming::NamingContext::NotFound& )
{
      cout << "NSAdmin: channel does not exist" << endl;
      return EXIT_FAILURE;
}

catch( CORBA::Exception& e )
{
      //some other error, abort search
      return EXIT_FAILURE;
}

//check for nil reference
if( CORBA::is_nil( obj ) )
{
      //nil reference found
      return EXIT_FAILURE;
}

//try to narrow the object to the EventChannel reference
try
{
      eventChannel = CosNotifyChannelAdmin::EventChannel::
        _narrow( obj );
}
catch( CORBA::Exception& )
{
      //the reference obtained from the naming service is
      //not an event channel
```

```
      return EXIT_FAILURE;
}

//check for nil reference
if( CORBA::is_nil( eventChannel ) )
{
      //nil reference found
      return EXIT_FAILURE;
}

...
```

Dohvaćena referenca na objekt komunikacijskog kanala je tipa `CosNotifyChannelAdmin::EventChannel`.

Na slici 7.8 prikazan je model komunikacijskog kanala korištenog u korisničkom sloju nadglednog sustava, a kojeg čini podskup modela usluge poruka.



NCA = CosNotifyChannelAdmin

NC = CosNotifyComm

Slika 7.8. Model komunikacijskog kanala

Entitet prijenosnog sloja, sukladno prikazanom modelu, nema pristupa objektu tvornici usluge poruka, pa time i ne može stvarati nove komunikacijske kanale. Za korištenje dohvaćenog kanala entitet pristupa podrazumijevanom objektu sučelja `NCA::EventChannel` i korištenjem metode `default_consumer_admin()` dohvaća objekt sučelja `NCA::ConsumerAdmin`. Objekt sučelja `NCA::ConsumerAdmin` služi kao tvornica objekata zastupnika, a koje entitet koristi za priključenje na komunikacijski kanal. Iako je moguće stvaranje i korištenje i

drugih, osim podrazumijevanog objekta sučelja `NCA::ConsumerAdmin`, osim u posebnim slučajevima za tim nema potrebe.

Korištenjem objekta sučelja moguće je (u modelu usluge poruka korištenom u nadglednom sustavu) stvoriti četiri vrste objekta zastupnika, u ovisnosti o načinu dobavljanja poruka iz kanala i o broju prenošenih poruka tijekom jednog poziva:

- objekt sučelja `CosNotifyComm::StructuredPullConsumer` – poruke se "izvlače" iz komunikacijskog kanala od strane entiteta, prenose se pojedinačno

- objekt sučelja `CosNotifyComm::StructuredPushConsumer` – poruke se "guraju" entitetu od strane komunikacijskog kanala, prenose se pojedinačno

- objekt sučelja `CosNotifyComm::SequencePullConsumer` – poruke se "izvlače" iz komunikacijskog kanala od strane entiteta, prenosi se više poruka u nizu

- objekt sučelja `CosNotifyComm::SequencePushConsumer` – poruke se "guraju" entitetu od strane komunikacijskog kanala, prenosi se više poruka u nizu.

Odluka o korištenju objekata koji podržavaju "guranje" ili "izvlačenje" poruka iz kanala ovisi o implementaciji pojedinog entiteta korisničkog sloja. Implementacija entiteta koji koriste guranje poziva složenija je stoga što entitet mora implementirati CORBA objekt sučelja `CosNotifyComm::*PushConsumer`, koji je zadužen za prihvaćanje poruka proslijeđenih od strane komunikacijskog kanala.

Sljedeći primjer prikazuje korištenje metode pojedinačnog izvlačenja poruka iz komunikacijskog kanala :

```
...

//get the consumer admin object of the notification channel
CosNotifyChannelAdmin::ConsumerAdmin_var consumerAdmin = eventChannel
 -> default_consumer_admin( );

//get the structured proxy supplier
CosNotifyChannelAdmin::ProxyID proxyId;
CosNotifyChannelAdmin::ProxySupplier_var proxySupplier;

try
{
      proxySupplier = consumerAdmin ->
       obtain_notification_pull_supplier(
```

```
       CosNotifyChannelAdmin::STRUCTURED_EVENT, proxyId );
}
catch(CosNotifyChannelAdmin::AdminLimitExceeded&)
{
    //cannot obtain proxy consumer
      return EXIT_FAILURE;
}

//narrow to the structured proxy pull supplier interface
CosNotifyChannelAdmin::StructuredProxyPullSupplier_var
  structuredProxyPullSupplier = CosNotifyChannelAdmin::
  StructuredProxyPullSupplier::_narrow( proxySupplier );

//check for the right reference type
if( CORBA::is_nil( structuredProxyPullSupplier ) )
      return EXIT_FAILURE;

// Connect to the Proxy Supplier
try
{
      //TODO do not register for callbacks
      structuredProxyPullSupplier->connect_structured_pull_consumer(
          CosNotifyComm::StructuredPullConsumer::_nil( ) );
}
catch(const CosEventChannelAdmin::AlreadyConnected&)
{
    return false;
}

//pull and display structured events from the channel
while( true )
{
      CosNotification::StructuredEvent* event =
       structuredProxyPullSupplier -> pull_structured_event();
      DisplayEvent(*event);
}

return EXIT_SUCCESS;

}
```

Filtriranje poruka moguće je na dvije razine: razini objekta sučelja `CosNotifyChannelAdmin::ConsumerAdmin` i na razini objekta zastupnika. Filtar definiran na razini objekta administratora vrijedi i za sve njime stvorene objekte zastupnike (grupu entiteta priključenu na isti kanal), dok filtar definiran na razini objekta zastupnika vrijedi samo za njega (tj. za pojedini entitet priključen na kanal korištenjem toga objekta zastupnika). Ukoliko filtar nije definiran, entitetu se prosljeđuju sve u kanal poslane poruke.

Filtar se opisuje u jeziku Extended-TCL, podržanom u svakoj implementaciji usluge poruka. Definicija se sastoji iz dva dijela: dijela koji opisuje tip poruke i dijela koje opisuje uvjete na sadržaj poruke. Oba uvjeta moraju biti zadovoljeni da bi poruka bila proslijeđena entitetu.

Sukladno definiciji poruka korištenih u nadglednom sustavu, svaka poruka u svom zaglavlju ima definirano:

- ime usluge (domene) – "`MonitoringService`"
- tip poruke (tj. događaja koji poruka opisuje) - "`CLIENT_COMM`", "`SERVER_COMM`" ili neki od korisnički definiranih tipova.

U prvom primjeru definiran je filtar koji entitetu korisničkog sloja propušta samo poruke vezane uz poslužiteljsku stranu komunikacije nadgledanih entiteta (svih entiteta priključenih na jedan logički kanal):

```
{ ["MonitoringService", "SERVER_COMM"]}
```

U drugom primjeru prikazana je definicija filtra koji propušta samo one poruke koje se odnose na objekte promatranih entiteta vezanih uz prilagodnik objekata pod imenom "zaba":

```
{["MonitoringService", "*"], "$adapterID = 'zaba'"}
```

Treći primjer prikazuje definiciju filtra koji propušta sve poruke o događajima u aplikaciji GPIBServer osim događaja vezanih uz objekt imena "GPIBadmin":

```
{["MonitoringService", "*"],$applicationName='GPIBServer'
and $objectID != 'GPIBadmin'"}
```

Sljedeći primjer prikazuje postavljanje filtra na komunikacijski kanal unutar implementacije usmjernika poruka:

```
...

