SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Goran Salamunićcar

Modeliranje pristupa kolekcijama raznorodnih podataka unutar informacijske infrastrukture

Magistarski rad

Zagreb, 2000.

Mentor: Prof. dr. sc. Vlado Glavinić

Magistarski rad ima 107 stranica i kao prilog jedan optički disk.

Rad br. _____.

Povjerenstvo za ocjenu rada:	Prof. dr. sc. Leo Budin	– predsjednik
	Prof. dr. sc. Vlado Glavinić	– mentor
	Doc. dr. sc. Bojana Dalbelo Bašić	

Povjerenstvo za obranu rada:	Prof. dr. sc. Leo Budin	– predsjednik
	Prof. dr. sc. Vlado Glavinić	– mentor
	Doc. dr. sc. Bojana Dalbelo Bašić	

Datum obrane rada: 20. lipnja 2000. g.

ZAHVALE

Ovom prilikom želio bih se zahvaliti svima koji su mi pomogli prilikom izrade magistarskog rada, kako kroz stručnu tako i kroz svaku drugu pomoć.

Svom mentoru prof. dr. sc. Vladi Glaviniću, dipl. ing. želim se zahvaliti, kako za pomoć prilikom izrade ovog rada, tako i na samom uvođenju u problematiku znanstvenog rada kroz moj završetak studija s naglaskom na znanstvenoistraživačkom radu, kada sam kao koautor s mentorom i objavio svoje prve znanstvene radove na međunarodnim konferencijama, te na literaturi koju mi je stavio na raspolaganje prilikom izrade ovog magistarskog rada.

Zahvaljujem se i svim ostalim suradnicima sa znanstvenoistraživačkog projekta 036033 *Elementi arhitektura za regionalne informacijske infrastrukture*, koji su mi svojim savjetima i komentarima pomogli da izrada ovog rada prođe u najboljem mogućem redu. Pri tome bih se posebno zahvalio i mr. sc. Borisu Motiku, dipl. ing. koji mi je među ostalim pomogao prilikom izbora samog područja magistarskog rada, te na literaturi koju mi je pritom posudio.

Također, želio bih se zahvaliti i svima iz firme u kojoj sam zaposlen od 1. lipnja 1997, *PELSYS, d.o.o.* Prije svega, direktoru mr. sc. Ivici Jovanoviću, dipl. ing. na razumijevanju za vrijeme koje mi je bilo potrebno da napišem ovaj rad, te na radnom mjestu Windows programera u Visual C++ jeziku koje mi je izuzetno pomoglo da sam rad napravim u skladu s profesionalnim pristupom koji se vezano za ovu problematiku koristi u svijetu. Posebno bih se želio zahvaliti i kolegama sa posla: Zdravku Nežiću, dipl. ing. i Dragi Balenu, dipl. ing. koji su mi svojim stručnim savjetima pomogli da riješim mnoge probleme vezane za samu problematiku ovog rada.

Na kraju, želio bih se zahvaliti i svojoj kako užoj tako i široj obitelji, na vremenu, razumijevanju i podršci prilikom izrade ovog rada. Prije svega, ovdje bih želio spomenuti majku Višnju, sestru Emu, Vladu, te posebno i baku Miru te djeda Petra.

Ovaj rad izrađen je u sklopu znanstvenoistraživačkog projekta 036033 *Elementi arhitektura za regionalne informacijske infrastrukture* s financijskom potporom Ministarstva znanosti i tehnologije Republike Hrvatske te Istarske Županije.

Mojoj majci Višnji, sestri Emi, i obitelji koju ću jednom imati :)

KAZALO

1. UVOD

Jedna od značajnih usluga koju trebaju pružati informacijske infrastrukture, globalne računalno -informacijske mreže koje objedinjavaju elemente računalne mreže i korisnički usmjerene elemente raspodijeljenih aplikacija, pristup je podacima raznorodnog tipa, formata i semantike. U svrhu olakšanja i poboljšanja interakcije s raspodijeljenim informacijskim sustavom podržanim informacijskom infrastrukturom, ideja je da se osigura jednoobraznost pristupa njegovim podacima definicijom standardnog sučelja. Da bi se osigurala jednoobraznost pristupa podacima informacijske infrastrukture, potrebno je provesti integriranje raznorodnih kolekcija raspodijeljenih podataka. To znači da model koji se razmatra u ovom radu treba znati pristupiti nekom podatku unutar informacijske infrastrukture, bez obzira na kojoj računalnoj arhitekturi i pripadajućem operacijskom sustavu je isti pohranjen. To mogu biti baze podataka za veliki broj korisnika kao npr. DB2, Oracle, SQL Server, Sybase i Informix, ali i za manji broj korisnika tipa Btrieve, dBase, Access i Paradox, ili ostali izvori podataka kao što su elektronička pošta, razno razne datoteke, tablice i sl.

Sljedeće što je potrebno osigurati jest pristup tim podacima s bilo koje računalne arhitekture odnosno pripadajućeg operacijskog sustava, kao što je npr. UNIX, Linux, OS2 Worp 4, Mac OS 8, MS-DOS, 16-bitni Windowsi, Windows CE, 32-bitni Windowsi (Windows 95, Windows 98, Windows NT 4.0 Workstation i Server) i mnogi drugi. Za svaku od ovih platformi potrebno je osigurati da korisnik može pristupiti ostatku sustava. Činjenica da se i sami podaci i korisnici mogu nalaziti na različitim operacijskim sustavima, samo je jedan od problema koji se javljaju prilikom razrade modela univerzalnog pristupa podacima. Štoviše, ova problematika obuhvaća dosta znanstvenih područja međusobno više ili manje povezanih, kojim se znanstvenici intenzivno bave negdje od 70-tih, usporedo i sa samim razvojem baza podataka uz koje je i najviše povezana.

Nekoliko takvih znanstvenih područja obuhvaćeno je problematikom *heterogenih i autonomnih baza podataka* [1][5][64][103][105][106][111][114][123]. Dobar pregled ove problematike dat je u nedavno objavljenoj knjizi [30] gdje nekoliko desetaka autora svaki za svoje područje daje uvod u problematiku, kao i pregled trenutno aktivnih projekata u svijetu koji se s istom bave.

Autonomnost baza podataka prvo je takvo područje, i proučava utjecaj zahtjeva za autonomnošću na integriranje i sposobnost zajedničkog djelovanja s drugim bazama podataka. Naime, često je čisto stvar politike kompanija odnosno organizacija da same žele imati potpunu kontrolu nad svojom bazom odnosno bazama podataka, te da žele ograničiti i imati kontrolu pristupa drugih. No isto tako često je poželjan i neki zajednički rad s drugim sustavima, koji je potrebno ostvariti, ali tako da se ne naruše principi autonomnosti. Ovi zahtjevi obično su proturječni, i općenito ni teoretski ne postoji savršeno rješenje već su uvijek potrebni neki kompromisi.

Sintaksne i semantičke različitosti su sljedeće područje, koje se posebno bavi problemom različitog značenja jednako nazvanog atributa u različitim bazama, kao što se može dogoditi i da različiti atributi u različitim bazama imaju isto značenje. Slično je i sa samim tablicama, kao i sa bilo kojom kombinacijom istih. Tu je i problem interpretacije; naime, ukoliko npr. i znamo da se radi o nekom iznosu i sredstvu plaćanja, sama valuta može biti implicitno određena ovisno o kojoj

zemlji se radi. Slično je i sa mjernim jedinicama, te različitim interpretacijama i drugih vrijednosti. Također, sama činjenica da smo došli do nekog podatka ne znači da smo došli i do neke informacije koja nam nešto znači. Problemi postoje i ukoliko se promatraju samo baze podataka unutar jednog jezičnog područja i jedne zemlje, no svakako su još samo složeniji ukoliko se promatraju globalno odnosno u cjelini.

Sljedeće područje bavi se *objedinjavanjem više zasebnih baza podataka* u neku veću cjelinu, što može biti jedna jedinstvena baza podataka ili nekoliko distribuiranih baza podataka. Ovaj problem često se javlja kad se primjerice dvije firme žele udružiti u jednu, te je nakon toga potrebno objediniti i njihove baze podataka. Problem se javlja i ukoliko firma već ima neku bazu podataka (recimo za knjigovodstvo), a proširivanje poslovanja kupnjom neke dodatne programske podrške (recimo za elektroničku trgovinu) zahtijeva dodatnu bazu podataka. U takvim slučajevima potrebno je objediniti rad tih baza. Ovdje također ne postoje savršena rješenja, već strategije kako se najjednostavnije odnosno najbrže ili najjeftinije može doći do cilja. Često se zna dogoditi da je objedinjavanje dva ili više sustava i skuplje rješenje nego sve raditi iz početka, što nam ukazuje na kompleksnost ovog područja. Dobar pregled takvih strategija dat je u [94].

Jedna od tehnika objedinjavanja više zasebnih relacijskih baza podataka je *integriranje njihovih shema* [89][132]. U literaturi se dosta često navodi jedna analiza dvanaest različitih strategija za objedinjavanja shema [4], ne toliko zato što je to dvanaest najboljih ili najčešće korištenih strategija, već upravo stoga što se vidi kako se i u ovom području ne nude savršena već kompromisna rješenja za različite specifične primjene, zbog čega je područje zanimljivo za znanstvena istraživanja, ali i zbog čega je zbog složenosti najbolje ako se u praksi može izbjeći potreba za objedinjavanjem više zasebnih baza podataka u neku veću cjelinu.

Protok poslova sljedeće je područje koje je također povezano kako s Internetom [10] tako i s bazama podataka. Naime, osnovne zamjerke prvim sustavima protoka poslova upravo je bila kako sami podaci nisu bili spremljeni u bazi odnosno bazama podataka. Pri tome se misli i na same podatke koji su bili predmet obrade protoka poslova, kao npr. neki elektronički dokument odnosno podaci neophodni za generiranje nekog elektroničkog dokumenta, tako i na same kontrolne podatke, kao npr. stanje protoka poslova, log datoteke tipe tko je što učinio i slično. Baze podataka su nešto gdje se iz sigurnosnih razloga kao i zbog pouzdanosti podaci tradicionalno drže, i velike kompanije nisu htjele prihvatiti takve sustave poslova upravo zato što su to zanemarili. Samim time što novi sustavi koriste baze podataka kako bi u njima držali relevantne podatke, s druge strane i nasljeđuju svu problematiku koja je neminovno i vezana sa samim bazama podataka.

Slična stvar je i s *elektroničkom trgovinom*. Zbog istih razloga kao i kod protoka poslova, ispravna rješenja jedino su ona koja podatke drže u bazi podataka. Kako broj korisnika Interneta (u Europi za 1998-u zabilježen porast za 98%), tako i količina ukupne trgovine putem Interneta (u svijetu za 1998-u zabilježen porast od 226%) raste izuzetno brzo, ovo je također izuzetno zanimljivo područje za daljnja istraživanja [20][31][36][62][102][119][128], posebno što je vezano i za baze podataka i za Internet. Među ostalim, u zadnje vrijeme postoje i projekti kojima je cilj već "klasičan" oblik elektroničke trgovine posebno proširiti kako agentima koji će pregovarati za robe i usluge tako i razno-raznim oblicima aukcijskog načina trgovanja [57].

Međusobna *povezanost baza i Interneta* sljedeće je područje koje je u zadnje vrijeme posebno zanimljivo [12][52], a inače je i posebno povezano sa prethodna dva područja odnosno elektroničkim trgovanjem i protokom poslova. Njegova posebna važnost direktno slijedi iz već objašnjene tradicionalne važnosti baza podataka, te s druge strane izuzetno velike popularnosti Interneta i njegovog brzog rasta.

Ovdje se i sam *progres Interneta*, kao danas najveće globalne računalne mreže, može također promatrati kao zasebno znanstveno područje [15][34][66][67][85]. Pri tom se primjena nalazi gotovo u svim područjima, od projektiranja integriranih krugova na mreži [42] do promidžbenih

stranica pojedinaca [120], kao i za dijeljenje zajedničkih resursa [45], stvaranje velike računalne snage za paralelno procesiranje [134], pametno pretraživanje koristeći agente [145] i slično. U zadnje vrijeme ima dosta radova koji pokušavaju predvidjeti do kojih promjena će doći kod Interneta [23][59][60][76], od čega ih je dosta vezano za slijedeću verziju Internetovog mrežnog protokola IPv6 [39][68] (Internet Protocol version 6), kao i za sljedeće verzije protokola HTTP (HyperText Transfer Protocol) [40][61] te proširivanja jezika IDL (Interface Definition Language) u WIDL (Web IDL) [90][137] i specifikacije dokumenata [144]. Na kraju, ovdje je i posebno zanimljivo promatrati daljnji progres pitanja intelektualnog vlasništva, odnosno utjecaj Interneta na pristup intelektualnom vlasništvu i inovacijama, patentima, kao i njihov utjecaj na daljnji razvoj Interneta [51][56][96].

Zasebno područje je i *pretraživanje vrlo velikih baza podataka* [17][38][49][50]. Ovdje je problem u tome što samo izvršavanje upita može trajati vrlo dugo, te se koriste tehnike koje već za vrijeme pretraživanja korisniku stavljaju na raspolaganje trenutno dostupne rezultate, koje on pak može koristiti za modifikaciju samog upita. Drugi segment ovog područje sastoji se u tome kako iz vrlo velike količine podataka izvući relevantne informacije, odnosno zaključiti što je bitno, a što ne.

Kao zasebno područje može se promatrati i proučavanje odnosno *osiguravanje zajedničkog rada* [22][70][100], iako ono zasigurno pokriva dobrim dijelom i područja već nabrojana. Osobitost ovog područja su arhitekture i paradigme koje pritom koristimo, kao npr. CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), i druge.

Kroz sva ova područja vezana sa samom problematikom univerzalnog pristupa podacima, vidi se da se same baze podataka u većoj ili manjoj mjeri redovito javljaju kao vrlo važan faktor, a treba naravno uzeti u obzir i da svakim danom pronalaze i nove primjene, kao npr. u sustavima za rad u realnom vremenu [125], i drugdje. Sljedeće što se može primijetiti jest da je ovo izuzetno široko znanstveno područje, gdje ni pojedini segmenti odnosno zasebna područja ne nude uvijek zadovoljavajuća rješenja, te stoga nije uvijek moguće pronaći niti zadovoljavajuće ukupno rješenje. Stoga na univerzalni pristup podataka nije ispravno gledati kao na nekakav zadatak kojeg je potrebno izvršiti, jer to jednostavno nije moguće, već kao skup znanstvenih područja gdje se u jednom ili drugom segmentu može dati veći ili manji doprinos.

Uz to, od važnosti za praktičnu primjenu je proučiti i paradigme razvoja programske podrške, koje će biti od najveće važnosti prilikom rješavanja konkretnih problema u pristupima podacima unutar neke informacijske infrastrukture. Uz sam model za pristup raznorodnim podacima, potrebno je razraditi i same tehnike programiranja koje će se koristiti prije svega za razvoj prototipa modela, a kasnije i same konkretne programske podrške. To je kao i u slučaju paradigma za razvoj programske podrške potrebno jednostavno stoga što sam pristup s jedne strane uvijek manje ili više ograničava što se sve može napraviti, a s druge određuje konkretne vremenske rokove unutar kojeg će nešto i biti napravljeno. Sve to izuzetno je važno gledano sa inženjerskog stajališta, gdje je uz teoretske osnove uvijek važnost i u primjenjivosti odnosno izvedivosti u samoj praksi.

Stoga je posebno zanimljivo proučiti primjenjivost objektno orijentiranog i komponentno temeljenog pristupa u izgradnji modela za pristup raznorodnim podacima. Važnost objektno orijentiranog pristupa je u ovom slučaju prvenstveno u ponovnoj iskoristivosti već napisanog koda, pošto se izvori podataka informacijske infrastrukture mogu grupirati u skupine koje mogu imati dosta zajedničkih elemenata, te se kod razrade programske podrške za pristup istim onda može među ostalim koristiti i nasljeđivanje i polimorfizam. Među ostalim, zanimljivo je pogledati primjenu objektno orijentiranog pristupa kod razvoja distribuiranih aplikacija [46][47], Internet aplikacija [41][82], kao i ostale moguće primjene, tehnike odnosno metrike [32][87]. Važnost komponentno temeljenog pristupa proizlazi iz složenosti samog modela za pristup raznorodnim podacima. Naime, prilikom razrade arhitekture istog prije ili kasnije morati će se doći do arhitekture, gdje će se pojedinim modulima podržavati pojedini izvori odnosno grupe izvora podataka. Takav pristup potreban je zato što unaprijed ne možemo znati koji će se sve izvori

podataka odnosno tipovi izvora podataka nalaziti unutar neke informacijske infrastrukture, kao što ne možemo znati niti kako će razvoj tehnologije utjecati na svojstva već postojećih. Stoga je takve potrebno podržati izgradnjom novih modula po već utvrđenim pravilima, koji će se jednostavno uklopiti u ostatak sustava, a ne mijenjanjem cijelog sustava razno raznim nadogradnjama. Dodatni argument za komponentno temeljen pristup je i svojevrstan trend zadnjih godina, da se ne samo vrlo velike već i srednje velike aplikacije izrađuju kao skup komponenata, a ne kao jedna monolitna aplikacija [84][88][99]. Pri tome je posebno zanimljivo promotriti komponentno temeljen pristup sa stajališta izgradnje distribuiranih aplikacija [71][79], sigurnosti, nekim specifičnim primjenama kao npr. u znanosti [28], predviđanja razvoja programske podrške u bližoj budućnosti [63], a naravno potrebno je proučiti i sve ostale kako pozitivne [86] tako i eventualne negativne [9] strane ovakvog pristupa.

Kroz drugo poglavlje, ukratko se rezimiraju sama istraživanja u svijetu vezana za područje informacijskih infrastruktura, daje kratki pregled projekata koji se bave ili su se bavili tom problematikom, te razmatra važnost informacijskih infrastruktura za razvoj društva u cjelini.

Kroz treće poglavlje, nakon kratkog uvoda u problematiku, razrađuje se osnovna arhitektura modela za pristup raznorodnim izvorima podataka unutar informacijske infrastrukture. Arhitektura modela se zasniva na suvremenom pristupu ostvarenja troredne aplikacije za Internet i intranete koristeći i objektno orijentirani i komponentno temeljeni pristup. Dalje se u radu definira i model posredničkog sučelja putem srednjeg reda modela, uključivo i korisničko sučelje raspodijeljenog informacijskog sustava koje se razrađuje u korisničkom redu modela.

Kroz četvrto poglavlje se razrađuje sama jezgra modela, njegov posrednički red. Opisuju se problemi koji se inače javljaju pri klasičnom razvoju aplikacija, kako bi se izbjegli prilikom implementacije posredničkog reda. Potom se daje pregled svojstava komponentno temeljene paradigme koja se zajedno s objektno orijentiranom koristi kao osnova izgradnje posredničkog reda i modela u cjelini. Zatim se opisuje jedna moguća implementacija posredničkog reda odnosno samog modela, te se pri tome daje i pregled tehnologija koje se pri tome mogu koristiti kako bi se posrednički red mogao implementirati na što jednostavniji način na operacijskom sustavu Windows NT Server. Dalje se u skladu s objektno orijentiranim i komponentno temeljenim pristupom razrađuje i sam COM (Component Object Model) kao tehnologija implementacije jezgre posredničkog reda. Pri tome je posebno stavljen naglasak na tome kako i koja svojstva COM-a koristiti prilikom implementacije različitih dijelova posredničkog reda, te kako proširiti mehanizme ponovne iskoristivosti, prije svega agregaciju, kako bi se COM što bolje prilagodio samoj arhitekturi modela za pristup raznorodnim podacima.

Kroz peto poglavlje, daje se postupak izgradnje modela specijaliziranog za sam pristup raznorodnim sustavima poslovanja relacijskih baza podataka. Prije svega obrazlaže se motivacija za uniformnim pristupom raznorodnim sustavima poslovanja relacijskih baza podataka, te se daje osnovna arhitektura prototipa modela za pristup istim koji je u osnovi pojednostavljena verzija općenitog modela predloženog i razloženog kroz prethodna poglavlja. Nakon toga, opisuje se ispitni poligon različitih sustava poslovanja relacijskih baza podataka nad kojim je sam model ispitivan, te se opisuju problemi na koje se naišlo prilikom implementacije kao i načini njihovog rješavanja kroz ovaj magistarski rad, i to prije svega oni vezani za rukovanje tablicama rezultata upita, internacionalizaciju/lokalizaciju, indeksiranje i integritet podataka, te raznolikost tipova podataka. Na kraju se ilustrira sam pristup kroz izradu virtualne knjižnice na mreži.

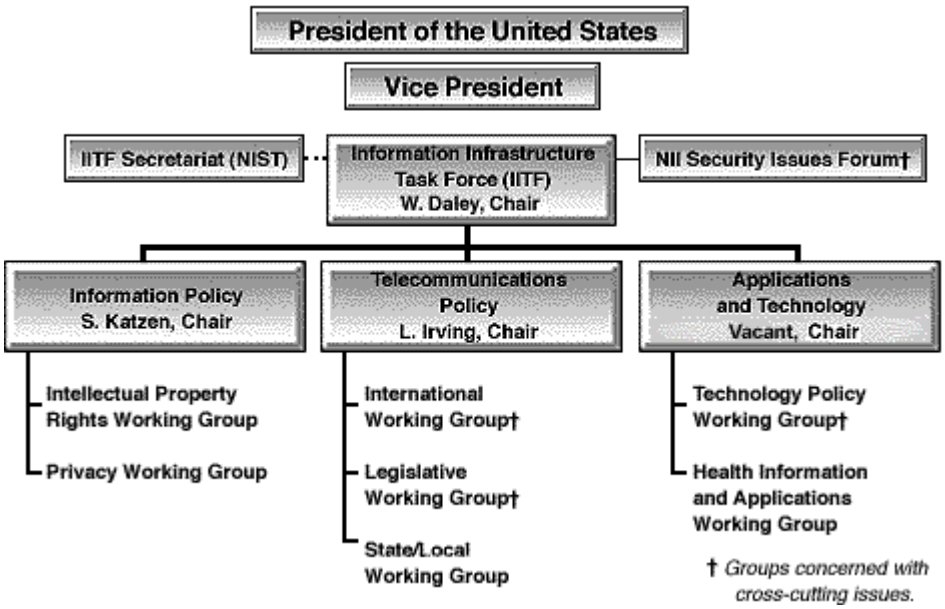
Kroz šesto poglavlje provodi se analiza svojstvenih funkcionalnosti modela, što uključuje i analizu otvorenosti modela koja je ključna za njegovu proširivost, i ostale elemente kao što su sigurnosni protokoli odnosno zaštita samih podataka. U sedmom poglavlju daje se sam zaključak.

2. INFORMACIJSKE INFRASTRUKTURE

Jedna od glavnih zadaća informacijskih infrastruktura je osigurati funkcionalniji, brži i jeftiniji pristup odnosno distribuciju raznorodnih podataka. Pri tome je od posebne važnosti uniformnost pristupa koja na te parametre direktno utječe. Kroz ovo poglavlje daje se kratki pregled istraživanja u svijetu vezanih za samo područje informacijskih infrastruktura, razmatra njihova važnost za razvoj društva u cjelini, te daje pregled i ostalih zadaća stavljenih pred njih.

Vlada SAD-a je 1993 formirala IITF (Information Infrastructure Task Force) radne grupe [http://iitf.doc.gov/], prikazane na Slici 1, kako bi se informacijska infrastruktura razvila i implementirala u skladu s njihovim administrativnim potrebama. Kroz IITF djeluju najviši državni dužnosnici koji odigravaju presudnu ulogu u razvoju i primjeni informacijskih i telekomunikacijskih tehnologija. IITF se sastoji od tri savjeta: za informacijska pitanja, za telekomunikacijska pitanja, te za primjenu informacijske infrastrukture i razvoj pripadnih tehnologija. Devet najznačajnijih principa odnosno ciljeva koje je IITF odredio su:

- poticanje investicija privatnog sektora;
- osigurati univerzalni pristup raznorodnim podacima kako bi isti bili dostupni svima uz pristupačnu cijenu;
- poticati razvoj i primjenu tehnoloških inovacija;
- poticati razvoj univerzalnih, interaktivnih usluga vođenih od strane korisnika;
- osigurati sigurnost podataka i pouzdanost mreže;
- usavršit sustav za dodjelu koncesija radio-frekvencijskih područja;
- zaštititi intelektualno vlasništvo;
- osigurati osiguravajuću koordinaciju s drugim vladinim uredima i drugim državama;
- osigurati pristup podacima vladinih ureda te zaštitu odnosno sigurnost istih.



Slika 1: IITF radne grupe.

U svrhu razvoja što boljeg i kvalitetnijeg globalnog rješenja, IITF surađuje sa sljedećim svjetskim strukovnim i interesnim organizacijama:

- ATM (Asynchronous Transfer Mode forum) – međunarodna ne-profitna organizacija za razvoj ATM proizvoda i usluga;
- GIIC (Global Information Infrastructure Commission) – udruga za globalni razvoj informacijskih infrastruktura čiji cilj je poticaj privatnih i privatno/javnih sektora za ulaganje u informacijske infrastrukture s ciljem razvoja informacijske mreže i usluga kako bi se osigurao globalni ekonomski rast te kvalitetnije obrazovanje i svakodnevni život [http://www.gii.org/];
- NSRC (Network Startup Resource Center) – ne-profitna organizacija čiji cilj je kroz razne projekte Azije, Afrike, Južne Amerike i Kariba, Bliskog Istoka i Oceanije, razvoj i integracija pripadajućih tehnologija računalnih mreža u svrhu kvalitetnijeg međusobnog povezivanja [http://www.nsrc.org/];
- NTIA (National Telecommunications and Information Administration) – agencija Ministarstva trgovine SAD-a čiji cilj je međunarodni razvoj telekomunikacijskih i informacijskih tehnologija, poticanje inovacija, otvaranje novih radnih mjesta, te pružiti korisnicima informacijskih infrastruktura više boljih i kvalitetnijih usluga uz prihvatljivije cijene [http://www.ntia.doc.gov/];
- PTC (Pacific Telecommunications Council) – pacifički telekomunikacijski koncil, ne-profitna ne-vladina međunarodna organizacija davatelja i korisnika komunikacijskih usluga [http://www.ptc.org/];
- INMARSAT (International Mobile Satellite Organization) – međunarodna organizacija koja je među prvima u svijetu 1979. započela razvoj globalnih komunikacija putem satelita, danas kroz tu organizaciju sudjeluju 84 zemlje u svrhu daljnjeg razvoja komunikacija putem satelita [http://www.inmarsat.org/];
- INTELSAT (International Telecommunications Satellite Organization) – komercijalni davaoci usluga satelitskih komunikacija [http://www.intelsat.int/];
- ITU (International Telecommunications Union) – međunarodna organizacija preko koje vlade i privatne kompanije koordiniraju globalne telekomunikacijske mreže i usluge [http://www.itu.ch/];
- OECD (Organization for Economic Co-operation and Development) – predstavlja udruženje 29 zemalja u organizaciji koja njihovim vladama daje osnovu za raspravu o ekonomskim i socijalnim pitanjima te koordinaciju samog rješavanja istih [http://www.oecd.org/].

Što se tiče akademskih istraživanja vezanih za informacijske infrastrukture, posebno se treba izdvojiti projekt sa Harvarda vezan za informacijske infrastrukture te Massachusetts Institute of Technology projekt vezan za razvoj inteligentnih informacijskih infrastruktura.

HIIP (Harvard Information Infrastructure Project) projekt [http://ksgwww.harvard.edu/iip/], zasnovan je 1989 kao rezultat zajedničkih aktivnosti "John F. Kennedy School of Government" Sveučilišta Harvard i "Harvard Law School". Osnovni cilj bio je osigurati neutralan, interdisciplinaran forum, koji će rješavati pitanja vezana za informacijske infrastrukture, njihov razvoj, korištenje i rast. Na projektu zajedno rade stručnjaci iz vlade, industrije i akademskih institucija, sa zadaćom da obuhvate iskustva menadžera, ekonomista, odvjetnika, znanstvenika koji se bave političkim znanostima te stručnjaka za nove tehnologije u svrhu razumijevanja i rješavanja problema vezanih za razvoj informacijskih infrastruktura. Projekt je još uvijek u tijeku, a izvještaj o dosadašnjim rezultatima istraživanja dostupan je preko Interneta [148].

MIT-ov projekt inteligentnih informacijskih infrastruktura, pokrenut je listopada 1993 s ciljem razvoja vrlo generalnog načina distribucije i pristupa podacima koji će se temeljiti na Internet protokolima. Cilj projekta je razvoj inteligentnih sustava za pohranu, pristup, rukovanje i

distribuciju informacija, i to koristeći primarno glavne Internet protokole kao što su elektronička pošta i World-Wide Web. Projekt se sastojao od sljedeća tri segmenta:

- razvoj usluga poslužitelja – cilj je strukturirati komunikaciju tako da su korištenjem inteligentnih agenata usluge na jednostavan način dostupne svim korisnicima informacijske infrastrukture; strukturirana komunikacija sastoji se od automatizirane obrade obrazaca, građenja modela uobičajenih interakcija s korisnikom, ili analize interakcije korisnika i informacijske infrastrukture za samog vremena pristupa; početne usluge sastoje se među ostalim od pretraživanja podataka, osiguravanja dostupnosti tih podataka, i traženja optimalnog (najbržeg) pristupa do tih podataka;
- razvoj organizacije poslovanja – cilj je razviti sustav koji će omogućiti velikim grupama korisnika da zajednički rješavaju probleme kroz sistematsku hijerarhijsku dekompoziciju zadataka te kroz ponovno integriranje njihovih rješenja; osnovna je ideja omogućiti organizacijama da na efikasniji način rukuju kompleksnošću takvih sustava koji su u principu temeljeni na razmijeni podataka između korisnika u obliku hiperteksta; ključno je pitanje rukovanje uskim grlima u takvim sustavima kad komunikacija između puno korisnika konvergira prema komunikaciji između par korisnika koji rezimiraju cijeli proces; tehnologija rješavanja ovog problema razvijena je i ispitana kroz sam projekt;
- praćenje prirodnog jezika – cilj je po smislu analizirati tekst u semantički prikaz iz kojeg će se onda moći generirati inteligentnije odnosno logički ispravnije rečenice; težište ovog istraživanja je na novom deklarativnom analizator-generator paru koji se može izgraditi na temelju proširenja već implementiranih istraživačkih sustava; kroz ovaj dio projekta primjenjuju se nove paradigme za semantičku percepciju, prezentaciju znanja i smisleno zaključivanje.

Projekt MIT-a služi kao centar za distribuciju elektroničkih publikacija Bijele Kuće, a tehnologije razvijene kroz sam projekt koriste se i kako bi vladine institucije postale dostupnije običnim građanima. Praktičnu primjenu projekt je najviše pokazao kroz automatsku distribuciju višemedijskih dokumenata, kroz razvoj okruženja za grupno donošenje odluka te kroz automatsko rukovanje protokom zadataka. Razvijene tehnologije uključuju statičku kategorizaciju dokumenata, provjeru njihove izvornosti, učenje računala (induktivni načini učenja kao i oni koji se zasnivaju na neuronskim mrežama), modularno dijeljenje podataka, sigurnost i kriptiranje, te alate za razvoj složene programske podrške. Projekt je zaključen u listopadu 1997 [http://www.ai.mit.edu/projects/iiip/home-page.html].

Na kraju se može spomenuti još i projekt SII (Science Information Infrastructure) [http://cse.ssl.berkeley.edu/sii/sii_sii.html] koga je pokrenula NASA u svrhu povezivanja znanstvenih i odgojnih institucija te istraživačkih centara, kako bi se ostvarila kvalitetnija svemirska istraživanja, te kako bi se osigurao kvalitetniji način pristupa rezultatima tih istraživanja.

Na osnovu ovih razmatranja može se zaključiti da su dosadašnja istraživanja vezana za informacijske infrastrukture rezultirala razvojem mnogih novih tehnologija, no isto tako da je to područje izuzetno široko i da se svi segmenti jednostavno nisu stigli obraditi, kao što je slučaj i sa samom problematikom kojom se bavi ovaj magistarski rad. Kroz spomenute projekte pokazana je kako važnost osiguravanja univerzalnog pristupa raznorodnim podacima unutar informacijske infrastrukture, tako i važnost same informacijske infrastrukture za razvoj društva u cjelini.

3. MODEL PRISTUPA RAZNORODNIM PODACIMA

Kroz ovo poglavlje, razrađuje se sama problematika pristupa podacima raznorodnog tipa, formata i semantike. Dolazi se do osnovne troredne arhitekture modela, koja se dalje razrađuje u okvirima standardnih arhitektura "programskih sabirnica" COM i CORBA. Osnovna zadaća "programskih sabirnica" je uskladiti komunikaciju i suradnju samih aplikacijskih procesa dok se uzima da je sam prijenos podataka od-kraja-do-kraja već riješen.

Prije svega daje se kratak uvod u problematiku pristupa raznorodnim podacima unutar informacijske infrastrukture, te se dolazi do osnovnog trorednog prikaza modela odnosno podjele na korisnički red (interakcija s korisnikom), posrednički red (pristup podacima) te na same podatke raznorodnog tipa, formata i semantike.

Nakon toga, analizira se gdje je u okviru informacijske infrastrukture najpovoljnije implementirati posrednički red (korisnički red nalazi se kod svakog zasebnog korisnika dok se sami podaci raznorodnog tipa formata i semantike mogu nalaziti bilo gdje u okviru informacijske infrastrukture). Zaključuje se da troredni model klijent-poslužitelj najbolje odgovara kao osnovna arhitektura modela, te se dolazi do okvirnih informacija gdje i kako koji dio modela treba dalje razrađivati. Na kraju se daje i kratka usporedba klasičnog dvorednog modela klijent-poslužitelj sa debelim klijentom i dvorednog modela klijent-poslužitelj sa debelim poslužiteljem, s modernijom trorednom arhitekturom. Također, uspoređuju se i klasične izvedbe odnosno namijene troredne arhitekture s trorednom arhitekturom modela za pristup raznorodnim podacima predloženog kroz ovaj rad.

Zatim se u više detalja analizira sam korisnički red modela te njegova interakcija sa posredničkim redom. Dolazi se do tri moguće implementacije korisničkog reda: prva je da se aplikacija izvodi na posredničkom redu dok se korisničkom redu šalje samo izgled korisničkog sučelja, druga je da posrednički red korisničkom redu šalje skup naredbi nekog jezika koje isti onda treba biti u stanju interpretirati, i treći slučaj je kad su korisnički i posrednički red jedna distribuirana aplikacija.

Nakon korisničkog, u više detalja analizira se i sam posrednički red, te se dolazi do detaljnije arhitekture istog koja se sastoji od sučelja prema korisničkom redu, objekata jezgre posredničkog reda, i samih objekata za pristup raznorodnim podacima. Rezimiraju se rezultati provedene analize, te se naposljetku dolazi i do cjelovite proširene arhitekture modela.

Na kraju poglavlja daju se i neke osnovne informacije o arhitekturi CORBA, te se pokazuje kako se model može proširiti kako bi se njime moglo pristupati i CORBA objektima. Time se ujedno ilustrira i otvorenost modela, odnosno osnovni princip kako proširiti njegovu funkcionalnost u svrhu pristupanja i svim onim raznorodnim podacima odnosno arhitekturama, koji prvotno nisu bili podržani odnosno predviđeni.

3.1. Problem pristupa raznorodnim podacima

Prvo je potrebno doći do toga što se već nalazi u okviru informacijske infrastrukture, zatim okvirno odrediti što je potrebno napraviti kako bi se osigurao uniformni pristup kolekcijama raznorodnih podataka uzevši u obzir koja parcijalna rješenja već postoje, i na kraju po mogućnosti izvršiti i neku podjelu na pod-dijelove istog s obzirom na prepoznatu funkcionalnost.

Podaci raznorodnog tipa, formata i semantike se mogu nalaziti bilo gdje u okviru informacijske infrastrukture. Njihovo glavno svojstvo jest da se vremenom mijenja kako količina podataka, tako i svi mogući tipovi podataka, formati podataka i semantike pristupa samim podacima. To među ostalim znači da ostatak sustava mora biti prilagodljiv stalnim promjenama.

Prvu uslugu koju mora pružiti model pristupa takvim podacima jest integriranje tih raznorodnih kolekcija podataka. To znači da taj dio modela mora znati fizički pristupiti bilo kojem podatku bez obzira na to gdje se isti nalazi i na koji način mu se treba pristupiti. Nakon što se pristupi samim podacima, potrebno je eventualno napraviti neku obradu te ih na dogovoren način proslijediti dalje. Stoga se ovaj dio modela naziva *posrednički red*, budući da služi kao posrednik kojeg koristimo prilikom pristupanja raznorodnim podacima.

Nakon što riješimo pristup svim podacima u okviru informacijske infrastrukture, i nakon što imamo jedinstven način njihovog zapisa, potrebno je te podatke prikazati u grafičkom višemedijskom obliku krajnjem korisniku. To je ujedno zadaća sljedećeg dijela modela. Budući da je krajnji korisnik onaj tko koristi ovaj red preko grafičkog sučelja, ovaj dio modela naziva se *korisnički red*. To je ujedno i posljednji red, budući da se iznad njega nalazi sam korisnik informacijske infrastrukture. Kroz daljnju razradu modela potrebno je ispitati prednosti i nedostatke nekih standardnih rješenja za korisničko sučelja kao što je pretraživač Interneta, sa nestandardnim koja će ponekad biti potrebno implementirati ukoliko će neki korisnici imati neke specifične zahtjeve.

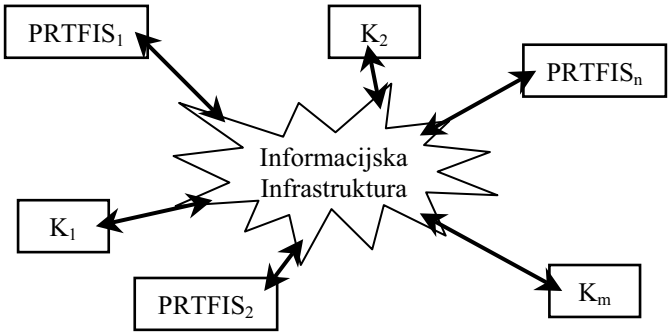
Ovom kratkom analizom dolazimo do trorednog prikaza modela kako je prikazano u Tablici 1. Ono što je potrebno napraviti u ovom radu jest postaviti i razraditi model korisničkog i podatkovnog reda. Prije svega, to znači odlučiti se gdje će isti u okviru informacijske infrastrukture biti implementirani i na koji način odnosno koristeći koje tehnologije, te provjeriti ispravnost samog pristupa i ispitati njegovu svojstvenu funkcionalnost.

Tablica 1: Troredni prikaz modela.

KORISNIČKI RED (KR) (osigurava pristup kroz grafičko višemedijsko sučelje)	– postaviti i razraditi model – provjeriti njegovu ispravnost i ispitati svojstvenu funkcionalnost
POSREDNIČKI RED (PR) (integrira raznorodne kolekcije raspodijeljenih podataka)	
PODACI RAZNORODNOG TIPA, FORMATA I SEMANTIKE (PRTFIS)	– postoje

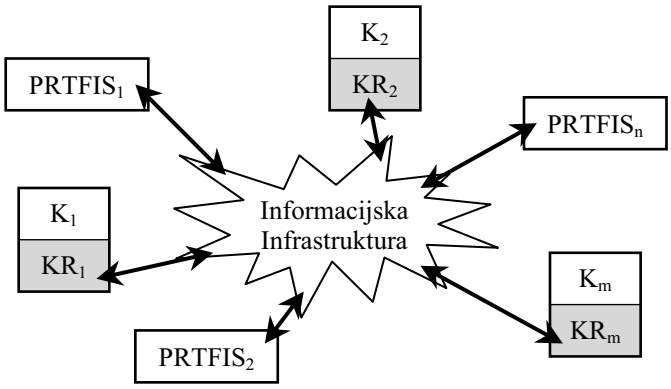
3.2. Osnovna arhitektura modela

Nakon što smo odredili okvirni troredni prikaz modela za uniformni pristup podacima, potrebno je razraditi i njegovu osnovnu arhitekturu. Među ostalim, to znači da moramo odrediti gdje će se i na koji način implementirati korisnički red (KR) i posrednički red (PR). Iako je troredna arhitektura odnosno arhitektura tankog klijenta već poznata, ista se prvenstveno koristila u svrhu jeftinijeg i bržeg razvoja i održavanja programske podrške. Stoga je istu potrebno dodatno razraditi u okvirima konteksta problematike vezane za pristup raznorodnim podacima. Na Slici 2 prikazano je početno stanje informacijske infrastrukture. Imamo korisnike K_1, K_2, …, i K_m; te izvore podataka raznorodnog tipa, formata i semantike $PRTFIS_1$, $PRTFIS_2$, …, i $PRTFIS_n$. Moguće su sve moguće kombinacije kao npr., K_1, K_3, K_7 i K_{14}, znaju pristupati izvoru $PRTFIS_1$; K_2, K_3, K_9 i K_{12}, znaju pristupati izvoru $PRTFIS_2$; izvoru $PRTFIS_7$ znaju pristupati K_1, K_8, K_9, K_{25}, K_{53} i K_{75}. Sasvim općenito, možemo imati *m K*-ova i *n PRTFIS*-ova dok isti mogu biti povezani na bilo koji način.



Slika 2: Početno stanje informacijske infrastrukture.

Budući da je jedina funkcija korisničkog reda implementacija grafičkog-višemedijskog sučelja prema korisniku, logično je da mora biti fizički implementiran kod svakog zasebnog korisnika, odnosno mora biti implementiran za sve moguće tipove operacijskih sustava krajnjih korisnika. Nakon što početnom stanju informacijske infrastrukture dodamo i implementaciju korisničkog reda, dolazimo do sljedećeg stanja u razradi arhitekture kako je prikazano na Slici 3.



Slika 3: Stanje informacijske infrastrukture nakon dodavanja korisničkog reda.

Sljedeće pitanje koje se postavlja jest gdje implementirati posrednički red? Da bi smo na njega odgovorili, pogledati ćemo sve moguće kombinacije gdje ga možemo implementirati, koja sve svojstva ima, i nakon toga koji od načina je najprihvatljiviji.

Posrednički red može biti implementiran kod nijednog korisnika, kod nekoliko njih ili kod svih korisnika. Isto tako može biti implementiran kod nijednog izvora podataka, kod nekoliko njih i kod svih. Također, on može biti implementiran i zasebno. To znači da ukupno imamo 18 mogućih

implementacija posredničkog reda kako je prikazano u Tablici 2. Pošto svaka od njih ima bar jedan nedostatak u odnosu na druge, potrebno ih je međusobno usporediti kako bi se došlo do optimalnog izbora.