//narrow to the structured proxy push supplier interface
CosNotifyChannelAdmin::StructuredProxyPushSupplier_var
structuredProxyPushSupplier = CosNotifyChannelAdmin::
   StructuredProxyPushSupplier::_narrow( proxySupplier );

//check for the right reference type
if( CORBA::is_nil( structuredProxyPushSupplier ) )
```

```
      throw DomainGatewayInterface::ChannelConnectionException();

//set the filter constraint (virtualDomainChannel != "")
try
{
      CosNotifyFilter::Filter_var filter = eventChannel ->
        default_filter_factory( ) -> create_filter( "EXTENDED_TCL" );
      CosNotifyFilter::ConstraintExp constraint;
      CosNotifyFilter::ConstraintExpSeq_var constraintSeq = new
        CosNotifyFilter::ConstraintExpSeq( );
      constraintSeq -> length( 1 );
      CosNotification::EventType event;
      event.domain_name = CORBA::string_dup("MonitoringService");
      event.type_name = CORBA::string_dup("*");
      CosNotification::EventTypeSeq_var eventSeq= new
CosNotification::
        EventTypeSeq( );
      eventSeq -> length( 1 );
      eventSeq[0] = event;
      constraint.event_types = eventSeq;
      constraint.constraint_expr = CORBA::string_dup
         ("$virtualChannelIOR != \'\'");
      constraintSeq[0] = constraint;
      filter -> add_constraints( constraintSeq );
      structuredProxyPushSupplier -> add_filter( filter );
}
catch( CORBA::Exception& e )
{
      cout << "filter error" << endl;
}

// Connect to the Proxy Supplier
try
{
    structuredProxyPushSupplier -> connect_structured_push_consumer(
      _eventPushConsumer );
}
catch( ... )
{
      throw DomainGatewayInterface::ChannelConnectionException();
}

...
```

Filtri se mogu dodavati i brisati u toku rada entiteta korisničkog sloja, u skladu s trenutnim zahtjevima na poruke.

7.3.1. Pretplate na poruke

Korištenje pretplata na poruke o događajima mora biti omogućeno prilikom pokretanja nadgledane komponente navođenjem argumenta naredbenog retka `-MSSubscriptions`. Mehanizam pretplata sličan je mehanizmu filtara, s jednom bitnom razlikom: filtri propuštaju ili odbacuju već postojeće poruke na krajevima komunikacijskog kanala, dok mehanizam pretplata omogućuje ili spriječava stvaranje poruke o događaju unutar nadgledne komponente. Korištenje jednog ne isključuje korištenje drugog mehanizma. Dapače, njihova pravilna kombinacija ima za posljedicu bolji rad nadglednog sustava, te minimalni utjecaj na nadgledani sustav.

Osnovna prednost mehanizma pretplata smanjenje je broja stvorenih poruka o događajima. Na strani nadgledne komponente stvaraju se samo oni događaji na koje postoji pretplata, a manji broj poruka ima, u većini slučajeva, za posljedicu smanjenje uvedenog kašnjenja prilikom poziva metoda udaljenih objekata. Nedostatka su dva: memorija unutar nadgledane komponente potrebna za pohranu informacija o pretplatama, te vrijeme potrebno za pronalaženje informacije o postojećim pretplatama tijekom stvaranje poruke o događaju. Stoga se nameće uvjet definiranja što manjeg broja pretplata sa što detaljnijim opisom događaja na koje se pretplaćuje.

Nadgledna komponenta stvarat će sve događaje iz unije pretplata na događaje jednog ili više entiteta korisničkog sloja priključenih na isti komunikacijski kanal. Stoga nije dovoljno korištenje samo pretplata da bi se ograničila vrsta primljenih poruka na strani pojedinog entiteta korisničkog sloja, već je potrebno i definiranje filtra na kraju komunikacijskog kanala.

U osnovi, postoje četiri načina korištenja pretplata na događaje i filtara:

- *Korištenje samo filtara na krajevima komunikacijskog kanala:* nadgledna komponenta stvara poruke o svim događajima, filtri prosljeđuju entitetu korisničkog sloja samo one poruke koje zadovoljavaju postavljene uvjete. U ovom slučaju svi entiteti koji koriste isti komunikacijski kanal moraju se oslanjati isključivo na filtre, niti jedan ne smije koristiti pretplate na događaje ni u kom obliku. Ako makar jedan entitet koristi pretplate, ograničava stvaranje poruka samo za one događaje za koje je zainteresiran.

- *Neovisno korištenje filtara i pretplata na događaje:* s pretplatama se ograničava volumen stvorenih poruka (na tipove poruka za koje postoje zainteresirani entiteti), a s filtrima se ograničava prosljeđivanje poruka pojedinim entitetima. Ovaj način korištenja pretplata omogućava detaljniji opis poruka na koje se entiteti pretplaćuju (detaljnije objašnjeno u daljnjem tekstu).

- *Vezano korištenje filtara i pretplata:* filtri se ovlašćuju na automatsko stvaranje pretplata, u skladu s njihovom definicijom: postavljanje ili promjena definicije filtra automatski, s obzirom na u njemu opisane tipove poruka, podrazumijeva i promjenu pretplata na poruke unutar nadgledne komponente. Ovaj način korištenja pretplata olakšava rad i jamči konzistentnost filtara i pretplata, no smanjuje mogućnost detaljnosti opisa događaja.

- *Korištenje samo pretplata:* uglavnom se koristi kada postoji samo jedan entitet priključen na komunikacijski kanal. Ovaj način korištenja omogućuje detaljniju definiciju pretplata, te brže prenošenje poruka uslijed nepostojanja potrebe za njihovim filtriranjem unutar komunikacijskog kanala.

Entitet korisničkog sloja definira pretplatu korištenjem metode `subscription_change` definirane unutar sučelja `NotifySubscribe`:

```
interface NotifySubscribe {

      void subscription_change(
            in CosNotification::EventTypeSeq added,
            in CosNotification::EventTypeSeq removed )
            raises ( InvalidEventType );

}; // NotifySubscribe
```

Sljedeći primjer prikazuje postavljanje pretplate na sve poruke o događajima tipa CLIENT_COMM, nevezano o (moguće) postavljenom filtru:

```
...
CosNotification::EventType event;
event.domain_name = CORBA::string_dup("MonitoringService");
event.type_name = CORBA::string_dup("CLIENT_COMM");

CosNotification::EventTypeSeq_var added = new
      CosNotification::EventTypeSeq( );
added -> length( 1 );
added[0] = event;

CosNotification::EventTypeSeq_var removed = new
      CosNotification::EventTypeSeq( );
removed -> length( 0 );

structuredProxyPullSupplier -> subscription_change( added,
      removed );
...
```

U drugom primjeru pokazano je vezano korištenje pretplata i filtra: korištenjem metode `attach_callback` zadaća definiranja pretplata prebacuje se na filtar. U ovom slučaju filtar će definirati pretplatu na sve događaje tipa SERVER_COMM pozivom metode `subscription_change` nad objektom `structuredProxyPullSupplier`, tj. komunikacijskim kanalom. Pretplata se vrši na one tipove poruka koji su navedeni unutar `constraint.event_types` polja filtra.

```
...
CosNotifyFilter::Filter_var filter = eventChannel ->
      default_filter_factory( ) -> create_filter( EXTENDED_TCL" );
CosNotifyFilter::ConstraintExp constraint;
CosNotifyFilter::ConstraintExpSeq_var constraintSeq = new
      CosNotifyFilter::ConstraintExpSeq( );
constraintSeq -> length( 1 );
CosNotification::EventType event;
event.domain_name = CORBA::string_dup("MonitoringService");
event.type_name = CORBA::string_dup("SERVER_COMM");
CosNotification::EventTypeSeq_var eventSeq = new
      CosNotification::EventTypeSeq( );
eventSeq -> length( 1 );
eventSeq[0] = event;
constraint.event_types = eventSeq;
constraint.constraint_expr = CORBA::string_dup("");
constraintSeq[0] = constraint;
filter -> attach_callback( structuredProxyPullSupplier );
structuredProxyPullSupplier -> add_filter( filter );
filter -> add_constraints( constraintSeq );
...
```

Standardni način definiranja tipova poruka na koje postoji pretplata omogućuje navođenje samo tipa poruke (CLIENT_COMM, SERVER_COMM, korisnički definirani tipovi), što je u većini slučajeva nedostatno (stvara se nepotrebno velik broj poruka). Stoga je, ukoliko se pretplate koriste nevezano s filtrima ili bez filtara, moguće koristiti i drukčiji format definiranja tipova poruka na koje se žele definirati pretplate: unutar polja `type_name` postavlja se niz karaktera sljedećeg formata:

```
<tip_poruke>[:<ime_poruke>[:<ime_aplikacije>[:<ime_prila-
godnika>[:<ime_objekta>]]]]
```

Unutar bilo kojeg od polja može se navesti "*", a što označava bilo koju vrijednost polja. Na primjer, ukoliko se želi pretplatiti na sve poruke objekata pripadnih prilagodniku objekata "ZABA", definicija pretplate izgledat će ovako:

```
...
CosNotification::EventType event;
event.domain_name = CORBA::string_dup("MonitoringService");
event.type_name = CORBA::string_dup("*:*:*:ZABA");
...
```

Pretplata na sve komunikacijske poruke aplikacije `banksvr` dana je u sljedećem primjeru:

```
...
CosNotification::EventType event1, event2;
event1.domain_name = CORBA::string_dup("MonitoringService");
event1.type_name = CORBA::string_dup("CLIENT_COMM:*:banksvr");
event2.domain_name = CORBA::string_dup("MonitoringService");
event2.type_name = CORBA::string_dup("SERVER_COMM:*:banksvr");

CosNotification::EventTypeSeq_var added = new
      CosNotification::EventTypeSeq( );
added -> length( 2 );
added[0] = event1;
added[1] = event2;
CosNotification::EventTypeSeq_var removed = new
      CosNotification::EventTypeSeq( );
removed -> length( 0 );
structuredProxyPullSupplier -> subscription_change( added,
      removed );
...
```

8. Primjer korištenja nadgledne usluge

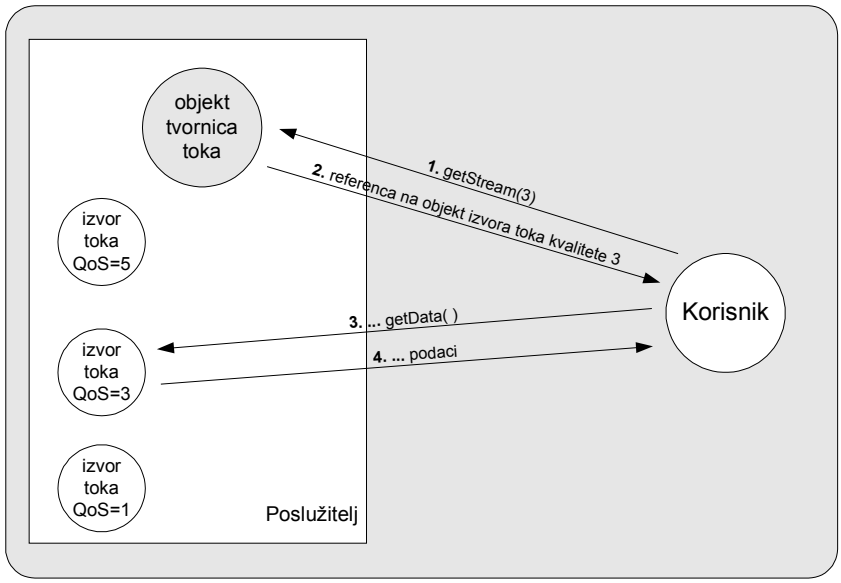
U ovom poglavlju bit će opisan primjer uporabe mehanizma meta programiranja u cilju dodavanja funkcionalnosti osnovnoj izvedbi sustava, te korištenje nadgledne usluge za nadgledanje i upravljanje radom raspodijeljenog sustava. Kao primjer raspodijeljenog sustava uzet je sustav temeljen na korisnik – poslužitelj komunikaciji, a kojemu je zadaća prenošenje i obrada podataka u čvrsto zadanom vremenu. Tipični predstavnici ovakvih sustava su sustavi za prijenos audio i video sadržaja temeljeni na toku podataka od strane poslužitelja do strane korisnika.

Poslužitelj toka podataka sastoji se od jednog objekta tvornice i više objekata izvora podataka. Svaki objekt izvora toka karakteriziran je parametrom kvalitete usluge (eng. *Quality of Service*), a koja u ovom sustavu opisuje količinu i kvalitetu prenošenih podataka. Predviđene su tri izvora podataka s kvalitetama usluge 1, 3 i 5. Sučelja implementirana objektima unutar poslužitelja toka definirana su jezikom IDL:

```
interface StreamServer;
interface StreamSource;

exception StreamRedirect {

      StreamServer server;
};

typedef sequence<octet> streamData;

interface StreamServer {

      StreamSource getStream(in short qos);
      void decreaseQoS( );
      void increaseQoS( );
};

interface StreamSource {

      streamData getData( );
      StreamServer getFactory( );