Tablica 2: Mogućnosti implementacije korisničkog reda.

redni broj mogućnosti	kod niti jednog KR-a	kod nekoliko KR-ova	kod svih KR-ova	kod niti jednog PRTFIS-a	kod nekoliko PRTFIS-a	kod svih PRTFIS-a	Zasebna imple-mentacija
1.	X			X			
2.	X			X			X
3.	X				X		
4.	X				X		X
5.	X					X	
6.	X					X	X
7.		X		X			
8.		X		X			X
9.		X			X		
10.		X			X		X
11.		X				X	
12.		X				X	X
13.			X	X			
14.			X	X			X
15.			X		X		
16.			X		X		X
17.			X			X	
18.			X			X	X

Budući da će se u informacijsku infrastrukturu često dodavati novi PRTFIS-i, to će značiti kako će se i sam posrednički red morati često mijenjati kako bi mogao pristupati tim novim dotada nepoznatim izvorima podataka. To znači da je prvo svojstvo posredničkog reda da mora biti pogodan izmjenama. Drugim riječima, u slučaju neke izmjene treba se samo zamijeniti neka komponenta ili dodati neka nova. Sljedeće što se može zaključiti o posredničkom redu jest da će biti izuzetno složen i zahtijevan prilikom realizacije. Naime mogućnosti koje imaju današnje baze podataka i sami operacijski sustavi su mnogobrojne, a posrednički red ih većinu mora moći podržati. Ukoliko neka baza korisniku pruža *x* različitih usluga, to znači da bilo koju od tih usluga korisnik može tražiti od posredničkog reda, on dotičnu mora prepoznati i mora znati kako da zahtjev proslijedi bazi. Ukoliko imamo samo jednu komercijalnu bazu, već je teško implementirati posrednički red. Pošto želimo da posrednički red zna pristupati svim komercijalnim sustavima poslovanja baza podataka, jasno je da će time njegova implementacija biti samo još bitno složenija. Primjerice može se navesti da je samo Microsoftov najnoviji standard za univerzalni pristup podacima OLE DB 1.1 [151] (Object Linking and Embedding DataBase Connectivity) dan na preko 700 stranica teksta, a da se dotičnom tehnologijom pokriva samo jedan dio funkcionalnosti

posredničkog reda. Zbog njegove visoke složenosti, možemo dalje zaključiti da će posrednički red biti izuzetno velik potrošač računalnih resursa (potrebna procesorska snaga i količina memorije za izvođenje, potreban prostor na tvrdom disku za instalaciju, i sl.). Sljedeće o čemu moramo voditi računa su licence. Budući da posrednički red među ostalim direktno pristupa i samim bazama podataka, a dotični pristup zahtijeva licencu, to znači i da ćemo za svaki primjerak posredničkog reda morati platiti i odgovarajuću licencu u takvom slučaju. Stoga nije svejedno da li će isti pristupiti nekoj bazi i nakon toga proslijediti informaciju mnogobrojnim korisnicima za što nam treba jedna licenca, ili će svaki korisnik imati svoj posrednički red za što nam treba onoliko licenci koliko imamo korisnika. I na kraju, u koliko želimo posrednički red implementirati za različite operacijske sustave, postavlja se pitanje da li se može implementirati u nekom jeziku koji se može izvoditi na svim tim različitim operacijskim sustavima tako da posrednički red ne moramo pisati više od jednom. Danas najprihvatljiviji jezik za takvu namjenu je *Java*, te se postavlja pitanje da li se taj jezik može koristiti u tu svrhu. Na žalost odgovor je odrečan, jer jezik koji bi bio dovoljno jednostavan da se može izvoditi na svim odnosno većini operacijskih sustava nije dovoljno funkcionalan da zadovolji kao jezik implementacije. Primjerice Java ne može pristupati COM objektima koji nemaju implementirano sučelje *IDispatch* i nemaju pripadnu biblioteku podataka. S obzirom na složenost dotičnog problema i s obzirom na zahtjev za pogodnost stalnim izmjenama, tehnologija izvedbe posredničkog reda treba biti i objektno orijentirana i komponentno temeljena, kako je već objašnjeno u samom uvodu. Nakon što smo razmotrili sva važna svojstva posredničkog reda, možemo dati njihov pregled:

a) PR mora biti izuzetno pogodan za stalne promjene i izmjene;

b) PR je izuzetno složen;

c) PR ima velike zahtjeve na računalne resurse;

d) PR je potrošač licenci;

e) PR je zahtjevan na jezik implementacije koji ne može biti neki univerzalni jezik kao npr. Java;

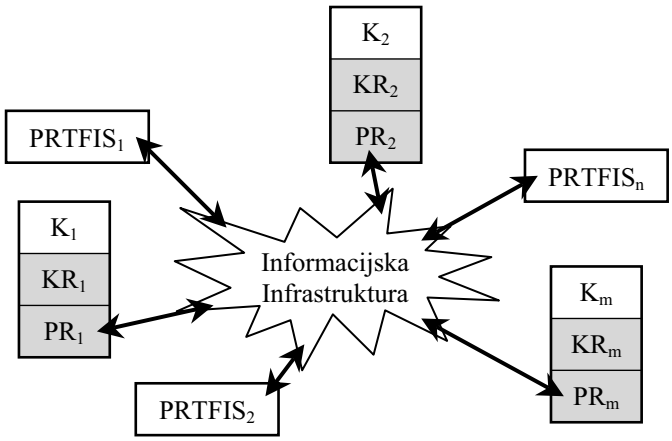
f) PR zahtijeva objektno orijentiranu tehnologiju izvedbe;

g) PR zahtijeva komponentno temeljenu tehnologiju izvedbe.

Ako sada ponovno pogledamo Tablicu 2, jasno se vidi da mnoge kombinacije uopće nemaju smisla. Naime ukoliko pretpostavimo da je funkcionalnost posredničkog reda implementirana kod samo nekih KR-ova, u tom slučaju postoje KR-ovi kod kojih ona nije uopće implementirana, što znači da se ionako u cijelosti posrednički red mora ponovno implementirati ili kod PRTFIS-a ili zasebno. No to znači da je kod onih nekoliko KR-ova nepotrebno implementiran, jer i oni mogu koristiti tu zajedničku implementaciju. To znači da su slučajevi 7 do 12 potpuno neprihvatljivi. Analogno tome možemo zaključiti da nema smisla niti da posrednički red bude implementiran kod samo nekoliko PRTFIS-a, što znači da su slučajevi 3, 4, 9, 10, 15 i 16 zbog istog razloga neprihvatljivi. Na isti način možemo zaključiti da ako posrednički red implementiramo zasebno, nema potrebe da se ponovno još negdje implementira te su zbog toga neprihvatljivi i slučajevi 4, 6, 8, 10, 12, 14, 16 i 18. Ostaju nam slučajevi 1, 2, 5, 13 i 17. Slučaj 1 neprihvatljiv je budući da posrednički red ustvari nije nigdje implementiran, dok je slučaj 17 neprihvatljiv jer je implementiran dva puta što je također neprihvatljivo zbog već spomenutih razloga. Na kraju nam ostaju samo slučajevi 2, 5 i 13 koji će se morati analizirati u malo više detalja.

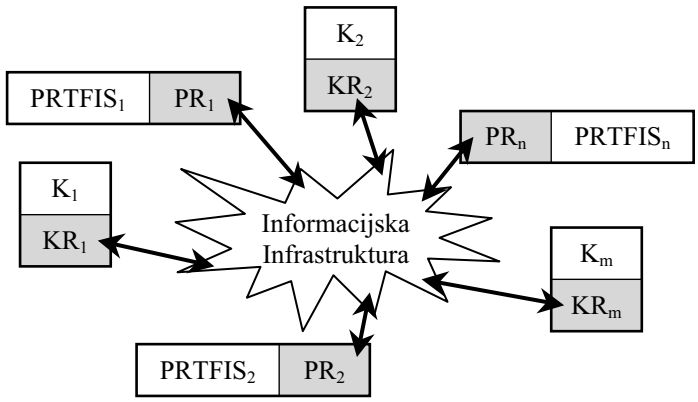
Slika 4 prikazuje slučaj 13 (debeli klijent) u kojem je posrednički red implementiran kod svakog zasebnog korisnika zajedno s korisničkim redom. Budući da korisnički red ionako moramo implementirati zasebno kod svakog korisnika, ovakvo rješenje se intuitivno nameće. No ukoliko se razmotre svojstva posredničkog reda, dolazi se do zaključka da je ovaj slučaj također neprihvatljiv. Zbog velike složenosti posredničkog reda (svojstvo *b*), te zbog toga što ga ne možemo napisati jednom u nekom univerzalnom jeziku i nakon toga prebaciti na sve pripadne operacijske sustave

različitih korisnika (svojstvo *e*), to bi značilo *m* puta implementirati jednu te istu izuzetno složenu funkcionalnost što je neprihvatljivo. Pitanje je i da li bi smo mogli implementirati ovaj red kod svakog zasebnog korisnika s obzirom na njegovu kompleksnost (svojstvo *c*), a javlja se i problem licenci (svojstvo *d*). Stoga nam preostaju još jedino mogućnosti 2 i 5.



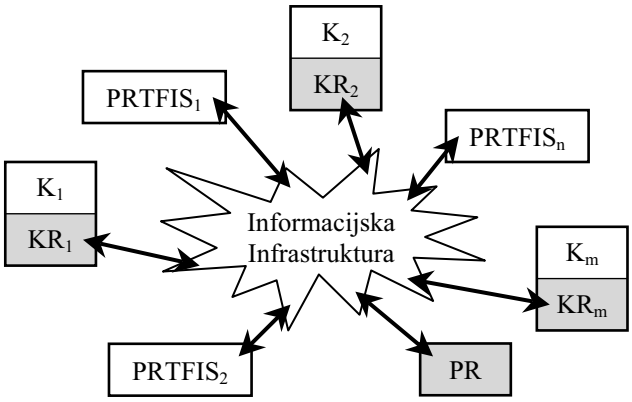
Slika 4: Slučaj izvedbe posredničkog reda kod svakog zasebnog korisnika.

Na Slici 5 prikazan je slučaj 5 (debeli poslužitelj) gdje je posrednički red implementiran kod svih zasebnih PRTFIS-a. Dok bi ovaj pristup (koji je inače i osnova za troredne arhitekture klijent -poslužitelj) bio sasvim prihvatljiv u slučaju da nas zanima razvoj i održavanje programske podrške za neku specifičnu platformu, kod modela za pristup kolekcijama raznorodnih podataka također se javlja problem da bi se sam posrednički red morao implementirati na više različitih platformi pošto se na istim ti podaci već nalaze. Zbog njegove velike kompleksnosti (svojstvo *b*), to je u ovoj primjeni također neprihvatljivo rješenje. Dodatno se javlja kao problem što bi se moglo dogoditi da je nedovoljno otvoren sam sustav gdje se nalazi neki PRTFIS iz sigurnosnih ili nekih drugih razloga, pa da zbog toga ne bi smo bili u stanju implementirati posrednički red u punoj funkcionalnosti. Stoga nam kao jedino rješenje ostaje slučaj 2 gdje je posrednički red implementiran na zasebnoj platformi.



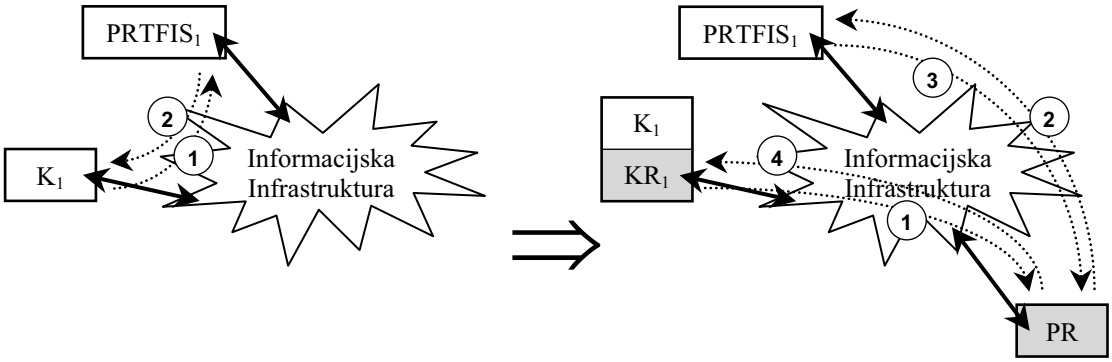
Slika 5: Slučaj izvedbe posredničkog reda kod svakog zasebnog izvora podataka.

Metodom isključivanja neprihvatljivih rješenja, došli smo do zaključka da je slučaj 2 prikazan na Slici 6 optimalan pristup. Posrednički red se u ovom slučaju implementira samo jednom, što uz to da možemo birati operacijski sustav na kojem će biti implementiran (pošto isti više nije vezan ni za operacijski sustav(e) korisnika ni za operacijski sustav(e) izvora podataka) garantira najjednostavniju, a time i najjeftiniju izvedbu.



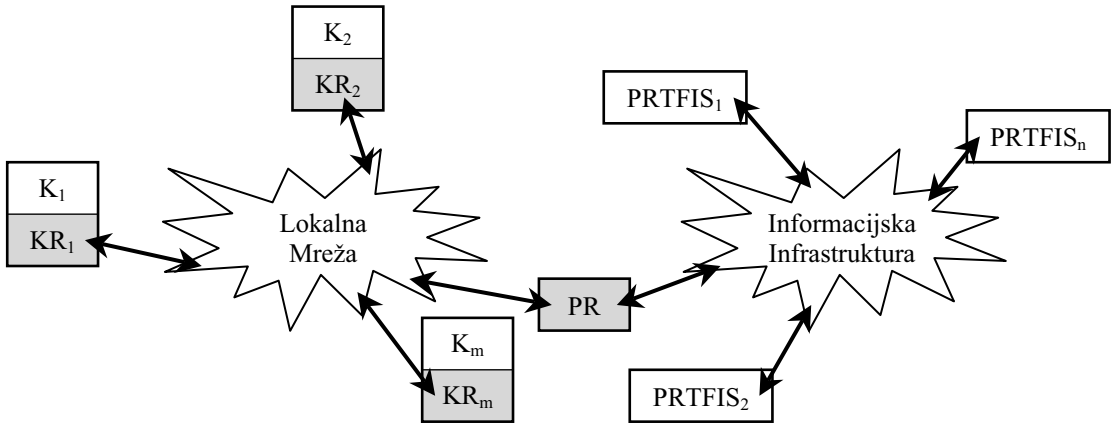
Slika 6: Slučaj izvedbe zasebnog posredničkog reda.

Mogući nedostatak ovog rješenja jest da se može pojaviti veći promet mrežom u odnosu na početno stanje kako se vidi iz Slike 7. Na Slici 7 (lijevo) imamo slučaj prije uvođenja modela gdje korisnik direktno dolazi do svojih podataka (ali zato npr. ne zna pristupiti ničemu drugom). Jedno čitanje sastoji se od zahtjeva (1), i dostave podataka (2). Na Slici 7 (desno) imamo slučaj nakon uvođenja modela gdje korisnik indirektno dolazi do svojih podataka preko posredničkog reda (no zato može pristupati i bilo čemu drugom). Ovdje se jedno čitanje sastoji od slanja zahtjeva PR-u (1), nakon toga PR prosljeđuje zahtjev PRTFIS-u (2), zatim PRTFIS šalje podatke PR-u (3) i na kraju PR prosljeđuje te podatke KR-u odnosno korisniku (4). Jasno je da se kao posljedica može javiti veći promet računalnom mrežom te da stoga možemo imati sporiji odziv.

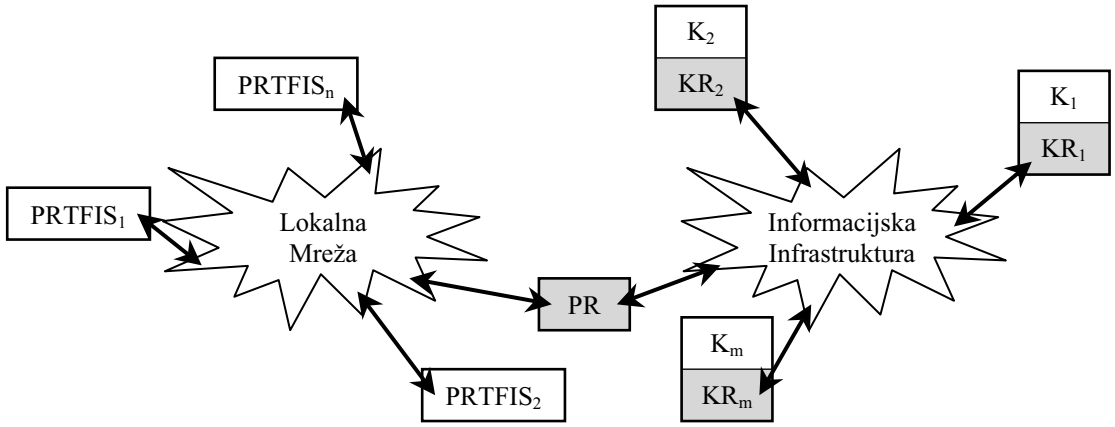


Slika 7: Može se kao nedostatak pojaviti veći promet mrežom.

Nedostatak se može efikasno otkloniti izborom lokacije računala na kojem je implementiran posrednički red kako se vidi na sljedeće dvije slike. Slika 8 prikazuje slučaj kada se implementira jedan PR za grupu korisnika, dok Slika 9 prikazuje slučaj kada se implementira PR za grupu izvora podataka. Ideja je da se posrednički red nalazi na prirodnom putu podataka, te se na taj način izbjegne kašnjenje i rušenje performansi same informacijske infrastrukture. Iz ovoga slijedi da se može javiti situacija da u okviru informacijske infrastrukture imamo posrednički red implementiran na nekoliko različitih računala radi poboljšanja performansa. Ovaj slučaj treba razlikovati u odnosu na slučajeve prikazane na slikama 4 i 5. Naime, ovdje se PR implementira uvijek na jednom te istom operacijskom sustavu, i to onom kojeg smo sami izabrali. Ne može se pojaviti niti problem udvostručivanja podataka ili neki slični problem iako sada korisnik na raspolaganju može imati i više od jednog PR-a. Izbor PR-a se ne smije mijenjati za vrijeme neke transakcije što jednostavno znači da će se prilikom bilo koje transakcije vidjeti samo jedan posrednički red. U slučaju da dotično računalo padne ili transakcija ne uspije zbog nekog drugog razloga, transakcija se poništava i ponavlja iz početka preko tog istog ili nekog drugog posredničkog reda. Ovim pristupom može se riješiti i dodatni problem koji se može pojaviti, a to je opterećenje računala-posrednika.

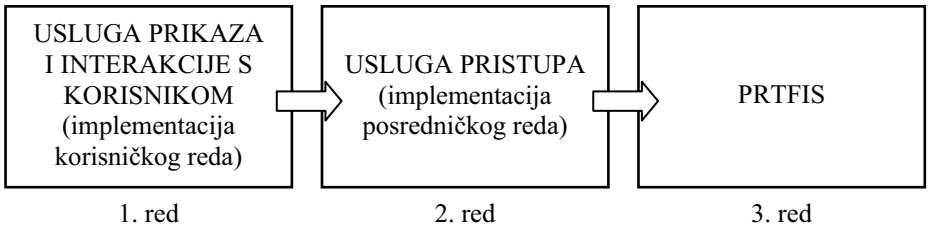


Slika 8: Slučaj kada se implementira jedan PR za grupu korisnika.



Slika 9: Slučaj kada se implementira PR za grupu izvora podataka.

Nakon svih ovih razmatranja, dolazimo do troredne arhitekture modela kako je prikazano na Slici 10. Treći red predstavlja podatke raznorodnog tipa, formata i semantike (PRTFIS) koje možemo naći u nekoj informacijskoj infrastrukturi. Posrednički red osigurava nam uslugu pristupa bilo kojem podatku koji se nalazi u okviru informacijske infrastrukture. Korisnički red osigurava nam uslugu višemedijskog prikaza tih podataka te uslugu interakcije s korisnikom.

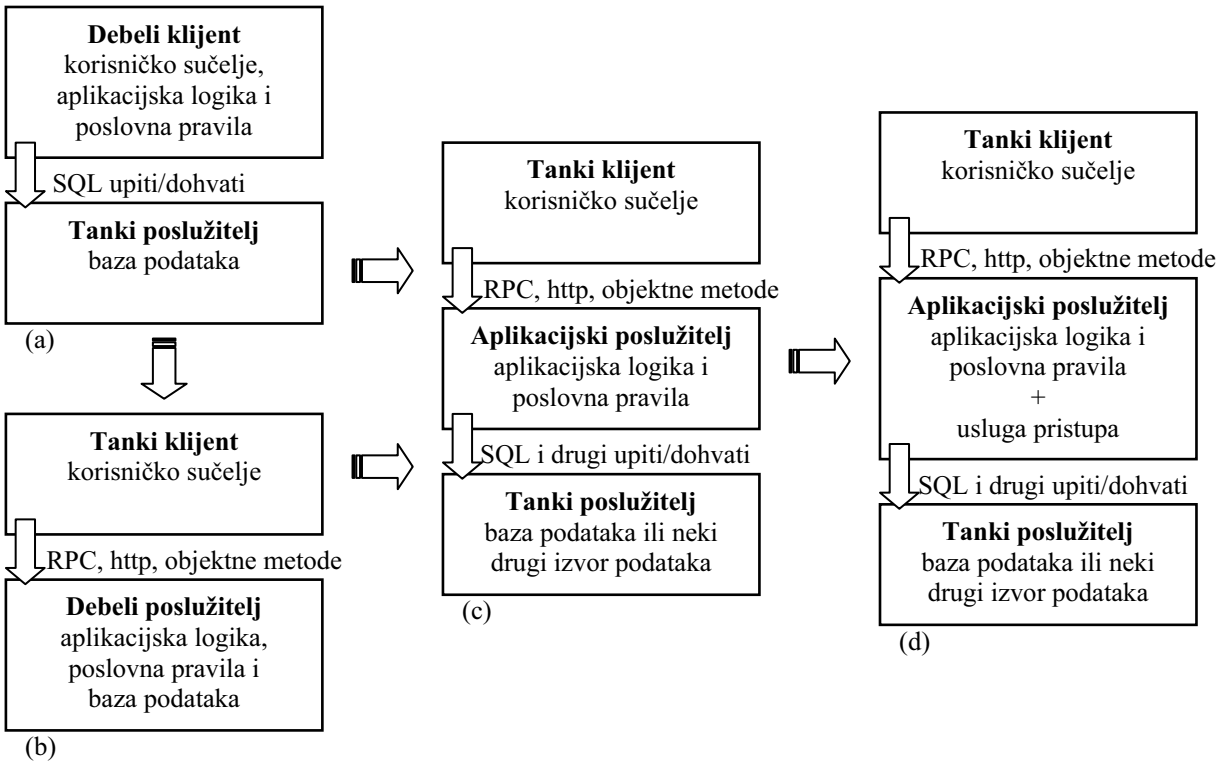


Slika 10: Troredna arhitektura modela.

Budući da možemo birati operacijski sustav i alat na kojem ćemo implementirati posrednički red, izabrati ćemo nešto što podržava i objektno orijentiranu i komponentno temeljenu paradigmu. Na taj način ćemo moći zadovoljiti zahtjeve *f* i *g*. Budući posrednički red ne moramo implementirati na više od jednog operacijskom sustavu, više nemamo potrebu za nekim univerzalnim jezikom tako da je riješen i zahtjev *e*. No to ujedno i ne znači da neke dijelove možda nećemo napisati i u *Javi*, naime komponentno temeljena paradigma osigurava da komponente možemo pisati u kojem god jeziku želimo, to može biti *C++*, *Java*, *Visual Basic* ili neki drugi. Posrednički red se implementira minimalan broj puta tako da se efikasno može riješiti i problem licenci odnosno zahtjev *d*. Što se tiče zahtijeva *c* na značajne računalne resurse, ono što će nam trebati, to ćemo ugraditi na poslužitelja na kojem će se nalaziti posrednički red, tako da je na taj način i to riješeno. Primjenom

komponentno temeljene tehnologije, mogu se vrlo efikasno izgrađivati i vrlo kompleksne aplikacije koje su zahvaljujući primjeni ove tehnologije unatoč svoje veličine ipak izuzetno pogodne stalnim promjenama i izmjenama čime možemo zaključiti da su zadovoljeni i zahtjevi *a* i *b*. Stoga možemo zaključiti da se je ovakvim pristupom zagarantirala optimalna arhitektura modela koja zadovoljava svim postavljenim zahtjevima.

Na Slici 11 (lijevo), prikazana je klasična dvoredna arhitektura debelog klijenta (a), klasična dvoredna arhitektura debelog poslužitelja (b) i klasična troredna arhitektura (c). Arhitektura debelog klijenta, tradicionalno još nazvana i klijent-poslužitelj arhitektura, najjednostavnija je za implementaciju i jedna od prvih koja se koristila, no postupno je bivala zamjenjivana arhitekturom debelog poslužitelja zbog sljedećih nedostataka: održavanje je teško pošto su aplikacijska logika i poslovna pravila implementirana zajedno sa korisničkim sučeljem; model je teško proširivati u slučaju potrebe za boljim performansama sustava; različite verzije debelog klijenta su potrebne za svaku različitu ciljnu platformu odnosno operacijski sustav; javljaju se problemi prilikom distribucije novijih verzija od kojih su najvažniji fizički sama distribucija te mogućnost da različiti klijenti koriste različite verzije; pristup podacima koji se ne nalaze unutar neke baze podataka nije podržan na standardan način. Dok se dobar dio ovih problema riješio se prelaskom na dvorednu arhitekturu debelog poslužitelja (primjerice kao tanki klijent se može koristiti pretraživač Interneta kojeg ne moramo sami implementirati i već podržava gotovo sve ciljne operacijske sustave), pojavili su se neki novi od kojih je jedan od najznačajnijih vezanost aplikacijske logike i poslovnih pravila sa samim sustavom poslovanja baza podataka. Ta vezanost ne samo da je prelazak sa jednog sustava poslovanja baza podataka na drugi učinila bitno složenijim, već su primjerice i same promjene sa jedne verzije na noviju ili promjene u skladu sa potrebom proširivanja performansi sustava, postajale previše složene i zahtjevne. Vremenom se uspostavilo da troredna arhitektura najbolje rješava nabrojane probleme. Njen nedostatak je u tome što je nešto složenija od dvoredne arhitekture debelog klijenta i debelog poslužitelja, no praksa je pokazala da uštede u održavanju odnosno prilikom proširivanja višestruko nadmašuju nešto veća ulaganja prilikom samog razvoja.



Slika 11: Migracija dvorednih arhitektura debelog klijenta (a) i debelog poslužitelja (b) u klasičnu troredu arhitekturu (c), te proširivanje iste u troredu arhitekturu prilagođenu pristupanju raznorodnim podacima untar informacijske infrastrukture (d).

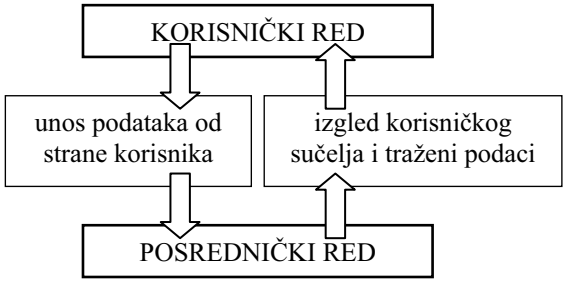
Nakon što je ustanovljeno da troredna arhitektura modela najbolje odgovara kao arhitektura modela za pristup raznorodnim podacima unutar informacijske infrastrukture, te da se troredna arhitektura i inače sve više koristi u odnosu na dvorednu te istu i zamjenjuje, potrebno je još pronaći i korelaciju između te *klasične* troredne arhitekture i troredne arhitekture modela za pristup podacima predložene kroz ovaj rad. Kako je prikazano na Slici 11 (desno), arhitektura modela za pristup podacima može se promatrati kao proširenje klasične troredne arhitekture *uslugom pristupa* odnosno implementacijom posredničkog reda. Daljnja analiza te same usluge pristupa te koje tehnologije i kako ih koristiti kako bi se ista implementirala je upravo sama jezgra problematike koja se dalje obrađuje kroz ovaj rad. No ista će se morati i dalje specijalizirati, odnosno ograničiti na primjerice samo sustave poslovanja relacijskim bazama podataka, kako bi se s jedne strane u tom znanstvenom području moglo otići dovoljno daleko gdje još uvijek ima materijala za znanstveno istraživanje, a s druge gdje će sam prototip biti dovoljno jednostavan kako bi se mogao i implementirati te isprobati u praksi. No prilikom tog procesa uvijek treba imati na umu kako primjenjivost na druge izvora podataka, tako i na samu aplikacijsku logiku te poslovna pravila koja se također implementiraju u sklopu Aplikacijskog poslužitelja.

Nakon što smo odredili osnovnu arhitekturu modela, kroz sljedeća dva poglavlja će se u više detalja analizirati kako izgraditi sam korisnički red i posrednički red. Nakon toga će se dati i sam detaljniji prikaz ove troredne arhitekture, s tim da će se posebna pažnja posvetiti samoj usluzi pristupa odnosno implementaciji posredničkog reda.

3.3. Korisnički red pristupa raznorodnim podacima

Budući da korisnički red moramo implementirati za svaku zasebnu platformu, a posrednički red implementiramo za samo jednu platformu, i to bilo koju po našem vlastitom odabiru, logično je da se u okviru posredničkog reda implementira sve što se u okviru njega može implementirati, dok se u okviru korisničkog reda treba implementirati samo ono što se mora implementirati kod svakog zasebnog korisnika, a to upravo i jest samo grafičko višemedijsko korisničko sučelje.

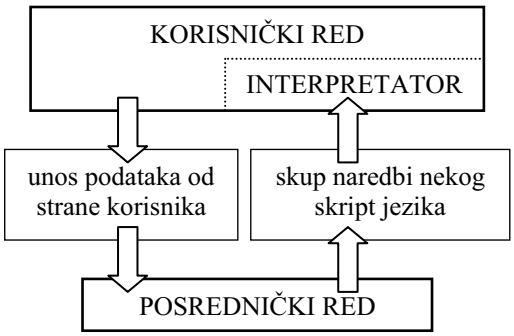
Ovim razmatranjem dolazimo i do prve moguće implementacije korisničkog reda kako je prikazano na Slici 12. Sama aplikacija se u potpunosti izvodi na posredničkom redu, dok se računalnom mrežom korisničkom redu šalje samo izgled korisničkog sučelja kojeg korisnički red prikazuje na zaslonu računala krajnjeg korisnika. Korisnički red prati unos s tipkovnice, miša, te prijenos podatka s jedinica vanjske memorije korisničkog računala tipa 3.5″ disketa, pogona optičkog diska i slično. Te podatke zatim šalje računalnom mrežom nazad posredničkom redu, a on ih interpretira kao da su isti uneseni na računalu na kojem je implementiran i na kojem se i izvodi sam posrednički red.



Slika 12: Aplikacija se izvodi na PR-u dok se KR-u šalje samo izgled korisničkog sučelja.

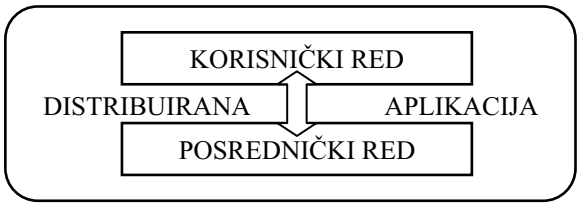
Sljedeće o čemu se može razmišljati jest kako dodatno optimirati predloženu arhitekturu odnosno promet mrežom. Komunikacija od korisničkog reda ka posredničkom redu ionako sadrži samo podatke koje je potrebno prenijeti tako da taj segment već je optimiran u neku ruku. Situacija je nešto drugačija ukoliko promatramo komunikaciju od posredničkog reda ka korisničkom redu. Iako se i sam izgled korisničkog sučelja može komprimirati nekim od postupaka kompresije za nepokretne slike (npr. JPEG) ili još bolje za pokretne slike (npr. M-JPEG), to kao i u prethodnom slučaju ne bi značilo neku drugu arhitekturu pošto se kompresija u stvarnom vremenu u principu uvijek može dodati kad se nešto šalje računalnom mrežom. No u ovom smjeru komunikacije, može se učiniti nešto više od same kompresije. Naime ideja je da posrednički red šalje korisničkom redu samo skup naredbi nekog (skriptnog) jezika koje će isti interpretirati i prikazati na željeni način. Ideja je da bi se na taj način moglo postići više nego primjenom same kompresije. Nedostatak u odnosu na prethodni slučaj jest u tome što korisnički red treba imati i interpretator tih naredbi kako je prikazano na Slici 13. Da dotični interpretator ne bi morali pisati onoliko puta za koliko različitih platformi želimo izgraditi korisnički red, za ovu svrhu može se koristiti bilo koji standardni pretraživač Interneta koji može u potpunosti predstavljati implementaciju korisničkog reda. Jezik u kojem se šalju naredbe može biti HTML (HyperText Markup Language) ili još bolje DHTML (Dynamic HTML), dok sam HTML dokument može sadržavati i mnogobrojne skripte. Na taj način može se prenijeti gotovo sve, od PDF (Portable Document Format) i Postscript dokumenata do pomične slike, zvuka odnosno višemedijskih dokumenata. Drugim riječima imamo prijenos i prikaz gotovo svih raznorodnih podataka, a ne moramo interpretirati korisnički red već možemo koristiti nešto što već postoji. Ukoliko standardni pretraživači Interneta u nekom segmentu ne pokažu dovoljnu funkcionalnost, ista se može dodati kroz razno-razna proširenja koja su za to namijenjena, a prije svega su to mnogobrojni danas dostupni skript jezici [11][25][69][73][83]. Kao što se iz prikazanog može zaključiti, posebna prednost ovog pristupa nije samo moguća ušteda u manjem

prometu mrežom, već upravo to što smo u potpunosti oslobođeni implementacije korisničkog reda pošto u gotovo svim slučajevima možemo koristiti standardni pretraživač Interneta koji je već napisan za skoro sve poznate platforme. Sve što je potrebno, jest dizajnirati Internet stranice na strani aplikacijskog poslužitelja. Čak i ukoliko neka funkcionalnost nije podržana standardnim pretraživačima, danas se ovo područje tako brzo razvija da će ista već vrlo skoro gotovo sigurno biti podržana.



Slika 13: PR šalje KR-u skup naredbi nekog jezika koje KR treba biti u stanju interpretirati.

Postoji još jedna moguća arhitektura koja se treba uzeti u obzir, a to je slučaj kada se i korisnički red i posrednički red nalaze na istim operacijskim sustavima i kada dotični operacijski sustav podržava razvoj distribuiranih aplikacija. U tom slučaju ta dva reda mogu biti implementirani kao jedna distribuirana aplikacija, čija logika se izvodi na jednom računalu dok se grafičko sučelje prikazuje na nekom drugom računalu. Osnovna prednost ovog pristupa u odnosu na prethodna dva su daleko bolje performanse u bilo kojem pogledu te jednostavniji razvoj. No ovo je samo jedan poseban slučaj koji se može razmatrati samo kada se dogodi da se korisnik nalazi na istom operacijskom sustavu kao što je i onaj na kojem smo se odlučili za implementaciju posredničkog reda, tako da se zbog svih ostalih korisnika moraju promatrati i prethodna dva slučaja. Ovo se može posebno pokazati zanimljivim kada se u informacijsku infrastrukturu dodaju novi korisnici radi optimiranja i samim tim jeftinije izvedbe. Pri tom treba uzeti u obzir da se pod pojmom "distribuirane aplikacije" ne misli na debelog klijenta odnosno na klasičnu dvorednu aplikaciju klijent-poslužitelj, pošto se posrednički red implementira zasebno odnosno nezavisno od izvora podataka te se istom može pristupati i na načine prikazane na slikama 12 i 13.

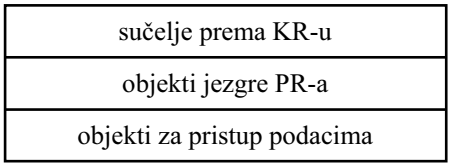


Slika 14: KR i PR su jedna distribuirana aplikacija.

Pitanje koje se postavlja jest koji od tri moguće implementacije korisničkog reda, kako je prikazano na slikama 12, 13 i 14, bi trebao podržati i sam posrednički red. U slučaju da se odlučimo za implementaciju posredničkog reda na Windows NT platformi, distribuirana aplikacija temeljena na Distributed COM tehnologiji najbolje je rješenje za korisnike koji koriste 32-bitne Windowse. Za ostale korisnike treba implementirati i bar jedno od prethodna dva rješenja.

3.4. Posrednički red pristupa raznorodnim podacima

Sam posrednički red sastoji se od tri podreda kako je prikazano na Slici 15. Sučelje prema korisničkom redu predstavlja skup lokalnih poslužitelja koji korisničkom redu osiguravaju pristup objektima jezgre posredničkog reda. Objekti za pristup podacima predstavljaju skup objekata koji znaju pristupiti podacima raznorodnog tipa, formata i semantike, dok objekti jezgre posredničkog reda rukuju kako tim podacima tako i samim objektima za pristup podacima. Oni moraju voditi računa o sigurnosnim protokolima odnosno o tome tko smije pristupati kojim podacima, dodatno moraju imati informaciju o tome koji objekt iz skupa objekata za pristup podacima treba pozvati da bi se pristupilo nekom podatku te kako ga treba inicijalizirati i pozvati. Također, objekti jezgre moraju ispravno interpretirati te podatke i proslijediti ih dalje, te po potrebi učiniti i potrebnu konverziju istih. Pri tome se pod pojmom objekta ne podrazumijevaju samo komponente, već i ostali objekti koji po potrebi mogu biti korišteni kao što je primjerice sustav za rukovanje komponentama ili transakcijama. S druge strane jasno je i da će pojedine grupe objekata odnosno komponenata imati i vrlo sličnu strukturu, tako da će uz komponentno temeljen i sam objektno orijentiran pristup imati svoju značajnu primjenu prilikom izgradnje posredničkog reda.



Slika 15: Tri podreda posredničkog reda.

Sljedeće što je potrebno jest razraditi samu arhitekturu sučelja prema korisničkom redu. U slučaju kad se korisniku šalje samo izgled korisničkog sučelja kako je prikazano na Slici 12, to znači implementaciju lokalnog poslužitelja koji će korisničkom redu slati izgled korisničkog sučelja i ostale višemedijske podatke, a od istog primati podatke od strane korisnika. Pri tome se pod pojmom lokalnog poslužitelja podrazumijeva poslužitelj koji će se nalaziti na istom računalu na kojem i sam posrednički red. U slučaju kad se korisniku šalje skup naredbi nekog skriptnog jezika kako je prikazano na Slici 13, to znači implementaciju lokalnog poslužitelja koji će korisničkom redu slati naredbe koje opisuju izgled korisničkog sučelja te putem njih i ostale višemedijske podatke, a od istog primati primljene podatke od strane korisnika kao i u prethodnom slučaju. U praksi to će najčešće značiti implementaciju nekog HTTP poslužitelja koji će znati generirati HTML odnosno DHTML. U slučaju razvoja distribuirane aplikacije kako je prikazano na Slici 14, sučelje prema korisničkom redu je prazno što znači da sama aplikacija direktno pristupa samim objektima jezgre PR-a. Nakon što uzmemo ovo u razmatranje, dolazimo do detaljnije arhitekture sučelja prema korisničkom redu kako je to prikazano na Slici 16 (gore).

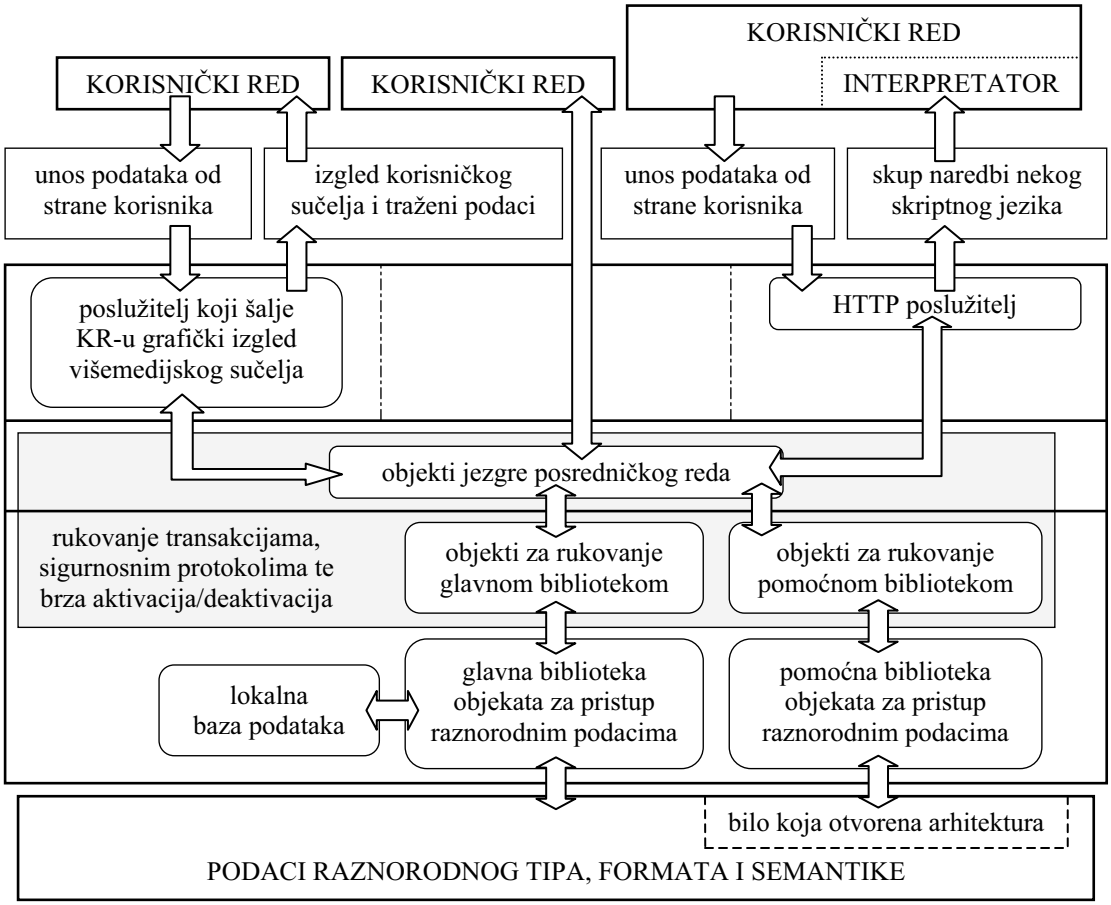
Sami objekti za pristup podacima predstavljaju u principu najveći dio posredničkog reda. Naime upravo oni u sebi sadržavaju svu potrebnu logiku koja mora znati pristupiti svemu što se nalazi unutar neke informacijske infrastrukture. Stoga ima smisla ovaj podred dalje razložiti na (glavnu) biblioteku objekata za pristup raznorodnim izvorima podataka te na objekte koji olakšavaju rukovanje tom bibliotekom, kako je također prikazano na Slici 16 (dolje), i na pomoćne biblioteke objekata i pripadnu podršku.

Objekti jezgre i objekti za rukovanje (glavnom) bibliotekom sučelja za raznorodne izvore podataka se tijekom rada posredničkog reda konstantno aktiviraju i deaktiviraju, a potrebno je zadovoljiti i neke druge zahtjeve kao što je npr. rukovanje transakcijama te sigurnosne protokole. Stoga ima smisla implementirati lokalnog poslužitelja za dotične objekte čime ćemo bitno

pojednostavniti njihovu izgradnju te osigurati efikasniji način njihovog izvođenja u realnom vremenu.

Moguća primjedba na trorednu arhitekturu prikazanu na Slici 16 u odnosu na dvodijelnu arhitekturu klijent-poslužitelj jest što nam za pohranu i pristup podacima trebaju uz korisnike još bar dva računala, ukoliko posrednički sloj nije razvijen za istu platformu za koju već imamo sustav poslovanja bazama podataka, što kod malih sustava može predstavljati financijsku neprihvatljivost. Dotični problem vrlo efikasno se može riješiti na taj način da se sklopu samog posredničkog reda doda i lokalna baza podataka koja je napravljena za isti operacijski sustav na kojem je i razvijen sam posrednički red.