};
```

Korisnička strana sustava, prilikom njena pokretanja, kontaktira objekt tvornicu poslužitelja toka i zahtijeva određenu kvalitetu usluge pozivom metode `getStream( in short QoS )`(slika 8.1). Reference na sve postojeće objekte tvornice (tj. poslužitelje toka) nalaze se zapisane unutar imeničke usluge, u okruženju /Streams,

pod različitim imenima. Objekt tvornica vraća korisniku referencu na objekt izvora podataka koji najbolje zadovoljava navedeni uvjet kvalitete usluge. Na osnovu vraćene reference na objekt izvor podataka korisnička strana sustava vrši pozive metode `getData( )`, na koji način dohvaća podatke sa strane poslužitelja, te ih obrađuje. Uvjet postavljen pred sustav je da pojedini ciklus dohvata i obrade podataka ne smije trajati više od 1 sekunde.



Slika 8.1. Građa nadgledanog raspodijeljenog sustava

Tijekom rada ovog sustava javljaju se dva osnovna problema:

- prekid dotoka podataka uslijed nedostupnosti poslužitelja
- promjene u radnoj okolini sustava čija je posljedica nemogućnost ispunjavanja zahtijevane kvalitete usluge.

Dio implementacije korisničke aplikacije prikazana je sljedećim programskim kodom:

```
...
      try
      {
            streamData_var data;
            data = streamSource -> getData();
            process( data );

      } catch(CORBA::Exception& e)
      {
            return 1;
      }
...
```

U slučaju da dođe do prekida rada poslužitelja, poziv metode `getData( )` izazvat će pojavu iznimke, te prekid rada programa.

Kako bi, u trenutku prekida rada poslužitelja, mogli izabrati drugi, aktivni poslužitelj, nužno je poduzeti sljedeće korake, istovremeno ne mijenjajući implementaciju samog programa korisnika:

- prepoznati iznimku koja označava grešku prekida veze s poslužiteljem
- pronaći aktivni poslužitelj
- dohvatiti referencu na objekt izvor podataka s istom kvalitetom usluge kao i kod prethodno korištenog objekta
- preusmjeriti neuspio poziv na novi objekt izvor podataka.

Opisana funkcionalnost implementirana je korištenjem prenosivih presretača. Objekt presretač razreda `FailOverInterceptor` nasljeđuje razred `ClientRequestInterceptor`, te implementira samo tijelo metode `receive_exception( )`. Navedena metoda poziva se u trenutku pojave iznimke kao rezultata poziva metode udaljenog objekta. Implementacije metode prikazana je sljedećim programskim kodom:

```
void FailOverInterceptor::receive_exception(
PortableInterceptor
  ::ClientRequestInfo_ptr info ) throw( PortableInterceptor
  ::ForwardRequest, CORBA::SystemException )
{
      if( setRecursiveFlag( info ) )
            return;

      if( ! strcmp(info -> operation(), "getData") )
      {
            CORBA::Any_var any = new CORBA::Any;
            any = info -> received_exception();
            CORBA::COMM_FAILURE* pException;
            if( any >>= pException )
            {

                  StreamServer_var streamServer;

                  streamServer = StreamServerLocator::
                    getStreamServer(FailOverInitializer::m_orb);

                  if( CORBA::is_nil( streamServer ) )
                  {
```

```
                    return;
                }

                StreamSource_var streamSource;

                try
                {
                streamSource = streamServer -> getStream(
                  QualityOfServiceHolder::m_qos);
                }
                catch( CORBA::Exception& e)
                {
                    return;
                }

                clearRecursiveFlag( );

                throw PortableInterceptor::ForwardRequest
                  (streamSource, CORBA::Boolean(true));
            }
        }

        clearRecursiveFlag( );
}
```

Unutar koda metode presretača ispituje se da li je poziv nastao kao rezultat poziva metode udaljenog objekta unutar implementacije presretača. Ako je, događaj se ne obrađuje. Zatim se provjerava da li je iznimka nastupila tijekom poziva metode `getData( )`, ako nije događaj se ne obrađuje. Također se ispituje da li je iznimka tipa `COMM_FAILURE` (prekid veze s poslužiteljem). Ako su svi navedeni uvjeti ispunjeni, poziva se metoda `getStreamServer`, koja vraća referencu na prvi aktivni poslužitelj registriran u okruženju `/Streams` imeničke usluge. Ako aktivni poslužitelj nije pronađen, presretnuta iznimka prosljeđuje se implementaciji aplikacije. Ako je poslužitelj pronađen, dobavlja se referenca na objekt izvor toka iste kvalitete usluge istovjetne prethodno korištenom objektu izvoru, te se podiže iznimka tipa `ForwardRequest`, koja će biti presretnuta od posredničke komponente, te će poziv metode udaljenog objekta biti preusmjeren na novi objekt izvor toka, transparentno u odnosu na programski kod aplikacije. Sve dok je makar jedan registrirani poslužitelj toka aktivan, program korisnik nastavit će s radom, ne primjećujući promjenu korištenih objekata izvora toka.

Dok je u prethodno opisanom primjeru korištenje nadgledne usluge bilo nepotrebno, u sljedećem, prilagodbi na promjene u radnom okruženju, od velikog je značaja, kao

pomagalo u odgovoru na dva od tri pitanja nužna za prilagodbu raspodijeljenog sustava – čemu se prilagoditi i kada izvršiti prilagodbu.

Kvaliteta usluge, zadana prilikom dohvaćanja reference objekta izvora toka, podložna je promjenama u radnoj okolini sustava – opterećenju računala na kojima se izvode poslužitelj i korisnik, te propusnošću računalne mreže. U slučaju da jedan od (ili oba) nabrojenih čimbenika u određenom vremenskom razdoblju ne omogućuje osiguranje prijenosa toka željene kvalitete, potrebno je smanjiti kvalitetu usluge, kako u pogledu kvantiteta tako i u pogledu kvalitete prenošenih podataka. Kao najočitiji primjer može se navesti prenošenje audio podataka određene kvalitete u stvarnom vremenu – kašnjenje u prijenosu podataka rezultirat će isprekidanim zvukom na korisničkoj strani sustava. Stoga je poželjno, u trenutku kada se zaključi da održavanje postojeće kvalitete usluge nije moguće, privremeno smanjiti kvalitetu usluge (npr. smanjenjem kvalitete prenošenih audio podataka, tj. smanjenjem količine prenošenih podataka) u cilju održavanja stalnosti toka.

Osnovni kriterij koji treba primijeniti prilikom nadgledanja rada sustava je da li korisnik uspijeva dohvatiti i obraditi podatke u za to predviđenom vremenu od 1 sekunde. Prosječno vrijeme ciklusa dohvata i obrade prati se u periodima od po pet sekundi, kako bi se izbjegle nepotrebne česte promjene kvalitete usluge uslijed kratkotrajnih smetnji u okolini. Kao uvjet povećanja kvalitete usluge (najviše do razine kvalitete navedene prilikom dohvaćanja reference na objekt izvor toka) uzima se trajanje prijenosa podataka manje od 0,4 sekunde.

Tri su moguća uzroka produljenja ciklusa dohvata i obrade:

- opterećenost računala na kojemu se izvršava poslužitelj toka
- opterećenost računala na kojemu se izvršava korisnik
- smanjena propusnost računalne mreže.

Za prepoznavanje navedenih uzroka potrebno je pratiti sljedeće parametre izvršavanja sustava:

- trajanje ciklusa dohvata i obrade podataka
- trajanje poziva metode na strani poslužitelja
- trajanje prenošenja podataka mrežom.

Ovi parametri mogu se izračunati na osnovi podataka sadržanih u porukama o događajima:

- trajanje ciklusa dohvata i obrade: vrijeme proteklo između pojave dvaju uzastopnih događaja `CLIENT_COMM.REQUEST`

- trajanje poziva metode na strani poslužitelja: vrijeme proteklo između pojave događaja `SERVER_COMM.REQUEST` i `SERVER_COMM.REPLY`

- trajanje prenošenja podataka mrežom: razlika između vremena trajanja poziva na strani korisnika (`CLIENT_COMM.REPLY`, `CLIENT_COMM.REQUEST`) i trajanja poziva metode na strani poslužitelja (`SERVER_COMM.REPLY`, `SERVER_COMM.REQUEST`).

Upravljanje radom sustava vrši se unutar entiteta korisničkog sloja – programa `StreamController` – koji prikuplja poruke o sustavu proslijeđene od strane nadglednih komponenti u prijenosni sloj. Nadgledne komponente aktivne su i unutar poslužitelja toka i unutar programa korisnika. Koriste se samo poruke o komunikacijskim događajima.

Prilikom pokretanja upravljačkog programa `StreamController` na komunikacijski kanal u koji se prosljeđuju poruke o nadgledanom sustavu postavlja se sljedeći filtar:

```
...
event.domain_name = CORBA::string_dup("MonitoringService");
event.type_name = CORBA::string_dup("*");
...
constraint.constraint_expr = CORBA::string_dup("$OPERATION ==
  \'getData\'");
...
```

Time se poruke proslijeđene upravljačkom programu ograničavaju samo na one komunikacijske poruke koje se odnose na pozive metode `getData( )` objekta izvora toka.

Poruke vezane uz jedan poziv udaljene metode grupiraju se na osnovu oznake niti izvršavanja raspodijeljenog sustava sadržane u polju `ExecutionTraceUUID` zaglavlja poruke.

Ako se analizom podataka prikupljenih dospjelim porukama zaključi da je potrebno prilagoditi trenutno korištenu kvalitetu usluge, poduzimaju se sljedeći koraci:

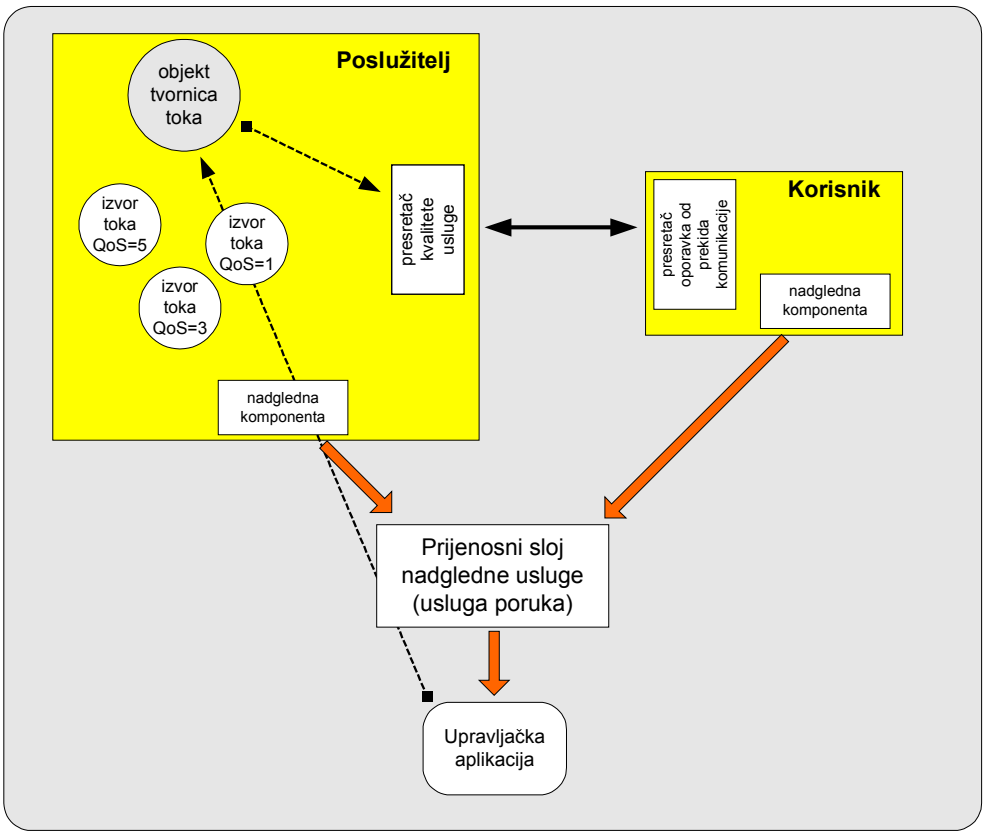
- iz polja EFFECTIVE_TARGET tijela poruke CLIENT_COMM.REQUEST dobavlja se referenca na pozivani objekt izvor toka

- na dobavljenoj referenci poziva se metode `getFactory( )`, koja vraća referencu na objekt tvornicu

- na objektu tvornici (sučelja `StreamServer`) poziva se jedna od metoda:
  - `increaseQoS( )` – povećanje trenutne kvalitete usluge
  - `decreaseQoS( )` – smanjenje trenutne kvalitete usluge.

Kao posljedica poziva jedne od gore navedenih metoda, prilikom prvog sljedećeg poziva metode `getData( )` objekta izvora toka, poziv će biti presretnut od strane presretača, te će biti preusmjeren na objekt izvor toka s manjom ili većom kvalitetom usluge (ovisno o pozvanoj metodi i postojanju objekta s većom ili manjom kvalitetom usluge).

Cjelokupni sustav sa svim dodatnom komponentama za upravljanje njegovim radom prikazan je na slici 8.2.:

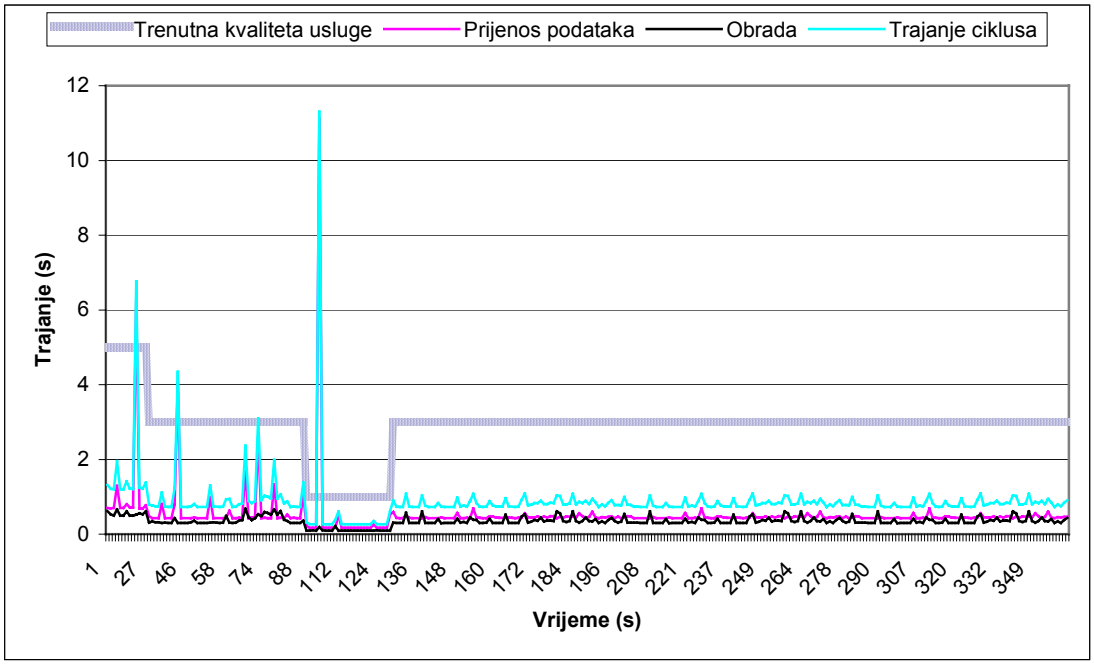


Slika 8.2. Slika sustava s dodatnim komponentama za upravljanje

Raspored komponenti nadgledanog sustava sličan je rasporedu korištenom prilikom mjerenja utjecaja nadgledanja (poglavlje 7.1.1): na jednom računalu izvršavao se

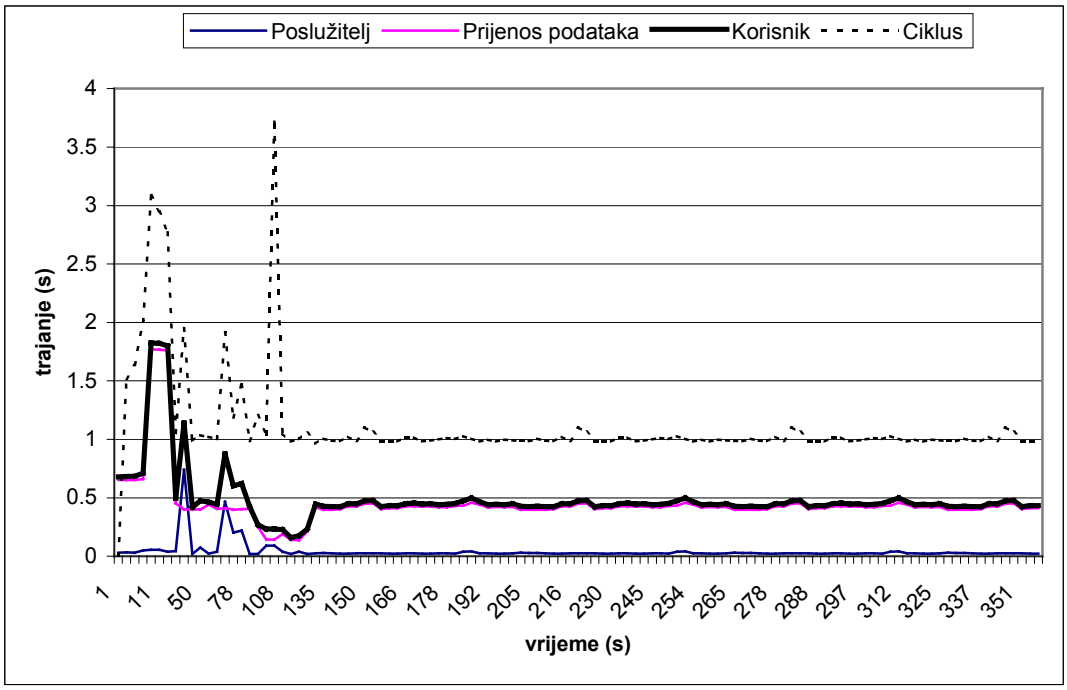
poslužitelj toka, na drugom program korisnik, a na trećem ostale komponente sustava: usluga poruka, imenička usluga i upravljački program.

Slika 8.3. prikazuje vremena izmjerena na strani korisnika tijekom izvršavanja sustava u trajanju od 5 minuta: na početku rada korisnik zahtijeva kvalitetu usluge razine 5, no vremenom se ispostavlja da željenu razinu nije moguće održati (vrijeme ciklusa dohvata i obrade kreće se oko 1.3 ms). U 29. sekundi izvršavanja kvaliteta usluge smanjuje se na 3 i održava stabilnom sve do prvog zastoja u radu (računalo na kojem se program korisnik izvršava postaje opterećeno). U 95. sekundi javlja se i dugačak zastoj, uzrokovan prebacivanjem neaktivnih stranica memorije na tvrdi disk, no smanjenje kvalitete usluge više nije moguće. Nakon određenog vremena opterećenje računala pada, te upravljač podiže kvalitetu usluge i održava je stabilnom do kraja rada sustava.



Slika 8.3. Podaci o izvršavanju sustava na strani korisnika

Na slici 8.4. prikazani su podaci o izvršavanju sustava na strani upravljačke aplikacije, na osnovu kojih su donošene odluke o promjeni kvalitete usluge. Treba napomenuti da se korišteni podrazumijevani parametri nadglednih komponenti (veličina međuspremnika i period slanja poruka u kanal), pri čemu je u nadgledanju ovog sustava bitan parametar samo period slanja poruka (5 sekundi). Za točnije nadgledanje i bržu reakciju na promjene mogao se koristiti i kraći period slanja, no u tom slučaju bi utjecaj nadglednog sustava na rad nadgledanog (i upravljanog) sustava bio veći. Ovim primjerom također je dokazano da nadgledna usluga koegzistira sa drugim presretačima aktivnima u nadgledanim komponentama raspodijeljenog sustava.



Slika 8.4. Podaci o izvršavanju sustava na strani upravljača

9. Zaključak

Jedna od osnovnih karakteristika današnjih računalnih sustava njihova je komunikacijska povezanost. Računalne mreže dominiraju kako u lokalnim sredinama (lokalne računalne mreže unutar organizacijskih jedinica), tako i na globalnom planu (Internet). Koristeći mogućnosti koje današnji računalni sustavi nude, organizacija i građa modernih programskih sustava slijedi koncepciju fizičke udaljenosti i komunikacijske povezanosti njenih građevnih komponenti.

Evolucijski proces razvoja mrežne infrastrukture i programskih sustava kao posljedicu ima raznorodnost korištene sklopovske i programske opreme. Uvođenje novih komponenti u sustav stoga postavlja dodatne zahtjeve na njihovo oblikovanje i implementaciju. U svrhu pojednostavljenja razvoja raspodijeljenih aplikacija uveden je tzv. međusloj. Umetnut između sloja implementacije aplikacije i sloja mrežne komunikacije, međusloj preuzima na sebe zadatke vezane uz komunikaciju i prevladavanje raznorodnosti komponenti raspodijeljenog sustava.