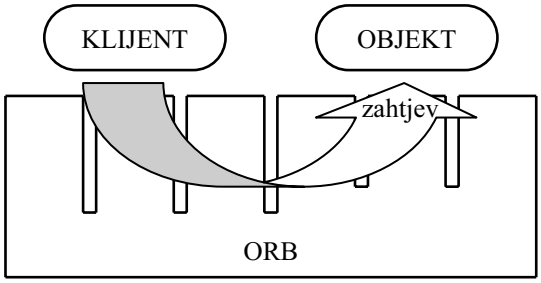
Na Slici 16 dodatno je pokazan i osnovni princip proširivosti modela. Većina komercijalnih sustava poslovanja relacijskih baza podataka podržava pristup prema standardu *X/Open DTP XA*, dok Microsoftovi proizvodi podržavaju pristup i prema specifikaciji *OLE Transaction*. Glavna biblioteka objekata za pristup raznorodnim izvorima podataka treba pokrivati ova dva protokola te time i većinu sustava poslovanja bazama podataka danas dostupnih. Ukoliko ipak postoji neki specifični izvor podataka, kojem se ne može pristupiti na prethodno opisani način, uvijek se može proširiti model tako da se izgradi pomoćna biblioteka objekata za tu klasu raznorodnih izvora podataka, a može se izgraditi i više pomoćnih biblioteka gdje svaka pokriva svoju klasu izvora podataka.



Slika 16: Detaljna arhitektura posredničkog reda s tri karakteristična podreda.

3.5. Model i CORBA

Standardna arhitektura posrednika zahtjevima objekata CORBA [155] je strukturirana na taj način da se omogući integriranje vrlo širokog kruga raznorodnih objekata pisanih u raznoraznim kako objektno orijentiranim, tako i onim samo objektno odnosno komponentno temeljenim jezicima [55]. Tu se posebno misli na objekte koji egzistiraju i trebaju imati sposobnost zajedničkog rada u okviru neke računalne mreže [91]. Želja odnosno zamisao je da se objekti ne pozivaju direktno, već da se zahtjevi za njihovim uslugama (znači obradom) šalju koristeći posrednik zahtjevima objekata ORB (Object Request Broker), kako je prikazano na Slici 17. U ovom slučaju *klijent* je objekt koji traži neku obradu dok je *objekt* poslužitelj koji je izvršava.



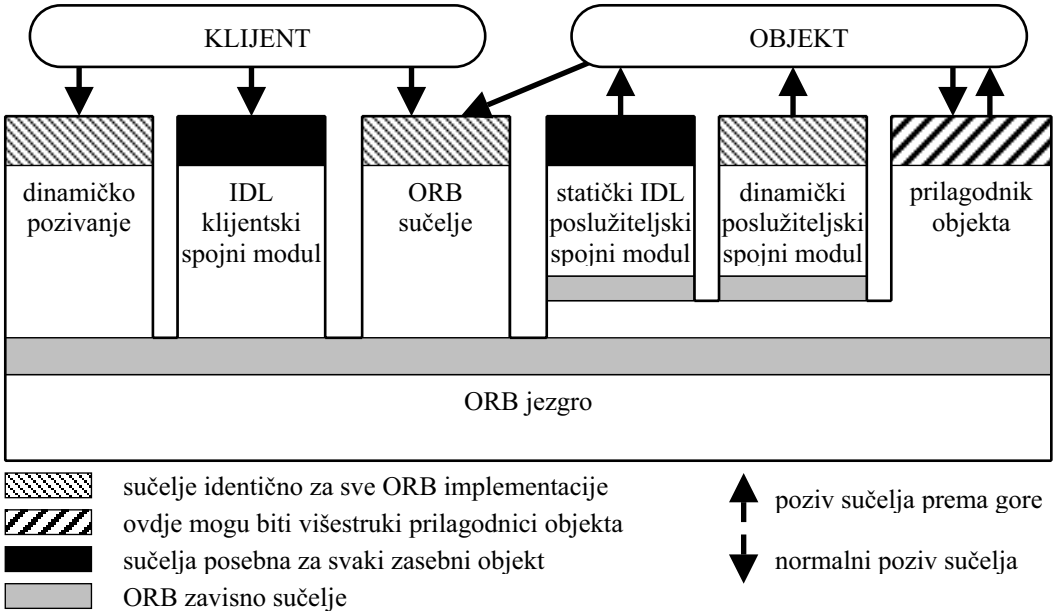
Slika 17: CORBA - klijent šalje zahtjev objektu preko ORB-a.

ORB je odgovoran za sve mehanizme koji su potrebni da se pronađe implementacija objekta koji će primiti zahtjev, pripremi implementacija objekta za prihvat zahtjeva, te proslijede podaci koji su sastavni dio samog zahtijeva. Sučelje koje klijent vidi je u potpunosti nezavisno o tome gdje se fizički nalazi sama implementacija objekta, u kojem jeziku je ista napisana, ili o bilo kojem drugom aspektu koji nije povezan sa samim sučeljem objekta.

Na Slici 18 imamo detaljniji prikaz arhitekture CORBA. Sama pojedinačna sučelja prema ORB-u prikazana su odgovarajući popunjenim pravokutnicima, dok same strelice označavaju da li imamo slučaj normalnog poziva sučelja ili poziv u suprotnom smjeru (prema gore).

Radi prosljeđivanja zahtjeva, klijent može koristiti sučelje za dinamičko povezivanje koje je identično za sve ORB implementacije, ili IDL klijentski spojni modul napravljen posebno za tu svrhu. Klijent također može i direktno pozivati funkcije ORB-a kroz ORB sučelje. Slična je situacija i s implementacijom objekta. On zahtjeve može dobivati kako preko dinamičkog poslužiteljskog spojnog modula koji je isti za sve ORB implementacije, ili preko statičkog IDL poslužiteljskog spojnog modula napravljenog specijalno za tu svrhu. Implementacija objekta također može komunicirati i sa svojim prilagodnikom i direktno sa ORB-om kroz ORB sučelje. Također, tu funkcionalnost može koristiti kako za vrijeme obrade zahtjeva tako i u intervalima između zahtjeva.

Sama sučelja mogu se opisati na dva načina. Prvi je statički, koristi se jezik za opis sučelja nazvan *OMG IDL*. Jezik definira tipove objekata u odnosu na operacije koje se mogu nad istim izvršit i pripadne parametre tih operacija. Drugi način je da se sučelja dinamički registriraju kod poslužitelja namijenjenog posebno za tu svrhu, nazvanog *Interface Repository*. Taj poslužitelj posreduje prilikom pristupa komponentama koje posjeduju takvo odnosno takva sučelja.



Slika 18: Detaljniji prikaz arhitekture CORBA.

CORBA je uz Microsoftov DCOM danas najzastupljeniji standard za razvoj distribuiranih aplikacija temeljenih na distribuiranim objektima i međusobnom povezivanju istih. Drugim riječima, mnogi sustavi poslovanja bazama podataka i općenito aplikacije već jesu i u bližoj budućnosti biti će napisane upravo prema CORBA standardu odnosno prema tom standardu će im se moći najlakše pristupiti [104]. To znači da mora biti moguće proširiti posrednički red tako, da zna pristupiti među ostalim i takvim izvorima podataka. U tom pogledu proširenje implementiramo kroz pomoćnu biblioteku objekata za raznorodne izvore podataka te objektima za rukovanje tom pomoćnom bibliotekom kako je prikazano na Slici 16. U našem slučaju to će biti biblioteka objekata za spajanje na CORBU. Na taj način kako god objekti jezgre sa Slike 16 pristupaju u jednom slučaju objektima za rukovanje glavnoj biblioteci i preko njih samoj glavnoj biblioteci objekata za pristup raznorodnim podacima, tako će u drugom slučaju pristupati objektima za rukovanje bibliotekom objekata prema arhitekturi CORBA i preko njih i same te biblioteke direktno CORBA objektima kako je također prikazano na Slici 16.

Stoga možemo zaključiti da arhitektura modela za pristup raznorodnim podacima unutar informacijske infrastrukture nije zatvorena, i da se na jednostavan način može proširiti tako da se preko nje može pristupati i svim onim raznorodnim podacima i arhitekturama, za koje nije prvotno bila namijenjena. To je vrlo važno pošto se uz arhitekturu CORBA mogu vremenom pojaviti i neke druge [18], kojima će se vjerojatno trebati moći pristupiti, a i sama CORBA će se sigurno dalje razvijati pa treba i to uzeti u obzir [118]. Zbog toga je svojstvo modela da je otvoren prema ovakvim zahtjevima vrlo važno. Ujedno to je i presudno da sam model može ispuniti svoju zadaću, a to je da može pristupiti bilo kojem podatku, bez obzira na njegov tip, format zapisa, ili semantiku odnosno protokol pristupa koji se mora pri tome ispoštovati. Stoga na arhitekturu prikazanu na Slici 16 treba gledati kao na osnovnu, principjalnu arhitekturu koja će se koristiti dalje u ovom radu, odnosno kao primjer kako se osnovna arhitektura može proširiti i time prilagoditi za pristup bilo kojem izvoru podataka unutar informacijske infrastrukture.

4. RAZRADA POSREDNIČKOG REDA MODELA

Kao što sami podaci raznorodnog tipa formata i semantike nisu posebno zanimljivi za daljnju analizu, u tom smislu što su oni nešto što već postoji u okviru informacijske infrastrukture odnosno nešto što se neće dalje modelirati, tako nije niti korisnički red, budući se isti sastoji ili od primjerice pretraživača Interneta kojeg krajnji korisnik već posjeduje, ili od neke Windows aplikacije koja koristeći DCOM protokol direktno komunicira s posredničkim redom, a koja se može implementirati na uobičajeni način kako se implementiraju i sve druge distribuirane Windows aplikacije. Stoga možemo zaključiti da je bit implementacije modela sama implementacija posredničkog reda. Uzevši u obzir i samu kompleksnost posredničkog reda, isti je potrebno dalje razraditi. Uz to je potrebno analizirati i dostupne tehnologije, te kako iste iskoristiti kako bi se posrednički red, a time i ukupan model u cjelini, uspješno i na vrijeme implementirali u praksi.

Prije svega analiziraju se uobičajeni problemi koji se javljaju prilikom razvoja distribuiranih aplikacija, kako bi se isti izbjegli prilikom implementacije samog posredničkog reda. To je važno budući da neki od njih kasno pokažu svoje nedostatke, te je ispravljanje takvih pogrešaka vrlo dugotrajan i skup proces (puno toga se mora raditi ispočetka ili se puno toga mora prepravljati).

Zatim se daje uvod u komponentno temeljenu paradigmu, pregled njenih svojstava, te se pokazuje kako je ista uz objektno orijentiranu paradigmu neophodna za uspješnu implementaciju modela. Jadno od najvažnijih svojstava komponentno temeljene paradigme je to što je dio aplikacije vrlo jednostavno zamijeniti drugim te što je nekoj postojećoj aplikaciji vrlo jednostavno dodavati nove module, čime se posrednički red bitno lakše prilagođava raznorodnim izvorima podataka. Pri tom se ne ulazi u obrazlaganje potrebitosti same objektno orijentirane paradigme, pošto se smatra da je općeprihvaćenost iste kroz dulji niz godina dovoljan argument da se koristi prilikom pisanja samih modula posredničkog reda.

Nakon toga, izabire se sama platforma odnosno operacijski sustav za implementaciju posredničkog reda, te se daje pregled tehnologija čijim se korištenjem može bitno pojednostavniti, kako implementacija posredničkog reda, tako i modela u cjelini. Izborom konkretne platforme za implementaciju posredničkog reda ne utiče se na samu otvorenost modela za pristup raznorodnim podacima, pošto se kako korisnički red odnosno korisnici, tako i sami podaci raznorodnog tipa formata i semantike mogu nalaziti na proizvoljnom operacijskom sustavu.

Na kraju se razrađuje i sama tehnologija COM, koja će poslužiti kao temelj za implementaciju gotovo svih dijelova posredničkog reda, odnosno samog modela. Ta tehnologija, među ostalim nam omogućava da prilikom projektiranja posredničkog reda kombiniramo objektno orijentiran i komponentno temeljen pristup. Dodatno se opisuje i način na koji je proširena tehnologija COM, kako bi se mogućnostima komponenata COM dodala neka nova svojstva koja će nam olakšati izgradnju samog posredničkog reda, ali ujedno i sačuvala kompatibilnost sa svim drugim COM komponentama izgrađenim na standardni način koji ne sadrži ta proširenja. Prije svega, opisuje se uvođenje proširenog algoritma agregacije kao zamijene za običnu agregaciju, te se opisuje kako se može implementirati dinamičko nasljeđivanje metoda sučelja za prikazne jezike WWW-a (World Wide Web).

4.1. Problemi pri klasičnom razvoju distribuiranih aplikacija

U ovom pod poglavlju opisati će se klasičan način razvoja aplikacija i distribuiranih aplikacija te će se navesti problemi koji se pri tome javljaju, i to posebno ukoliko se ne koristi objektno orijentirana i komponentno temeljena paradigma.

Praksa programskog inženjerstva ukazuje na prikladnost klasifikacije distribuiranih aplikacija u skladu s brojem klijenata. U tom smislu mogu se razlikovati:

- dvokorisničke aplikacije;
- srednjekorisničke aplikacije;
- mnogokorisničke aplikacije.

U slučaju dvokorisničkih aplikacija, imamo samo dva korisnika koji se nalaze na dva različita računala. To je najjednostavnije za realizirati, jer nije potreban gotovo nikakav poseban rad na protokolu. U slučaju srednjekorisničkih aplikacija, imamo tri ili više korisnika. Svaki od njih radi sa sustavom koristeći aplikaciju koja sa drugim komunicira po nekom protokolu koji je u ovom slučaju obično znatno složeniji. Pod ovim pojmom podrazumijeva se neka specifična aplikacija koju koristi uži krug ljudi (program za tele medicinu koji u isto vrijeme koristi nekoliko liječnika, sustav za video konferencije i sl.). Pod pojmom mnogokorisničkih aplikacija, podrazumijevaju se aplikacije koje su namijenjene za pristup preko Interneta najširem krugu ljudi, kao npr. knjižara ili virtualna trgovina i slično. U ovom slučaju koriste se aplikacije zasnovane na HTML-u odnosno DHTML-u.

Budući da se mnogokorisničke aplikacije oslanjaju na HTML/DHTML, mogućnosti su ograničene odnosno bitno manje nego kod srednjekorisničkih aplikacija. Naravno, njihova prednost u jednostavnosti jest što se u principu vrlo lako koriste, za što je potrebno imati samo standardni pretraživač Interneta. S druge strane isto je tako očito da su dvokorisničke aplikacije samo specijalan slučaj srednjekorisničkih aplikacija kao što su i nedistribuirane aplikacije samo specijalan slučaj distribuiranih. Stoga je kao primjer načina izvedbe jedne standardne aplikacije upravo izabrana jedna srednjekorisnička aplikacija. U našem slučaju to će biti neka aplikacija za tele medicinu.

Prvo je potrebno definirati module potrebne za izgradnju ove aplikacije. Prije svega, treba nam raspodijeljeni zaslon (engl. whiteboard). To je prozor u kojeg liječnici mogu učitavati slike, vršiti operacije nad njima te označavati važne detalje na slikama. Nakon što jedan liječnik učita neku sliku u raspodijeljeni zaslon, to trebaju vidjeti svi liječnici koji su spojeni u sustav. Isto tako, nakon što netko izvrši neku operaciju nad slikom ili nacrta nešto na njoj, promjenu moraju vidjeti svi. Sljedeće što nam treba jest zvukovna komunikacija. Vrlo je korisno ukoliko liječnici mogu usmeno razgovarati o dotičnoj slici odnosno slikama, jer isti mogu imati vrlo loše daktilografske sposobnosti. Sam protokol je bitno jednostavnije implementirati ukoliko istovremeno samo jedan liječnik mijenja sadržaj raspodijeljenog zaslona, i ukoliko samo jedan liječnik (ne mora biti taj isti) koristi mikrofon, jer u tom slučaju nije potrebno miješanje zvuka. U tom slučaju vrlo je praktično da se implementira i tekstualni IRC (Internet Relay Chat) prozor kojeg svi istovremeno mogu koristiti, nakon što netko napiše neku poruku ista će se pojaviti u pridruženim prozorima svih drugih korisnika. Tko želi može koristeći ga zatražiti riječ ili pravo rada nad raspodijeljenim zaslonom.

Sustav je najjednostavnije implementirati kao sustav jednog poslužitelja i nekoliko jednakih klijenata, gdje je i poslužitelj i klijent svaki za sebe zasebna monolitna aplikacija. Svaki liječnik se prilikom podizanja klijenta spaja na poslužitelj. Nakon povezivanja ima informaciju o tome da li je još netko u tom trenutku tamo spojen. Svi klijenti mogu direktno komunicirati sa poslužiteljem, a preko njega i s drugim klijentima. Prednost ovog načina komunikacije jest što klijent aplikacija ne

mora znati na kojem računalu se nalaze druge klijent aplikacije, odnosno svaka se u sustav uključuje automatski. Jednostavnost pri rukovanju je vrlo važna budući da korisnici ne moraju imati jako tehničko predznanje.

Sama komunikacija može se temeljiti primjerice na protokolu TCP/IP (Transmission-Control Protocol/Internet Protocol). Protokol razmjene poruka između klijenata i poslužitelja definira sam razvojni inženjer, dok samu razmjenu poruka osigurava operacijski sustav. U našem slučaju, najjednostavniji protokol može se implementirati uz korištenje značke za raspodijeljeni zaslon i značke za zvuk. Onaj tko dotičnu ima, ima i pravo rada na raspodijeljenom zaslonu odnosno pravo korištenja mikrofona. Liječnik koji se prvi spojio na poslužitelja može po dogovoru biti taj koji određuje kome će koju značku dati u kome trenutku i dodjeljuje mu istu. Ovo možda nije najjednostavnije sa stajališta korisnika, ali je najjednostavnije za implementaciju odnosno za ovaj primjer.

Sama implementacija se može izvesti na sljedeći način. Za svaki prozor klijent ima posebnu dretvu korisničkog sučelja kao i kod bilo koje druge jedno korisničke aplikacije. Dodatno svaki klijent može imati posebne radne dretve i to:

- radnu dretvu za lokalni prikaz raspodijeljenog zaslona;
- radnu dretvu za prijenos zvuka;
- radnu dretvu za IRC prozor;
- radnu dretvu za komunikaciju sa poslužiteljem.

U slučaju da liječnik koji ima značku za rad na raspodijeljenom zaslonu vrši nad istom neke operacije (učitava sliku, mijenja neku ili nešto na nekoj crta ili pokazuje), radna dretva za lokalni prikaz raspodijeljenog zaslona osvježava raspodijeljeni zaslon, ali i o dotičnim promjenama obavještava poslužitelja. On te promijene šalje svim ostalim klijentima gdje pripadne lokalne radne dretve za lokalni prikaz raspodijeljenog zaslona te iste promijene prikazuju. Zadatak radne dretve za prijenos zvuka kod onog klijenta kod kojeg liječnik ima značku za zvuk jest kompresija zvuka u realnom vremenu i slanje istog poslužitelju. Zadatak radnih dretva za prijenos zvuka kod ostalih klijenata jest primanje istog od poslužitelja, dekompresija u realnom vremenu i reprodukcija. U slučaju radne dretve za tekstualni prozor, svaka lokalne promijene prikazuje i šalje poslužitelju dok također i svaka prikazuje promijene koje dobije od poslužitelja. Ove radne dretve ne komuniciraju sa poslužiteljem direktno, već izmjenjuju poruke sa radnom dretvom za komunikaciju sa poslužiteljem. Ona direktno sa poslužiteljem komunicira TCP/IP protokolom.

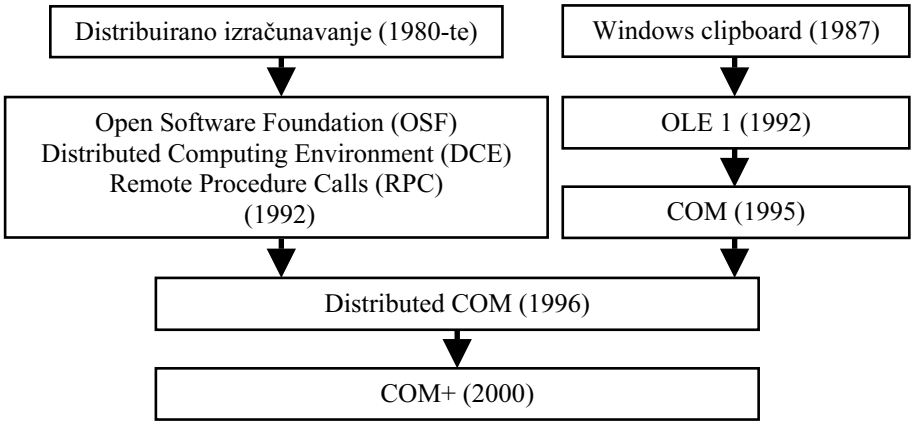
Osnovni nedostaci ovakvog pristupa su:

- imamo dvije monolitne aplikacije (klijenta i poslužitelja) čiji dijelovi se vrlo teško mogu iskoristiti prilikom izgradnje drugih aplikacija;
- cijeli sustav je vrlo složen i u najjednostavnijem obliku, vrlo je teško pronalaziti pogreške u kodu i ostale nepravilnosti u radu;
- sustav je neprilagodljiv izmjenama – bilo koja zahtijeva kompletno prevođenje cijelog programa;
- na razvoju sustava teško istovremeno može raditi više od jednog programera.

Zbog ovakvih i sličnih problema, došlo je do razvoja komponentno temeljene paradigme u zadnjih nekoliko godina. Uvod u tu tehnologiju, a slobodno možemo reći i filozofiju razvoja aplikacija, kako nedistribuiranih tako i onih distribuiranih, dat je u sljedećem poglavlju.

4.2. Pristup firme Microsoft komponentno temeljenoj paradigmi

Microsoft je u komponentno temeljenoj paradigmi (uz objektno orijentiranu), odnosno u projektiranju aplikacija modeliranjem komponenata, prepoznao budućnost i u zadnje vrijeme u to ulaže izuzetno puno. Kako se ta paradigma razvijala prikazano je na Slici 19 [107][29]. DCOM i COM+ su nastali kroz evoluciju razvoja, sa jedne strane metoda i postupaka za distribuirano izračunavanje, a s druge tehnologija koje je razvijao sam Microsoft. Budući da Microsoft danas drži i veliku većinu alata za razvoj aplikacija (Visual C++, Visual Basic, Visual J++), koji se kao i sami operacijski sustavi temelje na komponentnom pristupu, logičan zaključak jest da će ova paradigma obilježiti vrijeme koje dolazi.

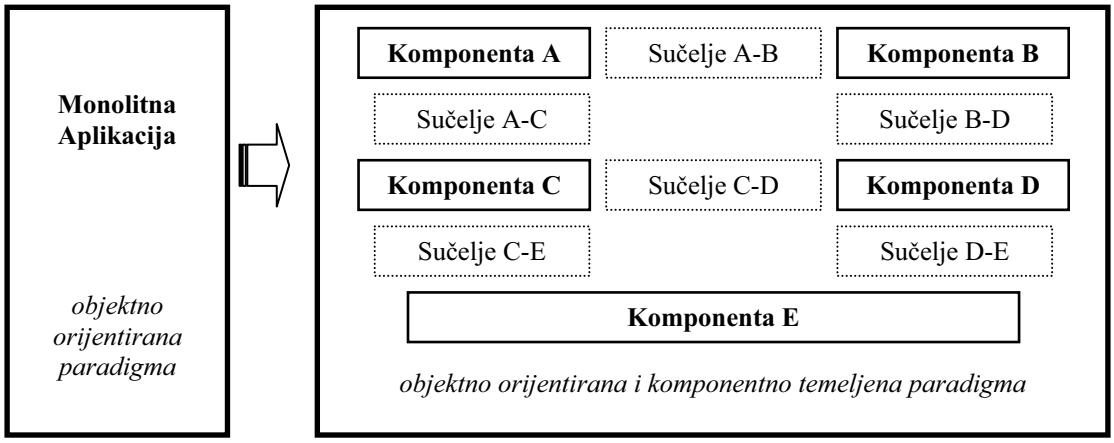


Slika 19: Razvoj tehnologije COM.

Sljedeći segment prilikom razvoja aplikacija o kojem se mora voditi računa jest podrška razvoju distribuiranih aplikacija. Pitanje koje se tu obično postavlja jest koliko je teže razvijati distribuirane aplikacije u odnosu na nedistribuirane. Ukratko možemo rezimirati da prilikom odabira okruženja za razvoj programske podrške treba voditi računa o tome da li je podržana objektno orijentirana paradigma, da li je podržana komponentno temeljena paradigma, te da li je podržan jednostavan razvoj distribuiranih aplikacija.

Izabrati neku komponentno orijentiranu paradigmu koristeći danas dostupnu tehnologiju u principu znači izabrati između COM-a [156] i JavaBeans komponenata [157]. Prednost prve tehnologije jest što se komponente mogu pisati u bilo kojem jeziku. Prednost druge jest u tome što su komponente napisane u Javi neovisne o operacijskom sustavu, no veliki nedostatak jest upravo u tome što se JavaBeans komponente moraju pisati upravo u Javi (manje su im mogućnosti).

Pitanje koje se postavlja jest koji je odnos između objektno orijentirane paradigme i komponentno temeljene paradigme? Budući da se same komponente mogu pisati i u asembleru, jasno je da komponentno temeljena paradigma ne mora nužno biti i objektno orijentirana, već je u nekim slučajevima samo objektno temeljena ili nije čak ni to. S druge strane, ni objektno orijentirana paradigma nije u svakom slučaju komponentno temeljena, najbolji primjer je razvoj neke monolitne aplikacije u jeziku C++. Zbog mnogih problema koji se javljaju prilikom izvedbe kompleksnih monolitnih aplikacija, a koji se vrlo uspješno rješavaju komponentno temeljenim pristupom, možemo zaključiti da su monolitne aplikacije pristup koji izumire. Rješenje je u kombinaciji komponentno temeljenog pristupa i objektno orijentiranog pristupa. Kako je prikazano na Slici 20, bit komponentno temeljene paradigme jest u tome da se monolitna aplikacija razloži na komponente, nakon čega se svaka komponenta može zasebno implementirati koristeći objektno orijentiranu paradigmu. U ovom slučaju dobro je što se ove dvije paradigme međusobno ne isključuju, već naprotiv, nadopunjuju.



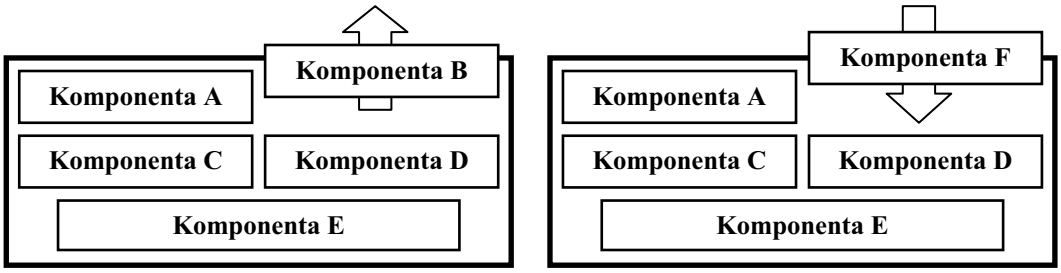
Slika 20: Kombinacija OO i komponentno temeljene paradigme je najbolje rješenje.

Danas postoje dva pristupa prilikom izgradnje distribuiranih aplikacija: CORBA [155] i DCOM [29][150]. U neku ruku možemo reći da su ova dva pristupa konkurencija jedan drugome. No isto tako, oni se ne isključuju. Aplikacija istovremeno može koristiti i DCOM i CORBA pristup. I sama CORBA specifikacija svjesna utjecaja COM-a je podržala sučelje prema istom. Dodatno, koristeći nasljednika COM-a – DCOM, distribuirane aplikacije mogu se razvijati analogno dosadašnjem razvoju nedistribuiranih aplikacije. I na kraju, koristeći nasljednika DCOM-a – COM+, tehnologiju koja se upravo razvija, predviđa se kako će buduće jedinstveno okruženje Windows DNA (WINDOWS Distributed interNet Application), zasigurno konačno podržati gotovo istovjetan način razvoja aplikacija, bilo da su to nedistribuirane aplikacije, aplikacije temeljene na modelu klijent/poslužitelj ili na Internetu [97]. Zahvaljujući toj tehnologiji, potpuno je transparentno da li se komponenta nalazi u drugom procesu lokalnog računala ili na bilo kojem drugom računalu u mreži. Tehnologijom COM+ se također kroz OLE DB razvija i univerzalni pristup podacima koji je dodatni vrlo važan element o kojem će se morati voditi računa prilikom razvoja aplikacija u budućnosti. Možemo zaključiti da se evolucijskim razvojem COM-DCOM-COM+ bitno unaprijedila tehnologija razvoja programske podrške.

Svaku COM komponentu (a COM komponenta je binarni izvršni kod napisan po COM standardu) možemo podijeliti na dva dijela: sučelje COM komponente i tijelo COM komponente. Samo sučelje je jedini dio kojeg vidi korisnik. Tijelo predstavlja unutrašnju građu komponente (programski kod) koja mora biti u potpunosti inkapsulirana. Da bi se shvatila važnost komponentno temeljene paradigme, potrebno je razmotriti neke od najvažnijih prednosti prilikom izgradnje aplikacija koristeći tu tehnologiju:

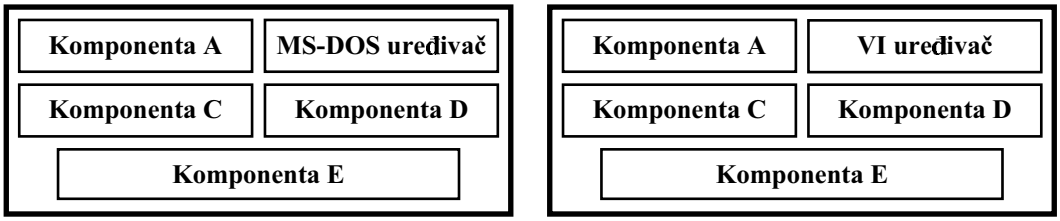
- dio aplikacije je vrlo jednostavno zamijeniti novim;
- aplikacije se lakše prilagođavaju različitim korisnicima;
- biblioteke komponenata omogućuju brz i jeftin razvoj;
- komponente neke aplikacije se mogu nalaziti na različitim računalima u mreži;
- razvoj kompleksnih aplikacija je bitno pojednostavljen.

Pošto je aplikacija prvenstveno sastavljena od komponenata koje su već prevedeni i povezani objekti dati u izvršnom obliku (DLL ili EXE datoteka), jasno je da je promjena neke komponente odnosno nekog dijela aplikacije vrlo jednostavna kako je to prikazano na Slici 21. U slučaju monolitnih aplikacija, ne samo da je potrebno ponovno prevesti i povezati cijelu aplikaciju, već je često potrebno i ručno prepravljati neke dijelove samog koda što je mukotrpan i dugotrajan te samim tim skup i neprihvatljiv proces, koji je uz to često i uzrokom mnogih previđenih grešaka u kodu.



Slika 21: Zamjena jedne komponente neke aplikacije novom je vrlo jednostavno.

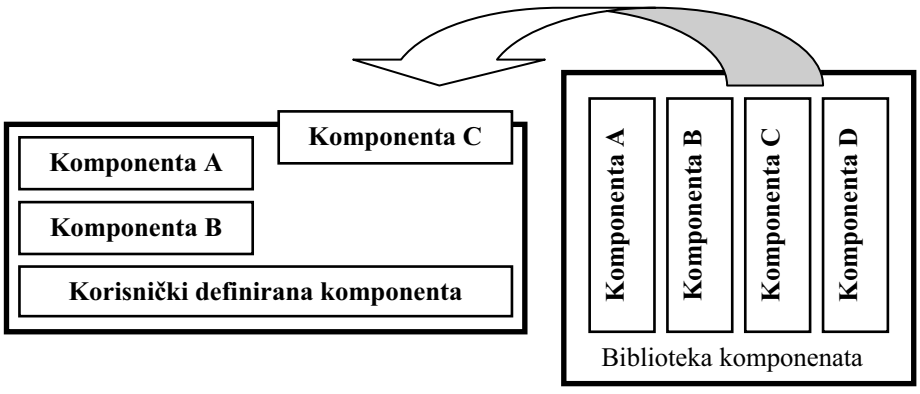
Samom činjenicom da se neka komponenta odnosno dio aplikacije vrlo lako može zamijeniti drugim, kako je prikazano na Slici 22 aplikacije se puno lakše prilagođavaju različitim korisnicima. Korisnik sam može birati koju verziju koje komponente će koristiti kao i da li će neku komponentu uopće koristiti. Dodatno, on sve to može raditi i za vrijeme izvođenja same aplikacije. Na taj način on može cijelu aplikaciju prilagoditi svojim potrebama, odnosno sve u njoj postaviti na način na koji – on to radi. Korisnici puno češće biraju za rad ovakve nego neprilagodljive aplikacije.



Slika 22: Aplikacije se puno lakše prilagođavaju različitim korisnicima.

Biblioteke komponenata bitno ubrzavaju razvoj novih aplikacija kako je to prikazano na Slici 23. Ne moramo razvijati komponente koje već postoje, a pošto je vrijeme razvoja kraće, puno je veća prilagodljivost promjenjivim zahtjevima tržišta. Posebno je važno što ne moramo trošiti vrijeme na provjeru ispravnosti komponenata za koje držimo da dolaze od pouzdanih proizvođača. Firme mogu pisati i svoje lokalne biblioteke te time izbjeći da se ista stvar razvija dva ili više puta.

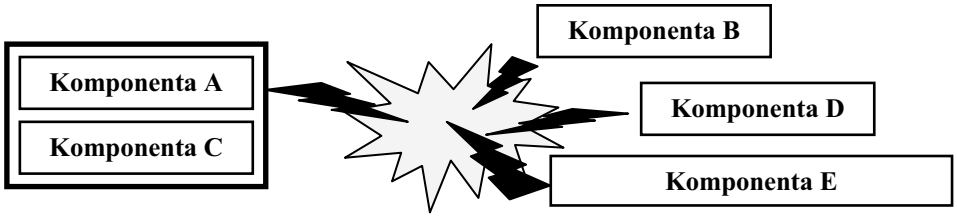
Budući da je ovo nova tehnologija, potrebno je temeljito proučiti sve ono što je vezano za sam izazov korištenja gotovih komercijalnih komponenata [136]. Pri tome posebno treba obratiti pozornost na realnu cijenu takvog razvoja [127], provjeru ispravnosti dostupnih komponenata odnosno certifikaciju istih [135], te na ostale sigurnosne rizike koji se javljaju prilikom korištenja te tehnologije [80][146], naravno ukoliko ne koristimo komponente koje smo sami napravili.



Slika 23: Biblioteke komponenata bitno ubrzavaju razvoj novih aplikacija.

Komponente jedne te iste aplikacije se mogu nalaziti na različitim računalima u mreži kako je prikazano na Slici 24. Time je bitno pojednostavljena izgradnja distribuiranih aplikacija. Dodatno,

komponenta koja se izvodi na udaljenom računalu može na njemu izvršiti operacije koje inače ne bi mogle biti izvršene na standardni način, te samim tim dobivamo i veće mogućnosti aplikacije.



Slika 24: Komponente neke aplikacije se mogu nalaziti na različitim računalima u mreži.

Pošto je opće poznato da se kompleksni problemi bitno jednostavnije rješavaju ukoliko se mogu razložiti na dijelove, intelektualni napor prilikom razvoja kompleksnih aplikacija, bitno je pojednostavljen u odnosu na razvoj monolitnih aplikacija. Dodatna prednost jest u tome što kod komponentno temeljene paradigme na jednom projektu može raditi puno veći broj ljudi – svaki radi na svojoj komponenti dok voditelj projekta sve to komponira u završnu aplikaciju.

Zahtjevi koji se postavljaju na sve COM komponente (isto tako i na DCOM i COM+ komponente samo što tamo postoje još neki dodatni zahtjevi) su:

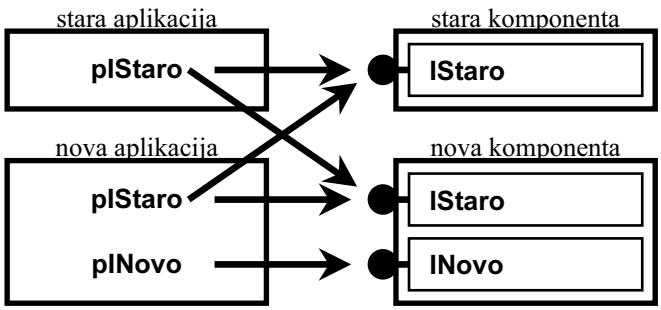
- mogućnost zamjene komponente za vrijeme dok se aplikacija izvodi;
- sve što korisnik vidi kod neke komponente jest njeno sučelje, ostatak mora biti inkapsuliran;
- moraju sakriti jezik u kojoj su napisane, odnosno moraju se isporučiti u binarnom obliku;
- nove komponente trebaju podržavati postojeće aplikacije;
- moraju biti transparentne na mreži.

Vrlo važan zahtjev jest da se komponente mogu mijenjati za vrijeme izvođenja aplikacije. Ovo je važno da se omogući, iako se krajnjem korisniku neće uvijek davati kao opcija. Ovaj zahtjev znači imati mogućnost da se komponente dinamički povezuju. Zašto je to važno? Pa pretpostavimo da nismo osigurali taj zahtjev. Nakon što zamijenimo samo jednu komponentu, aplikaciju bi trebalo statički povezivati. Čak i ako korisnik ima odgovarajući povezivač, nismo sigurni da ga zna ispravno koristiti, a ako i zna, možda neće imati odgovarajuću verziju. Sve to je potpuno neprihvatljivo jer takva aplikacija se ne bi u mnogome razlikovala od monolitne aplikacije. Stoga je zahtjev da se komponente mogu dinamički zamjenjivati vrlo važan.

Zahtjev da sve što korisnik vidi kod neke komponente jest njeno sučelje dok ostatak mora biti inkapsuliran ne postoji samo radi toga da se olakša izrada samih aplikacija. On direktno slijedi iz prethodnog zahtijeva. Nakon što jednu komponentu zamijenimo drugom sa istim sučeljem, ostatak aplikacije ne smije vidjeti nikakvu promjenu. To znači da unutrašnja struktura komponente mora biti potpuno inkapsulirana.

Na sličan način možemo zaključiti i da jezik implementacije komponente mora biti sakriven, odnosno da komponente moraju biti isporučivane već prevedene, povezane i spremne za upotrebu (u binarnom obliku). U protivnom bi aplikacija morala znati u kojem jeziku je neka komponenta napisana što bi značilo neku dodatnu međuzavisnost. Drugim riječima sučelje komponente ne bi bilo jedino što znamo i što trebamo znati o njoj.

Nove komponente moraju podržavati stare aplikacije kao što i nove aplikacije moraju podržavati stare komponente, kao što je prikazano na Slici 25. U protivnom se ne bi moglo predvidjeti kod kompleksnih aplikacija, ili bi to bilo neprihvatljivo teško, da li će ista i dalje raditi nakon što, npr. samo jednu komponentu zamijenimo s npr. novijom verzijom te iste komponente.



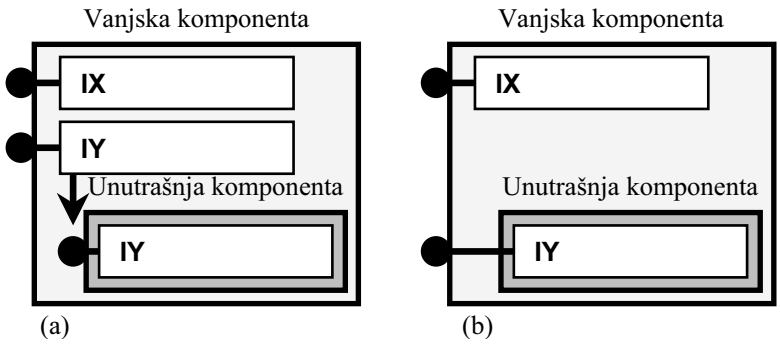
Slika 25: Nove komponente moraju podržavati postojeće aplikacije i obrnuto.

Zahtjev da komponente moraju biti transparentne na mreži ustvari znači da komponente pišemo istovjetno, bez obzira da li će se ta komponenta nalaziti unutar istog procesa kao i aplikacija koja je koristi, ili će se nalaziti unutar nekog drugog procesa na lokalnom ili udaljenom računalu. Prednost ovog pristupa jest u tome što jedna te ista komponenta može biti građevni element kako nedistribuirane tako i neke distribuirane aplikacije.

Dodatni (neobavezni) zahtjevi koje bi bilo dobro da podržavaju COM odnosno DCOM komponente su:

- agregacija (sadržavanje podržavaju automatski);
- posebno sučelje za makro jezike;
- apartman-dretve (engl. apartment thread), slobodne-dretve (engl. free thread) ili oboje (engl. both).

Osnovna razlika između sadržavanja i agregacije prikazana je na Slici 26. U slučaju sadržavanja, unutrašnja komponenta se u potpunosti nalazi unutar vanjske koja mora poznavati sva sučelja koja unutrašnja komponenta posjeduje. U slučaju agregacije, vanjska komponenta ima informacije samo o tome koju komponentu agregira, ostalo je ne zanima. Jedno od pravila koje svaka COM komponenta mora zadovoljiti jest da bilo koje sučelje možemo pitati da li ta komponenta podržava i bilo koje drugo sučelje. U slučaju sadržavanja, jasno je da sučelja *IX* i *IY* jednostavno zadovoljavaju to pravilo budući da su to sučelja jedne te iste komponente. U slučaju agregacije, situacija više nije tako jednostavna budući da korisnik pristupa direktno sučelju unutrašnje komponente *IY*, a sama ta komponenta ne može u naprijed znati u koje će sve komponente biti agregirana pa samim tim ne može automatski korisniku ni odgovoriti da, primjerice u našem slučaju, komponenta koju koristi podržava i *IX* sučelje. Stoga postoji i poseban mehanizam kojim komponente međusobno komuniciraju, tako da se njime rješava ovaj problem. Ako ga komponente imaju implementiranim, za njih se kaže da podržavaju agregaciju. Bitna prednost agregacije je u tome što korisnik direktno pristupa agregiranim komponentama pa je sam rad brži. Sljedeća prednost je u tome što vanjska komponenta ne mora znati što sve sadržava komponenta koju agregira. U našem slučaju, npr. možemo zamisliti da unutrašnju komponentu zamijenimo novijom verzijom koja sadržava i sučelje *IZ*. U slučaju sadržavanja, vanjsku komponentu moramo također prilagoditi da podržava to novo sučelje, dok će u slučaju agregacije to ona učiniti automatski (ukoliko je ispravno napisana odnosno napisana da to podržava). Stoga je zbog ovih prednosti vrlo važno da COM komponente podržavaju agregaciju.



Slika 26: Primjer sadržavanja (a) i agregacije (b).

Prilikom stvaranja same komponente, ona nam daje kazaljku na njeno sučelje. Iako je to vrlo prihvatljivo rješenje u slučaju programskog jezika C++, klijenti napisani u Visual Basicu ili Javi ne mogu komponentama pristupati na taj način. Stoga neke komponente podržavaju i *IDispatch* sučelje namijenjeno isključivo osiguranju pristupa iz takvih jezika. Da bi se preko tog sučelja pristupilo komponenti, dovoljan nam je tekstualni zapis metode i pridruženih argumenata. Stoga preko tog sučelja komponenti mogu pristupati kako makro jezici tako i svi drugi jezici koji ne podržavaju kazaljke. To je velika prednost, no na ovaj je način pristup sporiji i korištenjem samo ovog načina mogućnosti su manje nego koristeći "standardan" C++ pristup. Stoga je idealan slučaj kada neka komponenta uz standardna sučelja namijenjena za C++ korisnike podržava i *IDispatch* sučelje. Novije verzije Visual Basica i Jave mogu COM komponentama pristupati i koristeći biblioteke sučelja, no *IDispatch* sučelje potrebno je zbog drugih jezika koji to ne mogu, kao što je primjerice HTML.

Svaka komponenta ne mora podržavati višezadaćnost, ili može podržavati apartman dretve, slobodne dretve ili oboje. Ukoliko neka komponenta npr. podržava slobodne dretve, to znači da dok joj preko jednog sučelja pristupa jedna radna dretva preko nekog drugog joj istovremeno može pristupati neka druga radna dretva. Pošto operacijski sustav Windows NT može istovremeno podržavati i do 32 procesora, moguće je da se jedna radna dretva izvodi na jednom, druga na drugom, a komponenta na trećem, tako da imamo i stvarnu, a ne samo prividnu istovremenu izvodivost. Koji oblik višezadaćnosti neka komponenta podržava, zapisano je u Windows registratoru. Windows registrator inače se koristi za zapis sistemskih parametara, a potrebno ga je dobro poznavati kako bi se COM klase uspješno pisale.

4.3. Izbor tehnologija za izvedbu posredničkog reda

Ono što je potrebno izabrati jest platforma odnosno operacijski sustav na kojem će se implementirati posrednički red modela sa Slike 16. Iako to može biti bilo koja komercijalna razvojna okolina, od UNIX-a i LINUX-a, pa do MAC OS 8, OS/2 Worp 4 i 32-bitnih Windowsa, Windows NT Server 4.0 je odabran zbog pratećih tehnologija koje nam ta razvojna okolina pruža [14][37][121]. Sustav naravno ne gubi na otvorenosti izborom neke konkretne platforme.

Prvo što moramo imati jest sustav, u kojeg ćemo moći programske module jednostavno dodavati prema potrebi. Ovim pristupom, kada se u okviru informacijske infrastrukture pojavi neki novi, do tada ne podržan izvor podataka, potrebno je napisati modul koji će podržati pristup istom. Jezik u kojem ćemo pisati te module mora biti dovoljno otvoren da se iz njega može pozvati bilo koji sistemski poziv u bilo kojem trenutku, jer upravo to nam garantira da ćemo moći pristupiti praktički bilo čemu. Logičan izbor je C++, budući da je upravo u tom jeziku i pisan operacijski sustav Windows NT Server 4.0. Samim tim, prilikom izbora vrste modula za inkapsulaciju C++ koda, logično se nameće COM, budući da je to najpodržanija komponentno temeljena tehnologija razvoja programske podrške za sustave Windows, te da je upravo C++ jedan od najčešćih jezika u kojem se implementiraju COM komponente. Na taj se način model može razvijati koristeći paradigmu koja je i objektno orijentirana (C++) i komponentno temeljena (Distributed COM).

Sljedeće što je potrebno jest pogledati što svakom od dijelova posredničkog reda prikazanim na Slici 16 odgovara kao implementacija u praksi, što od toga već postoji i kao takvo se može iskoristiti, što samo djelomično zadovoljava traženu funkcionalnost, te što će se eventualno trebati pisati od početka. Također treba pogledati za one dijelove koji će se pisati i dalje razvijati, kakvu dodatnu komercijalnu podršku imamo.

Na Slici 27 prikazano je kako se posrednički red modela može implementirati na Windows NT Server 4.0 OS-u, dat je pregled dostupnih tehnologija koje možemo koristiti za razvoj modela, te su prikazana i moguća proširenja koja se mogu dodati po potrebi. Možemo prepoznati tri podreda posredničkog reda. Sučelje prema korisničkom redu sastoji se od sljedećih dijelova: Windows Terminal Server, Citrix MetaFrame, IIS (Internet Information Server) i ASP [6][75] (Active Server Pages). Objekti jezgre posredničkog reda implementirani su kao objekti COM. Objekti za pristup podacima sastoje se od ADO [74][77][81][159] (ActiveX Data Objects) objekata, OLE DB objekata [7][8][48] te od dodatne opcije - lokalne baze podataka koja primjerice može biti Access baza podataka (64 slobodne licence uključene u cijenu svakog Windows NT Server 4.0 sustava). Tu se još nalazi i OrbixCOMet sa pripadnim COM objektima za rukovanje tom bibliotekom za pristup arhitekturi CORBA. Izvore podataka raznorodnog tipa, formata i semantike u praksi predstavljaju prije svega: velike baze odnosno baze za vrlo velik broj korisnika (DB2, Oracle, SQL Server [152], Sybase, Informix, NCR Teradata), ISAM (Indexed Sequential Access Method) baze podataka (Btreive, dBase, Paradox, Microsoft FoxPro, Lotus 1-2-3) te svi ostali izvori raznorodnih podataka tipa zapisi tabličnih proračuna, elektronička pošta, datoteke, tekst i sl. Također, kako se iz slike vidi, komponente jezgre posredničkog reda i ADO objekti [19] mogu se izvršavati u sklopu lokalnog MTS (Microsoft Transaction Server) poslužitelja.

Windows Terminal Server i Citrix MetaFrame kao cjelina predstavljaju poslužitelja koji šalje korisničkom redu grafički izgled višemedijskog sučelja kako je prikazano na Slici 16. Na taj način omogućuje se da se na samom Windows NT operacijskom sustavu izvršava neka aplikacija, dok se sučelje iste može prikazati na danas skoro svim operacijskim sustavima, kako je prikazano na Slici 27. Sam Windows Terminal Server je Microsoftov proizvod, i njegov glavni nedostatak jest u tome što podržava samo 32-bitne Windowse i Windows CE operacijski sustav. Također, Windows Terminal Server prenosi preko mreže samo grafičko sučelje, dok prijenos zvuka odnosno općenito

višemedije nije podržan. Firma Citrix sa svojim Citrix MetaFrame proizvodom predstavlja nadogradnju na Windows Terminal Server, time što su podržani skoro svi operacijski sustavi te što je dodatno podržan zvuk odnosno višemedija. Zbog ovog posljednjeg, bolje je koristeći Citrixov MetaFrame spajati i 32-bitne Windowse i Windowse CE. Inače, treba voditi računa i o tome da se za sada Windows Terminal Server isporučuje samo kao zasebni operacijski sustav [92], stoga što se morala modificirati sama jezgra Windows NT 4.0 operacijskog sustava, da bi dotična funkcionalnost bila podržana. Budući da Windows Terminal Server neće biti podržan kao usluga odnosno dio samog Windows NT Server operacijskog sustava do dolaska novije verzije [92] (Windows 2000 Server), koja još nije komercijalno dostupna u vrijeme pisanja ovog magistarskog rada, te budući da ovaj dio sučelja posredničkog reda prema korisničkom redu možemo već sada gledati kao nešto što je komercijalno nabavljivo, tom dijelu se neće više posvećivati posebna pažnja.

Za implementaciju HTTP poslužitelja u Slici 16, može se u praksi iskoristiti IIS kako je prikazano na Slici 27. Dotični poslužitelj može generirati HTML procesiranjem ASP stranica i tako osigurati pristup svim korisnicima koji posjeduju neki standardni pretraživač Interneta. Za korisnike koji posjeduju Internet Explorer 4.0 ili noviju verziju, IIS može generirati i DHTML [58] čime se korisnicima može pružiti još veća funkcionalnost samog pristupa. Same stranice na računalu gdje je implementiran posrednički red mogu biti napisane koristeći razno razne skripte odnosno mogu biti napisane koristeći tehnologiju ASP. Ta tehnologija omogućuje da se iz samog HTML-a odnosno DHTML-a direktno pozovu COM objekti jezgre posredničkog reda i na taj način pristupi podacima, te ih se u okviru HTML/DHTML formata proslijedi korisniku. Stoga je potrebno dobro poznavati ASP tehnologiju [53], te interakciju iste s objektima jezgre posredničkog reda.

Za implementaciju sučelja prema korisničkom redu kako je prikazano na Slici 14, može se koristiti Distributed COM tehnologija i tako omogućiti svim aplikacijama koje se izvode na 32-bitnim Windowsima, da direktno pristupaju objektima jezgre posredničkog reda, bez obzira što se iste izvode na fizički različitim računalima. Windows NT Server 4.0 ili ostale novije verzije Windows operacijskih sustava imaju Distributed COM podršku, dok recimo Windows-95 operacijski sustav treba proširiti Distributed COM proširenjem da bi se mogla koristiti dotična tehnologija (instalira se odgovarajući paket koji se može besplatno skinuti s Interneta).

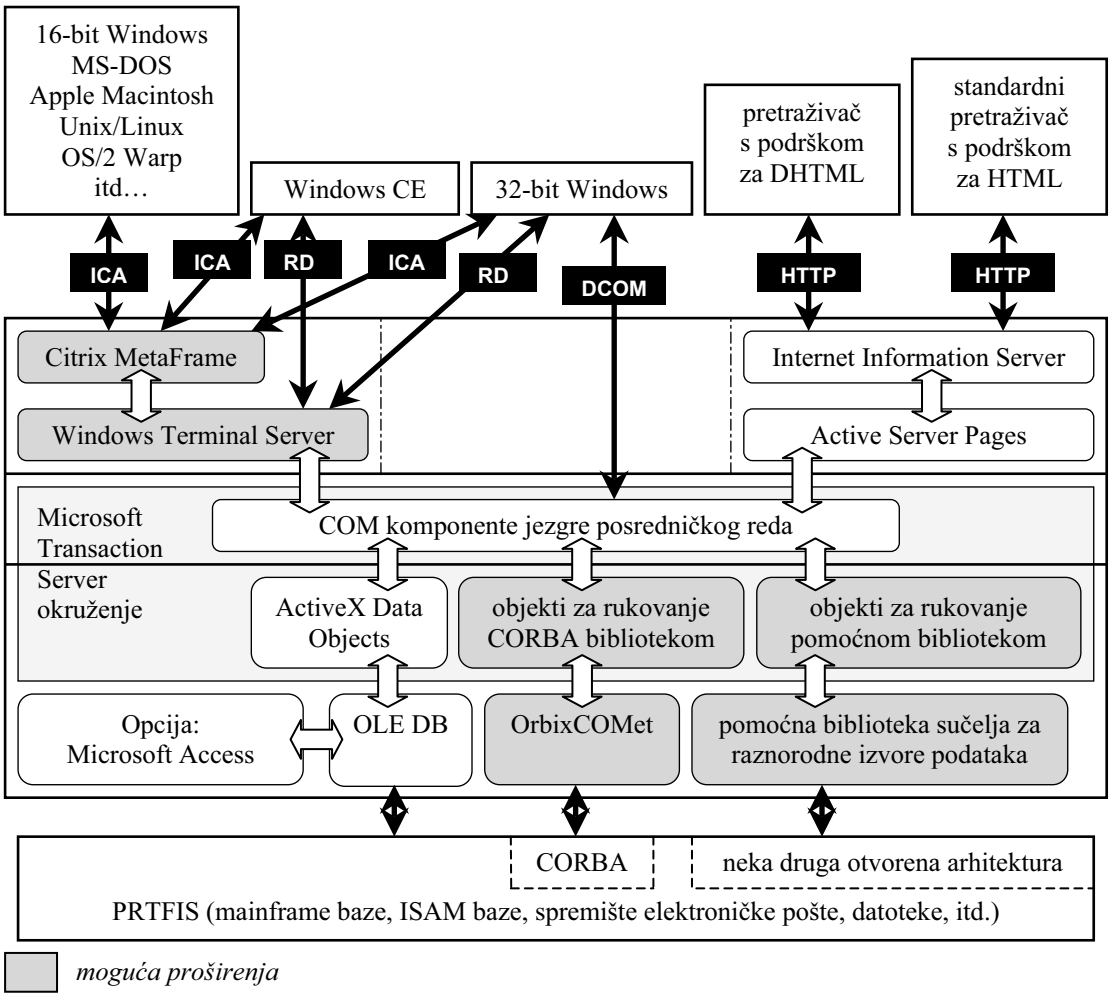
COM objekti jezgre posredničkog reda i ADO objekti mogu se zajedno izvršavati koristeći usluge lokalnog poslužitelja za objekte sa brzom aktivacijom odnosno deaktivacijom koji je u našem slučaju Microsoft Transaction Server [54]. Ovo je posebno zanimljivo područje budući da je MTS relativno nova tehnologija. Posebno je zanimljivo kako pisati komponente za dotično okruženje, te kakve su performanse sustava napisanog na taj način u realnom vremenu. Budući da su same komponente jezgre posredničkog reda COM komponente, iste mogu biti pisane među ostalim u jezicima Visual C++, Visual J++ i Visual Basic.

Biblioteka sučelja za pristup raznorodnim podacima jest skup COM komponenata podržanih kroz OLE DB. Microsoft Transaction Server poslužitelj uz OLE transakcije podržava i X/Open DTP XA standard [29]. XA je dvo-fazni protokol podrške podržan od strane skoro svih UNIX baza podataka, što znači da se koristeći OLE DB tehnologiju možemo spajati kako na Microsoftove proizvode, tako i na sve baze vezane za UNIX, a tu spadaju gotovo sve ostale.

Lokalnu bazu podataka najjednostavnije je implementirati koristeći recimo Access, posebno ukoliko nam je potrebna neka manja funkcionalnost, te ukoliko nam neće trebati više od 64 istovremena pristupa. U protivnom, ukoliko trebamo implementirati veliku odnosno složenu i zahtjevnu bazu podataka za više korisnika, lokalno ili na drugom računalu možemo imati i neki drugi sustav poslovanja relacijskih baza kao npr. SQL Server [160], ili čak i neki drugi ne Microsoftov sustav poslovanja relacijskih baza [140] kao recimo DB2, Oracle ili Sybase.

Ukoliko se želi omogućiti pristup arhitekturi CORBA kako je prikazano na Slici 16, može se koristiti OrbixCOMet proizvod Iona Tecnology firme. U tom slučaju bi COM klase za rukovanje odnosno pristup CORBA biblioteci trebali napisati sami, po uzoru kako je ADO napisan za OLE DB.

Možemo zaključiti, da ono što je najvažnije proučiti prilikom razrade modela odnosno posredničkog reda jest sama COM [93][138][139][142] odnosno Distributed COM [21][29][33] tehnologija. Naime svi objekti koji se izvršavaju u sklopu MTS okruženja su COM objekti, uključujući objekte jezgre posredničkog reda, ADO objekte, OLE DB objekte i općenito gotovo sve druge objekte koji grade posrednički red.



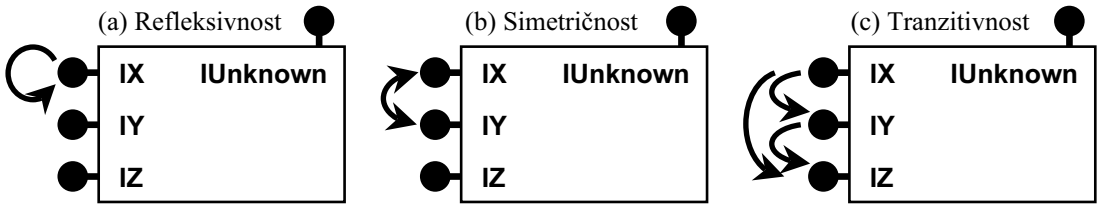
Slika 27: Pregled tehnologija koje se mogu koristiti za izvedbu posredničkog reda.

4.4. COM kao tehnologija implementacije jezgre posredničkog reda

4.4.1. Uvod u tehnologiju COM

Komponenta COM je binarni standard izvršnog koda koji zadovoljava točno određena pravila. Među ostalim to znači da se svako sučelje nasljeđuje od sučelja *IUnknown*, te stoga mora podržavati metode *QueryInterface*, *AddRef* i *Release* koje su uvijek prve tri metode svakog sučelja. Metoda *QueryInterface* služi da se dozna da li neka komponenta podržava i neko drugo sučelje, te da mu se može pristupiti. Metode *AddRef* i *Release* služe za kontroliranje životnog vijeka komponente. Svaki put kad se počne koristiti neko sučelje, poziva se *AddRef* i unutrašnje brojilo komponente se povećava za jedan. Svaki put kad se prestane koristiti neko sučelje, poziva se *Release* i unutrašnje brojilo komponente smanjuje se za jedan. Nakon što vrijednost brojila dođe na 0, komponenta se uništava i time se oslobađaju memorija i ostali resursi koje je koristila. Komponenta se dijeli na sučelje i tijelo koje mora biti inkapsulirano (napisano u nekom jeziku, prevedeno, povezano i pripremljeno za upotrebu). Metoda *QueryInterface* mora zadovoljavati sljedeća svojstva:

- za neko sučelje mora uvijek vraćati jednu te istu kazaljku ukoliko se poziva više puta za redom;
- poziv metode za bilo koje sučelje mora uvijek uspjeti ili nikada ne uspjeti što znači da je skup sučelja koje podržava neka COM komponenta statičan, a ne dinamičan;
- metoda mora biti refleksivna kako je prikazano na Slici 28 (a); primjerice ako se od sučelja *IX* traži da vrati kazaljku na sučelje *IX*, mora se dobiti već postojeća kazaljka;
- metoda mora biti simetrična kako je prikazano na Slici 28 (b); primjerice ako se od sučelja *IX* traži da vrati kazaljku na sučelje *IY* i poziv uspije, onda također mora uspjeti i ako se od sučelja *IY* traži da vrati kazaljku na sučelje *IX*;
- metoda mora biti tranzitivna kako je prikazano na Slici 28 (c); primjerice ako se od sučelja *IX* traži da vrati kazaljku na sučelje *IY* i poziv uspije, te se nakon toga od sučelja *IY* traži da vrati kazaljku na sučelje *IZ* i poziv opet uspije, onda također mora uspjeti i ako se od sučelja *IX* traži da vrati kazaljku na sučelje *IZ*.

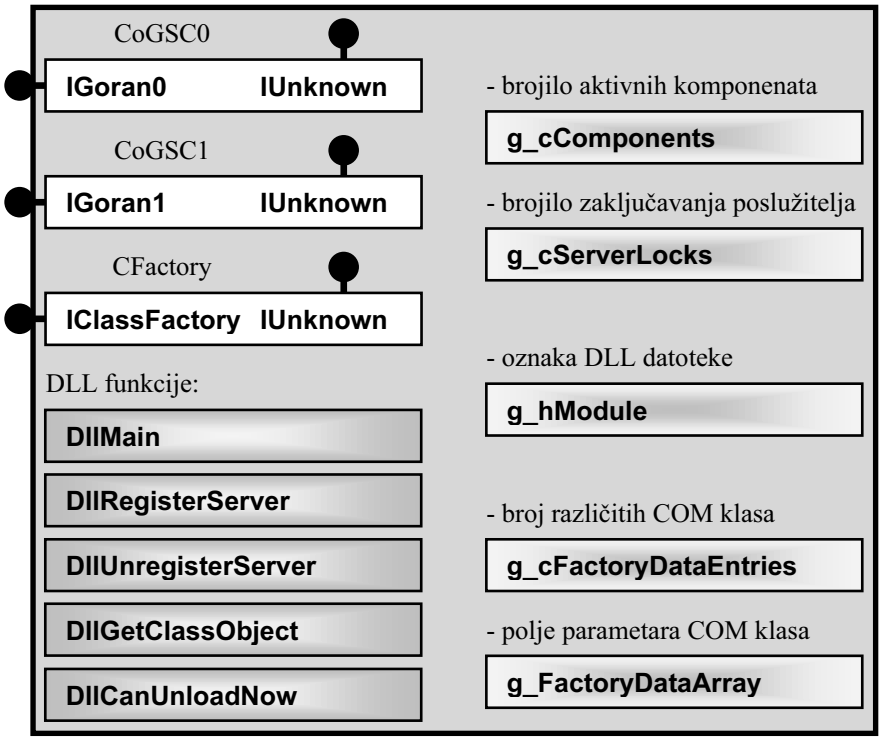


Slika 28: Svojstvo ekvivalencije metode *QueryInterface*.

Za imenovanje sučelja, komponenata i biblioteka koriste se dvojna imena, koja se sastoje od korisnički definiranog imena i UUID-a (Universally Unique IDentifier). Ukoliko bi dva različita proizvođača za ime svoje biblioteke, COM komponente ili nekog sučelja uzeli isto ime, došlo bi do kolizije. Stoga se svakom imenu u datoteci IDL pridružuje i UUID, globalni 128-bitni identifikator, koji jednoznačno određuje dotični objekt. UUID se generira u zavisnosti od mrežne adrese što garantira da su brojevi generirani na različitim računalima različiti, te u zavisnosti od vremena kad se generira što garantira i da su UUID-i generirani na jednom te istom računalu međusobno također različiti. Time se garantira jedinstvenost UUID-ova.

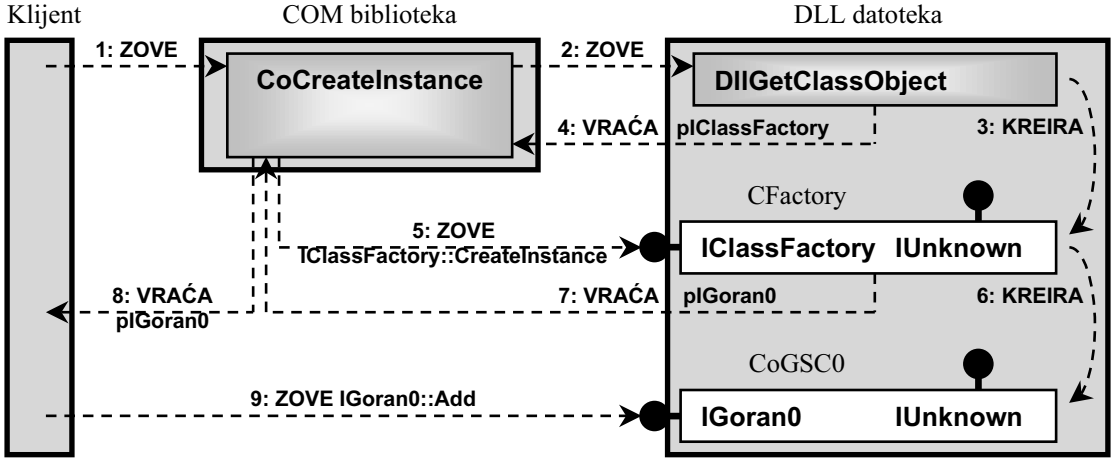
Na Slici 29 prikazani su osnovni elementi datoteke DLL, koja može sadržavati jednu ili više COM klasa. Pri tom pod pojmom klasa nije obuhvaćena puna funkcionalnost u smislu objektno orijentiranog pristupa, što bi trebalo biti podržano tek sa COM+ tehnologijom. *CFactory* dinamički kreira primjerke COM klasa, što su u našem slučaju *CoGSC0* i *CoGSC1*. Sve COM klase izvedene

su iz C++ klase *CUnknown*, kako bi objekt *CFactory* mogao njima polimorfno rukovati. *CFactory* ima vrlo specifičnu arhitekturu u odnosu na druge COM klase. Među ostalim nije izvedena iz *CUnknown*, a uz *IUnknown* podržava još jedino sučelje *IClassFactory*. Također, njen konstruktor i destruktor za razliku od drugih COM klasa ne povećavaju odnosno smanjuju za jedan globalno brojilo *g_cComponents*, koje inače sadrži broj trenutno aktivnih komponenata u memoriji. Metodom *IClassFactory::CreateInstance* korisnik dinamički kreira primjerke COM komponenata, dok metodom *IClassFactory::LockServer* zaključava i otključava DLL povećavajući odnosno smanjujući za jedan globalno brojilo *g_cServerLocks*, čime može spriječiti da se DLL izbriše iz memorije i u trenutku kad vrijednost brojila *g_cComponents* padne na nulu. *DllMain* je glavna funkcija u nekoj DLL datoteci, u našem slučaju njezin jedini zadatak jest da nakon poziva, u globalnu varijablu *g_hModule* spremi oznaku (engl. handle) na DLL datoteku koja je potrebna drugim funkcijama. Funkcije *DllRegisterServer* i *DllUnregisterServer* služe za registraciju odnosno odregistraciju svih COM klasa pripadajućeg DLL-a u Windows registratoru. Funkcija *DllGetClassObject* služi da kreira primjerak klase *CFactory* i inicijalizira je ovisno o tome koju je COM klasu korisnik tražio, te da sučelje *IClassFactory* klase *CFactory* vrati korisniku. Funkciju *DllCanUnloadNow* poziva operacijski sustav kad želi provjeriti da li se DLL datoteka smije izbrisati iz memorije. Globalno polje *g_FactoryDataArray* za svaku pojedinu COM klasu sadrži element tipa *CFactoryData*, koji o istoj sadrže sve specifične informacije potrebne DLL funkcijama i klasi *CFactory*, dok globalna varijabla *g_cFactoryDataEntries* sadrži broj elemenata tog polja odnosno broj COM klasa koje se nalaze u nekom DLL-u. Sama arhitektura DLL datoteke nije specificirana COM standardom, no za prikazanu na Slici 29 je odlučeno da je najprihvatljivija pošto se kao razvojni jezik koristio Visual C++. Inače, COM klase se osim u DLL-u mogu nalaziti i u nekoj EXE datoteci.

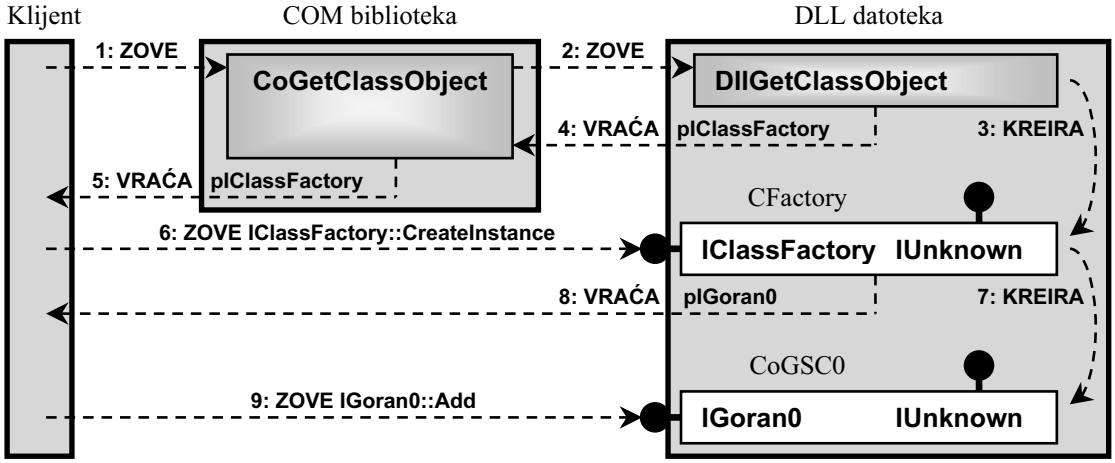


Slika 29: Osnovni elementi datoteke DLL koja sadrži komponente COM.

Prvo što klijent mora učiniti da bi koristio COM komponente jest inicijalizacija COM-a pozivom funkcije *CoInitialize* ili funkcije *CoInitializeEx*, dok je zadnja stvar koju mora učiniti deaktivacija COM-a pozivom funkcije *CoUninitialize*. Neka komponenta se može pozvati i koristiti na dva načina, da se koristi ili funkcija *CoCreateInstance* kako je prikazano na Slici 30, ili da se koristi funkcija *CoGetClassObject* kako je prikazano na Slici 31.



Slika 30: Pozivanje neke COM komponente koristeći *CoCreateInstance* funkciju.



Slika 31: Pozivanje neke COM komponente koristeći *CoGetClassObject* funkciju.

Na Ispisu 1, prikazana je međusobna povezanost funkcija *CoCreateInstance* i *CoGetClassObject*. Ukoliko je sve što nas zanima da dobijemo valjanu referencu na objekt neke COM klase, možemo koristiti jednostavniju funkciju *CoCreateInstance*. U tom slučaju ne možemo direktno pristupati klasi *CFactory*, koja nam je među ostalim potrebna kad želimo zaključati/otključati DLL pozivom metode *IClassFactory::LockServer*. Dodatna prednost korištenja druge funkcije je i kada trebamo kreirati velik broj objekata jedne ili više različitih COM klasa, objekt tipa *CFactory* se u ovom slučaju kreira te oslobađa iz memorije samo jednom, dok bi se u prvom slučaju to dogodilo po jednom za svaki pojedini objekt. Zaključak je da je prednost pristupa prikazanog na Slici 30 jednostavnost (do željene reference dolazimo samo jednim pozivom), dok je prednost pristupa sa Slike 31 veća funkcionalnost koja je ponekad potrebna.

```
HRESULT CoCreateInstance(const REFCLSID& refclsid, IUnknown* pUnknownOuter,
    DWORD dwClsContext, const REFIID& refiid, void** ppv)
{
    IClassFactory* pIClassFactory;
    HRESULT hr = CoGetClassObject(refclsid, dwClsContext, NULL,
        IID_IClassFactory, (void**)&pIClassFactory);
    if(SUCCEEDED(hr))
    {
        hr = pIClassFactory->CreateInstance(pUnknownOuter, refiid, ppv);
        pIClassFactory->Release;
    }
}
```

Ispis 1: Međusobna povezanost funkcija *CoCreateInstance* i *CoGetClassObject*.

4.4.2. Pametne kazaljke na COM sučelja

Dok se nasljeđivanja i višestruka nasljeđivanja koriste kako bi se olakšalo pisanje COM klasa, pametne kazaljke koriste se kako bi se olakšalo pisanje klijenta odnosno kako bi se olakšalo rukovanje COM objektima.

Pametna kazaljka se ponaša vrlo slično običnim kazaljkama, no razlika je u tome što pruža daleko veću funkcionalnost. Ona je ovojnica oko stvarne kazaljke na neko COM sučelje. Osnovno pojednostavljenje sastoji se u tome, što je pametna kazaljka objekt koji u svojem destruktoru automatski poziva metodu *Release*. Zahvaljujući tome ne moramo voditi računa da ispravni broj puta pozovemo tu metodu prilikom završetka rada klijenta, već se dereferenciranje kazaljki na sučelja automatizira. Budući da je pametna kazaljka C++ objekt, možemo iskoristiti njen konstruktor za automatsku inicijalizaciju na vrijednost *NULL*.

Sam rad s pametnom kazaljkom, bitno je jednostavniji nego rad s običnom kazaljkom. Naime ovdje se može iskoristiti svojstvo jezika C++ da podržava dodefiniranje operatora, te se zahvaljujući tome mogu automatizirati neke operacije koje su potrebne pri radu s kazaljkama na neko COM sučelje. Prilikom pisanja COM klijenata u jeziku C++, mogu se javiti neke pogreške koje sadašnji C++ prevodioci ne prepoznaju. Klasični primjer je poziv metoda *CoCreateInstance* i *QueryInterface* gdje referenca na UUID i traženo sučelje moraju biti u korelaciji. Ovi problemi također se uspješno rješavaju u okviru pametnih kazaljki tako, da iste jednostavno podržavaju član metodu *CoCreateInstance* koja uzima samo prva tri argumenta od korisnika, te interno generira četvrti i peti element koji su u korelaciji. Dodatno, dodefinirani operator "=" potpuno eliminira potrebu za *QueryInterface* metodom, budući da istu pametne kazaljke po potrebi same pozivaju.

Korištenjem pametnih kazaljki dobivamo i pregledniji kod. Naime u slučajevima kad se ne koriste pametne kazaljke, sam kod implementacije klijenata ima puno uvlačenja u odnosu na lijevu marginu, što nastaje zbog referentnog brojenja i potrebe da klijent "čisto" izađe i u slučaju da poziv neke metode ne uspije. Kod kompleksnih klijenata, ovo pojednostavljenje itekako puno znači, jer upravo dobra preglednost koda jedna je od najboljih garancija da neće biti puno previđenih grešaka.

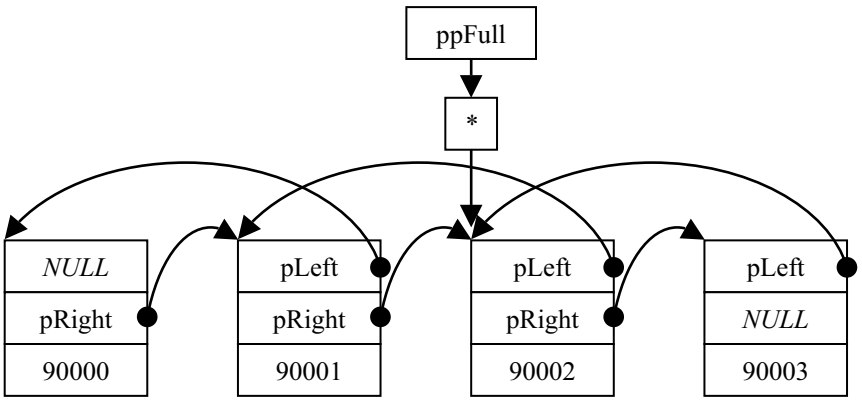
Na Ispisu 2, prikazana je definicija makroa pametnih kazaljki *ISP* i *IUSP*, te kako deklarirati neku varijablu koja je tipa "pametna kazaljka".

```
#define ISP(T) ISPtr<T, &IID_##T>
#define IUSP IUnknownSPtr
//  Koristi:  ISP(IX) spIX;  !!Ne koristi sa IUnknown jer se ISP(IUnknown) neće prevesti!!
//  Koristi:  IUSP spIUnknown;
```

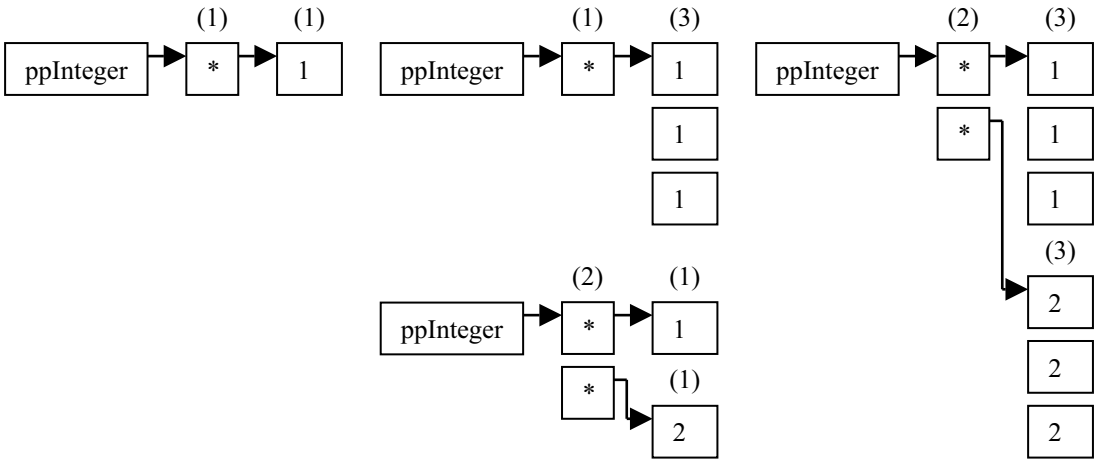
Ispis 2: Definicija makroa pametnih kazaljki *ISP* i *IUSP*, te ilustracija njihovog korištenja.

4.4.3. Biblioteke sučelja, programski jezik IDL i prevodilac MIDL

Kad se pređu granice jednog procesa ili jednog računala, programi više ne mogu direktno komunicirati. Ukoliko se naime dogodi da neki program želi namjerno ili nenamjerno pisati po adresnom području nekog drugog programa/procesa, operacijski sustav ga u tome sprečava. Takva arhitektura gdje je svaki proces zasebno adresno područje uvedena je iz sigurnosnih razloga. Stoga kada želimo prebaciti neke podatke iz jednog procesa u drugi, bio on na istom ili nekom drugom računalu, to ne smijemo i ne možemo raditi direktno, već moramo koristiti pozive koje nam sam operacijski sustav stavlja na raspolaganje. U tom slučaju nije dovoljno prenijeti samo kazaljku, već je potrebno prenijeti i sve podatke odnosno strukturu na koju ista pokazuje, a koja može biti i relativno složena kao što je npr. dvostruko povezana lista prikazana na Slici 32. Dodatni problem je i jednoznačnost definiranja onoga što se prenosi; npr. na Slici 33 prikazane su četiri moguće interpretacije kazaljke na kazaljku cijelog broja. Da bi se razriješili ovi problemi, sučelja COM komponenata opisujemo u jeziku IDL. Njegova primjena nam omogućava da komponente pišemo istovjetno, bez obzira da li se iste koriste u okviru neke nedistribuirane ili distribuirane aplikacija.



Slika 32: Dvostruko povezana lista koju klijent može slati/primati od poslužitelja.



Slika 33: Četiri moguće interpretacije vrijednosti “int**” u C++ jeziku.

Na Ispisu 3 dano je nekoliko primjera opisa sučelja u IDL jeziku. Koristeći prevodilac MIDL (Microsoft IDL), dobijemo datoteke koje zatim uključujemo u naš projekt (*c*, *h* i *tlb* u slučaju da smo kao jezik pisanja komponenata izabrali C++).

```
HRESULT FxStringIn([in, string] wchar_t* szIn);
HRESULT FxStringOut([out, string] wchar_t** szOut);
HRESULT FyCount([out] long* sizeArray);
HRESULT FyArrayIn([in] long size1, [in, size_is(size1)] long array1[]);
HRESULT FyArrayOut([out, in] long* psize2, [out, size_is(*psize2)] long array2 []);
HRESULT FzStructIn([in] Point3d pt);
HRESULT FzStructOut([out] Point3d* pt);
```

Ispis 3: Opis sučelja u IDL jeziku.

Neki tipovi podataka u jeziku C++ kao što su npr. *short*, *int* i *long*, u praksi mogu predstavljati različite zapise kod različitih implementacija i na različitim arhitekturama. To se ne smije dozvoliti ako želimo osigurati univerzalni način komunikacije kojeg će razumjeti raznorodne implementacije na raznorodnim arhitekturama. Stoga IDL jezik kao prvo definira svoje vlastite jedinstvene tipove podataka prikazanih u Tablici 3. To su *boolean*, *byte*, *char*, *double*, *float*, *handle_t*, *hyper*, *int*, *long*, *short*, *small* i *wchar_t*. Tip *wchar_t* prenosi 16-bitne znakove, te se stoga koristi za prijenos UNICODE-a. To konkretno znači da će se među ostalim i naša slova Č, Ć, Đ, Š, Ž, č, ć, đ, š i ž ispravno prenijeti iz jednog procesa u drugi na istom ili udaljenom računalu. Dodatno, jezik IDL koristi format prijenosa podataka mrežom NDR (Network Data Representation) [29], te je time garantirano da će podatke ispravno primiti neka druga arhitektura, bez obzira da li se tamo koristi interni zapis tipa "little endian" ili "big endian".

Tablica 3: Osnovni tipovi podataka u jeziku IDL.

Osnovni tip:	Opis:
boolean	Booleov tip podatka koji može sadržavati vrijednost *TRUE* ili vrijednost *FALSE*.
byte	8-bitni podatak koji se prenosi bez ikakvih izmjena.
char	8-bitni podatak bez predznaka.
double	64-bitni realan broj.
float	32-bitni realan broj.
handle_t	Oznaka koja se koristi za RPC (Remote Procedure Call) povezivanje ili serijalizaciju podatka.
hyper	64-bitni cijeli broj koji se može deklarirati sa ili bez predznaka.
int	32-bitni cijeli broj koji se može deklarirati sa ili bez predznaka.
long	32-bitni cijeli broj koji se može deklarirati sa ili bez predznaka.
short	16-bitni cijeli broj koji se može deklarirati sa ili bez predznaka.
small	8-bitni cijeli broj koji se može deklarirati sa ili bez predznaka.
wchar_t	16-bitni znak, služi za prijenos UNICODE-a.

Za razliku od jezika opće namjene, kao što je npr. C++, u jeziku IDL se mora moći specificirati u kojem se smjeru prebacuju podaci. Za tu svrhu se koriste atributi smjera dati u Tablici 4. Neka metoda može imati više ulaznih parametara smjera *[in]*, više izlaznih parametara smjera *[out]*, više ulazno-izlaznih parametara smjera *[in, out]*, no samo jedan smjera *[out, retval]*. Da bi bio moguć pristup COM komponentama i iz viših jezika kao što je skriptni jezik unutar HTML-a, dodan je atribut *retval* koji označava koji argument će se automatski proslijediti u tom slučaju. Inače, sve funkcije direktno vraćaju *HRESULT* koji sadržava kod pogreške koja se dogodila prilikom poziva nekog sučelja, ili vrijednost *S_OK* ukoliko je poziv prošao uspješno.

Tablica 4: Atributi smjera jezika IDL.

Atribut smjera:	Opis:
[in]	Podaci se prenose od klijenta ka poslužitelju, to je jednosmjeran prijenos.
[out]	Podaci se prenose od poslužitelja ka klijentu, to je jednosmjeran prijenos.
[in, out]	Podaci se prenose prvo od klijenta ka poslužitelju, a zatim od poslužitelja ka klijentu, to je dvosmjeran prijenos.
[out, retval]	Podaci se prenose od poslužitelja ka klijentu, to je jednosmjeran prijenos, no uz to ovaj argument uz *HRESULT* predstavlja i povratnu vrijednost funkcije.

IDL atributi polja podataka dodani su kako bi se mogla specificirati stvarna veličina nekog polja koje se šalje mrežom. To je potrebno specificirati u jeziku IDL, kako bi se npr. nakon što se nekom sučelju proslijedi kazaljka na neko polje podataka, točno znalo koliko podataka ima u tom polju i koliko ih se mora prenijeti mrežom. Atributi polja podataka dati su u Tablici 5. Atributi *max_is*, *min_is* i *size_is* određuju koliko prostora će biti dodijeljeno na strani klijenta ili poslužitelja te se stoga te vrijednosti ne smiju prekoračiti. Pošto se uvijek ne treba prenijeti cijelo dodijeljeno polje nego samo jedan njegov dio, da bi se odredilo koji je to dio služe atributi *first_is*, *last_is* i *length_is*. Uvijek se ne koriste svi atributi, nego samo oni koji su nam potrebni. Tako nam recimo ne treba atribut *min_is* (koji je dat isključivo radi cjelovitosti), a atribut *max_is* je uvijek za jedan manji od *size_is* tako da koristimo ili jednog ili drugog. Isto tako, ako koristimo atribute *first_is* i *last_is*, ne treba nam i atribut *length_is* jer uvijek je *length_is = last_is - first_is + 1*.

Tablica 5: Atributi polja podataka jezika IDL.

Atribut polja podataka:	Opis:
max_is	Najveći dozvoljeni indeks polja.
min_is	Najmanji dozvoljeni indeks polja (uvijek 0).
size_is	Najveći dozvoljeni broj elemenata polja kojeg treba prenijeti.
first_is	Indeks prvog elementa kojeg treba prenijeti.
last_is	Indeks zadnjeg elementa kojeg treba prenijeti.
length_is	Ukupan broj elemenata polja kojeg treba prenijeti.

Tablica 6: Atributi kazaljki jezika IDL.