Standard CORBA čini najrasprostranjeniji platformski- i jezično-neovisan međusloj, danas korišten u izgradnji raspodijeljenih aplikacija. CORBA se temelji na opisu sučelja raspodijeljenih objekata korištenjem opisnog jezika sučelja neovisnom o krajnje korištenom programskom jeziku implementacije i platfomski-neovisnom programskom sučelju. No iako od pomoći, CORBA ne uklanja bitne razlike između oblikovanja i implementacije neraspodijeljenih i raspodijeljenih sustava. Latencija mrežne komunikacije, pristup dijeljenoj memoriji, usporednost izvršavanja i pojava djelomičnog zatajenja sustava svojstveni su raspodijeljenim sustavima. Njima uzrokovane pogreške ne mogu se detektirati i otklanjati standardnim alatima predviđenim za razvoj i otklanjanje pogrešaka u neraspodijeljenim sustavima.

Novije generacije raspodijeljenih sustava temeljenih na CORBA-i, osim ispunjavanja pred njih postavljenih funkcionalnih zahtjeva (rješavanje zadataka iz domene problema), zahtijevaju i prilagodbu rada sustava promjenama u radnom okruženju. Za uspješno provođenje prilagodbe nužan je ugrađen mehanizam prilagodbe (eksplicitan ili implicitan). Također, nužno je poznavanje stanja čitavog (ili dijela) sustava, kako bi se mogao prepoznati uzrok i pravi trenutak provođenja prilagodbe.

Nadgledanjem rada raspodijeljenog sustava temeljenog na CORBA-i moguće je pronalaženje pogrešaka u radu, kao i prikupljanje informacija o stanju dijela sustava ili sustava u cjelini, te njihovo korištenje tijekom prilagodbe. Nadgledanje raspodijeljenih sustava mora ispuniti dva, često oprečna zahtjeva: pružiti cjelovitu i ispravnu sliku rada sustava, te imati minimalan utjecaj na nadgledani sustav. Pod

utjecajem na sustav podrazumijevaju se potrebne izmjene u programskom kodu sustava i utjecaj nadgledanja na izvršavanje nadgledanog sustava.

Postojeći sustavi za nadgledanje rada raspodijeljenih sustava temeljenih na CORBA-i ne zadovoljavaju jedan ili oba prethodno navedena zahtjeva, ili posjeduju druge nedostatke koji ograničavaju njihovu uporabu. Nedostaci se očituju u nužnosti velikih promjena programskog koda nadgledanih sustava, nepredvidljivosti utjecaja nadgledanja na rad sustava, ograničenosti rješenja na samo jedan programski jezik implementacije, zatvorenosti rješenja samo na rješenja ponuđena od stane jednog proizvođača sustava ili neprenosivosti rješenja između različitih implementacija sustava CORBA.

Stoga je osmišljen i razvijen novi sustav za nadgledanje rada raspodijeljenih sustava temeljenih na CORBA-i. Osnovne prednosti ovog sustava su transparentnost postupka nadgledanja s obzirom na nadgledan sustav, minimalan utjecaj na rad nadgledanog sustava, prenosivost sustava na sve platforme, jezike implementacije i većinu implementacija sustava CORBA, te prilagodljivost različitim načinima njegove uporabe.

Postupak nadgledanja oslanja se na praćenju događaja unutar nadgledanog sustava, te stvaranju i prosljeđivanju poruka o događajima zainteresiranim stranama. Poruke o događajima čine trag izvršavanja sustava, sukladno modelu asinkrone mreže. Uporabom mehanizama meta-programiranja sustava CORBA izbjegnuto je grupiranje raznorodnih funkcionalnosti na razini entiteta domene problema, čime je postignuta neovisnost i transparentnost programskog koda nadgledanja (djelomično i prilagodbe) s obzirom na nadgledani sustav.

Nadgledni sustav građen je modularno, od više zasebnih komponenti svrstanih u tri sloja. Prvi sloj sustava čine nadgledne komponente smještene unutar izvršnih okruženja komponenti nadgledanog sustava. Posebna pozornost posvećena je implementaciji nadgledne komponente s ciljem unošenja što manjeg kašnjenja u rad nadgledanog sustava. Srednju razinu sustava čini sustav komunikacijskih kanala za prenošenje poruka zainteresiranim entitetima trećeg sloja. Odjeljivanjem proizvođača od potrošača poruka sustavom komunikacijskih kanala postignuta je njihova neovisnost i predvidljivost u nadgledani sustav unesenog kašnjenja. Filtriranjem poruka i pretplatama na promatrane događaje omogućeno je praćenje samo onih događaja u sustavu za koje postoji zainteresiranosti, smanjujući time utjecaj na nadgledani sustav i količinu prenošenih podataka. Treći sloj sustava čine entiteti – potrošači poruka. Cjelokupna komunikacija među komponentama nadglednog sustava ostvarena je isključivo putem CORBA-e, čime je osigurana otvorenost sustava i neovisnost pojedinih komponenti sustava o korištenoj računalnoj platformi ili jeziku implementacije.

Mogućnosti poboljšanja sustava postoje, prvenstveno bi ih trebalo tražiti u segmentu slanja poruka u komunikacijske kanale, a što trenutno unosi najveće kašnjenje u rad nadgledanog sustava. Automatsko upravljanje strukturom komunikacijskih kanala i filtara na osnovu nadgledanog sustava moglo bi se pokazati od važnosti u nadgledanju raspodijeljenih sustava s vrlo velikim brojem komponenti. Također, nužan je razvoj općenitih alata korisničkog sloja u cilju jednostavnog praćenja rada, vizualizacije i validacije nadgledanih sustava. Dodatna područja koje bi nadgledna usluga trebala obuhvatiti čine sustavi CORBA za rad u stvarnom vremenu, nadolazeći standard CORBA temeljenih komponenti, te integracija s raspodijeljenim arhitekturama temeljenim na Javi. Primjena autonomnih pokretnih objekata kao dijela nadglednog sustava također čini jedno od područja koje bi trebalo pomnije ispitati.

10. Literatura

[1] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.4 edition", October 2000.

[2] Grady Booch, "Object-oriented analysis and design with applications", 2nd edition", Addison-Wesley Publishing Company, 1995.

[3] Object Management Group, "CORBAServices", December 1997.

[4] Pattie Maes, "Concepts and Experiments in Computational Reflection", Proceedings of OOPSLA'87, ACM SIGPLAN Notices, 22, 147-155, ACM Press

[5] G.S. Blair, G. Coulson, P. Robin, M. Papathomas, "An Architecture for Next Generation Middleware", Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, The Lake District, England, November 1998.

[6] N. Wang, K. Parameswaran, D. Schmidt, "The Design and Performance of Meta-Programming Mechanisms for Object Request Broker Middleware", 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), San Antonio, Jan/Feb, 2001.

[7] Nancy. A. Lynch, "Distributed Algorithms", Morgan Kaufmann Publishers, Inc., 1996.

[8] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[9] M. Henning, S. Vinoski, "Advanced CORBA Programming with C++", Addison-Wesley, 1999.

[10] S. Baker, "CORBA Distributed Objects Using Orbix", Addison-Wesley, 1997.

[11] Object Management Group, "Portable Interceptors, Joint Revised Submission", December 1999.

[12] Waldo, J., Wyant, G., Wollrath, A., Kendall, S., "A Note on Distributed Computing", Sun Microsystems Laboratories, November 1994.

[13] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. C. Magalhães, and R. H. Campbell, "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB", IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000). New York, April 3-7, 2000

[14] I. Sommerville, "Software Engineering, 5th edition", Addison-Wesley, 1995.

[15] Jeffrey J.P. Tsai, Yaodong Bi, Steve J.H. Yang, Ross A.W. Smith, “Distributed Real-Time Systems – Monitoring, Visualization, Debugging and Analyisis”, John Wiley & Sons, Inc., 1996.

[16] The CORBA-Assistant, White Paper, Fraunhofer Institut, June 1997.

[17] MAScOTTE Project, White Paper, ESPRIT Project 20804, May 1997.

[18] Günther Rackl, “Monitoring Globus Components with MIMO”, Technical Report, Institut für Informatik, Technische Universitat München, November 1999.

[19] Günther Rackl, “Multi-layer Monitoring in Distributed Object Environments”, Institut für Informatik, Technische Universitat München, November 1999.

[20] D. C. Schmidt, “The Distributed Object Visualization Environment”, White Paper, Washington University, St. Louis, 1997.

[21] Jina Mao, “Monitoring and Analyzing Method Invocations in the 2k Operating System”, M. Sc. Thesis, University of Illinois, 1999.

[22] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L.C. Magalháes, R.H. Campbell, “Monitoring, Security and Dynamic Configuration with the dynamicTAO Reflective ORB”, Proceedingsof the Middleware 2000 Conference , ACM/IFIP, April 2000.

[23] N. K. Diakov, H. J. Batteram, H. Zandbelt, M. J. van Sinderen, “Monitoring of Distributed Component Interactions”, Workshop on Reflective Middleware RM’2000, New York, April 2000.

[24] N. K. Diakov, M. J. van Sinderen, D. Quartel, “Monitoring Extensions for Component-based Distributed Software”, 5th International Conference on Protocols for Multimedia Systems, Cracow, October 2000.