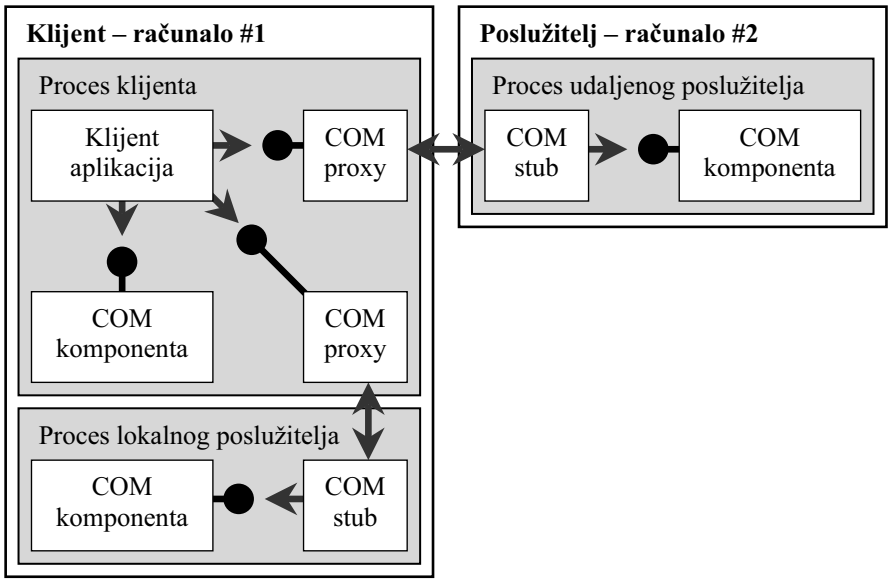
Atribut kazaljki:	Svojstva:
ref	- mogu pokazivati na bilo koju memorijsku lokaciju osim na *NULL* - ne smije promijeniti vrijednost za vrijeme poziva - ne smije više kazaljki pokazivati na istu memorijsku lokaciju
unique	- mogu pokazivati na sve memorijske lokacije pa i na *NULL* - smiju promijeniti vrijednost za vrijeme poziva i to čak sa ne *NULL* vrijednosti u *NULL* i obrnuto (ta promjena se ignorira sa strane IDL jezika) - ne smije više kazaljki pokazivati na istu memorijsku lokaciju
ptr	- dodatno smije više kazaljki pokazivati na istu memorijsku lokaciju

Atributi kazaljka jezika IDL služe da bi mogli specificirati kakvu kazaljku iz jezika C++ prenosimo, da bi se mogao optimirati sam kod za prenošenje. Moguće vrijednosti su date u Tablici 6. Kazaljke tipa *ref* su podskup kazaljka tipa *unique*, a kazaljke tipa *unique* podskup kazaljka tipa *ptr*. Dodatno ograničenje koje moraju poštovati sve kazaljke jest da se ne smiju prekoračiti atributi polja podataka dati u Tablici 5. Možemo zaključiti da se kazaljke tipa *ref* koriste isključivo kao reference, imamo najviše ograničenja, no stoga je prebacivanje podataka na koje iste pokazuju brzo i jednostavno. S druge strane kazaljke tipa *ptr* su najsličnije kazaljkama u jeziku C++, gdje gotovo da nemamo ograničenja, no prebacivanje je složeno budući da kod za prebacivanje mora interno raditi kazalo kazaljki tako da iste podatke ne prebacuje višestruko odnosno da ne uđe u mrtvu petlju.

Biblioteke sučelja komponenata COM su binarni zapisi njihovih sučelja koja su izvorno obično dana u jeziku IDL. Osnovna primjena biblioteka sučelja je u jednostavnijem pristupu COM komponentama iz viših programskih jezika, i u jednostavnijem razvoju distribuiranih aplikacija. Prvotno se nekoj COM komponenti moglo pristupati iz nekog višeg programskog jezika, jedino ukoliko je ta komponenta imala implementirano *IDispatch* sučelje. Koristeći biblioteke sučelja, danas se vrlo jednostavno pristupa bilo kojem sučelju bilo koje COM komponente, kako iz Visual Basica, tako i iz Jave. Budući da je u nekoj biblioteci sučelja dat binarni oblik svih sučelja naših komponenata, iste možemo jednostavno prebaciti na bilo koje računalo u mreži, a koristeći DCOM, operacijski sustav će ih sam automatski povezati s pripadnim klijentom/klijentima. Ukoliko za jezik implementacije izaberemo C++, sučelja ionako opisujemo u *IDL* datoteci tako da je najjednostavniji način da se generira biblioteka sučelja da se koristi prevodilac MIDL, od koga jednostavno zatražimo da nam među ostalim izgenerira i nju (*TLB* datoteka). Ovdje je važno napomenuti da se u samoj *IDL* datoteci umjesto *pointer_default(unique)* treba kod svih sučelja upisati zastavica *oleautomation*. Ukoliko se to ne učini, biblioteka sučelja će se ispravno izgenerirati, ali ne i ispravno registrirati u Windows registratoru, te istu nećemo moći koristiti.

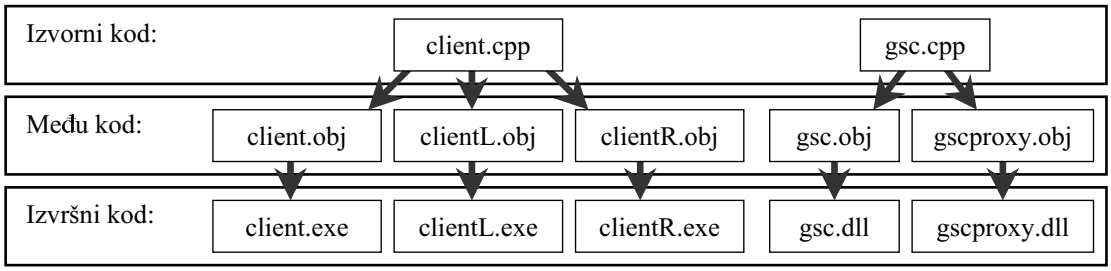
4.4.4. Pozivi između različitih procesa i različitih računala

Kako je prikazano na Slici 34, komponenta se može nalaziti unutar istog procesa kao i aplikacija koja je koristi, unutar nekog drugog procesa na istom računalu, ili unutar nekog drugog procesa na udaljenom računalu. Osnovni princip je da klijent ne vidi razliku između COM proxy-a (spojni modul na strani klijenta) i same COM komponente, dok komponenta ne vidi razliku između COM stub-a (spojni modul na strani komponente) i samog klijenta.



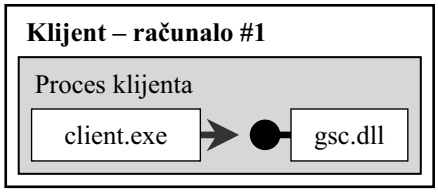
Slika 34: Tri moguća načina pozivanja komponenata od strane korisnika.

COM stub se ne mora zasebno pisati, već se može koristiti standardni nadomjestni program *dllhost.exe* koji se isporučuje zajedno s operacijskim sustavom. Zahvaljujući tome što je biblioteka sučelja integrirana u *gsc.dll* datoteku gdje se nalazi i sama komponenta odnosno više njih, standardni nadomjestni program *dllhost.exe* im zna pristupati. Kako je prikazano na Slici 35, COM proxy također se ne mora zasebno implementirati, već se taj proces može automatizirati, kao što je također i klijenta potrebno opisati samo jednom, dok se automatski generira verzija koja komponentu poziva u vlastiti proces, drugi proces na istom računalu ili drugi proces na udaljenom računalu.

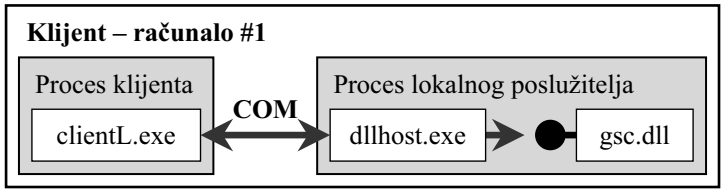


Slika 35: Postupak automatiziranog generiranja izvršnog koda.

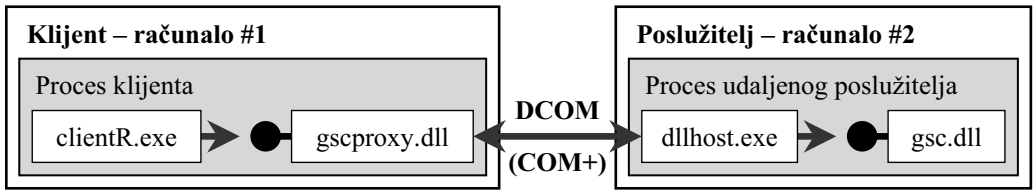
Na slikama 36, 37 i 38, prikazano je kojim izvršnim datotekama sa Slike 35 zamjenjujemo elemente principijelne sheme sa Slike 34. Kao što se može primijetiti, svaka *EXE* datoteka izvršava se u zasebnom procesu, dok se *DLL* datoteke učitavaju u procese u kojem se već izvodi neki *EXE*. Dodatno se može primijetiti da nam datoteka *gscproxy.dll* nije potrebna, zahvaljujući tome što je biblioteka sučelja integrirana u samu *gsc.dll* datoteku, ne samo u slučaju kad se klijent i komponenta nalaze unutar istog procesa, već i kad se nalaze unutar različitih procesa istog računala.



Slika 36: Klijent i poslužitelj nalaze se unutar istog procesa.



Slika 37: Klijent i poslužitelj nalaze se unutar različitih procesa na istom računalu.



Slika 38: Klijent i poslužitelj nalaze se unutar različitih procesa na različitim računalima.

Što se tiče terminologije, prvo se koristila riječ *COM*, a nakon što je podržan poziv između različitih računala, *Distributed COM* odnosno *DCOM*, kako bi se naglasila ta nova mogućnost. No vremenom, "Distributed" se počelo ispuštati tako da se i pod samim COM podrazumijevalo da je podržan i poziv između različitih računala (kako je prikazano na Slici 38). Ove različitosti u terminologiji razriješene su najnovijom verzijom COM standarda – *COM+*, gdje je odlučeno da se "Distributed" odnosno "D" više neće navoditi, iako sam COM+ naravno podržava pozive između različitih računala, što se onda i implicitno podrazumijeva.

4.4.5. Višedretveno programiranje u COM-u

COM podržava višedretveno programiranje, a dobro poznavanje procesa i dretvi posebno je važno prilikom implementacije distribuiranih aplikacija. Svaki proces može se sastojati od jedne ili više dretvi, a postoje 4 vrste:

- dretva korisničkog sučelja (engl. user-interface thread);
- radna dretva (engl. worker thread);
- apartman dretva (engl. apartment thread) (komponenta);
- slobodna dretva (engl. free thread) (komponenta).

Dretva korisničkog sučelja sastavni je dio bilo kojeg grafičkog sučelja prema korisniku i njen zadatak je dinamičko osvježavanje istog. Unutar nekog procesa, zajedno sa dretvom korisničkog sučelja može biti i jedna ili više radnih dretvi, koje istovremeno mogu izvršavati neke dodatne operacije. Kod modeliranja komponenata, imamo apartman dretve i slobodne dretve. Komponente mogu biti građene tako da podržavaju prve, druge, obje ili nijednu. To svojstvo neke komponente zapisano je u Windows registratoru. Razlika je u tome što je jednostavnije napisati komponentu koja podržava apartman dretvu, pošto se oko prebacivanja podataka i sinkronizacije automatski brine COM, no mogućnosti takve komponente su manje u odnosu na one koje podržavaju slobodne dretve. Tu se neke stvari mogu/trebaju ručno napraviti, ali su zato mogućnosti komponente veće, npr. različitim sučeljima komponente se može istovremeno pristupati iz različitih radnih dretvi. Samim tim postiže se i veća brzina rada koja u nekim slučajevima može biti važna. Kod procesa i dretvi, najvažniji pojmovi su:

- prebacivanje podataka (engl. marshaling);
- sinkronizacija (engl. synchronization).

Pod prebacivanjem podataka podrazumijeva se da se ne prenosi samo kazaljka već i sami podaci na koju ista pokazuje. Pod sinkronizacijom podrazumijeva se sprječavanje neželjenog istovremenog pristupa dijeljenim podacima. Kao što prebacujemo podatke ukoliko pozivamo komponentu koja se nalazi u drugom procesu ili na drugom računalu, tako zbog dodatnih zahtijeva koje postavlja višedretvno programiranje to ponekad moramo činiti i ukoliko se nalaze unutar istog procesa. Pri tome generalno vrijede sljedeća pravila:

- pozivi između procesa uvijek zahtijevaju prebacivanje podataka;
- pozivi unutar iste dretve nikad ne zahtijevaju prebacivanje podataka;
- pozivi komponente unutar apartman dretve uvijek zahtijevaju prebacivanje podataka;
- pozivi komponente unutar slobodne dretve ponekad zahtijevaju prebacivanje podataka;
- pozivi komponente unutar apartman dretve su uvijek sinkronizirani;
- pozivi komponente unutar slobodne dretve nisu sinkronizirani;
- pozivi unutar iste dretve su sinkronizirani od same dretve.

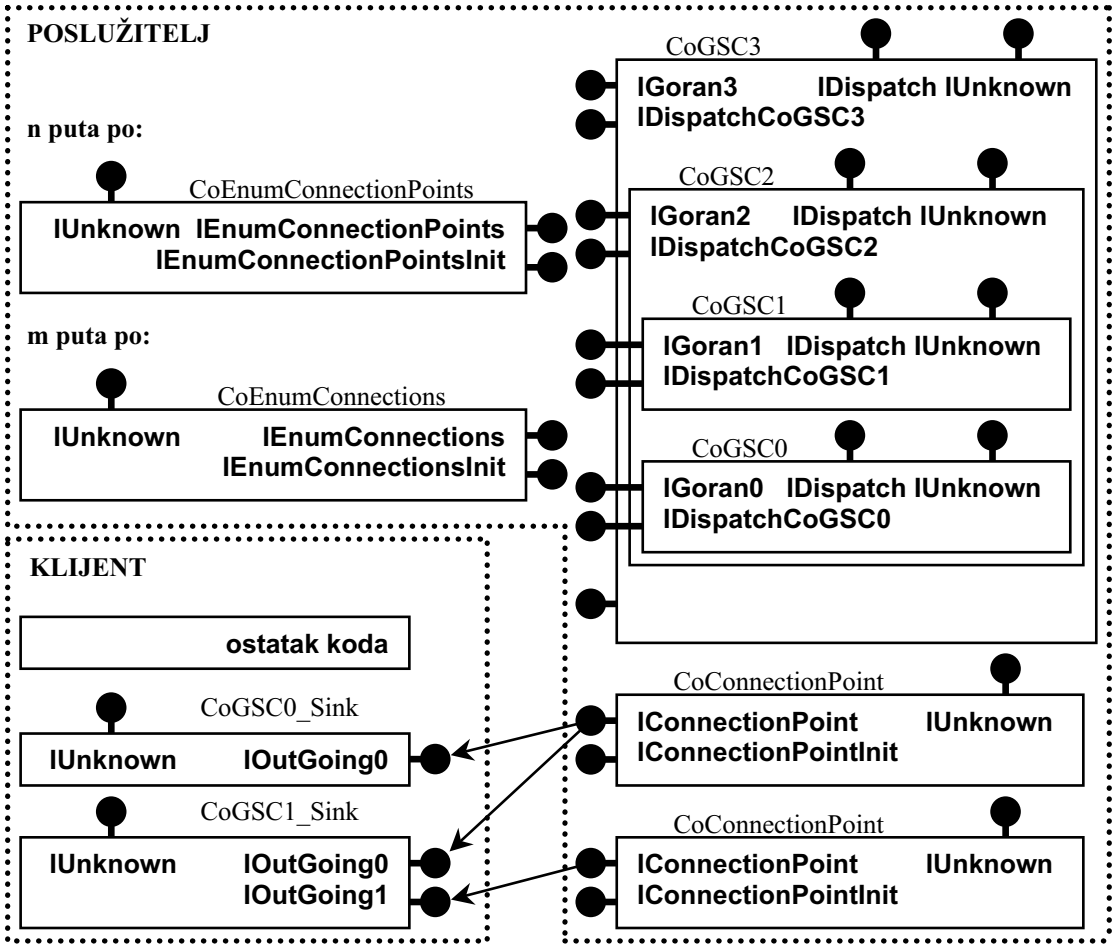
Prebacivanje podataka potrebno je kada prelazimo granice procesa ili kada se komponenta nalazi unutar apartman dretve, a pristupa joj netko izvan te dretve. Sinkronizacija je potrebna kada se komponenta nalazi unutar slobodne dretve, a pristupa joj netko izvan te dretve. Pošto se komponenti može pristupati u ovom slučaju istovremeno iz više različitih dretvi, ona sama mora osigurati sinkronizaciju. Pristup dijeljenim varijablama je jedan primjer gdje je potrebno o tome voditi računa, npr. ukoliko jedna metoda nekog sučelja povećava istu, a druga smanjuje. U našem slučaju to su varijabla koja označava da li je *DLL* datoteka zaštićena od iščitavanja iz memorije *g_cServerLocks*, ukupan broj aktivnih komponenata *DLL* datoteke *g_cComponents*, i referentno brojilo *m_cRef* koga ima kako klasa *CFactory* tako i svaka komponenta derivirana iz klase

CUnknown. Ono na što treba obratiti pažnju je povećavanje za jedan i smanjivanje za jedan tog brojila budući da za to može trebati više od jedne naredbe u strojnom kodu (dotična vrijednost se prvo mora pročitati, zatim izmijeniti te nakon toga ponovo zapisati u memoriju). Stoga pristup toj varijabli mora biti zaštićen. U tu svrhu, jednostavno se koriste funkcije *InterlockedIncrement* i *InterlockedDecrement* koje nam garantiraju da se varijabli neće istovremeno pristupati iz neke druge dretve za vrijeme dok mijenjamo njeno stanje.

Na kraju je još potrebno i u Windows registratoru upisati za svaku komponentu koje dretve podržava. Komponente iz ovog rada podržavaju i apartman dretve i slobodne dretve, odnosno potrebno je upisati na odgovarajuće mjesto *"Both"*. Kako bi se ti podaci automatski unijeli u Windows registrator prilikom registracije *gsc.dll* datoteke, potrebno je datoteci *gsc.*cpp u polju *g_FactoryDataArray* tipa *CfactoryData* za svaku komponentu upisati *"Both"*. U slučaju da neka komponenta podržava samo slobodne dretve upisali bi *"Free"*, u slučaju da podržava samo apartman dretve upisali bi *"Apartment"*, dok bi u slučaju da ne podržava niti jedne niti druge upisali *"NULL"*.

4.4.6. Pozivi unazad i enumeratori

Ukoliko iz komponente sa strane poslužitelja želimo pozivati komponente sa strane klijenta (poziv unazad), kako je prikazano na Slici 39, možemo koristiti spojne točke. Na taj način, komponenta poslužitelja može obavijestiti komponentu klijenta da se dogodila neka promjena, ili može zatražiti od nje da učini nešto što ona sama ne može, jer se npr. nalazi na drugom računalu. Enumeratori služe za rukovanje velikim brojem istovjetnih objekata, kao npr. u slučaju da imamo veliki broj spojnih točaka preko kojih su povezani komponente klijenta i poslužitelja, odnosno za rukovanje vezama koje su preko svake pojedine spojne točke ostvarene. Poslužitelj može imati više spojnih točaka, klijent može imati više komponenata na koje se iste spajaju (odvodne komponente), pojedina odvodna komponenta može imati jedno ili više izlaznih sučelja, preko jedne spojne točke poslužitelj se može spojiti na jednu ili više odvodnih komponenata, a na jednu odvodnu komponentu može se spojiti s jednom ili sa više spojnih točaka.



Slika 39: Poslužitelj poziva metode klijenta koristeći spojne točke.

Najjednostavniji oblik poziva unazad jest da klijent od poslužitelja prvo traži kazaljku na *IConnectionPointContainer* sučelje, nakon toga od tog sučelja za svako svoje izlazno sučelje traži spojnu točku, i nakon toga sa svakom spojnom točkom uspostavlja jednu ili više veza. Na kraju komunikacije, raskida veze i oslobađa sučelja. Rukovanje većim brojem spojnih točaka možemo olakšati koristeći *CoEnumConnectionPoints* komponentu, do koje možemo doći preko *IConnectionPointContainer* sučelja. Analogno spojnim točkama, rukovanje većim brojem veza možemo olakšati koristeći *CoEnumConnections* komponentu, do koje možemo doći preko *IConnectionPoint* sučelja koje ima svaka spojna točka.

Budući da svaka spojna točka sadrži standardno *IConnectionPoint* sučelje preko kojeg joj se pristupa, spojne točke ne mogu biti agregirane od strane u našem slučaju *CoGSC3* komponente sa Slike 39, jer jedna COM komponenta ne može imati više istih sučelja (u našem slučaju bi *CoGSC3* imala dva puta implementirano *IConnectionPoint* sučelje). Ukoliko bi ipak agregirali spojne točke, pristup bi funkcionirao, no samo dok se poslužitelj i komponenta nalaze u istom procesu, jer se tada kazaljke prenose direktno. U slučaju kad se nalaze u dva različita procesa, ili na dva različita računala, pristup sa agregacijom ne bi funkcionirao. Stoga se same spojne točke izvode kao zasebne COM klase. Uz sučelje *IConnectionPoint* one imaju i sučelje *IConnectionPointInit* preko kojeg se iste i inicijaliziraju. Enumeratori se izvode kao zasebne COM klase iz istog razloga zbog kojeg i spojne točke. U ovom primjeru, imamo enumerator spojnih točaka *CoEnumConnectionPoints*, i enumerator veza *CoEnumConnections*. Komponenta *CoEnumConnectionPoints* podržava standardno sučelje *IEnumConnectionPoints* preko kojeg se pristupa enumeratoru spojnih točaka, te sučelje *IEnumConnectionPointsInit* preko kojeg se dotični enumerator inicijalizira. Analogno tome, komponenta *CoEnumConnections* podržava standardno sučelje *IEnumConnections* preko kojeg se pristupa enumeratoru veza, te sučelje *IEnumConnectionsInit* preko kojeg se isti inicijalizira.

Implementirati spremnik spojnih točaka u okviru neke COM komponente ustvari znači da ta komponenta mora imati implementirano *IConnectionPointContainer* sučelje. Opis metoda tog sučelja date su u sljedećoj tablici.

Tablica 7: Metode IConnectionPointContainer sučelja.

Metode:	Opis:
QueryInterface, AddRef i Release	Naslijeđeno od *IUnknown* sučelja.
EnumConnectionPoints	Ova metoda kreira novu *CoEnumConnectionPoints* komponentu, inicijalizira je, i vraća kazaljku na kazaljku na njeno *IEnumConnectionPoints* sučelje.
FindConnectionPoint	Ova metoda za svako izlazno sučelje pronalazi odgovarajuću spojnu točku, te vraća kazaljku na kazaljku na njeno *IConnectionPoint* sučelje.

Spojne točke, zasebne su COM komponente koje moraju imati implementirano sučelje *IConnectionPoint*, u našem slučaju dodatno imaju implementirano i sučelje *IConnectionPointInit*. Opis metoda tih sučelja date su u sljedeće dvije tablice. Član varijabla *m_pIID* sadrži kazaljku na IID od izlaznog sučelja na koji se spojna točka spaja. Na taj način imamo fleksibilnu arhitekturu koja nam omogućava da spojnu točku možemo spojiti na bilo koje izlazno sučelje. Član varijabla *m_cTotal* sadrži najveći mogući broj veza, a *m_pCONNECTDATA* je kazaljka na polje elemenata tipa *CONNECTDATA* koje sadrži popis svih uspostavljenih veza i njihovih identifikatora. Član varijabla *m_pIConnectionPointContainer* sadrži kazaljku na *IConnectionPointContainer* sučelje spremnika unutar kojeg se nalazi.

Tablica 8: Metode IConnectionPoint sučelja.

Metode:	Opis:
QueryInterface, AddRef i Release	Naslijeđeno od *IUnknown* sučelja.
GetConnectionInterface	Ova metoda vraća kazaljku na IID od sučelja na koje je dotična spojna točka spojena ili inicijalizirana da se spoji.
GetConnectionPointContainer	Ova metoda vraća kazaljku na kazaljku na sučelje *IConnectionPointContainer* spremnika u kojem se nalazi.
Advise	Ova metoda služi za uspostavu veze. Prosljeđuje joj se

	kazaljka na *IUnknown* sučelje odvodne komponente, a metoda vraća identifikator veze.
Unadvise	Ova metoda služi za raskidanje veze. Prosljeđuje joj se identifikator veze.
EnumConnections	Ova metoda kreira novu *CoEnumConnections* komponentu, inicijalizira je, i vraća kazaljku na kazaljku na njeno *IEnumConnections* sučelje.

Tablica 9: Metode IConnectionPointInit sučelja.

Metode:	Opis:
QueryInterface, AddRef i Release	Naslijeđeno od *IUnknown* sučelja.
Initialize	Ovom metodom, inicijaliziraju se član varijable *CoConnectionPoint* komponente i to: *m_pIID*, *m_cTotal*, *m_pCONNECTDATA* i *m_pIConnectionPointContainer*.

Svi enumeratori su međusobno slični, što je posljedica toga da su svi oni građeni kako bi olakšali rukovanje velikim brojem istovjetnih elemenata. Njihovo *IEnum*... sučelje, što je u našem slučaju sučelje *IEnumConnectionPoints* odnosno sučelje *IEnumConnections*, podržava uz metode *IUnknown* sučelja četiri dodatne metode kako je prikazano sljedećom tablicom: *Next*, *Skip*, *Reset* i *Clone*. I član varijable svih enumeratora su međusobno također vrlo slične. Svaki enumerator ima varijablu *m_cCur* koja pokazuje trenutni element na kojeg enumerator pokazuje, te varijablu *m_cTotal* koja sadrži maksimalan broj elemenata kojim enumerator može rukovati. Enumeratori također imaju i kazaljku na polje elemenata koje enumeriraju, što su u našem slučaju elementi tipa *IConnectionPoint* odnosno *CONNECTDATA*. Sučelja kojim se inicijaliziraju enumeratori, *IEnumConnectionPointsInit* odnosno *IEnumConnectionsInit*, inicijaliziraju iste pozivom metode *Initialize* analogno kao i u slučaju spojnih točaka.

Tablica 10: Metode sučelja IEnumConnectionPoints i IEnumConnections.

Metode:	Opis:
QueryInterface, AddRef i Release	Naslijeđeno od *IUnknown* sučelja.
Next	Ovom metodom od enumeratora se traži popis od sljedećih *cConnections* spojnih točaka odnosno veza. Enumerator vraća kazaljku na broj pronađenih elemenata popisa *lpcFetched*, te same elemente.
Skip	Ovom metodom od enumeratora se traži da preskoči sljedećih *cConnections* elemenata.
Reset	Ovom metodom, enumerator se postavlja u inicijalno stanje.
Clone	Ovom metodom, od enumeratora se traži da sam sebe klonira u identičnog enumeratora, i da nam vrati kazaljku na kazaljku na njegovo *IEnumConnectionPoints* odnosno *IEnumConnections* sučelje.

4.4.7. Monikori

Osnovna zadaća monikora je da olakšaju pronalaženje, pokretanje i inicijalizaciju COM komponenata. Koristeći *CFactory::CreateInstance* metodu koja služi za standardno učitavanje u memoriju COM komponenata, ne možemo iste odmah i inicijalizirati. U tu svrhu možemo za svaku COM komponentu koja zahtijeva inicijalizaciju kreirati vlastito sučelje za pokretanje iste, i dodati ga klasi *CFactory*. Monikori nam dodatno služe kao standardni način da se pozivaju ta sučelja dodana klasi *CFactory*, kako iz jezika C++ tako i iz drugih jezika. Po potrebi, može se na standardan način u COM-u implementirati i vlastiti monikor, kao COM komponenta koja podržava *IMoniker* sučelje. U sljedećoj tablici, dat je pregled monikora podržanih od strane operacijskog sustava. Kao i svaku drugu COM komponentu, tako i svakog monikora možemo kreirati koristeći standardnu COM funkciju *CoCreateInstance*. No budući da se monikori u pravilu koriste za kreiranje neke komponente koja zahtijeva inicijalizaciju, prvo je potrebno sam monikor inicijalizirati tim parametrima. U tu svrhu, za svaki monikor imamo i posebnu funkciju koja nam olakšava kreiranje istog. U slučaju *Class* monikora, to je *CLSID* (CLaSs IDentifier) komponente koju ćemo njime kreirati.

Tablica 11: Monikori operacijskog sustava.

Monikor:	Funkcija kojom se kreira:	Opis:
File Moniker	*CreateFileMoniker*	Služi kao ovojnica putanje do neke datoteke.
Item Moniker	*CreateItemMoniker*	Služi da se identificira objekt sadržan u nekom drugom objektu.
Pointer Moniker	*CreatePointerMoniker*	Služi da identificira objekt koji može postojati samo u aktivnom stanju (stanju izvođenja).
Anti-Moniker	*CreateAntiMoniker*	Inverz nekog drugog monikora.
Composite Moniker	*CreateCompositeMoniker*	Ovaj monikor sastavljen je od više monikora.
Class Moniker	*CreateClassMoniker*	Služi kao ovojnica oko *CLSID*-a svake COM komponente odnosno klase.
URL Moniker	*CreateURLMoniker*	Ovaj monikor služi za rukovanje nekom URL adresom.

Pregled i opis metoda *IMoniker* sučelja, za slučaj da želimo implementirati vlastiti monikor npr. *CoMyMoniker*, dat je u Tablici 12. Vlastiti monikor može se napisati kako bi se jednostavnije kreirale komponente koje zahtijevaju inicijalizaciju, kao npr. *CoConnectionPoint*, *CoEnumConnectionPoints* i *CoEnumConnections*. Budući da je *IMoniker* sučelje izderivirano iz sučelja *IPersistStream* koje je pak izderivirano iz sučelja *IPersist*, potrebno je implementirati i ta dva sučelja. Dodatno, imamo i sučelje *IMonikerInit* koje nam služi za samu inicijalizaciju monikora. Kao što se iz tablice vidi, monikor može imati implementiranu samo metodu *BindToObject*, te u tom slučaju ostale koje se ne koriste vraćaju *E_NOTIMPL*. Prilikom implementacije bilo kojeg monikora, dodatno je potrebno da klasa *CFactory* podržava sučelje *IParseDisplayName*, te pripadnu metodu *ParseDisplayName*.

Tablica 12: Metode IMoniker sučelja (za slučaj monikora CoMyMoniker).

Metode:	Opis:
QueryInterface, AddRef i Release	Naslijeđeno od *IUnknown* sučelja.
GetClassID	Naslijeđeno od *IPersist* sučelja. (*E_NOTIMPL*)

IsDirty	Naslijeđeno od *IPersistStream* sučelja. (*E_NOTIMPL*)
Load	Naslijeđeno od *IPersistStream* sučelja. (*E_NOTIMPL*)
Save	Naslijeđeno od *IPersistStream* sučelja. (*E_NOTIMPL*)
GetSizeMax	Naslijeđeno od *IPersistStream* sučelja. (*E_NOTIMPL*)
BindToObject	Ova metoda služi za pokretanje, inicijalizaciju i spajanje na traženi objekt. U našem slučaju, vraća nam kazaljku na *ImyClassFactory0* sučelje kojim možemo u jednom koraku kreirati i inicijalizirati traženu COM komponentu.
BindToStorage	(*E_NOTIMPL*)
Reduce	(*E_NOTIMPL*)
ComposeWith	(*E_NOTIMPL*)
Enum	(*E_NOTIMPL*)
IsEqual	(*E_NOTIMPL*)
Hash	(*E_NOTIMPL*)
IsRunning	(*E_NOTIMPL*)
GetTimeOfLastChange	(*E_NOTIMPL*)
Inverse	(*E_NOTIMPL*)
CommonPrefixWith	(*E_NOTIMPL*)
RelativePathTo	(*E_NOTIMPL*)
GetDisplayName	(*E_NOTIMPL*)
ParseDisplayName	(*E_NOTIMPL*)
IsSystemMoniker	(*E_NOTIMPL*)

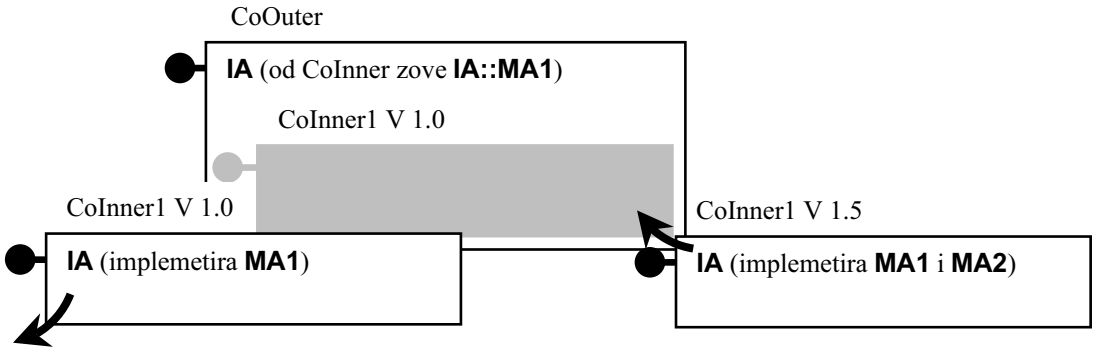
Koristeći funkciju *MkParseDisplayName*, monikor *CoMyMoniker* se može pozvati na isti način kao i svaki drugi monikor. Toj funkciji, jednostavno se proslijedi string L"CoMyMoniker:ConnectionPoint", L"CoMyMoniker:EnumConnectionPoints" ili L"CoMyMoniker:EnumConnections", već ovisno o tome koju COM komponentu želimo kreirati. Funkcija nam vraća *IMoniker* sučelje. Nakon pozivanja pripadne *BindToObject* metode, imamo kazaljku na *IMyClassFactory0* sučelje. Koristeći metode iste i to *CreateConnectionPoint*, *CreateEnumConnectionPoints* i *CreateEnumConnections*, u samo jednom koraku možemo i kreirati i inicijalizirati pripadnu komponentu.

Monikor se može kombinirati i sa drugim monikorima. U tom slučaju se sa znakom L"!" odvajaju dijelovi stringa vezani za različite monikore. Tako recimo možemo imati npr. L"CoNekiDrugiMoniker:NjegoviParametri!CoMyMoniker:ConnectionPoint", pa će monikor *CoNekiDrugiMoniker* biti prvi pozvan, a on će nakon toga pozvati monikora *CoMyMoniker* i proslijediti mu dugi dio stringa. Veliko slovo *L* označava da se radi o UNICODE stringu gdje je svaki znak kodiran sa 16 bita.

4.4.8. Proširivanje mehanizma agregacije COM klasa

Ponovna iskoristivost nasljeđivanjem odnosno višestrukim nasljeđivanjem nije moguća kod COM klasa, budući da je COM binaran standard te različite COM klase mogu biti pisane u različitim jezicima, od kojih neki i ne moraju biti objektno orijentirani. Stoga su razvijeni sadržavanje (delegiranje) i agregacija, kako bi se omogućila ponovna iskoristivost već napisanih COM klasa. No ti mehanizmi imaju dosta nedostataka odnosno postavljaju mnoga ograničenja, te se stoga u dosta slučajeva ne mogu koristiti, kao npr. kad više od jedne agregirane COM klase ima implementirano jedno od najpoznatijih COM sučelja – *IDispatch*. To standardno COM sučelje je sučelje između COM klase (koja može biti napisana u bilo kojem jeziku) i bilo kojeg drugog jezika odnosno skriptnog jezika iz kojeg se korisnički definirana COM sučelja ne mogu direktno pozivati. Najčešće korištenje sučelja *IDispatch* je kad se iz ASP stranica pozivaju COM klase prilikom procesa generiranja HTML/DHTML stranica. U dostupnoj literaturi vezanoj za agregaciju u COM-u [13][29][44][107][130], kao niti u literaturi vezanoj za samo *IDispatch* sučelje [24][131][149] te vezanoj za COM+ [98][147], nema zadovoljavajućih odgovora na ova pitanja.

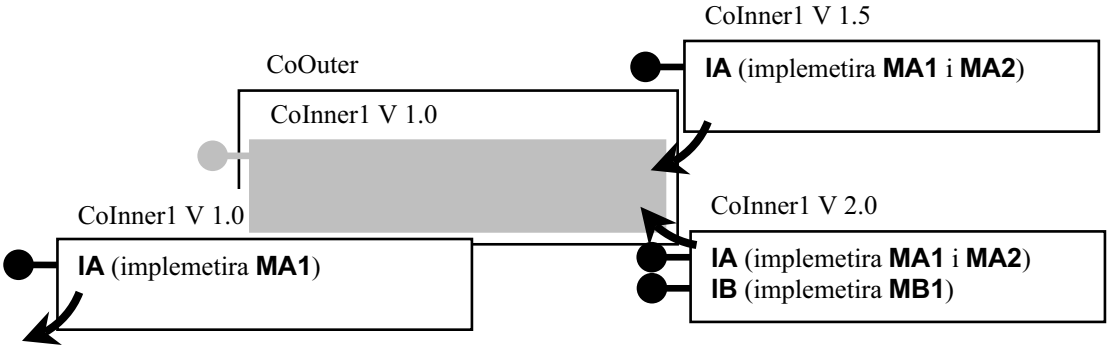
Osnovni princip sadržavanja je u tome da jedna COM klasa (kontejner) u potpunosti sadržava drugu. Prvi nedostatak ovog pristupa je što kontejner mora ručno delegirati sve pozive prema sadržanoj klasi odnosno sadržanim klasama, što utječe na performanse sustava u cjelini. Još važniji nedostatak očituje se kad sadržanu klasu želimo zamijeniti novijom verzijom, koja sadrži ista sučelja, ali zato jedno ili više sučelja podržava nove metode kako je prikazano na Slici 40. Te nove metode neće biti dostupne krajnjem korisniku, dok god se ručno vanjskoj klasi odnosno vanjskim klasama ne doda kod za delegiranje poziva tim novim metodama, i iste nakon toga ponovno ne prevedu u izvršibilni kod. U slučaju sa Slike 40, morali bi ručno dodati potreban kod COM klasi *CoOuter*, kako bi mogla delegirati poziv metodi *MA2* dodanoj sučelju *IA*. Ovaj problem posebno je naglašen ukoliko je *IA* u konkretnom slučaju sučelje *IDispatch*. Budući da je to jedino sučelje preko kojeg se može pristupiti COM klasi iz npr. nekog skriptnog jezika, ono mora imati preslikane sve potrebne nam metode svih drugih sučelja, čime se praktički ne može izbjeći da *IDispatch* sučelje novije verzije COM klase ima jednu ili više novih metode, ili da npr. novija verzija već postojeće metode zahtijeva veći broj parametara.



Slika 40: Sadržavanje ne rješava problem kad se nekom sučelju doda neka nova metoda.

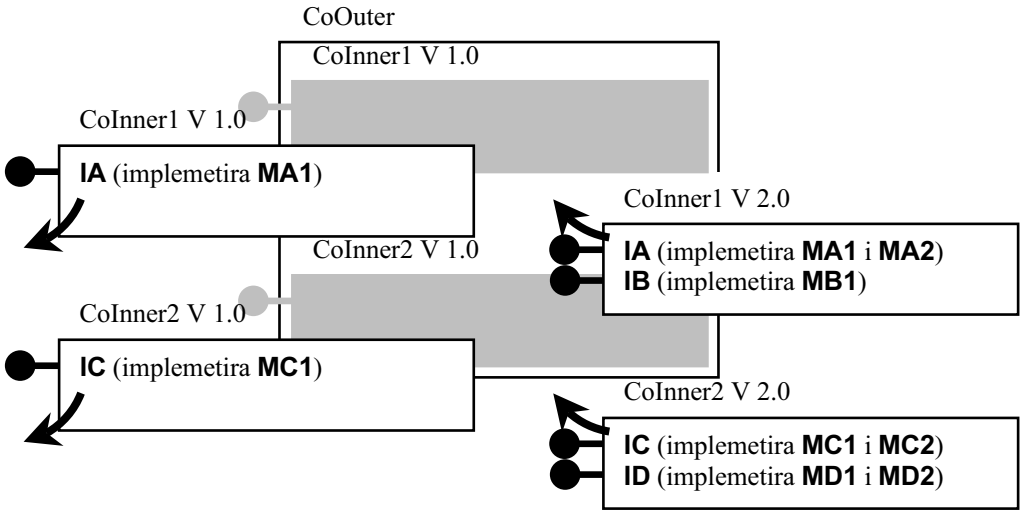
Kako bi se riješili opisani problemi, razvijena je agregacija. Ukoliko i COM klasa za koju želimo da agregira i COM klasa za koju želimo da bude agregirana podržavaju agregaciju, prva može klijentu osigurati direktan pristup sučelju odnosno sučeljima druge. Tako kad zamijenimo COM klasu *CoInner1 V 1.0* sa *CoInner1 V 1.5* kako je prikazano na Slici 41, ne moramo ručno modificirati sam kod COM klase *CoOuter* i istu nakon toga prevoditi u izvršni kod kako je bio slučaj kod sadržavanja. No problem ipak ostaje ukoliko COM klasu *CoInner1 V 1.0* želimo zamijeniti sa verzijom *CoInner1 V 2.0*, koja dodatno podržava i sučelje *IB*. Čak i ukoliko koristimo agregaciju, krajnji klijent neće moći pristupiti tom dodanom sučelju dok ponovno ručno ne modificiramo i ponovno prevedemo u izvršibilni kod klasu *CoOuter*. To ne samo da nije praktično,

već je i mogući izvor pogrešaka u kodu, a u nekim slučajevima kad nam nije dostupan izvorni kod vanjske klase, zahtjevanu promjenu nije niti moguće unijeti. Stoga je potrebno proširiti COM mehanizam agregacije, kako bi se među ostalim riješio i ovaj problem.



Slika 41: Problem kad se COM klasi agregirane klase doda neko novo sučelje.

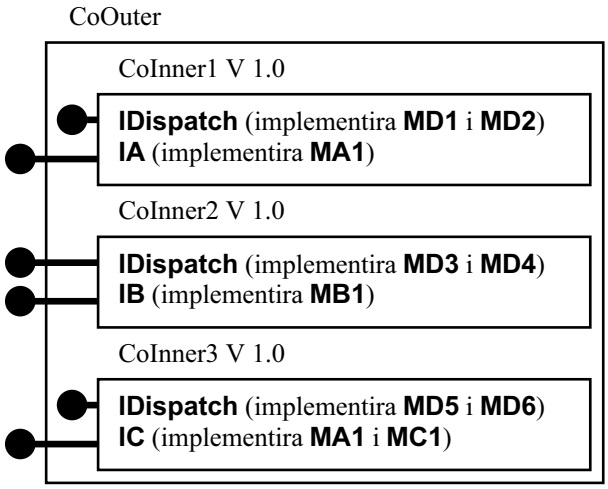
COM klasa može agregirati jednu, ali i više drugih COM klasa. Dodatni problem je u tome što svaka od agregiranih klasa može s novijom verzijom biti zamijenjena sa drugom koja uz postojeće sadrži jedno ili više novih sučelja, kako je prikazano na Slici 42. Tako novija verzija *V 2.0* COM klase *CoInner1* dodatno podržava sučelje *IB*, dok novija verzija *V 2.0* COM klase *CoInner2* dodatno podržava sučelje *ID*. Zbog istog razloga kao i u prethodnom slučaju, sučelja *IB* i *ID* neće biti dostupna krajnjem klijentu dok god ručno ne dopišemo potreban kod u COM klasi *CoOuter* i isti nakon toga ne prevedemo u izvršni kod. Razlika između proširenja koje je potrebno napisati kako bi se riješio ovaj problem u odnosnu na prethodni, otprilike odgovara razlici između jednostrukog i višestrukog nasljeđivanja u objektno orijentiranom programiranju.



Slika 42: COM klasa može agregirati jednu ali i više drugih COM klasa.

Dodatni problem koji se nadovezuje na prethodni slučaj je kad jedno te isto sučelje imaju dvije ili više agregiranih COM klasa kako je prikazano na Slici 43. To se često zna dogoditi sa standardnim sučeljima kao što je *IDispatch*, budući da njega mora imati svaka COM klasa za koju želimo da bude dostupna iz npr. skriptnog jezika kao što je ASP. Pošto svaka COM klasa bilo koje sučelje može imati samo jednom, u našem slučaju *CoOuter* može imati samo jednom sučelje *IDispatch* pa se krajnjem klijentu može omogućiti pristup *IDispatch* sučelju od samo jedne agregirane COM klase. Stoga su metode do kojih bi se inače moglo doći preko *IDispatch* sučelja drugih, kao što su u našem primjeru *MD1*, *MD2*, *MD5* i *MD6*, za njega nedostupne. Slučaj kad

različita sučelja imaju iste metode nije problem, pošto su npr. *IA::MA1* i *IC::MA1* dva potpuno različita poziva gdje ne može doći do kolizije.