[25] C. Gransart, P. Merle, J.M. Geib, “GoodeWatch: Supervision of CORBA Applications”, ECOOP'99 Workshop, Object-Orientation and Operating Systems, June 1999.

[26] M. J. Katchabaw, S. L. Howard, H. L. Lutfiyya, A. D. Marshall, M. A. Bauer, “Making Distributed Applications Manageable Through Instrumentation”, Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE’97), 1997.

[27] S: Majumdar, I. Ahmad, E. Shen, I.A. Fatah, “Achieving High Performance Through Middleware Reconfiguration”, Workshop on Reflective Middleware RM’2000, New York, April 2000.

[28] D. C. Schmidt, V. Kachroo, Y. Krishnamurthy, F. Kuhns, “Developing Next-generation Distributed Applications with QoS-enabled Middleware”, IEEE Communications vol.17, October 2000.

[29] N. Wang, M. Kircher, D. C. Schmidt, “Applying Reflective Middleware Techniques to Optimize a QoS-enabled CORBA Component Model Implementation”, Proceedings of the COMPSAC 2000, Taipei, Taiwan, October 25-27, 2000.

[30] Object Management Group, “Notificaton Service Specification 1.0”, June 2000.

[31] Dougles C. Schmidt, “The Adaptive Communication Environment”, Technical Paper, Department of Computer Science, Washington University, St. Louis, 1993.

[32] A. Elrayess, J. A. Rolia, “Reducing Coupling in Distributed Management Frameworks”, IEEE Third International Workshop on Systems Management, Newport, Rhode Island, 22-24 April, 1998.

[33] Java Management Extensions White Paper, Sun Microsystems,

[34] G. Bullen, “AppCenter White Paper”, Borland

[35] A. S. Tannenbaum: “Distributed Operating Systems”, Prentice Hall, 1995.

[36] Object Management Group, “Discussion of the OMA”, January 1997.

[37] Object Management Group, “Telecom Log Service Specification 1.0”, January 2000.

[38] Armin Stranjak, “Autonomni pokretni objekti u raspodijeljenim računalnim sustavima”, magistarski rad, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2000.

A Sažetak

Ovaj rad opisuje načine i mehanizme prilagodbe raspodijeljenih računalnih sustava temeljenih na arhitekturi CORBA. Sukladno potrebi prilagodbe novije generacije raspodijeljenih sustava promjenama u radnom okruženju, osmišljen je i izrađen sustav praćenja njihova rada. Objašnjene su tehnologije raspodijeljenih sustava s posebnim osvrtom na arhitekturu CORBA. Analizirani su problemi svojstveni fazama oblikovanja, izgradnje i korištenja sustava, te njihovo rješavanje temeljeno na postupcima nadgledanja. Dan je kraći pregled postojećih metoda nadgledanja i njihovih svojstava. Izložen je pojam računalne refleksije i mehanizmi meta-programiranja u sustavu CORBA. Detaljno je opisana građa i izvedba usluge praćenja rada raspodijeljenog sustava, načini njena korištenja, te analizirana njena svojstva. Naposljetku, predstavljen je primjer praćenja i upravljanja radom postojećeg sustava korištenjem razvijene usluge.

Ključne riječi: raspodijeljeni sustav, CORBA, nadgledanje, prilagodba, upravljanje

B Summary

Adaptive Object Oriented Distributed Computer Systems

This thesis addresses methods and mechanisms of CORBA-based distributed system adaptation. Regarding the need of adaptation to environmental changes present in new generation of distributed systems, a new monitoring system has been designed and implemented. Distributed system technologies have been explained, primarily focusing on CORBA. Problems present in design, implementation and operation phases have been analyzed and solutions based on system monitoring proposed. Concept of reflection in computer systems and CORBA related meta-programming mechanisms have been elucidated. Design and implementation of the CORBA monitoring service have been described in detail, usage explained and properties analyzed. Finally, an example of monitoring and application steering on the existing distributed system has been presented using the developed service.

Keywords: distributed system, CORBA, monitoring, adaptation, control

C Životopis

Igor Čavrak rođen je 15. XI. 1971. godine u Rijeci. Studij na Elektrotehničkom fakultetu započeo je 1990. godine, a diplomirao u ljeto 1996. godine na smjeru Računarska tehnika. Zaposlen je na Fakultetu elektrotehnike i računarstva od rujna 1996. godine. Poslijediplomski studij , smjer Jezgra računarskih znanosti, upisao je u ožujku sljedeće godine. Bavi se raspodijeljenim sustavima, s naglaskom na sustavima arhitekture CORBA. Objavio je pet radova na znanstvenim skupovima u zemlji i inozemstvu.