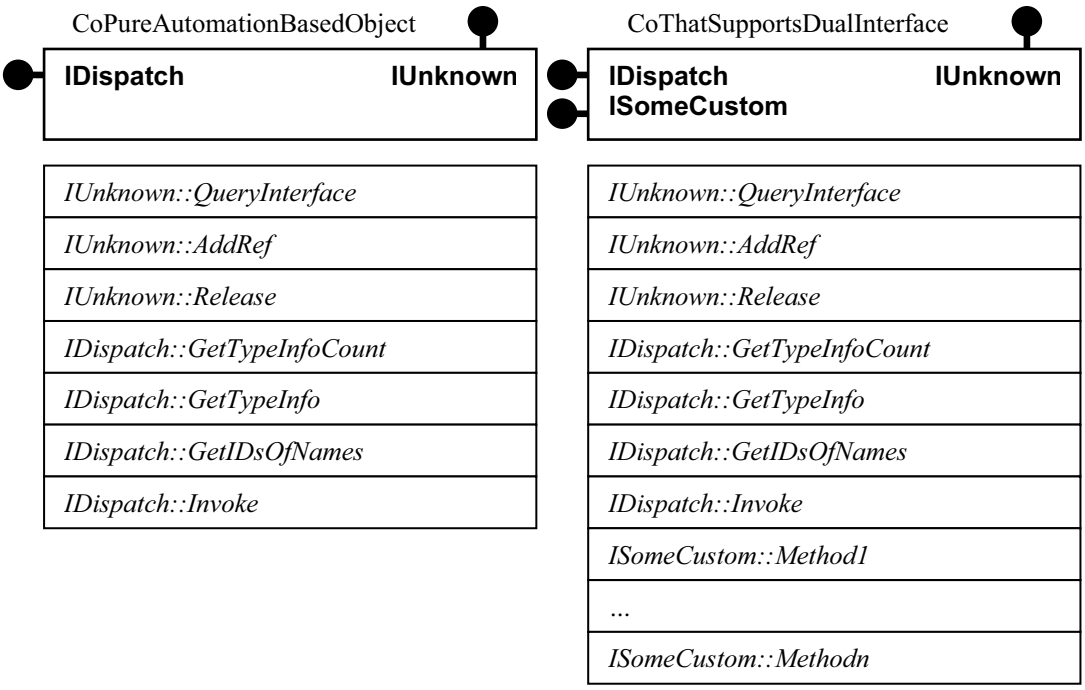
Slika 43: Neko sučelje može imati više od jedne agregirane COM klase.

Prije razrade algoritma kojim će se proširiti standardni način agregacije COM klasa, potrebno je kategorizirati vrste COM klasa sa stajališta agregacije, kako bi se obuhvatili svi mogući slučajevi. Osnovni princip jest da bi jedna COM klasa agregirala drugu, obje moraju podržavati agregaciju. Općenito, imamo sljedeće kategorije COM klasa s obzirom na podržavanje agregacije:

- COM klasa ne može biti agregirana i ne može agregirati drugu COM klasu;
- COM klasa ne može biti agregirana ali zato može agregirati drugu COM klasu;
- COM klasa može biti agregirana ali zato ne može agregirati drugu COM klasu;
- COM klasa može biti agregirana i može agregirati drugu COM klasu;
- neka COM klasa može biti građena tako da može agregirati samo jednu dodatnu COM klasu, ili tako da može agregirati i više od jedne odnosno proizvoljan broj dodatnih COM klasa.

Dodatno se mora uzeti u obzir i činjenica da svaka COM klasa može kroz *IDispatch* pružati nijednu, jednu ili više metoda. Također, potrebno je proučiti i standardne načine implementacije *IDispatch* sučelja, kako bi se ista također mogla proširiti sa svrhom rješavanja problema vezanih za agregaciju COM klasa koje imaju implementirano ovo specifično sučelje.

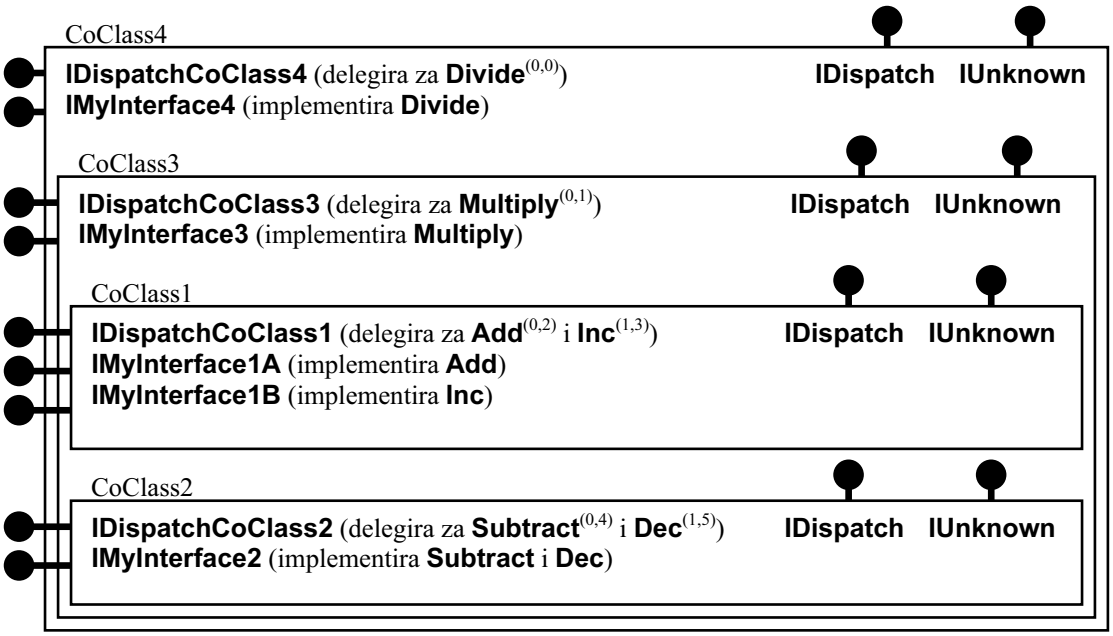
Na Slici 44 prikazana su dva najčešća načina implementacije *IDispatch* sučelja. COM klasa *CoPureAutomationBasedObject* predstavlja minimalnu moguću implementaciju, gdje COM klasa uz *IDispatch* nema implementirano niti jedno dodatno sučelje, kako je prikazano na Slici 44 (lijevo). Iako *IDispatch* sučelje pruža sasvim zadovoljavajuću funkcionalnost makro i skriptnim jezicima, pozivi koristeći to sučelje ne mogu se po performansama mjeriti sa pozivima koji direktno koriste korisnički definirana sučelja. Dodatno, iz jezika kao što je C++ bitno je lakše odnosno jednostavnije pozivati metode korisnički definiranih sučelja, nago sučelja *IDispatch*. Iz tih razloga, u zadnje vrijeme se bitno češće koristi takozvana dvojna implementacija sučelja *IDispatch*, kako je prikazano na Slici 44 (desno), gdje COM klasa uz *IDispatch* ima i korisnički definirano sučelje. Iako standardnom dvojnom implementacijom sučelja *IDispatch* nije predviđeno da svaka COM klasa uz *IDispatch* može podržavati i više od jednog korisnički definiranog sučelja, to će također biti obuhvaćeno algoritmom proširene agregacije, pošto ponekad kod COM klasa koje imaju implementirano sučelje *IDispatch* imamo potrebu za više korisnički definiranih sučelja zbog istog razloga zbog kojeg imamo i kod svih drugih COM klasa.



Slika 44: Dva najčešća načina implementacije IDispatch sučelja.

Budući da svako COM sučelje mora biti derivirano iz *IUnknown*, *IDispatch* time direktno nasljeđuje metode *QueryInterface*, *AddRef* i *Release*. Uz njih, *IDispatch* ima metode *GetTypeInfoCount*, *GetTypeInfo*, *GetIDsOfNames* i *Invoke*. Prilikom daljnje razrade proširenog algoritma agregacije, ovdje su nam posebno zanimljive zadnje dvije metode. Korištenje *IDispatch* sučelja počinje tako da klijent, kao UNICODE string, metodi *GetIDsOfNames* proslijedi imena pridruženih metoda koje namjerava koristiti. Ukoliko COM klasa podržava te pridružene metode, *GetIDsOfNames* korisniku vrača polje podataka tipa DISPID (DISPatch IDentifier), gdje je DISPID 32-bitan broj koji jedinstveno identificira svaku pridruženu metodu unutar klase. Nakon toga, korisnik poziva željenu pridruženu metodu šaljući metodi *Invoke* taj identifikator i parametre pakirane u strukturi tipa DISPPARAMS. Korisnik naravno može lokalno spremiti dobivene identifikatore tipa DISPID, te je time metodu *GetIDsOfNames* potrebno pozvati samo jednom prilikom inicijalizacije klijenta.

Kako bi sistematički implementirali *IDispatch* kao dio svake COM klase, svaka klasa sadrži jedno sučelje derivirano iz *IDispatch* čije ime sadrži ime klase. Tako, kao što je prikazano na Slici 45, COM klasa *CoClass1* implementira *IDispatchCoClass1* i još dva dodatna korisnička sučelja *IMyInterface1A* i *IMyInterface1B*, COM klasa *CoClass2* implementira *IDispatchCoClass2* i korisničko sučelje *IMyInterface2*, itd. Koristeći ovaj pristup, sve metode kojim se može direktno pristupiti iz jezika koji mogu direktno pristupiti korisnički definiranim sučeljima, kao što su primjerice Visual C++, Visual J++ i Visual Basic, ne moraju biti ograničene da budu dio samo jednog dodatnog korisnički definiranog sučelja, kao što je to slučaj kod klasične dvojne implementacije *IDispatch* sučelja kako je prikazano na Slici 44 (desno).



Slika 45: COM klase koje podržavaju proširen oblik agregacije.

Također, može se primijetiti da slučajevi sa Slike 45 obuhvaćaju sve kategorije COM klasa s obzirom na podržavanje agregacije. Jedini zahtjev koji moramo ispuniti prilikom implementacije agregacije, je da COM klasa koja agregira mora krajnjem klijentu izgledati isto kao da su sva sučelja agregiranih COM klasa implementirana kao njena vlastita. To znači da moramo ispravno implementirati metode *QueryInterface*, *AddRef* i *Release*.

Kako bi podržali proširenu agregaciju, prvo je potrebno svim COM klasama dodati sučelje *INondelegatingUnknown*. To sučelje je samo za internu upotrebu, i nije direktno dostupno vanjskom klijentu odnosno klijentima. Ono podržava tri metode: *NondelegatingQueryInterface*, *NondelegatingAddRef* i *NondelegatingRelease*. Metode *QueryInterface*, *AddRef* i *Release* sučelja *IUnknown* samo će delegirati pozive tim trima metodama sučelja *INondelegatingUnknown*.

Nakon inicijalizacijske faze, nakon što COM klasa *CoClass4* kreira COM klasu *CoClass3* koja interno kreira COM klase *CoClass2* i *CoClass1*, svaka COM klasa mora imati kazaljku na *INondelegatingUnknown* od *CoClass4*. Tako će se za vrijeme izvođenja svi pozivi *QueryInterface* metode biti delegirani za *NondelegatingQueryInterface* od *CoClass4*, kao što će i svi pozivi prema *AddRef* i *Release* biti delegirani za *NondelegatingAddRef* i *NondelegatingRelease* od *CoClass4*. To znači da su jedine metode koje će povećavati i smanjivati referentno brojilo *NondelegatingAddRef* i *NondelegatingRelease* od *CoClass4*. Kad referentno brojilo od *CoClass4* dođe na vrijednost 0, zadaća te klase je da uništi sebe ali i *CoClass3* koju agregira, kao što je pritom i zadaća od *CoClass3* da uništi sebe i *CoClass2* i *CoClass1* koje sama agregira. Dok je jednostavno implementirati referentno brojenje odnosno metode *NondelegatingAddRef* i *NondelegatingRelease*, nešto je složenija implementacija metode *NondelegatingQueryInterface*.

Na Ispisu 4 prikazano je kakva bi bila uobičajena implementacija metode *NondelegatingQueryInterface* za COM klasu *CoClass4*. Za vrijeme inicijalizacijskog procesa, COM klasa *CoClass4* može dobaviti i spremiti kazaljke na agregirana sučelja, ili može u hodu zvati *NondelegatingQueryInterface* sučelje agregirane COM klase *CoClass3*. Sučelje *NondelegatingQueryInterface* ostalih COM klasa može se implementirati na ekvivalentan način. Nedostatak ovakvog klasičnog pristupa očituje se u slučajevima koji su prikazani na slikama 41 i 42, kad se jedna verzija agregirane COM klase zamjenjuje novijom koja sadrži jedno ili više novo-dodanih sučelja. Ukoliko želimo riješiti taj problem, moramo razviti algoritam koji neće ovisiti o tome koja korisnički definirana sučelja agregirana COM klasa ima. Koristeći takav pristup,

moći ćemo agregirane COM klase zamjenjivati i novijim verzijama koje imaju više sučelja od originalne koju zamjenjuju, i COM klasa koja agregira će ih automatski podržati.

```
if( korisnik_zahtijeva_kazaljku_na_IMyInterface4 )
{
    povećaj_referentno_brojilo;
    return kazaljku_na_IMyInterface4;
}
else if( korisnik_zahtijeva_kazaljku_na_IMyInterface3 )
{
    uzmi_kazaljku_na_IMyInterface3_iz_lokalne_varijable_ili_je_traži_od_CoClass3;
    povećaj_referentno_brojilo_ako_je_kazaljka_uzeta_iz_lokalne_varijable;
    return kazaljku_na_IMyInterface3;
}
else if( korisnik_zahtijeva_kazaljku_na_IMyInterface2 )
{
    uzmi_kazaljku_na_IMyInterface2_iz_lokalne_varijable_ili_je_traži_od_CoClass3;
    povećaj_referentno_brojilo_ako_je_kazaljka_uzeta_iz_lokalne_varijable;
    return kazaljku_na_IMyInterface2;
}
else if( korisnik_zahtijeva_kazaljku_na_IMyInterface1 )
{
    uzmi_kazaljku_na_IMyInterface1_iz_lokalne_varijable_ili_je_traži_od_CoClass3;
    povećaj_referentno_brojilo_ako_je_kazaljka_uzeta_iz_lokalne_varijable;
    return kazaljku_na_IMyInterface1;
}
else
    return E_NOINTERFACE;
```

Ispis 4: Uobičajena implementacija NondelegatingQueryInterface metode.

Prvi dio algoritma koji poslužuje pozive za sučeljima implementiranih od same klase može ostati isti. Također, *IUnknown* i *INondelegatingUnknown* moraju biti implementirani u svakoj COM klasi koja podržava agregaciju. Jedina dodatna informacija o kojoj algoritam može ovisiti je lista COM klasa koje neka COM klasa direktno agregira. "Direktno" znači da će npr. lista od *CoClass4* sadržavati jedino element *CoClass3*, dok će lista od *CoClass3* sadržavati dva elementa i to *CoClass2* i *CoClass1*.

Dodatni zahtjev je da algoritam ne smije ovisiti o broju agregiranih COM klasa. Taj zahtjev je važan utoliko što nam osigurava da algoritam trebamo implementirati samo jednom, kako bi ga koristili u svim COM klasama koje trebaju podržavati ovaj prošireni oblik agregacije. Budući je lista agregiranih COM klasa statična, u tu svrhu možemo koristiti polje kazaljki na sučelja *INondelegatingUnknown* agregiranih COM klasa. Veličina tog polja jednaka je broju agregiranih COM klasa, a isto se može dinamički alocirati za vrijeme inicijalizacije kao što se u tom slučaju treba i dealocirati za vrijeme deaktivacije. Na Ispisu 5 prikazana je usavršena implementacija *NondelegatingQueryInterface* metode koja zadovoljava ove zahtjeve. Kako bi dokazali da je *NondelegatingQueryInterface* metoda ispravno implementirana, moramo dokazati da je zadovoljeno svojstvo ekvivalencije *QueryInterface* metode. Nakon toga, potrebno je i pokazati da smo riješili problem prikazan na slikama 41 i 42, koji se javlja kad se jedna verzija agregirane COM klase zamjenjuje novijom koja sadrži jedno ili više novo-dodanih sučelja.

```
if(korisnik_zahtijeva_kazaljku_na_neko_moje_sučelje )
    { povećaj_referentno_brojilo; return kazaljku_na_to_sučelje; }
for ( i = 0; i < broj_agregiranih_COM_klasa; i++ )
    if( pitaj_COM_klasa_[i]_da_li_ima_traženo_sučelje_i_ako_ima_onda )
        return traži_COM_klasu_[i]_kazaljku_na_to_sučelje_i_proslijedi_je_korisniku;
return E_NOINTERFACE;
```

Ispis 5: Usavršena implementacija NondelegatingQueryInterface metode.

Može se primijetiti da će svi pozivi biti delegirani metodi *NondelegatingQueryInterface* vanjske COM klase *CoClass4*. Uz to, ponašanje *NondelegatingQueryInterface* je deterministično i statično (pošto se agregirane klase ne zamjenjuju drugim za vrijeme izvođenja programa), što znači da ta metoda uvijek vraća iste izlazne parametre za bilo koju kombinaciju ulaznih, čak i u slučaju kad dvije ili više agregiranih komponenata imaju ista sučelja.

Jedini način na koji korisnik može dobiti ispravnu kazaljku jest da istu traži od metode *NondelegatingQueryInterface* COM klase *CoClass4*. Budući je ta metoda deterministička, ukoliko on ponovno zahtijeva kazaljku na to isto sučelje, čak i od njega samog (npr. ukoliko traži kazaljku na *IX* koristeći *IX::QueryInterface*), ponovno će primiti istu kazaljku. To znači da je zadovoljeno svojstvo refleksivnosti.

Ukoliko je korisnik dobio ispravnu kazaljku na *IY* koristeći *IX::QueryInterface*, i nakon toga traži kazaljku na IX koristeći *IY::QueryInterface*, poziv će opet biti delegiran *NondelegatingQueryInterface* od COM klase *CoClass4*, i opet će vratiti istu kazaljku na sučelje *IX* kao što je to vraćeno i prvi put. To znači da je zadovoljeno i svojstvo simetričnosti.

Ukoliko je korisnik primio ispravnu kazaljku na *IY* koristeći *IX::QueryInterface*, i ispravnu kazaljku na *IZ* koristeći *IY::QueryInterface*, možemo zaključiti da *NondelegatingQueryInterface* od COM klase *CoClass4* vraća ispravnu kazaljku na *IZ*. Ukoliko korisnik kasnije traži kazaljku na *IZ* koristeći *IX::QueryInterface*, taj poziv će također biti delegiran metodi *NondelegatingQueryInterface* COM klase *CoClass4*, i korisnik će ponovno dobiti ispravnu kazaljku na sučelje *IZ*. To znači da je zadovoljeno i svojstvo tranzitivnosti.

Budući da su zadovoljena svojstva simetričnosti, refleksivnosti i tranzitivnosti, možemo zaključiti da je svojstvo ekvivalencije metode *QueryInterface* također zadovoljeno, i da smo ispravno implementirali *NondelegatingQueryInterface* metodu.

Ukoliko sad korisnik zamijeni jednu ili više COM klasa sa novijim verzijama koje podržavaju i neka nova sučelja, koristeći usavršenu implementaciju metode *NondelegatingQueryInterface* koja je prikazana na Ispisu 5, on ne mora ručno dopisivati kod u svim COM klasama koje ih agregiraju. Predloženi algoritam će dinamički pronaći ta nova sučelja, ekvivalentno kako pronalazi i od prije implementirana. Time su i riješeni problemi prikazani na slikama 41 i 42.

Budući svaka COM klasa može bilo koje sučelje imati samo jednom, klijent vidi jedino *IUnknown* i *IDispatch* od vanjske COM klase, u našem slučaju to je *CoClass4*. Stoga COM klase moraju rukovati sa pozivima metoda *GetTypeInfoCount*, *GetTypeInfo*, *GetIDsOfNames* i *Invoke* na sličan način kao i sa pozivima *QueryInterface* metode (mora biti razvijena neka vrsta delegacije).

Algoritam koji rukuje *IDispatch* sučeljem mora ispravno dinamički dodjeljivati DISPID-ove svim pridruženim metodama *IDispatch* sučelja. Unutar bilo koje COM klase, dvije različite pridružene metode *IDispatch* sučelja moraju uvijek imati različite DISPID-ove. Pošto COM klase *CoClass1* i *CoClass2* ne mogu znati da će biti agregirane od strane neke druge COM klase, *CoClass3* isto kao i *CoClass4* moraju osigurati dinamičko dodjeljivanje DISPID-ova.

Ukupan broj pridruženih metoda *IDispatch* sučelja koje neka COM klasa sama implementira (ne računajući agregirane COM klase) je spremljen u varijablu *m_cMyDispatchIDs*. Ukupan broj pridruženih metoda *IDispatch* sučelja koje neka agregirana COM klasa implementira zajedno sa COM klasama koje sama rekurzivno agregira je spremljen u polje *m_cInnerDispatchIDsArray*, dok je kazaljka ne njeno *IDispatch* sučelje spremljeno u polje *m_pDispatchInnerArray*, i tako za sve direktno agregirane COM klase. Te vrijednosti svaka COM klasa postavlja za vrijeme inicijalizacije. U tu svrhu može se koristiti prvi dio algoritma kako je prikazano na Ispisu 6. Inače, kod poziva metode *GetIDsOfNames*, *riid* ima vrijednost *IID_NULL*. Vrijednost *IID_IDispatch* može se iskoristiti kako bi se od metode zatražio ukupan broj metoda pridružen sučelju *IDispatch*.

Koristeći ove informacije, COM klasa koja agregira može za vrijeme izvođenja adekvatno mijenjati DISPID-ove koje dobiva od agregiranih COM klasa, kako bi osigurala jedinstvenost svakog DISPID-a kojeg dobiva klijent (drugi indeksi sa Slike 45). Taj algoritam je sastavni dio metode *GetIDsOfNames*. Kako bi se algoritam pojednostavnio, sve COM klase interno broje metode pridružene *IDispatch* sučelju u rastućem redoslijedu počinjući od 0 (prvi indeksi sa Slike 45). Nakon što korisnik pošalje kao UNICODE string metodi *GetIDsOfNames* ime pridružene metode koju zahtijeva, svaka COM klasa prvo pretražuje svoju biblioteku sučelja. Ukoliko COM klasa ne podržava tu pridruženu metodu, ciklički delegira taj poziv svim agregiranim COM klasama, dok god neka od njih ne potvrdi da podržava tu pridruženu metodu, odnosno dok sve ne potvrde da ju ne podržavaju. Za to vrijeme, algoritam adekvatno povećava lokalnu varijablu *dispid* kako bi osigurao da klijent uvijek dobije jedinstven DISPID.

```
HRESULT _stdcall GetIDsOfNames(REFIID riid, LPOLESTR* rgszNames, \
    UINT cNames, LCID lcid, DISPID* rgDispId) \
{ \
    if(riid == IID_IDispatch) \
    { \
        *rgDispId = m_cMyDispatchIDs; \
        for(ULONG i=0; i<m_cUnknownInnerEntries; i++) \
            *rgDispId += m_cInnerDispatchIDsArray[i]; \
        return S_OK; \
    } \
    if(riid != IID_NULL) \
        return DISP_E_UNKNOWNINTERFACE; \
    HRESULT hr; \
    if(m_cMyDispatchIDs != 0) \
    { \
        hr = DispGetIDsOfNames(m_pTypeInfo, rgszNames, cNames, rgDispId); \
        if(SUCCEEDED(hr)) \
            return hr; \
    } \
    DISPID dispid; \
    dispid = m_cMyDispatchIDs; \
    for(ULONG i=0; i<m_cUnknownInnerEntries; i++) \
    { \
        hr = m_pDispatchInnerArray[i]->GetIDsOfNames(riid, rgszNames, cNames, lcid, \
            rgDispId); \
        if(SUCCEEDED(hr)) \
        { \
            *rgDispId += dispid; \
            return hr; \
        } \
        else \
            dispid += m_cInnerDispatchIDsArray[i]; \
    } \
    return DISP_E_UNKNOWNNAME; \
} \
```

Ispis 6: C++ implementacija metode GetIDsOfNames.

Drugi dio algoritma implementiran je kao dio metode *Invoke* kako je prikazano na Ispisu 7. Osnovni zadatak metode je da iz primljenog jedinstvenog DISPID-a odredi koja COM klasa te koja pridružena metoda *IDispatch* sučelja te klase je određena tim DISPID-om. Drugim riječima, metoda mora rekonstruirati originalan DISPID (prvi indeksi sa Slike 45) iz primljenog DISPID-a (drugi indeksi sa Slike 45), te delegirati zahtjev (inverzna funkcionalnost od metode *GetIDsOfNames*).

Algoritam prvo provjerava da li je primljen DISPID manji od vrijednosti spremljene u varijabli *m_cMyDispatchIDs*: ukoliko je, *IDispatch* od te same COM klase podržava traženu pridruženu metodu i poziv se delegira vlastitoj implementaciji sučelja *IDispatch*. U suprotnom,

algoritam smanjuje primljen DISPID za ukupan broj pridruženih metoda sučelja *IDispatch* te COM klase (spremljen u varijabli *m_cMyDispatchIDs*), te rekurzivno ponavlja postupak na analogan način sve dok ne dođe do agregirane COM klase u okviru koje je implementirana pridružena metoda određena primljenim DISPID-om.

```
HRESULT _stdcall Invoke(DISPID dispIdMember, REFIID riid, LCID lcid, WORD wFlags, \
DISPPARAMS* pDispParams, \
VARIANT* pVarResult, EXCEPINFO* pExcepInfo, UINT* puArgErr) \
{ \
    if(riid != IID_NULL) \
        return DISP_E_UNKNOWNINTERFACE; \
    if(dispIdMember < m_cMyDispatchIDs) \
        return DispInvoke(static_cast<IDispatch##CoName*>(this), m_pTypeInfo, \
            dispIdMember, wFlags, pDispParams, pVarResult, pExcepInfo, puArgErr); \
    DISPID dispid; \
    dispid = dispIdMember - m_cMyDispatchIDs; \
    for(ULONG i=0; i<m_cUnknownInnerEntries; i++) \
    { \
        if(dispid < m_cInnerDispatchIDsArray[i]) \
            return m_pDispatchInnerArray[i]->Invoke(dispid, riid, lcid, wFlags, pDispParams, \
                pVarResult, pExcepInfo, puArgErr); \
        dispid -= m_cInnerDispatchIDsArray[i]; \
    } \
    return DISP_E_MEMBERNOTFOUND; \
}
```

Ispis 7: C++ implementacija metode Invoke COM klase CoClass4.

Ukoliko svaka COM klasa ima implementirane metode *GetIDsOfNames* i *Invoke* (kako je prikazano na ispisima 6 i 7), riješili smo i problem kad konkretno *IDispatch* sučelje ima više od jedne agregirane COM klase, kako je prikazano na Slici 43. Zahvaljujući mehanizmu delegiranja poziva *IDispatch* sučelja, koji je razvijen kao sastavni dio proširenog algoritma agregacije, klijent uvijek prima za svaku metodu pridruženu *IDispatch* sučelju jedinstven DISPID, dok se za vrijeme samog poziva iz tog DISPID-a rekurzivno rekonstruira originalni DISPID, kao što se i sam poziv delegira točno COM klasi koja implementira traženu pridruženu metodu i koja je generirala sam originalni DISPID.

Ovim proširenjem COM algoritma agregacije, dobili smo razvojnu okolinu koja je preduvjet za uspješnu implementaciju objekata jezgre posredničkog reda. Naime, budući da klasa odnosno grupa raznorodnih izvora tipova podataka ima proizvoljno puno, pojedini programer odnosno pojedina grupa programera ne može imati dovoljno znanja odnosno iskustva i vremena kako bi napisala potrebne module za sve njih. Pristup proširenog algoritma agregacije omogućava da svaki programer odnosno grupa programera zaduženih za implementaciju COM klase odnosno COM klasa, koje će podržati pristup nekom specifičnom izvoru podataka, može raditi na svojem projektu neovisno od drugih. Pritom mogu postojati neka specifična sučelja koje će po dogovoru implementirati svi, radi međusobne razmijene podataka odnosno ostvarivanja sposobnosti zajedničkog rada, no i svakom zasebnom modulu odnosno COM klasi mogu se po potrebi dodavati i sučelja specifična za taj izvor podataka, koja će zahvaljujući algoritmu proširene agregacije biti dostupni krajnjem klijentu odnosno korisniku. Time drugim riječima imamo otvorenu arhitekturu jezgre posredničkog reda, gdje se pojedini izvori podataka podržavaju izgradnjom i dodavanjem odgovarajućih modula.

4.4.9. Pristup komponentama COM iz različitih okruženja

Na kraju, kako bi se provjerilo da je pristup predložen kroz algoritam proširene agregacije moguće relativno jednostavno implementirati u praksi, koristeći IDL i Visual C++ implementirane su COM klase prikazane na Slici 45. Nakon toga, provjereno je da pristup funkcionira kako kad se sam klijent također piše u jeziku Visual C++, tako i za slučajeve kad se COM klase pozivaju iz HTML-a, ASP-a, te Visual Basic makro jezika. Također, slučaj kad je klijent pisan u jeziku Visual C++ provjeren je i za slučaj kad su klijent i komponenta unutar istog procesa na istom računalu, kad su unutar različitih procesa na istom računalu, kao i kad su na različitim računalima.

Ispis 8 prikazuje slučaj poziva sa udaljenog računala s klijentom napisanim u jeziku C++.

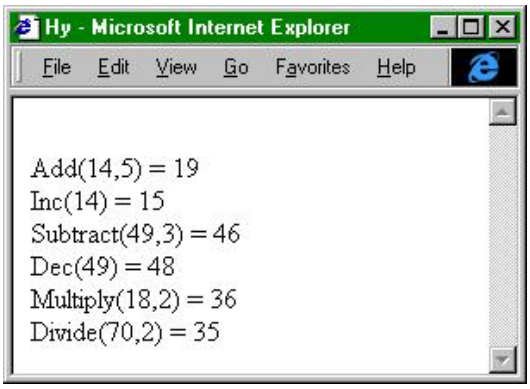
```
client.exe: Calling CoInitialize()
client.exe: Calling ClientBody()
client.exe: Calling CoClass4 CoCreateInstance() for IMyInterface1A
client.exe: Calling IMyInterface1A::QueryInterface() for IMyInterface1B
client.exe: Calling IMyInterface1A::QueryInterface() for IMyInterface2
client.exe: Calling IMyInterface1A::QueryInterface() for IMyInterface3
client.exe: Calling IMyInterface1A::QueryInterface() for IMyInterface4
client.exe: Calling IMyInterface1A::Add(8,4)
client.exe: Calling succeeded => Add(8,4)=12
client.exe: Calling IMyInterface1B::Inc(8)
client.exe: Calling succeeded => Inc(8)=9
client.exe: Calling IMyInterface2::Subtract(8,4)
client.exe: Calling succeeded => Subtract(8,4)=4
client.exe: Calling IMyInterface2::Dec(8)
client.exe: Calling succeeded => Dec(8)=7
client.exe: Calling IMyInterface3::Multiply(8,4)
client.exe: Calling succeeded => Multiply(8,4)=32
client.exe: Calling IMyInterface4::Divide(8,4)
client.exe: Calling succeeded => Divide(8,4)=2
client.exe:  :)
client.exe: Calling CoUninitialize()
client.exe: the end
```

Ispis 8: Izlaz klijenta clientR.exe – CoClass4 poziva se sa udaljenog računala.

Na Ispisu 9 prikazano je kako COM klase možemo pozvati direktno iz HTML-a, dok je na Slici 46 prikazan rezultat izvršavanja tog koda.

```
<HTML>
   <HEAD>
      <TITLE>Hy</TITLE>
   </HEAD>
   <BODY>
      <OBJECT ID="MyComponent" CLASSID="CLSID:19720706-0E04-11D2-830D-006008545093">
      </OBJECT>
      <SCRIPT LANGUAGE="LiveScript">
         document.write(
            "<br>Add(14,5) = ",          MyComponent.Add(14, 5),
            "<br>Inc(14) = ",            MyComponent.Inc(14),
            "<br>Subtract(49,3) = ",     MyComponent.Subtract(49, 3),
            "<br>Dec(49) = ",            MyComponent.Dec(49),
            "<br>Multiply(18,2) = ",     MyComponent.Multiply(18, 2),
            "<br>Divide(70,2) = ",       MyComponent.Divide(70, 2));
      </SCRIPT>
   </BODY>
</HTML>
```

Ispis 9: HTML klijent.



Slika 46: Rezultat izvođenja HTML klijenta sa Ispisa 9.

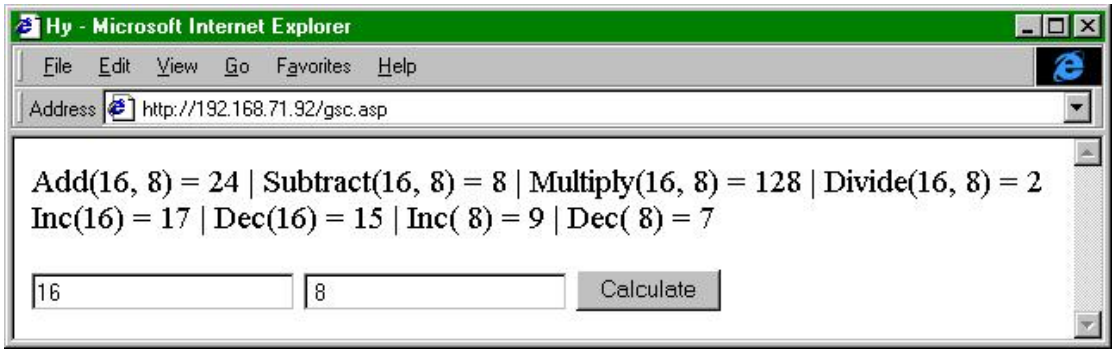
Na Ispisu 10, dat je klijent u ASP-u. Ispis 11 prikazuje rezultat procesiranja ASP koda, generirani HTML kod. Na Slici 47 prikazan je rezultat izvršavanja ASP koda sa Ispisa 10.

```
<%@ LANGUAGE="VBSCRIPT" %>
<%Response.Expires = 0%>
<HTML>
   <HEAD>
      <TITLE>
         Hy
      </TITLE>
   </HEAD>
   <BODY>
      <%
      Set MyComponent = Server.CreateObject("Component.CoClass4")
      p1 = Request.Form("dFirst")
      p2 = Request.Form("dSecond")
      If p1 = 0 Then
         p1 = 1
      End If
      If p2 = 0 Then
         p2 = 1
      End If
      %>
      Add(<%=p1%>,<%=p2%>) = <%= MyComponent.Add(p1, p2) %>
      | Subtract(<%=p1%>,<%=p2%>) = <%= MyComponent.Subtract(p1, p2) %>
      | Multiply(<%=p1%>,<%=p2%>) = <%= MyComponent.Multiply(p1, p2) %>
      | Divide(<%=p1%>,<%=p2%>) = <%= MyComponent.Divide(p1, p2) %> </br>
      Inc(<%=p1%>) = <%= MyComponent.Inc(p1) %>
      | Dec(<%=p1%>) = <%= MyComponent.Dec(p1) %>
      | Inc(<%=p2%>) = <%= MyComponent.Inc(p2) %>
      | Dec(<%=p2%>) = <%= MyComponent.Dec(p2) %> </br>
      <FORM METHOD="POST" ACTION="http://192.168.71.92/gsc.asp">
         <P>
            <INPUT TYPE="txt" NAME="dFirst" VALUE="16">
            <INPUT TYPE="txt" NAME="dSecond" VALUE=" 8">
            <INPUT TYPE="SUBMIT" VALUE="Calculate">
         </P>
      </FORM>
   </BODY>
</HTML>
```

Ispis 10: ASP klijent.

```
<HTML>
    <HEAD>
        <TITLE>
            Hy
        </TITLE>
    </HEAD>
    <BODY>
        Add(16, 8) = 24
        | Subtract(16, 8) = 8
        | Multiply(16, 8) = 128
        | Divide(16, 8) = 2 </br>
        Inc(16) = 17
        | Dec(16) = 15
        | Inc( 8) = 9
        | Dec( 8) = 7 </br>
        <FORM METHOD="POST" ACTION="http://192.168.71.92/gsc.asp">
            <P>
                <INPUT TYPE="txt" NAME="dFirst" VALUE="16">
                <INPUT TYPE="txt" NAME="dSecond" VALUE=" 8">
                <INPUT TYPE="SUBMIT" VALUE="Calculate">
            </P>
        </FORM>
    </BODY>
</HTML>
```

Ispis 11: HTML kao rezultat izvođenja ASP koda sa Ispisa 10.

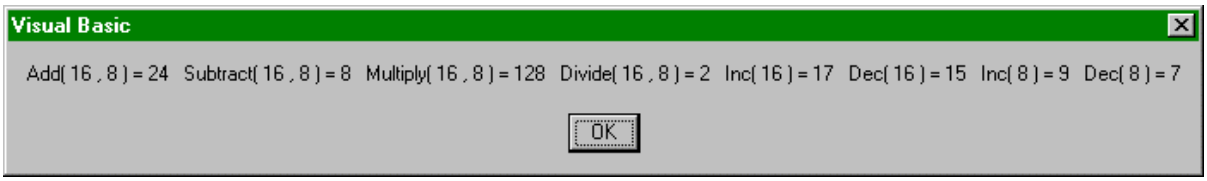


Slika 47: Rezultat izvođenja ASP klijenta sa Ispisa 10.

Na Ispisu 12, dat je klijent u Visual Basic makro jeziku, dok je na Slici 48 prikazan rezultat izvršavanja tog koda.

```
Sub TestComponent()
'DESCRIPTION: Macro to test CoClass4 component using IDispatch.
    Dim MyRef
    Set MyRef = CreateObject("Component.CoClass4")
    MsgBox "Add( 16 , 8 ) = " & MyRef.Add(16,8) _
        & "   Subtract( 16 , 8 ) = " & MyRef.Subtract(16,8) _
        & "   Multiply( 16 , 8 ) = " & MyRef.Multiply(16,8) _
        & "   Divide( 16 , 8 ) = " & MyRef.Divide(16,8) _
        & "   Inc( 16 ) = " & MyRef.Inc(16) _
        & "   Dec( 16 ) = " & MyRef.Dec(16) _
        & "   Inc( 8 ) = " & MyRef.Inc(8) _
        & "   Dec( 8 ) = " & MyRef.Dec(8)
End Sub
```

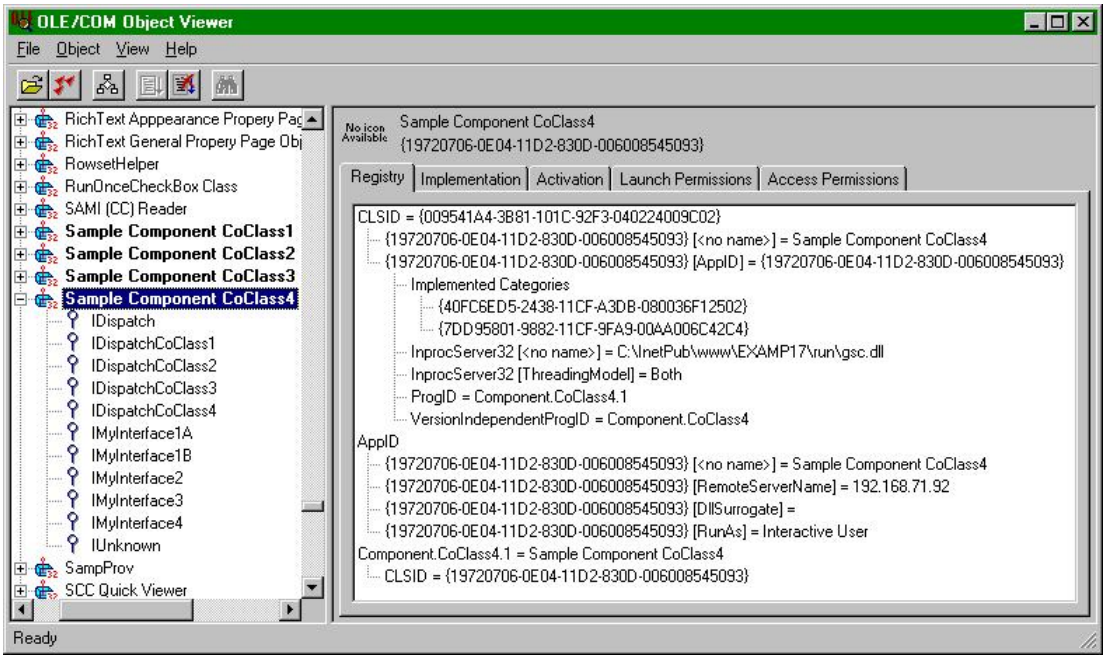
Ispis 12: Klijent napisan u Visual Basic makro jeziku.



Slika 48: Rezultat izvođenja Visual Basic klijenta sa Ispisa 12.

Iz priloga se vidi ne samo da se sam proširen oblik agregacije COM klasa da relativno lako implementirati koristeći jezike IDL i Visual C++, već i da pristup funkcionira bez obzira na koji način se same COM klase koriste, odnosno iz kojeg jezika se poziva *IDispatch* sučelje.

Na kraju, možemo još pogledati kako izgledaju same COM klase koristeći vizualno sučelje *OLE/COM Object Viewer* aplikacije. Kako se vidi sa Slike 49, sam operacijski sustav odnosno sama aplikacija za COM klasu *CoClass4* prikazuje da ima implementirana kako vlastita sučelja, i to: *IUnknown*, *IDispatch*, *IMyInterface4* i *IDispatchCoClass4*; tako i sučelja COM klasa koje agregira, i to: *IMyInterface3*, *IMyInterface2*, *IMyInterface1A*, *IMyInterface1B*, *IDispatchCoClass3*, *IDispatchCoClass2* i *IDispatchCoClass1*. Time imamo i povratnu informaciju da je i sama biblioteka sučelja, koja se u našem slučaju nalazi u DLL datoteci zajedno sa samim COM klasama, ispravno implementirana algoritmom proširene agregacije.



Slika 49: OLE/COM Object Viewer – vizualni pregled COM klasa instaliranih na računalo.

4.4.10. Alati, biblioteke te ostale preporuke za pisanje COM klasa

Dodatno se prilikom pisanja COM klasa može koristiti:

- *MFC (Microsoft Foundation Class)* – biblioteka standardnih Windows C++ klasa [108];
- *ATL (Active Template Library)* – biblioteka standardnih COM klasa i standardnih elemenata COM klasa [117].

MFC se općenito može koristi kod kreiranja korisničkih sučelja, u slučaju da kao alat implementacije koristimo Visual C++, bilo da sam kod pišemo kao COM komponente ili da ga pišemo samo kao C++ klase (svaka COM klasa može biti upravo C++ klasa ukoliko kao jezik implementacije izaberemo npr. Visual C++). *ATL* nam s druge strane direktno olakšava pisanje COM klasa, pružajući već gotove implementacije odnosno prototipove implementacija najčešće potrebnih struktura koda. Također, mogu se koristiti i pojedini alati koji automatski generiraju "kosture" COM klasa, specifičnih za neko posebno područje, dok je onda na programeru da ručno doda što je već potrebno.

Sve se može još dodatno pojednostavniti korištenjem makro naredbi, čime se dio opisane procedure automatizira, tako da samo korištenje monikora bude maksimalno jednostavno. Sve što je potrebno u tom slučaju jest upisati traženi string, i pozvati sučelje *IMyClassFactory0* kojim kreiramo i inicijaliziramo traženu komponentu.

I na kraju, koji jezik izabrati za implementaciju samih komponenata? Za sučelje, kao što je već rečeno, najbolje je koristiti jezik IDL. Za samo tijelo komponente, preporučeno je koristiti C++ odnosno Visual C++ [72][126][141][161]. Komponente se mogu pisati i u bilo kojem drugom jeziku čiji prevodilac podržava COM binarni standard [65], no tada će njihove mogućnosti u nekim slučajevima biti manje. Na primjer, mogu se među ostalim koristiti i jezici Visual Basic i Visual J++. Naravno, ukoliko se to želi, može se koristiti i sam strojni jezik pošto su komponente binaran standard. No ovo ipak ne treba posebno izdvajati pošto i sam C/C++ dozvoljava da se neki (vremenski kritični) dijelovi pišu upravo u strojnom jeziku, tako da kada se kaže C++, onda se u pravilu misli da je tu po potrebi obuhvaćen i C i strojni jezik. Preporuka je da se komponente pišu u jeziku C++, a ako ih koriste i neki drugi jezici, jednostavno se implementira i *IDispatch* sučelje odnosno biblioteka sučelja, tako da isti mogu komponenti pristupati, kao što joj u slučaju implementacije *IDispatch* sučelje vrlo jednostavno mogu pristupati i makro jezici.

5. PROTOTIP POSREDNIKA ZA PRISTUP RAZNORODNIM SUSTAVIMA POSLOVANJA RELACIJSKIH BAZA PODATAKA

Dok je kroz prethodno poglavlje razvijena sama *infrastruktura* za izgradnju posredničkog reda, kroz ovo poglavlje opisuje se konkretna implementacija prototipa modela za pristup raznorodnim sustavima poslovanja relacijskih baza podataka. Kroz uspješnu praktičnu realizaciju samog modela za neko konkretno područje, želi se prije svega pokazati kako primjenjivost u praksi, tako i opravdati sam pristup predložen kroz prethodno odnosno prethodna poglavlja.

Prije svega obrazlaže se sama motivacija za postizanjem jedinstvenog pristupa raznorodnim sustavima poslovanja relacijskih baza podataka. Među ostalim se ukratko opisuju kako nedostatci komercijalnih rješenja, tako se i obrazlaže važnost samih relacijskih baza podataka, kako sasvim općenito tako i u okviru informacijske infrastrukture. Drugim riječima, obrazlaže se zašto je baš ovo područje posebno zanimljivo kao primjer implementacije prototipa modela posredničkog reda.

Nakon toga opisuje se osnovna struktura samog prototipa modela, te se ista uspoređuje sa arhitekturom posredničkog reda predloženom kroz prethodna poglavlja. Može se zaključiti da je model za pristup raznorodnim sustavima poslovanja relacijskih baza podataka samo jedno pojednostavljenje općenitijeg modela predloženog kroz prethodna poglavlja, koji se uvijek može proširiti kako bi se pristupalo i podacima koji nisu u nekoj relacijskoj bazi. Također se ukratko i opisuje sam rad prototipa modela odnosno protok podataka kroz aplikacijski poslužitelj, te se obrazlaže odabir tehnologije na kojoj je i zasnovan cijeli sustav.

Zatim se opisuje i sam ispitni poligon raznorodnih sustava poslovanja relacijskih baza podataka, nabrajaju se točne verzije nad kojim je pristup testiran, te je dana i sama shema iz koje se najbolje vidi kako je istim s jedne strane pristupao sam aplikacijski poslužitelj, te s druge strane kako se pristupalo samom aplikacijskom poslužitelju.

Nakon toga opisuju se četiri glavne grupe problema na koje se naišlo prilikom implementacije prototipa modela, te načini na koje su isti riješeni odnosno djelomično riješeni [113]. To su problemi vezani za rukovanje tablicama rezultata upita, problemi vezani za internacionalizaciju -lokalizaciju, problemi vezani za indeksiranje i integritet podataka te problemi vezani za raznolikost tipova podataka.

Na kraju poglavlja opisuje se izvedba jedne Internet aplikacije, virtualne knjižnice na mreži [112]. Ta Internet aplikacija koristeći prototip modela razvijenog i implementiranog kroz ovo poglavlje, pokazuje kako se u praksi koristeći taj prototip vrlo jednostavno može implementirati jedinstven pristup relacijskim bazama podataka koje se nalaze pod raznorodnim sustavima poslovanja relacijskih baza podataka.

5.1. Motivacija za uniformnim pristupom

Na svakoj kako globalnoj tako i lokalnoj računalnoj mreži, kao što su primjerice Internet i intranet, odnosno sama informacijska infrastruktura, može se nalaziti ne samo mnogo različitih relacijskih baza podataka, već se i istim može upravljati različitim sustavima poslovanja relacijskim bazama podataka kao što su primjerice DB2, Oracle, SQL Server, Sybase, Access, i drugi.

Iako postoji standardni jezik za pristup relacijskim bazama podataka SQL (Structured Query Language), razlike u interpretaciji odnosno različitim implementacijama su tolike da isti ne samo da ne garantira, već nije niti dovoljan da bi se osigurao uniformni pristup. Uz sam zajednički jezik za rad sa bazom, potrebno je i neko zajedničko programsko sučelje odnosno API (Application Programming Interface). Ovdje je također problem u tome što dok svaki proizvođač sustava poslovanja relacijskih baza podataka nudi adekvatne alate za razvoj programske podrške za njegov sustav, ti isti alati u praksi se pokazuju praktički neupotrebljivi za bilo koji sustav poslovanja relacijskih baza podataka drugog proizvođača.

Konkretno se u to može uvjeriti na jednom primjeru vezanim za ADO. To je proizvod Microsofta, i namijenjen je prije svega kako bi se iz ASP-a osigurao pristup bazama podataka koristeći tehnologiju OLE DB. Kao prvi korak, instalirani su razni sustavi poslovanja relacijskih baza podataka, te napravljena vrlo jednostavna baza podataka koja se sastojala od samo jedne tablice i to za svaki od njih. Nakon toga je u ASP-u napisana skripta koja koristeći ADO čita vrijednosti zapisane u samoj tablici. Iako je zahtjev bio trivijalno jednostavan, pokazalo se da je pristup funkcionirao samo kod Microsoftovih sustava poslovanja relacijskih baza podataka SQL Server i Access te kod sustava poslovanja relacijskih baza podataka IBM DB2, dok kod ostalih pristup nije funkcionirao, i to među ostalim kod Oracle, Sybase, Informix i NCR Teradata. Kod jednog drugog testa, pristup nije niti sa IBM DB2 ispravno funkcionirao.

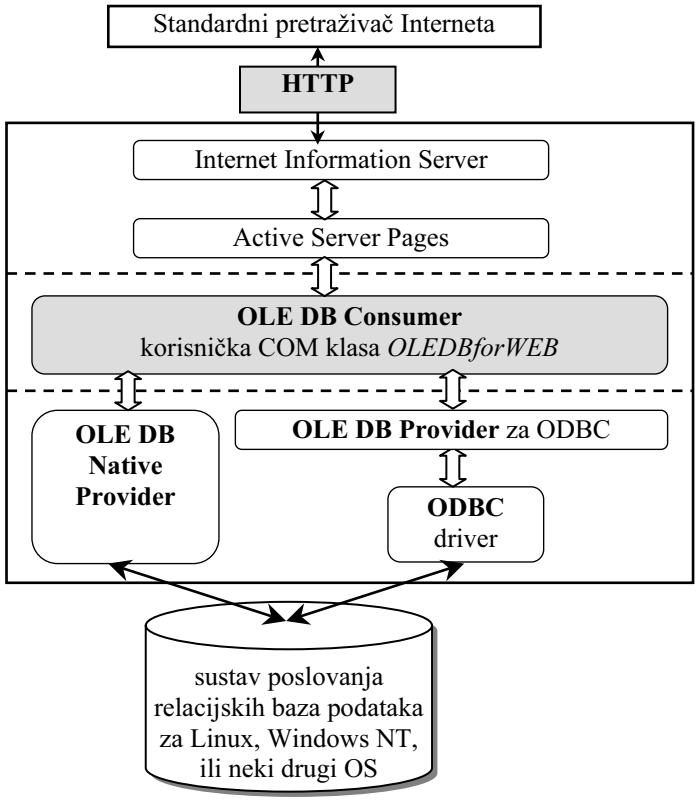
Najveći problem kod ADO objekata je u tome što sam izvorni kod nije dostupan (Microsoft samo dozvoljava da se besplatno sa mreže skine izvršni kod). Stoga isti nije moguće prilagoditi kako bi radio i sa raznorodnim sustavima poslovanja relacijskih baza podataka drugih proizvođača.

Osigurati uniforman pristup, važno je kako prilikom pristupa već postojećim aplikacijama, tako i prilikom izgradnje novih aplikacija. Naime ukoliko gradimo neku WWW aplikaciju odnosno Internet stranice koje pristupaju nekoj bazi podataka, u svakom slučaju je velika prednost ako će se te iste stranice moći koristiti i kad jedan sustav poslovanja relacijskih baza podataka zamijenimo sa drugim. To je posebno važno pošto dosta korisnika već ima neki sustav poslovanja relacijskih baza podataka. Ukoliko bi neka WWW aplikacija bila napravljena za neki drugi sustav, ili bi korisnik morao kupiti taj drugi sustav, ili bi se aplikacija morala ponovo pisati. Ovdje je važno napomenuti da alati različitih proizvođača sustava poslovanja relacijskih baza podataka često i kad su namijenjeni za istu svrhu, kao recimo spajanje baza na Internet, često koriste različite skriptne jezike odnosno različita proširenja HTML-a, te stoga redovito čak niti dijelovi neke WWW aplikacije nisu od koristi kad se piše WWW aplikacija za neki drugi sustav poslovanja relacijskih baza podataka, pa stoga praktički i nemamo ponovnu iskoristivost već napisanog koda.

Stoga iz svih ovih razloga proizlazi da je posebno zanimljivo na konkretnom primjeru upravo pokazati kako napisati posrednički red koji će osigurati uniformnost pristupa raznorodnim sustavima poslovanja relacijskih baza podataka. Posebno što s jedne strane same relacijske baze podataka tradicionalno imaju izuzetno veliku važnost stoga što garantiraju sigurnost, pouzdanost odnosno ACID (engl. Atomic, Consistent, Isolated, and Durable properties of transactions) svojstva svih transakcija, a s druge strane što i sam Internet svakim danom ima sve veću važnost odnosno koristi ga sve veći broj ljudi. Upravo stoga je i razlika u odnosu na neke starije pristupe [3], u izgradnji sučelja upravo za Internet odnosno mrežu.

5.2. Osnovna arhitektura prototipa posrednika

Osnovna arhitektura prototipa modela, aplikacijskog poslužitelja, prikazana je na Slici 50. Kao što se sa slike može primijetiti, ista je izgrađena za Windows NT 4.0 Server platformu koristeći OLE DB, COM i ASP, odnosno izgrađena je u skladu sa arhitekturom posredničkog reda kako je prikazano na Slici 27. Osnovna razlika između ovog pojednostavljenog modela i arhitekture prikazane na Slici 16 odnosno na Slici 27 je u tome što je podred objekata jezgre posredničkog reda za sada ostavljen prazan, te što se umjesto ADO objekata koristiti kroz ovaj rad projektirana COM klasa *OLEDBforWEB*.



Slika 50: Osnovna arhitektura prototipa modela za pristup raznorodnim sustavima poslovanja relacijskih baza podataka.

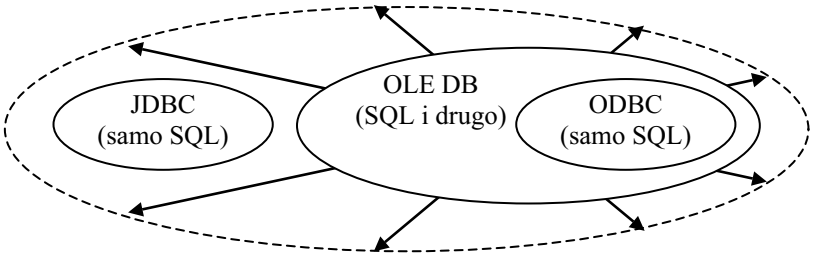
Ovdje je važno napomenuti da iako se sam posrednički red nalazi na Windows NT Server 4.0 platformi, kako korisnici sa svojim pretraživačima Interneta tako i same baze podataka zajedno sa pripadnim sustavom poslovanja mogu se nalaziti i na bilo kojem drugom operacijskom sustavu. To što su same Internet stranice napisane u ASP-u ne predstavlja ograničenje na pretraživače Interneta, pošto se sam ASP kod izvršava na strani aplikacijskog poslužitelja, te se klijentima šalje čist HTML. Tako pretraživač može biti i neki Microsoftov kao npr. Internet Explorer, ili od nekog drugog proizvođača kao npr. Netscape Communicator/Navigator i drugi. Isto tako i OLE DB Provider-i i ODBC (Open DataBase Connectivity) driver-i mogu direktno pristupati svojim sustavima poslovanja relacijskih baza podataka, bez obzira na kojoj su platformi isti.

Nakon što klijent otvori neku ASP stranicu, IIS je izvršava kako bi se generirao HTML ili po potrebi DHTML. Prilikom tog procesa, iz ASP-a se može pristupiti korisnički projektiranoj COM klasi *OLEDBforWEB* kako bi se preko nje pristupilo nekom sustavu poslovanja relacijskih baza podataka. Sam *OLEDBforWEB* u tu svrhu može direktno pristupiti nekom OLE DB Provider-u, a isto tako može i indirektno preko OLE DB Provider-a za ODBC driver-e pristupati i ODBC

driver-ima. Prednost prvog pristupa je u tome što je to prirodan pristup u okviru OLE DB okruženja, što znači da je najbrži i da nam pruža najviše mogućnosti. Prednost drugog pristupa je u tome što je ODBC još uvijek bitno bolje podržan nego OLE DB, pošto je to bitno starija tehnologija koja već je široko prihvaćena te za skoro svaki sustav poslovanja relacijskih baza podataka gotovo redovito postoji i pripadni ODBC driver.

Za OLE DB tehnologiju, odlučeno je zbog sljedećih razloga, odnosno glavnih prednosti u odnosu na ODBC i JDBC [122][157] (Java DataBase Connectivity):

- API-i za pristup podacima kao što su ODBC i JDBC zahtijevaju da sam izvor podataka podržava odnosno pruža pristup podacima isključivo kroz SQL, što je nedostatak ukoliko gradimo sustav koji u budućnosti želimo proširiti da može pristupati i drugim izvorima podataka;
- OLE DB pojednostavnjuje pristup za jednostavne izvore podataka zahtijevajući od njih jedino da prikazuju svoje podatke kao tablice, odnosno osim SQL izvora podataka podržava i druge, kako je prikazano na Slici 51, te je samim tim i bolji kandidat ukoliko nam treba što općenitiji pristup kao što je kod nas i slučaj;
- OLE DB koristi infrastrukturu COM, što je važno budući da moderan razvoj programske podrške danas zahtjeva ne samo korištenje objektno orijentiranog pristupa, nego i komponentno temeljenog;
- klasa COM može biti napisana u bilo kojem jeziku koji podržava binarni standard COM, što uključuje jezike kao što su Visual C++, Visual J++ i Visual Basic [143][158], a to je važno pošto se unutar većine operacijskih sustava iz jezika C++ može pozvati bilo koji sistemski API poziv u bilo kojem trenutku [72], te stoga što je samim jezikom C++ među ostalim podržan i objektno orijentiran i komponentno temeljen razvoj programske podrške.



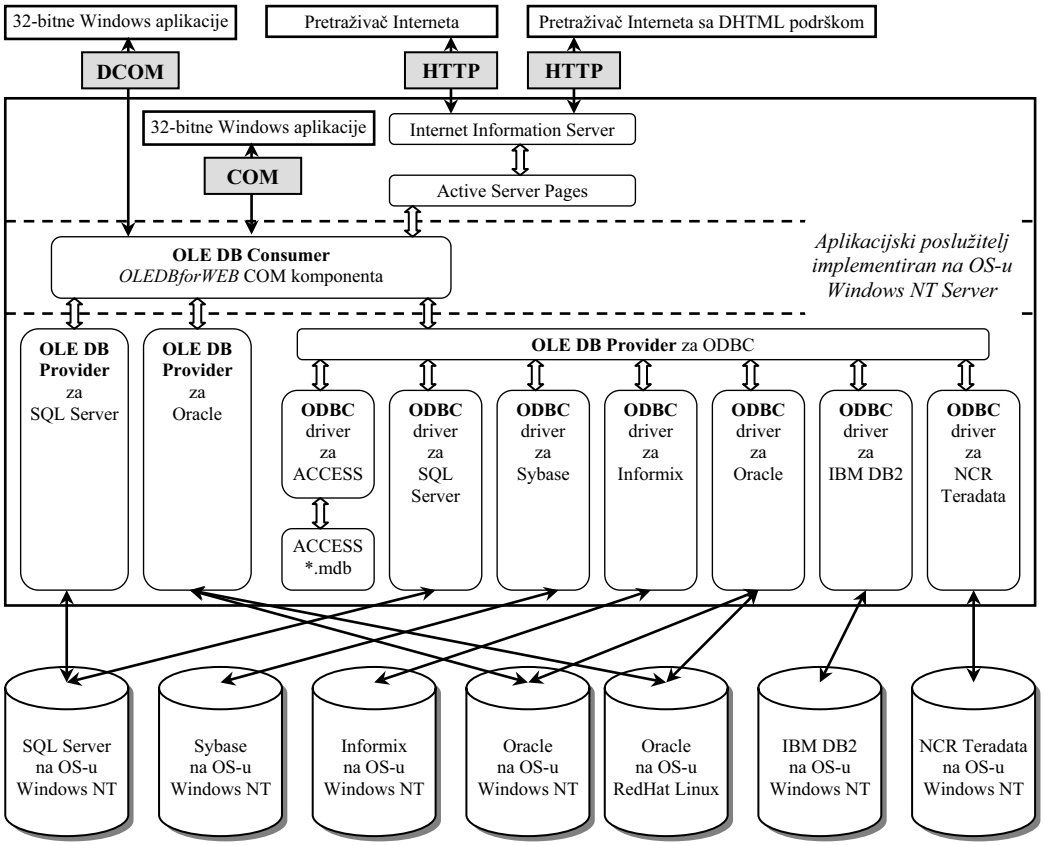
Slika 51: Odnos OLE DB tehnologije te ODBC-a i JDBC-a.

I na kraju, uz tehnologiju potrebno je još odabrati i sam jezik implementacije. Iako Java ima puno prednosti u odnosu na druge jezike i to posebno prilikom izgradnje distribuiranih aplikacija [109], te iako se u zadnje vrijeme dosta radi kako bi se proširila funkcionalnost tog jezika [27][43], u našem slučaju pošto posrednički red trebamo implementirati na samo jednoj platformi, prenosivost koda nije važna tako da je zbog većih mogućnosti i bitno veće brzine izvođenja i od Visual J++ i od Visual Basic koda, Visual C++ najbolji izbor za implementaciju, kako OLE DB COM klasa općenito, tako i same COM klase *OLEDBforWEB*.

5.3. Ispitni poligon

Na Slici 52, prikazan je ispitni poligon različitih sustava poslovanja relacijskih baza podataka koji su korišteni za ispitivanje korisnički projektirane COM klasa *OLEDBforWEB*:

- IBM DB2 Universal Database Version 5.2.0 Enterprise Edition za Windows NT Server 4.0 SP4;
- Oracle8 Release 8.0.3 Database za Windows NT Server 4.0 SP4;
- Oracle8 Release 8.0.5 Database za Red Hat Linux release 5.2 (Apollo) - Kernel 2.0.36;
- Microsoft SQL Server 6.50.201 SP1 Database za Windows NT Server 4.0 SP4;
- Microsoft Access 97 Database za Windows NT Server 4.0 SP4;
- Sybase Adaptive Server 11.5 Enterprise Database za Windows NT Server 4.0 SP4;
- Informix Dynamic Server 7.30.TC3 Database za Windows NT Server 4.0 SP4;
- NCR Teradata Version 02.00.01 Database za Windows NT Server 4.0 SP4.



Slika 52: Ispitni poligon za raznorodne sustave poslovanja relacijskih baza podataka.

Uz to što se pokazalo da sami raznorodni sustavi poslovanja relacijskih baza podataka ne moraju biti niti na istom računalu niti na istom operacijskom sustavu kao i posrednički red, kao niti sama pripadajuća baza odnosno baze, također je i pokazano da se i svakom od njih može pristupiti kako različitim pretraživačima Interneta, tako i direktno sa istog računala gdje se nalazi korisnički red koristeći COM odnosno sa nekog računala u mreži koristeći DCOM.

5.4. Problemi vezani za rukovanje tablicama rezultata upita

Problemi vezani za rukovanje tablicama rezultata upita spadaju u prvu klasu problema koji su se javili prilikom implementacije korisnički projektirane COM klasa *OLEDBforWEB*. To su prije svega problemi vezani za rukovanje tablicama koje su rezultati nekih SQL upita, kao što je primjerice SELECT.

Kad se koristi ADO prilikom izrade Internet stranica za pristup raznorodnim sustavima poslovanja relacijskih baza podataka, neke metode kao npr. *AbsolutePage* i *PageCount* se ne mogu koristiti ukoliko se koristi Sybase, Informix, Oracle ili NCR Teradata ODBC driver. ADO prijavljuje kako dotični driver nije implementiran kako bi podržao te metode te da se iste ne mogu pozivati iz ASP aplikacije. Dodatno se kod korištenja Intersolv ODBC driver-a za Informix javlja problem, da metode *BOF* i *EOF* nisu dostupne, tako da čak nije niti moguće znati koliko redaka je zahvaćeno nekim SQL upitom. Posljedica toga je da čak i u slučaju kad neka tablica u bazi sadržava samo jedan redak, isti se kod ispisa ne nekoj stranici ponavlja dok god se ne dosegne predefinirani limit.

API za rukovanje tablicom odnosno tablicama rezultata SQL upita je ključan prilikom izrade Internet stranica. Budući da svi rezultati upita obično ne mogu stati na samo jednu stranicu, potrebno je dotične rezultate prezentirati korisniku kroz nekoliko dinamički izgeneriranih stranica međusobno povezanih linkovima. To je vrlo teško napraviti unutar ASP-a ukoliko se ne može koristiti neki prikladan API. Posebno to nije moguće ukoliko čak niti ne znamo koliko smo redaka dobili iz baze nekim upitom. Dodatno i zbog tih razloga, *OLEDBforWEB* je implementiran kako bi se direktno spojio na OLE DB Provider-e, te kako bi se time izbjegli ADO objekti gdje nije dostupan izvorni kod.

Nakon izvršavanja nekog SQL upita, *OLEDBforWEB* interno kreira tablicu ukoliko je tablica rezultat upita, te u tom slučaju istu dodatno i spremi u pričuvnu memoriju. Ukoliko se direktno spaja na OLE DB Provider-e preko COM klase *OLEDBforWEB*, primijećeno je da sve OLE DB funkcije za rukovanje tablicama funkcioniraju dobro ukoliko se na sam *OLEDBforWEB* spajamo direktno koristeći COM/DCOM, no ne i kad se na sam *OLEDBforWEB* spajamo iz ASP-a; problem ostaje jedino kod Intersolvovog ODBC driver-a za Informix. To je vjerojatno posljedica različitog konteksa unutar kojeg se sam driver nalazi za vrijeme izvođenja. No čak i u slučaju tog driver-a koji za razliku od svih drugih očito nije jednako dobro napisan, problemu je moguće doskočiti na relativno jednostavan način. *OLEDBforWEB* prestaje uzimati nove retke ukoliko se dogodi da su dva uzastopna retka identična i u tom slučaju zadnjeg zanemari. Poseban slučaj je ukoliko su uzeta samo dva retka i zaključeno je da su identični, tada se dodatno provjerava da li je prvi redak potpuno prazan i u tom slučaju se zaključuje da naš upit nije pronašao ništa u bazi. Koristeći ovakav pristup, može se zaključiti da je ova klasa problema u potpunosti riješena na adekvatan način. Jedino na što je važno paziti prilikom ovog pristupa, jest da se prilikom svakog SQL SELECT upita moraju navesti i elementi tablice koji sačinjavaju primarni ključ, budući da se tako ne može dogoditi da sam algoritam "preskoči" retke koje inače ne bi trebao.

Dok koristeći ovaj pristup odnosno korisnički projektiranu COM klasu *OLEDBforWEB* ASP aplikacija može koristiti API vrlo sličan kao i onaj od samog ADO-a, ovi principi osiguravaju transparentnost u rukovanju tablicama prilikom pristupa različitim sustavima poslovanja relacijskih baza podataka.

5.5. Problemi vezani za internacionalizaciju/lokalizaciju

Problemi vezani za internacionalizaciju/lokalizaciju spadaju u drugu klasu problema koji su se javili prilikom implementacije korisnički projektirane COM klasa *OLEDBforWEB*. Tu su prije svega obuhvaćeni problemi vezani za različite znakove pojedinih pisama odnosno kodne stranice, kao i problemi vezani za specifičnosti zapisa datuma i brojeva te vezani za specifičnosti lokalnih valuta.

Što se tiče kodnih stranica, jedino je UNICODE konačno rješenje koje bi u potpunosti zadovoljilo. Dok su COM i *OLEDBforWEB* UNICODE orijentirani gdje je svaki znak zapisan kao 16-bitna vrijednost, svi testirani sustavi poslovanja relacijskih baza podataka za sada prvenstveno koriste kodne stranice, zastarjeli način zapisivanja znakova koristeći samo 8-bitne vrijednosti. Stoga je prilikom ulaza/izlaza potrebna konverzija. COM klasa *OLEDBforWEB* rješava taj problem tako da čita iz Windows registratora lokalni identifikator LCID (LoCal IDentifier), te vrši konverziju koristeći kodnu stranicu koja je dodijeljena tom specifičnom LCID-u. Ovaj pristup testiran je tako što su regionalne postavke postavljene na Hrvatsku, te su se nakon toga različiti sustavi poslovanja relacijskih baza podataka ispitali u radu sa Hrvatskim znakovima koji nisu dio Engleske abecede (č, ć, š, đ, ž, Č, Ć, Š, Đ i Ž). Procedura generalno funkcionira dobro, i problemi su nastupili jedino kod Intersolvovog ODBC driver-a za Informix, i to samo kod znakova ž i Ž. Ovdje je posebno zanimljivo napomenuti kako pristup funkcionira čak i unatoč tome što neki sustavi poslovanja relacijskih baza podataka uopće niti nisu podržavali neku od kodnih stranica koja sadrži naše znakove.

Kod brojeva, neki koriste znakove kao što su jednostruki navodnik i točka za odvajanje svake tri znamenke, a općenito i za samu decimalnu točku neki koriste zarez, a neki točku. Ovakve raznolike notacije stvaraju probleme i to posebno prilikom komunikacije s bazom. Čak i ukoliko se ne koristi niti jedan znak za odvajanje svake tri znamenke, ukoliko se kao decimalna točka koristi zarez a ne točka, postoji problem prilikom rada sa bazom. Naime sam zarez se koristi prilikom odvajanja pojedinih parametara SQL upita, te se stoga isti ne može koristiti i kao decimalna točka. U protivnom, svaki realan parametar baza bi protumačila kao dva cjelobrojna, i javila bi poruku o grešci. Stoga se u ovom radu kako bi se osigurala transparentnost u radu s raznorodnim sustavima poslovanja relacijskih baza podataka, kao decimalna točka koristi upravo točka, dok se sve ostale znamenke pišu zajedno (ne odvajaju se svake tri niti sa nekim znakom niti sa samim razmakom). Ovdje je važno napomenuti da ova pravila vrijede samo prilikom komunikacije između COM klase *OLEDBforWEB* i sustava poslovanja relacijskih baza podataka kao i između korisničkog programa i *OLEDBforWEB*-a, dok korisnički program krajnjem korisniku brojeve uvijek može prikazati u bilo kojem obliku.

Kod datuma i lokalnih valuta, problem je u tome što ne postoji standardni način kojeg bi se pridržavali svi raznorodni sustavi poslovanja relacijskih baza podataka kao niti neki zajednički tip podatka. Stoga se za interni zapis istih koriste realni brojevi, koji su kao tip FLOAT podržani od svih ispitivanih sustava. To ne predstavlja nikakav problem, pošto se svaki iznos neke valute može jednoznačno zapisati kao realni broj, kao što se i svakom trenutku u vremenu također može jednoznačno pridružiti neki realan broj. Preslikavanje između iznosa u valuti i realnog broja je očito, recimo iznosu od 54 kn i 36 lp možemo pridružiti realan broj 54.36 iz kojeg opet možemo rekonstruirati da se radilo o 54 kn i 36 lp. Kod datuma i vremena, pretvorba u realan broj vrši se tako da se godine pomnože s 10000, tome se doda broj mjeseci pomnožen sa 100 i broj dana, te broj sati podijeljen sa 100, broj minuta podijeljen sa 10000 te broj sekunda podijeljen sa 1000000 gdje broj sekunda može biti bilo koji realan broj. Tako je recimo 19991014.163045 datum 14^{ti} listopad 1999. godine u 16 sati 30 minuta i 45 sekundi. Ovakvom notacijom, direktno se rješava i sam problem 2000-te godine. Kao i u slučaju samih cjelih i realnih brojeva, i ovdje korisnički program može krajnjem korisniku prikazati datume i valute u bilo kojem obliku.

5.6. Problemi vezani za indeksiranje i integritet podataka

Problemi vezani za indeksiranje i integritet podataka spadaju u treću klasu problema koji su se javili prilikom implementacije korisnički projektirane COM klasa *OLEDBforWEB*. U svrhu testiranja, koristila se SQL skripta sa Ispisa 13, dok su u Tablici 13 dati sami rezultati testiranja. Lijevi stupac sadrži broj testa, dok prvi redak sadrži imena sustava poslovanja relacijskih baza podataka nad kojim su sami testovi izvršeni.

```
DROP TABLE Titles
DROP TABLE Publishers
DROP TABLE Authors
CREATE TABLE Authors(
    AuthorID integer not null,
    FirstName char(100) not null,
    LastName char(100) not null,
    Birthdate integer not null,
    email char(100) not null,
    www char(100) not null,
    PRIMARY KEY (AuthorID))
CREATE TABLE Publishers(
    PublisherID integer not null,
    CompanyName char(100) not null,
    Address char(100) not null,
    Zip integer not null,
    City char(100) not null,
    State char(100) not null,
    Country char(100) not null,
    Telephone char(100) not null,
    Fax char(100) not null,
    email char(100) not null,
    www char(100) not null,
    PRIMARY KEY (PublisherID))
CREATE TABLE Titles(
    ISBN char(100) not null,
    PublisherID integer not null,
    AuthorID integer not null,
    YearPublished integer not null,
    PaperTitle char(100) not null,
    PRIMARY KEY (ISBN),
    FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID))
CREATE UNIQUE INDEX idx_1 ON Authors(FirstName)
```

Ispis 13: SQL skripta za ispitivanje podržavanja indeksiranja i integriteta podataka.

U prvom testu, izvršena je skripta sa Ispisa 13, ali bez podcrtanog dijela koda, kako bi se pokazalo da se skripta može uspješno izvršiti kod svih nabrojanih sustava poslovanja relacijskih baza podataka, ukoliko se izuzme funkcionalnost vezana za indeksiranje i integritet podataka. Kao što se vidi iz Tablice 13, svi su uspješno prošli prvi test, što znači da su uspješno kreirane tablice *Titles*, *Publishers* i *Authors* zajedno sa svim pripadnim atributima.

U drugom testu, izvršena je cjelokupna skripta sa Ispisa 13 uključujući i podcrtan kod. Rezultati su prikazani u Tablici 13, kako se vidi svi osim Accessa podržavaju da se integritet podataka definira iz samog SQL-a (X u pripadnom polju znači da dotični sustav poslovanja relacijskih baza podataka nije prošao dotični test), te da se kreira jedinstveni indeks nad atributom koji nije niti primaran niti strani ključ.

U trećem i četvrtom testu, pokušan je kreiran indeks nad atributima koji su primarni odnosno strani ključ. Pošto Access nije prošao prvi test, jasno je da nije mogao niti ova dva. No vidimo da niti Informix nije omogućio da se kreira indeks nad primarnim ključem, dok uz Informix nad stranim ključem nisu dopustili da se kreira indeks niti Sybase, niti Oracle, niti NCR Teradata. Jedino SQL Server i DB2 dopuštaju da se kreira indeks, bez obzira da li je neki atribut primaran ključ, strani ključ, ili nije niti primaran niti strani ključ.

Peti test je osnovni test provjere integriteta podataka, pokušalo se je ubaciti neki element u tablicu *Titles* dok su tablice *Authors* i *Publishers* bile prazne. Svi sustavi poslovanja relacijskih baza podataka koji su prošli prvi test su to zabranili kako su i trebali, pošto se u tablicu *Titles* smiju ubacivati samo oni zapisi koji referenciraju bar jednog autora i bar jednog izdavača.

U šestom testu nekoliko redaka je ubačeno u svaku tablicu i nakon toga se pokušalo obrisati tablicu *Authors*. Uz Access, to također nisu spriječili niti Informix, niti DB2, iako su trebali, jer nema smisla zabraniti nekome da obriše jedan ili više redaka neke tablice ako mu je dopušteno da obriše cijelu tablicu.

Tablica 13: Rezultati testiranja podrške indeksiranja i integriteta podataka.

Test:	Access	SQL Server	Sybase	Informix	Oracle	DB2	NCR Teradata
1							
2	X						
3	X			X			
4	X		X	X	X		X
5	X						
6	X			X		X	

Ova heterogenost posljedica je implementacije samih sustava poslovanja relacijskih baza podataka, i nije je moguće jednostavno izbjeći unutar same COM klase *OLEDBforWEB*. Na žalost, jedini mogući način da se izbjegne ova netransparentnost jest da se integritet podataka posebno implementira unutar same COM klase *OLEDBforWEB*, odnosno same aplikacije, primjerice u ASP-u. Problem je u tome što je to dugotrajan i vrlo zahtjevan postupak podložan greškama. Upravo stoga i je predviđeno da sam integritet podataka bude podržan od sustava poslovanja relacijskih baza podataka, dok samo indeksiranje također nema smisla implementirati zasebno od istog.

Za razliku od prve dvije grupe problema, ovu nije moguće u potpunosti riješiti već se mora prilaziti nekim kompromisnim rješenjima. Osnovni problem nije u pogrešno izabranoj arhitekturi ili u nedostatku OLE DB tehnologije, već se nalazi dublje u samim sustavima poslovanja relacijskih baza podataka i jedino što se može napraviti u smislu idealnog rješenja jest pričekati dok svi, ili bar važniji proizvođači raznorodnih sustava poslovanja relacijskih baza podataka, to ne riješe u okviru samih njih na zadovoljavajući odnosno transparentan način.

5.7. Problemi vezani za raznolikost tipova podataka

Problemi vezani za raznolikost tipova podataka spadaju u četvrtu klasu problema koji su se javili prilikom implementacije korisnički projektirane COM klasa *OLEDBforWEB*. U svrhu testiranja, koristila se SQL skripta sa Ispisa 14, dok su sami rezultati testiranja dati u tablicama 14, 15 i 16. Jedino što se mijenjalo kod SQL skripte prikazane na Ispisu 14 jest podcrtani dio koda odnosno sam tip podatka koji se ispitivao, te je u tu svrhu isti ciklički bio ubacivan umjesto "CHAR(254)".

Tablica 14: ANSI SQL tipovi podataka.

Tip podatka:	Access	SQL Server	Sybase	Informix	Oracle	DB2	NCR Teradata
CHAR	STR	STR	STR	STR	STR	STR	STR
CHAR(n)	0 ≤ n ≤ 255 STR	1 ≤ n ≤ 255 STR	1 ≤ n ≤ 255 STR	1 ≤ n ≤ 32767 STR	1 ≤ n ≤ 2000 STR	1 ≤ n ≤ 254 STR	1 ≤ n ≤ 32000 STR
VARCHAR	STR	STR	STR	X	X	X	X
VARCHAR(n)	0 ≤ n ≤ 255 STR	1 ≤ n ≤ 255 STR	1 ≤ n ≤ 255 STR	1 ≤ n ≤ 255 STR	1 ≤ n ≤ 4000 STR	1 ≤ n ≤ 4000 STR	1 ≤ n ≤ 32000 STR
CLOB	X	X	X	X	BYTES (X)	X	X
CLOB(n)	X	X	X	X	X	1 ≤ n < 2**20 BYTES	X
DECIMAL	X	NUMERIC	NUMERIC	R8	NUMERIC	NUMERIC*	NUMERIC
DECIMAL(n)	X	1 ≤ n ≤ 28 NUMERIC	1 ≤ n ≤ 38 NUMERIC	1 ≤ n ≤ 32 R4(n≤8) R8(n ≥9)	1 ≤ n ≤ 38 NUMERIC	1 ≤ n ≤ 31 NUMERIC*	1 ≤ n ≤ 18 NUMERIC
DECIMAL(n,m)	X	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC*	0 ≤ m ≤ n NUMERIC
NUMERIC	R8	NUMERIC	NUMERIC	R8	NUMERIC	NUMERIC*	NUMERIC
NUMERIC(n)	X	1 ≤ n ≤ 28 NUMERIC	1 ≤ n ≤ 38 NUMERIC	1 ≤ n ≤ 32 R4(n≤8) R8(n≥9)	1 ≤ n ≤ 38 NUMERIC	1 ≤ n ≤ 31 NUMERIC*	1 ≤ n ≤ 18 NUMERIC
NUMERIC(n,m)	X	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC	0 ≤ m ≤ n NUMERIC*	0 ≤ m ≤ n NUMERIC
BIT	BOOL	BOOL	BOOL	X	X	X	UI1
TINYINT	X	UI1	UI1	X	X	X	I1
SMALLINT	I2	I2	I2	I2	NUMERIC	I2	I2
INTEGER	I4	I4	I4	I4	NUMERIC	I4	I4
BIGINT	X	X	X	X	X	STR	NUMERIC
REAL	R4	R4	R4	R4	R4 (R5)	R4	R5
FLOAT	R8	R8	R8	R8	R8	R8	R8
DOUBLE PRECISION	X	R8	R8	R8	R8	R8	R8
DATE	DBTIMESTAMP	X	X	DBDATE	DBTIMESTAMP	DBDATE	DNDATE
TIME	DBTIMESTAMP	X	X	X	X	DBTIME	DBTIME
TIMESTAMP	DBTIMESTAMP	BYTES	X	X	X	DBTIMESTAMP	STR
BINARY	BYTES	BYTES	BYTES	X	X	X	BYTES
BINARY(n)	0 ≤ n ≤ 255 BYTES	1 ≤ n ≤ 255 BYTES	1 ≤ n ≤ 255 BYTES	X	X	X	1 ≤ n ≤ 32000 BYTES
VARBINARY	BYTES	BYTES	BYTES	X	X	X	X
VARBINARY(n)	0 ≤ n ≤ 255 BYTES	1 ≤ n ≤ 255 BYTES	1 ≤ n ≤ 255 BYTES	X	X	X	1 ≤ n ≤ 32000 BYTES
BLOB	X	X	X	X	BYTES (*)	X	X
BLOB(n)	X	X	X	X	X	1≤n<2**20 BYTES	X

Tablica 15: Access tipovi podataka koji nisu u prethodnoj tablici.

Tip podatka:	Access	SQL Server	Sybase	Informix	Oracle	DB2	NCR Teradata
BYTE	UI1	X	X	BYTES*	X	X	BYTES
BYTE(n)	X	X	X	X	X	X	$1 \le n \le 32000$ BYTES
COUNTER	I4	X	X	X	X	X	X
CURRENCY	CY	X	X	X	X	X	X
DATETIME	DBTIMESTAMP	DBTIMESTAMP	DBTIMESTAMP	X	X	X	X
DOUBLE	R8	X	X	X	X	R8	R8
GUID	BYTES	X	X	X	X	X	X
LONG	I4	X	X	X	STR (*)	X	X
LONGBINRY	BYTES*	X	X	X	X	X	X
LONGTEXT	STR*	X	X	X	X	X	X
SHORT	I2	X	X	X	X	X	X
SINGLE	R4	X	X	X	X	X	X
TEXT	STR	STR*	STR*	STR*	X	X	X
TEXT(n)	$0 \le n \le 255$ STR	X	X	X	X	X	X
MEMO	STR*	X	X	X	X	X	X
NUMBER	R8	X	X	X	R8 (VARNUMERIC)	X	X
NUMBER(n,m)	X	X	X	X	NUMERIC	X	X

Tablica 16: SQL Server tipovi podataka koji nisu u jednoj od prethodne dvije tablice.

Tip podatka:	Access	SQL Server	Sybase	Informix	Oracle	DB2	NCR Teradata
INT	I4	I4	I4	I4	NUMERIC	I4	I4
SMALLDATETIME	X	DBTIMESTAMP	DBTIMESTAMP	X	X	X	X
IMAGE	BYTES*	BYTES*	BYTES*	X	X	X	X
SMALLMONEY	X	CY	CY	X	X	X	X
MONEY	CY	CY	CY	NUMERIC	X	X	X

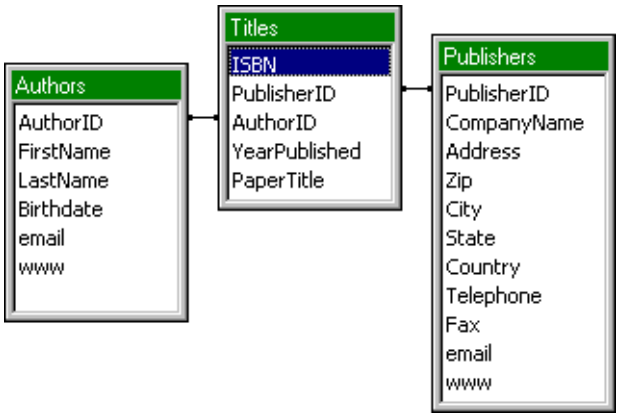
```
DROP TABLE DataTypes
CREATE TABLE DataTypes(dt CHAR(254))
SELECT * FROM DataTypes
```

Ispis 14: SQL skripta za ispitivanje podržavanja pojedinih tipova podataka.

Ove probleme nije moguće jednostavno riješiti iz istog razloga kao i probleme iz prethodne grupe. Ukoliko je cilj postići transparentnost u pristupu, jedino što preostaje jest koristiti samo one tipove podataka koji su podržani od svih sustava poslovanja relacijskih baza podataka. Iako je takvih relativno malo, upravo oni se najčešće koriste, tako da je moguće u praksi izrađivati aplikacije koje će moći transparentno pristupati pripadnim sustavima. To je i osnovni razlog zbog kojeg je odlučeno da se među ostalim datum/vrijeme kao i iznos u nekoj valuti u bazu zapisuju kao realni brojevi, kako je opisano kroz prethodno pod poglavlje.

5.8. Primjer aplikacije: virtualna knjižnica na mreži

Kao primjer jedne jednostavne aplikacije koja koristi prototip modela razvijenog odnosno implementiranog kroz ovaj rad, izabrana je jednostavna virtualna knjižnica na mreži. Sama relacijska baza podataka knjižnice sastoji se od tri tablice, kako je prikazano na Slici 53. Svi atributi tablica su ili tipa *INTEGER* ili tipa *CHAR(n)*, gdje je $1 \le n \le 254$, pošto svi testirani sustavi poslovanja relacijskih baza podataka podržavaju ta dva tipa podataka. Inače, sama relacijska baza podataka (sastavljena od te tri tablice), može se kreirati koristeći SQL skriptu sa Ispisa 13 (dio koji nije podcrtan).



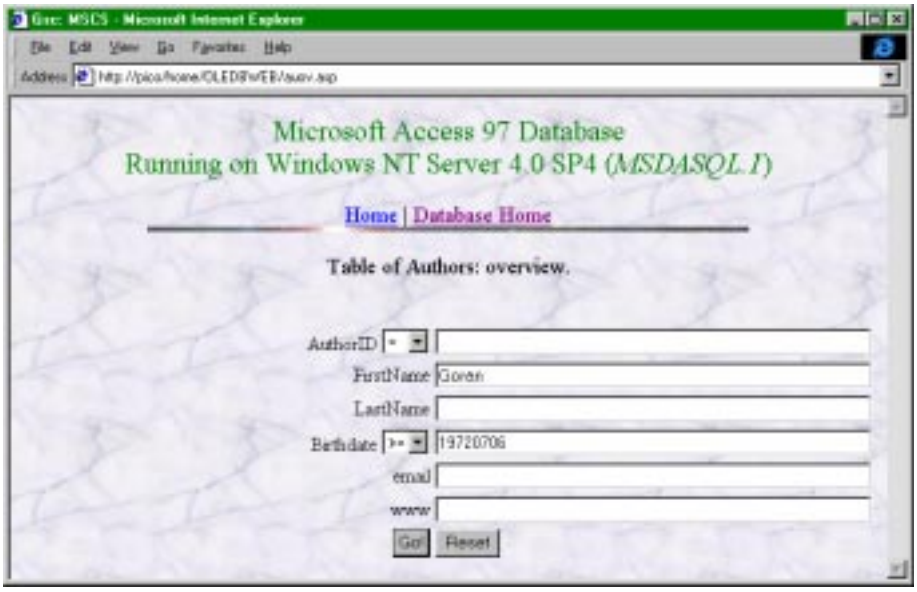
Slika 53: Relacijska baza podataka virtualne knjižnice sastavljena od tri tablice.

Slika 54 prikazuje izgled inicijalne stranice. Iznad horizontalne linije se nalaze često potrebni linkovi koji se prema potrebi dinamički generiraju za vrijeme rada aplikacije, dok je ispod linije predviđen prostor gdje se unose vrijednosti i prikazuju rezultati upita. Svaka od tablica prikazanih na Slici 53 može se pretraživati, može joj se dodati novi zapis, vrijednosti postojećeg zapisa se mogu mijenjati, a bilo koji postojeći zapis se također može i obrisati. Dodatno, moguće je pretraživati i spoj dviju tablica; primjerice mogu se pretraživati autori vezani za nekog izdavača kao i izdavači vezani za nekog autora (spoj preko tablice *Titles*). Opcija *Custom Query* kao i opcije *drop*/*create* neke tablice dodane su u ilustrativne svrhe.

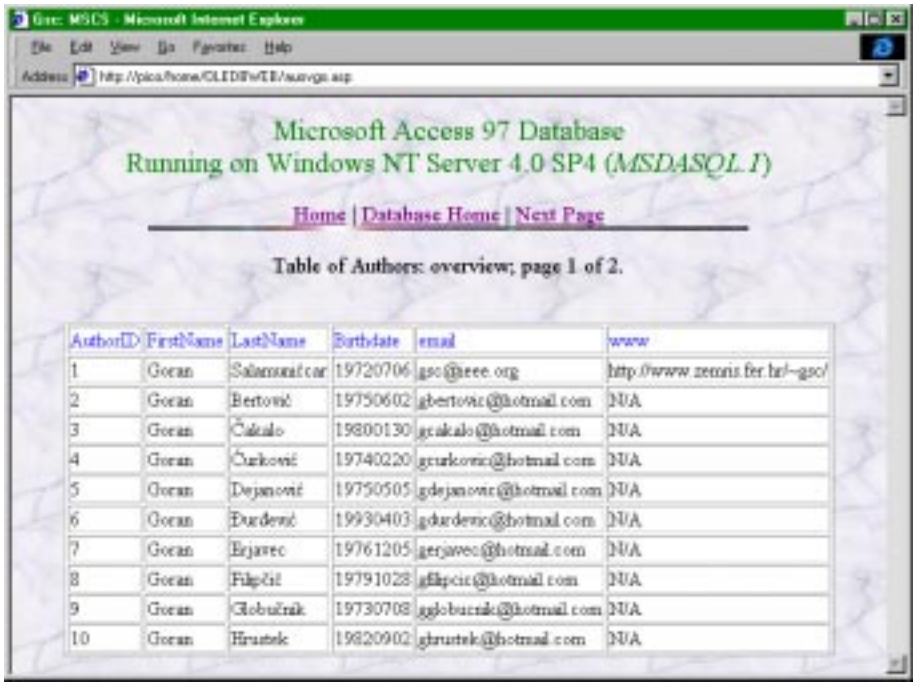
Slika 54: Inicijalna stranica Internet aplikacije s linkovima na sve ostale glavne stranice.

Na Slici 55, prikazan je primjer obrasca za unos/izmjenu vrijednosti, do koje korisnik može doći tako da klikne link *overview* prikazan na Slici 54, desno od *Authors*. U slučaju da se radi o pretraživanju, što je slučaj kod *overview* linkova uz tablice *Authors*, *Publishers* i *Titles*, te kod linkova *Authors releated with some Publisher* i *Publishers releated with some Author*, obrazac je

početno prazan i korisnik unosi kriterij za pretraživanje. U slučaju da se radi o izmjeni vrijednosti, što je slučaj kod *edit existing one* linkova uz tablice *Authors*, *Publishers* i *Titles*, korisnik u obrascu inicijalno dobiva vrijednosti koje su trenutno u bazi, s mogućnošću da svaku od njih izmjeni. U slučaju *add new one* linkova uz tablice *Authors*, *Publishers* i *Titles*, korisnik također dobiva prazan obrazac gdje može upisati vrijednosti koje će se spremiti u bazu. U slučaju sa Slike 55, korisnik je odlučio pretražiti tablicu *Authors*, i to sve autore čije ime je *Goran* i koji su rođeni šestog srpnja 1972 ili nakon toga. Ukoliko korisnik ostavi obrazac prazan, dobiti će sve zapise iz baze, a u protivnom baza se pretražuje samo po onim atributima za koje je korisnik u obrascu specificirao neke vrijednosti. Isto tako kod izmjene nekog zapisa u bazi, mijenjaju se vrijednosti samo onih atributa koje je korisnik u obrascu izmijenio, dok se kod unosa novog zapisa moraju specificirati svi atributi, ili se mogu koristiti inicijalne vrijednosti.



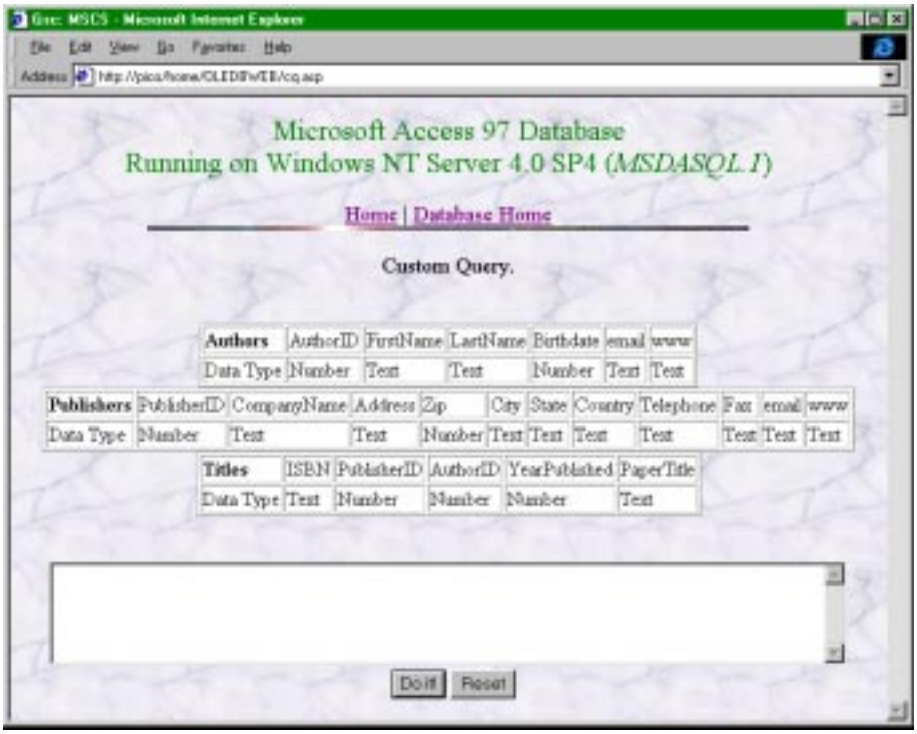
Slika 55: Obrazac gdje korisnik upisuje podatke, u ovom slučaju za pretraživanje baze.



Slika 56: Prikaz rezultata pretraživanja baze.

Na Slici 56, prikazani su rezultati pretraživanja koristeći upit dan na Slici 55. Link *Next Page* zajedno sa linkom *Previous Page* koristi se ukoliko svi rezultati ne stanu na samo jednu stranicu, za prijelaz između istih. Ti linkovi se dinamički generiraju prema potrebi. Kako bi odredili kad je potrebno generirati te linkove, kao i za rukovanje svim tim podacima dobivenim nekim pretraživanjem baze, koristi se API COM klase *OLEDBforWEB*. U takvim prilikama, *OLEDBforWEB* dodatno privremeno spremi u svojoj pričuvnoj memoriji podatke koji su dobiveni iz baze, kako bi se osigurala brža i jednostavnija navigacija među više stranica koje sadrže rezultat nekog upita, te kako bi se koristilo manje procesorske snage.

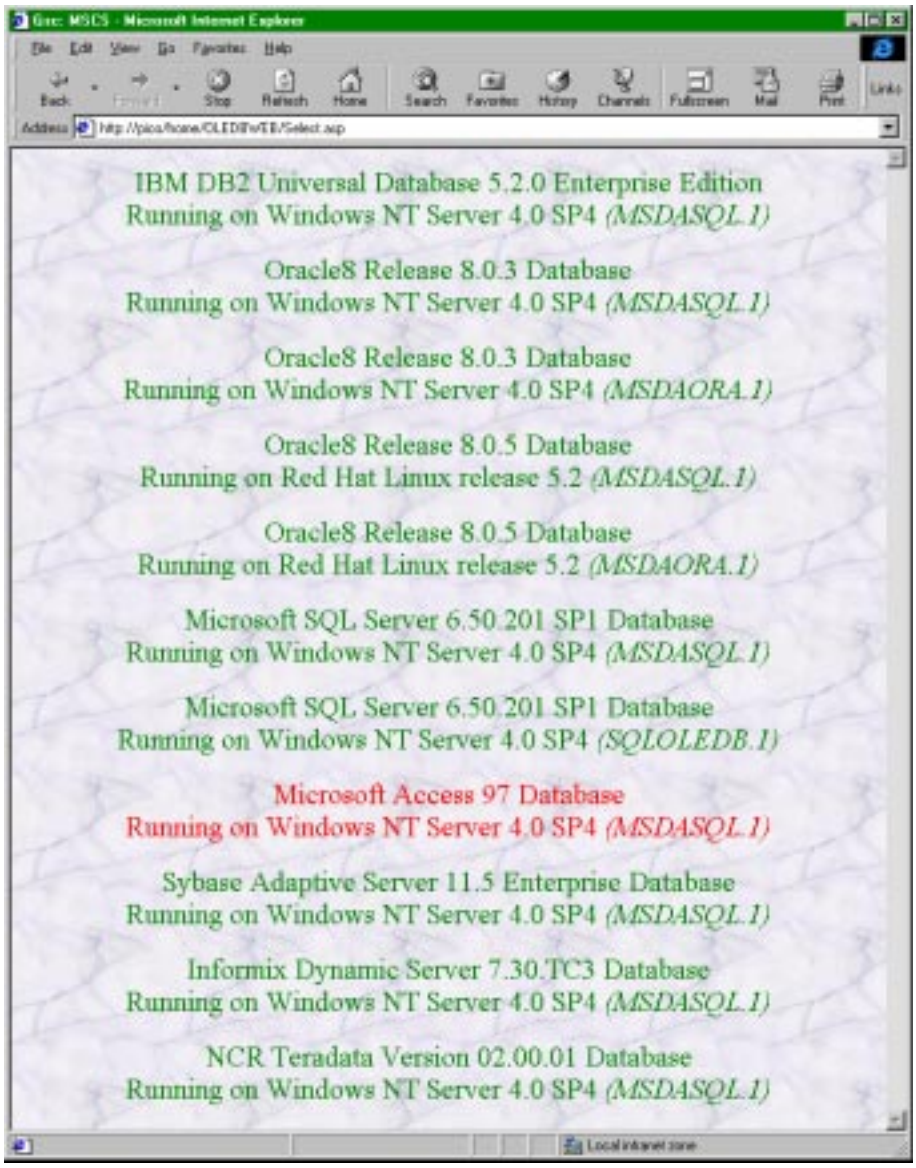
Na Slici 57 prikazana je opcija *Custom Query*, koja omogućava unos bilo koje SQL naredbe. Prije svega, korisnik ima pregled atributa svih tablica baze, a u predviđenom prostoru može unijeti bilo koju željenu SQL naredbu i izvršiti je. Ova opcija je korisna za samog razvojnog inženjera, dok su same Internet stranice još u razvoju i dok nije uz pomoć njih moguće vršiti sve potrebne operacije sa bazom. No ova opcija nije predviđena da je koristi krajnji korisnik, naravno, prije svega iz sigurnosnih razloga pošto se istom ne smije dozvoliti da može nad bazom izvršiti bilo koju operaciju, a i stoga što su za krajnje korisnike primjereniji obrasci pošto se od istih ne može očekivati poznavanje SQL jezika.



Slika 57: Opcija koja omogućava unos bilo koje SQL naredbe.

Dok je inicijalna implementacija napravljena za Access 97 koji se nalazio na istom računalu kao i WWW poslužitelj, kako bi se sustav testirao kao cjelina, na umreženom računalu instalirani su i drugi sustavi poslovanja relacijskih baza podataka kako je prikazano na Slici 52. Pri tome su na njemu napravljene dvije particije. Na prvoj je instaliran Windows NT Workstation 4.0 sa sljedećim raznorodnim sustavima poslovanja relacijskih baza podataka: IBM DB2 verzija 5.2.0 Enterprise Edition, Oracle 8 verzija 8.0.3, Microsoft SQL Server verzija 6.50.201 SP1 [124], Sybase Adaptive Server verzija 11.5 Enterprise Edition [154], Informix Dynamic Server verzija 7.30.TC3 i NCR Teradata verzija 02.00.01. Na drugoj particiji je instaliran Red Hat Linux i sustav poslovanja relacijskih baza podataka Oracle 8 verzija 8.0.5. Ukoliko isključimo Informix, gdje je bilo manjih problema, mrežna Internet aplikacija inicijalno razvijena za Access 97 radila je bez problema nakon prelaska na bilo koji drugi sustav poslovanja relacijskih baza podataka, zahvaljujući tome što smo sve različitosti riješili kroz samu COM klasu *OLEDBforWEB*, dok sam *OLEDBforWEB* Internet

aplikaciji u ASP-u pruža jedinstveno sučelje koje ne ovisi o tome na koji sustav poslovanja relacijskih baza podataka se preko njega spajamo. Dodatno, željeni sustav poslovanja relacijskih baza podataka može se izabrati i za vrijeme dok aplikacija radi, jednostavno selekcijom kako je prikazano na Slici 58. Uz svaki sustav poslovanja relacijskih baza podataka piše i na kojem operacijskom sustavu se dotični nalazi, kako i preko kojeg OLE DB Provider-a se na istog spajamo (vrijednosti u zgradama). Naravno, jedino ograničenje je u tome što se na one koji se nalaze na Windows NT particiji možemo spojiti samo kad su dignuti NT-i, kao i što se na one koji se nalaze na Linux particiji možemo spojiti smo kad je dignut Linux. No to naravno nije ograničenje modela razvijenog kroz ovaj rad, već jednostavno posljedica toga da na jednom računalu istovremeno možemo imati dignut samo jedan operacijski sustav. Da su raznorodni sustavi poslovanja relacijskih baza podataka bili instalirani na različita računala, što je inače i slučaj u praksi, ovog ograničenja naravno ne bi bilo. Samim tim što sustav poslovanja relacijskih baza podataka i WWW poslužitelj ne moraju biti na istom računalu, imamo i mogućnost da između njih smjestimo računalo za kontrolu prometa mrežom, kao što i više WWW poslužitelja može biti spojeno na jedan sustav poslovanja relacijskih baza podataka čime dobivamo i izvrsnu mogućnost proširivosti u slučaju da broj korisnika koje je potrebno posluživati značajno naraste [16][35][78].



Slika 58: Korisnik može izabrati na koji sustav poslovanja relacijskih baza podataka će se spojiti Internet aplikacijom.

6. ANALIZA MODELA ZA PRISTUP RAZNORODNIM PODACIMA

Kroz ovo poglavlje, daje se kratki pregled analize modela kako bi se provjerila njegova ispravnost odnosno ispitala svojstvena funkcionalnost.

Prije svega analizira se sama funkcionalnost modela predloženog ovim radom, odnosno analizira se jednostavnost pristupa podacima raznorodnog tipa formata i semantike sa stajališta korisnika. Pošto isti podacima može pristupati kako koristeći standardni pretraživač Interneta, tako i kroz aplikaciju napravljenu specijalno za tu svrhu, potrebita funkcionalnost se može zadovoljiti.

Pošto se sama informacijska infrastruktura dinamički mijenja, u skladu s tim i model se mora moći prilagođavati tim promjenama kako bi se mogao vremenom proširivati odnosno kako bi mogao pristupati novododanim izvorima podataka. Stoga se ukratko analizira i sama otvorenost modela, odnosno pokazuje se kako je arhitektura modela za pristup raznorodnim podacima, prvenstveno time što je temeljena na komponentnom pristupu, otvorena i pogodna za prilagodbe raznorodnim izvorima podataka.

Na kraju se analiziraju i sami sigurnosni protokoli, odnosno kakvu zaštitu samih podataka od neovlaštenog pristupa sam model pruža. Zaključuje se da je ovakvim pristupom moguće ostvariti zadovoljavajuću sigurnost podataka, te da će isključivo o tome kolika sigurnost će biti potrebna, te koliko će se u njenu implementaciju uložiti, ovisiti i kolika sigurnost će se ostvariti prilikom neke konkretne implementacije.

6.1. Funkcionalnost modela

Prilikom provjere funkcionalnosti modela, potrebno je obratiti pažnju na jednostavnost korištenja i na kvalitetu usluge.

Ukoliko se posredničkom redu pristupa iz pretraživača Interneta, jasno je da je sam pristup podacima raznorodnog tipa, formata i semantike, krajnje jednostavan. Uz to, svatko koristi pretraživač koji mu najviše odgovara i s kojim mu je najlakše raditi. Također, nije potrebno voditi računa o distribuciji programske podrške, budući da pretraživač Interneta ionako već svi korisnici redovito imaju. Koristeći ovaj pristup, također se može prenijeti gotovo sve, od teksta i slika, pa do zvuka i pokretnih slika. Drugim riječima, možemo zaključiti da je u potpunosti podržana višemedija. Također, možemo zaključiti da je pristup osiguran i sa gotovo svih raznorodnih platformi. Bez obzira da li se radi o nekom računalu UNIX, računalu Mac, ili nečem trećem, svi oni mogu s pripadnim pretraživačem pristupiti standardnom HTML-u, a isto tako i standardnom HTML-u kojem je dodana Java ili neke druge skripte. Nedostatak ovog pristupa je u tome što su mogućnosti ipak ograničene u odnosu na mogućnost aplikacije projektirane po nekim specifičnim zahtjevima krajnjeg korisnika.

Ukoliko se posredničkom redu pristupa direktno iz neke Windows aplikacije koristeći DCOM protokol, jasno je da smo po performansama dostigli praktički sve što se današnjom tehnologijom može napraviti. Na ovaj način možemo učiniti i sve ono što bi mogli koristeći pretraživač Interneta, budući da je i sam pretraživač Interneta samo jedna aplikacija. Nedostatak ovog pristupa je u tome što se programska podrška mora distribuirati na sva ciljna računala, a naravno istu je potrebno i razviti što je skup proces. Dodatni nedostatak je i u tome što smo vezani na Windows operacijske sustave (što i nije neki nedostatak ukoliko razvijamo nov sustav). Stoga se ovaj pristup preporuča samo ukoliko prethodni zbog nekog razloga ne zadovolji.

Ova dva pristupa možemo i kombinirati uz korištenje ActiveX kontrola. Ovaj pristup, kao i prethodni, vezan je za slučajeve gdje su nam korisnici ujedno i korisnici Windows operacijskih sustava. Prednost korištenja ActiveX kontrola je automatska distribucija programske podrška. Ovdje je važno napomenuti da to ne mora nužno značiti i veći rizik odnosno manju pouzdanost pri radu. Naime, koristeći Microsoft Internet Explorer 4.0 ili noviji, može se postaviti da se dozvoljava pokretanje ActiveX kontrola samo sa računala na kojem se nalazi implementiran posrednički red. Na taj način imamo automatsku distribuciju programske podrške, uz zadovoljavajuću pouzdanost cijelog sustava. Možemo zaključiti da je ovo kompromisno rješenje u odnosu na prethodna dva, koje nudi veću funkcionalnost od rješenja koje koristi samo pretraživač Interneta, a jednostavniju izvedbu od rješenja koje nudi korištenje DCOM tehnologije i izgradnje zasebnih korisničkih aplikacija.

Na kraju ovog poglavlja, možemo još spomenuti i mogućnost da se korisniku šalje samo višemedijski izgled korisničkog sučelja. Budući da je ova tehnologija dosta nova, i da će u sklopu operacijskog sustava najvjerojatnije biti podržana tek od verzije operacijskog sustava Windows 2000, potrebno je pričekati da se vidi kako će se ta tehnologija pokazati u praksi, te kako se eventualno može iskoristiti u svrhu jednostavnijeg pristupa podacima raznorodnog tipa, formata i semantike.

6.2. Otvorenost modela

Budući da se radi o sustavu koji mora moći pristupati raznorodnim podacima, bez obzira na format, tip i semantiku istih, jasno je da se isti vremenom mora moći nadograđivati, te time i prilagođavati promjenama u informacijskoj infrastrukturi.

Zasnovanost sustava na COM tehnologiji garantira nam upravo to – otvorenost. Svaka COM klasa može se ili sadržati ili agregirati u neku drugu COM klasu što nam garantira ponovnu iskoristivost već napisanog koda, kao i sama činjenica da se iste pišu koristeći objektno orijentiran jezik Visual C++.

Za svaki izvor podataka kojem možemo pristupiti uz pomoć OLE DB tehnologije, jednostavno napišemo zaseban OLE DB poslužitelj. Važno je napomenuti da na taj način možemo pristupiti gotovo svim komercijalnim sustavima poslovanja bazama podataka, kako onih sa strane Microsofta, tako i svih onih koje podržavaju X/Open DTP XA standard, a tu spadaju gotovo svi poznati sustavi poslovanja bazama podataka koji se izvorno nalaze na UNIX operacijskom sustavu. Na ovaj način možemo pristupiti i većini ostalih raznorodnih izvora podataka, kao što su npr. datoteke i sl. Nakon što napišemo dotični OLE DB poslužitelj, tu COM komponentu jednostavno dodamo posredničkom redu.

Ukoliko ipak postoji neki specifični izvor podataka, kojem se ne može pristupiti na prethodno opisani način, uvijek se može proširiti model na način kako je to pokazano za slučaj da je potrebno osigurati pristup CORBA objektima.

Prvo je potrebno, po prepoznatoj funkcionalnosti grupirati sve te dodatne izvore podataka raznorodnog tipa formata i semantike. Nakon toga, za svaku takvu grupu možemo napisati pomoćnu biblioteku objekata za pristup dotičnim specifičnim izvorima podataka kako je prikazano na Slici 16. Sučelje prema nekom specifičnom izvoru podataka u našem slučaju je COM klasa koja mu zna pristupati, a kojoj s druge strane pristupamo na istovjetan način kao i svim drugim COM klasama u grupi. Za svaku takvu grupu objekata, potrebno je još u COM-u napisati i objekte koje njima rukuju, a kojima objekti jezgre posredničkog reda pristupaju na istovjetan način kao i ADO objektima odnosno COM klasi *OLEDBforWEB* u slučaju objekata za rukovanje OLE DB bibliotekom. Objekti za rukovanje bibliotekama sučelja mogu također imati implementirano *IDispatch* sučelje kako bi im se moglo pristupati i direktno iz ASP stranica.

Ovdje je važno napomenuti i još jednu mogućnost proširenja. Budući da se i sama tehnologija OLE DB temelji na COM-u, istu je također moguće proširiti te time povećati svojstvenu funkcionalnost iste. Ovo ipak nije preporučljivo, budući da Microsoft trenutno ulaže vrlo puno upravo u OLE DB, te se ta tehnologija u ovom trenutku vrlo dinamički mijenja [133]. Koristeći upravo ove principe, Microsoft je već i proširio OLE DB sa OLAP (On-Line Analytical Processing) proširenjem [129]. No s druge strane, upravo to da Microsoft vrlo puno ulaže u ovu tehnologiju, garancija nam je da ćemo sa standardnim COM klasama podržanih kroz OLE DB moći pristupati svakim danom sve većem broju raznorodnih izvora podataka, te da nećemo morati za to pisati vlastite COM klase odnosno pripadne biblioteke istih.

Kroz ovaj rad, pokazano je kako se statička agregacija podržana sadašnjim COM-om može proširiti, te kako se mogu konstruirati *IDispatch* sučelja koja podržavaju taj proširen oblik agregacije. Zahvaljujući tim proširenjima, možemo zaključiti da sustav ne samo da je otvoren, već se i prilikom nadogradnje istog mogu koristiti funkcionalni oblici ponovne iskoristivosti već napisanog koda.

6.3. Sigurnosni aspekti modela

Budući da se posredničkom redu modela pristupa preko računalne mreže pripadne informacijske infrastrukture, jasno je da o sigurnosnim protokolima pristupa moramo voditi računa [2][26]. Pri tome je, zbog same specifičnosti problematike vezane za sigurnost, potrebno prije svega oslanjati se na standardna rješenja [115][116].

Budući da se posrednički red nalazi na Windows NT operacijskom sustavu, najjednostavniji oblik zaštite podataka od neovlaštenog pristupa jest koristiti njegov sigurnosni model. Od svakog tko se spaja na sustav, može se zahtijevati korisničko ime i zaporka. Na osnovi zaporke, provjerava se da li je to stvarno korisnik predstavljen dotičnim korisničkim imenom, te mu se nakon toga dodjeljuju ovlaštenja koja ovise o tome u koju grupu korisnika dotični spada. Sve COM klase koje se pokrenu nakon toga, u okviru MTS poslužitelja ili izvan njega, automatski nasljeđuju privilegije spojenog korisnika. Na taj način, garantira se da svatko može pristupati samo onim podacima unutar informacijske infrastrukture za koje ima dopuštenje.

Svaka COM klasa koristeći usluge MTS poslužitelja (točnije rečeno koristeći *ISecurityProperty* sučelje kontekst objekta kojeg joj pridružuje MTS poslužitelj) dodatno može imati i informacije o tome tko je pokrenuo akciju koja je uzrokovala njeno kreiranje, odnosno o tome tko je pokrenuo akciju koja je uzrokovala njeno pozivanje. Posebno se može postaviti, koje grupe korisnika imaju pravo da kreiraju neku komponentu, odnosno da pozivaju njene metode. Na taj način se postiže dodatna sigurnost, pošto se točno zna tko ima pravo da pokrene neku komponentu odnosno da pozove metodu iste, te pošto i sama komponenta ima podatke o tome, pa u skladu s njima može i poduzimati odnosno ne poduzimati odgovarajuće akcije.

Što se tiče samog prometa mrežom, ukoliko se koriste aplikacije temeljene na pretraživačima Interneta, mogu se koristiti sigurne veze odnosno SSL (Security Socket Layer). U slučaju kada se koristiti DCOM, može se koristiti dodatno kriptiranje po izboru budući da se kriptiranje i dekriptiranje vrlo jednostavno mogu implementirati kao zasebne metode bilo koje COM klase.

Koristeći Internet Explorer 4.0 ili noviji, kod svakog korisnika se može postaviti s kojih računala je dozvoljeno pokretati ActiveX komponente. Koristeći to svojstvo, kod svih korisnika možemo postaviti da je pokretanje ActiveX komponenata dozvoljeno samo sa računala na kojem se nalazi posrednički red. Na taj način možemo spriječiti mogućnost da korisnik nehotice pokrene ActiveX s nekog drugog računala (što može imati vrlo neugodne posljedice), a same ActiveX komponente kao sredstvo automatske distribucije programske podrške, a samim tim i komponenata koje će nam osiguravati potrebno kriptiranje podataka. Koristeći ovaj pristup, i upotrebom samo pretraživača Interneta možemo osigurati istu sigurnost kao i u slučaju kad se koristi Distributed COM.

Sa sljedećom verzijom Windows operacijskog sustava (Windows 2000), biti će dostupan i sigurnosni protokol Kerberos [29], te će se i to po potrebi moći iskoristiti prilikom same implementacije posredničkog reda. Također, dosta se može očekivati i od Internet sigurnosnih protokola na kojima se intenzivno radi i koji se mogu očekivati sa sljedećom verzijom Internet standarda [95][101][110].

Na kraju možemo zaključiti da je ovakvim pristupom moguće ostvariti zadovoljavajuću sigurnost podataka, a kolika sigurnost će se ostvariti prilikom konkretne implementacije, ovisiti će isključivo o tome kolika sigurnost će biti potrebna, odnosno koliko će se u njenu implementaciju uložiti.

7. ZAKLJUČAK

U ovom radu su analizirani problemi vezani za pristup podacima raznorodnog tipa, formata i semantike, koji se javljaju prilikom interakcije s raspodijeljenim informacijskim sustavom podržanim informacijskom infrastrukturom. U svrhu olakšanja i poboljšanja pristupa, predložena je arhitektura modela za univerzalan odnosno uniformni pristup raznorodnim podacima unutar informacijske infrastrukture.

Analizom problematike te usporedbom klasičnih implementacija dvorednih i troredних arhitektura, zaključilo se da troredna arhitektura koja se sastoji od korisničkog reda, posredničkog reda, i samih podataka raznorodnog tipa, formata i semantike, najbolje odgovara kao osnovna arhitektura modela za univerzalan odnosno uniformni pristup raznorodnim podacima. Dalje se razradio kako sam korisnički red odnosno tri moguće implementacije korisničkog reda i pripadne komunikacije s posredničkim redom, tako i sama arhitektura posredničkog reda, te su se pri tom proučile i razradile postojeće tehnologije važne za uspješnu izvedbu samog modela. Arhitektura modela zasniva se na suvremenom pristupu ostvarenja troredne Internet/intranet aplikacije koristeći i objektno orijentiran i komponentno temeljen pristup. Da bi se pokazalo kako je model jednostavno proširiti u svrhu mogućnosti pristupanja i svim onim raznorodnim podacima odnosno arhitekturama za koje prvotno nije bio namijenjen, razmotreno je jedno moguće proširenje u sklopu osnovne arhitekture kako bi se podržalo pristupanje CORBA objektima.

Dalje je razrađena i arhitektura same jezgre modela, njegov posrednički red. Prije svega opisani su problemi koji se inače javljaju pri klasičnom razvoju aplikacija, kako bi se isti izbjegli prilikom implementacije posredničkog reda. Nakon toga, daje se pregled svojstava komponentno temeljene paradigme koja se zajedno s objektno orijentiranom koristi kao osnova izgradnje posredničkog reda i modela u cjelini. Zatim se opisuje jedna moguća implementacija posredničkog reda odnosno samog modela, te se pri tome daje i pregled tehnologija koje se mogu koristiti kako bi se posrednički red implementirao na što jednostavniji način. Pošto se ta implementacija temelji na Windows NT Server 4.0 operacijskom sustavu, dalje se u skladu s objektno orijentiranim i komponentno temeljenim pristupom razrađuje i sam COM kao tehnologija implementacije jezgre posredničkog reda. Pri tome je posebno stavljen naglasak na to kako i koja svojstva COM-a koristiti prilikom implementacije različitih dijelova posredničkog reda, te kako proširiti COM mehanizme ponovne iskoristivosti, prije svega agregaciju, kako bi se COM prilagodio samoj arhitekturi modela za pristup raznorodnim podacima.

U nastavku se daje i postupak izgradnje prototipa specijaliziranog za sam pristup raznorodnim sustavima poslovanja relacijskih baza podataka, kao primjer pojednostavljene verzije općenitijeg modela za pristup podacima koji i ne moraju biti u nekoj bazi podataka, kao što su primjerice datoteke, dokumenti i slično. Prije svega obrazlaže se posebna važnost odnosno motivacija za uniformnim pristupom različitim sustavima poslovanja bazama podataka, te se dalje kao korisnički definirana COM klasa *OLEDBforWEB* implementira i sam prototip modela. Isti se ispituje nad poligonom različitih sustava poslovanja bazama podataka uključujući: DB2, Oracle, SQL Server, Sybase, Access, Informix i NCR Teradata. Opisuju se problemi na koje se naišlo prilikom implementacije te načini kako su isti riješeni, i to prije svega problemi vezani za rukovanje tablicama rezultata upita, internacionalizaciju/lokalizaciju, indeksiranje i integritet podataka, te raznolikost tipova podataka. Zatim se ilustrira i kako se definirana COM klasa *OLEDBforWEB* može koristiti, kroz izradu jedne primjer-aplikacije, virtualne odnosno mrežne knjižnice.

Na kraju je izvršena i sama analiza modela. Prije svega to je uključilo procjenu jednostavnosti rukovanja modelom, koja je važna kako bi se osigurao jednostavan pristup raznorodnim podacima neke informacijske infrastrukture, procjena njegove funkcionalnosti, odnosno mogućnosti pristupa podacima raznorodnog tipa, formata i semantike, procjena njegove otvorenosti koja je osnova daljnje proširivosti, odnosno mogućnosti da se modelu može jednostavno dodati modul za svaki novi izvor podataka neke informacijske infrastrukture, te procjenu pripadne zaštite podataka koju model pruža, odnosno sigurnosne protokole koji se pri tom koriste. U cjelini, model zadovoljava pretpostavljena očekivanja.

Kombinacija objektno orijentiranog i komponentno temeljenog pristupa osnovni je princip koji se koristio prilikom implementacije prototipa modela, i to prvenstveno kroz upotrebu programskog jezika Visual C++ i tehnologije Distributed COM, koji zajedno čine osnovu razvoja samog posredničkog reda te time i samog modela u cjelini. Komponentno temeljen pristup osigurava nam modularno izgrađenu arhitekturu gdje različiti moduli podržavaju različite izvore podataka odnosno klase izvora podataka, i gdje je podržavanje nekog novog izvora podataka moguć izgradnjom novog modula bez da se time utiče na rad već implementiranog dijela sustava (osim što mu se proširuje funkcionalnost). Objektno orijentiran pristup, s druge strane nam osigurava maksimalnu fleksibilnost odnosno efikasnost prilikom izgradnje tih istih modula. Problemi koji su se javili u posredničkom redu prilikom korištenja mehanizama za ponovnu iskoristivost već napisanog koda u COM-u, sadržavanja i agregacije, riješeni su na taj način da je kroz ovaj rad dat prijedlog proširene arhitekture za agregaciju COM komponenata. U okviru tog algoritma proširenog oblika agregacije, riješeni su i problemi koji se mogu javiti prilikom korištenja *IDispatch* sučelja, koje je inače osnovno sučelje preko kojeg se iz ASP stranica pristupa jezgri posredničkog reda.

Sam model, ili točnije rečeno njegov posrednički red, razvijen je u skladu s tehnologijom dostupnom početkom 2000, odnosno u skladu s arhitekturom Windows DNA. Pri tom je najviše korištena tehnologije OLE DB kao osnova za pristup raznorodnim podacima, COM kao paradigma pisanja komponenata, te jezici ASP i Visual C++ za samu implementaciju prototipa. S druge strane, korisnički red kao i sami podaci raznorodnog tipa, formata i semantike mogu se nalaziti praktički na bilo kojoj platformi unutar informacijske infrastrukture. Koristeći takav pristup, s jedne strane zadržana je otvorenost sustava u cjelini, a s druge prototip same jezgre sustava odnosno posredničkog reda, implementiran je koristeći najnovije tehnologije.

LITERATURA

[1] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, M. C. Shan, "The Pegasus Heterogeneous Multidatabase System", *Computer*, IEEE, prosinac 1991, pp. 19-27.

[2] W. A. Arbaugh, J. R. Davin, D. J. Farber, J. M. Smith, "Security for Virtual Private Intranets", *Computer*, IEEE, rujan 1998, pp. 48-55.

[3] D. Arnold, P. Cannata, L. A. Glasson, G. Hallmark, B. McGuire, S. Newman, R. Odegard, H. Sabharwal, "SQL Access: An implementation of the ISO Remote Database Access standard", *Computer*, IEEE, prosinac 1991, pp. 74-78.

[4] C. Batini, M. Lenzerini, S. B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys*, Vol. 18, No. 4, prosinac 1986.

[5] R. J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, D. Woelk, "The InfoSleuth Project", *Proc. SIGMOD'97 – ACM SIGMOD Int'l Conf. on Management of Data*, AZ, SAD, 1997, pp. 543-545.

[6] M. Betz, "Active Data Objects & ASP", *Dr. Dobb's Journal*, SAD, svibanj 1998, pp. 88-91.

[7] J. A. Blakeley, "Data Access for the Masses through OLE DB", *Proc. SIGMOD'96 – ACM SIGMOD Int'l Conf. on Management of Data*, Montreal, Kanada, lipanj 1996, pp. 161-172.

[8] J. A. Blakeley, M. J. Pizzo, "Microsoft Universal Data Access Platform", *Proc. SIGMOD'98 –ACM SIGMOD Int'l Conf. on Management of Data*, Seattle - WA, SAD, lipanj 1998, pp. 502-503.

[9] B. Boehm, C. Abts, "COTS Integration: Plug and Pray?", *Computer*, IEEE, siječanj 1999, pp. 135-137.

[10] G. A. Bolcer, G. Kaiser, "SWAP: Leveraging the Web To Manage Workflow", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 85-88.

[11] J. Bosak, "Media-Independent Publishing: Four Myths about XML", *Computer*, IEEE, listopad 1998, pp. 120-122.

[12] A. Bouguettaya, B. Benatallah, M. Ouzzani, L. Hendra, "WEBFINDIT: An Architecture and System for Querying Web Databases", *Internet Computing*, IEEE, srpanj/kolovoz 1999, pp. 30-41.

[13] K. Brown, "ActiveX/COM Q & A", *u [153]*, SAD, 1997.

[14] L. F. Cabrera, B. Andrew, K. Peltonen, N. Kusters, "Advances in Windows NT Storage Management", *Computer*, IEEE, listopad 1998, pp. 48-54.

[15] L. Cardelli, R. Davies, "Service Combinators for Web Computing", *Tr. on Software Engineering*, IEEE, svibanj/lipanj 1999, Vol. 25, No. 3, pp. 309-316.

[16] V. Cardellini, M. Colajanni, P. S. Yu, "Dynamic Load Balancing on Web-Server Systems", *Internet Computing*, IEEE, svibanj/lipanj 1999, pp. 28-39.

[17] S. Chakrabarti, B. E. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, J. Kleinberg, "Mining the Web's Link Structure", *Computer*, IEEE, kolovoz 1999, pp. 60-67.

[18] J. Charles, "Middleware Moves to the Forefront", *Computer*, IEEE, svibanj 1999, pp. 17-19.

[19] J. Chen, R. Patterson, "ADO and SQL Server Developer's Guide", *u [153]*, SAD, 1998.

[20] S. Y. Choi, A. B. Whinston, "The Future of E-Commerce: Integrate and Customize", *Computer*, IEEE, siječanj 1999, pp. 133-134, 138.

[21] P. Clip, "DCOM: Microsoft Enhances DCE", *BYTE*, SAD, ožujak 1998, pp. 47-48.

[22] C. Collet, M. N. Huhns, W. M. Shen, "Resource Integration Using a Large Knowledge Base in Carnot", *Computer*, IEEE, prosinac 1991, pp. 55-62.

[23] R. Comerford & R. Blokzijl, V. Cerf, S. Deering, D. Heath, C. Huitema, L. Landweber, K. Neggers, J. Postel, "State of the Internet: Roundtable 4.0", *Spectrum*, IEEE, SAD, listopad 1998, pp. 69-79.

[24] A. Denning, *ActiveX™ Controls Inside Out (2nd ed.)*, Microsoft Press, SAD, siječanj 1997.

[25] R. Dobson, "ECMAScript: The Holy Standard?", *BYTE*, srpanj 1998, pp. 47-48.

[26] P. W. Dowd, J. T. McHenry, "Network Security: It's Time to Take It Seriously", *Computer*, IEEE, rujan 1998, pp. 24-28.

[27] J. Dreystadt, "Storing Java in a Relational Database", *BYTE*, lipanj 1998, pp. 61-62.

[28] P. F. Dubois, "Scientific Components Are Coming", *Computer*, IEEE, ožujak 1999, pp. 115-117.

[29] G. Eddon, H. Eddon, *Inside Distributed COM*, Microsoft Press, SAD, ožujak 1998.

[30] A. Elmagarmid, M. Rusinkiewicz, A. Sheth, *Management of Heterogeneous and Autonomous Database Systems*, Morgan Kaufmann Publishers, San Francisco, California, SAD, 1999.

[31] M. Fan, J. Stallaert, A. B. Whinston, "A Web-Based Financial Trading System", *Computer*, IEEE, travanj 1999, pp. 64-70.

[32] K. Fertalj, D. Kalpić, "An Object Based Software Development Technique", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 469-474.

[33] P. Flanigan, J. Karim, "NCSA Symera: Distributed parallel-processing using DCOM", *Dr. Dobb's Journal*, SAD, studeni 1998, pp. 20-27.

[34] J. Fomook, "Moving Mission-Critical Apps to the Web", *BYTE*, srpanj 1998, pp. 49-50.

[35] R. Fryer, "Data-Warehouse Scalability", *BYTE*, lipanj 1998, pp. 63-64.

[36] J. Gallaugher, "Challanging the New Conventional Wisdom of Net Commerce Strategies", *Communications*, ACM, srpanj 1999, pp. 27-29.

[37] R. Gamache, R. Short, M. Massa, "Windows NT Clustering Service", *Computer*, IEEE, listopad 1998, pp. 55-62.

[38] V. Ganti, J. Gehrke, R. Ramakrishnan, "Mining Very Large Databases", *Computer*, IEEE, kolovoz 1999, pp. 38-45.

[39] L. Garber & S. Deering, "Steve Deering on IP Next Generation", *Computer*, IEEE, travanj 1999, pp. 11-13.

[40] L. Garber & G. Lawton, "HTTP-Next Generation Is in the Works", *Computer*, IEEE, studeni 1998, pp. 18-20.

[41] H. W. Gellersen, M. Gaedke, "Object-Oriented Web Application Development", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 60-68.

[42] L. Geppert, "IC Design on the World Wide Web", *Spectrum*, IEEE, lipanj 1998, pp. 45-50.

[43] K. Golomb, T. Sorgie, "How Do I Ensure Secure Communications from a Java Applet", *Dr. Dobb's Journal*, SAD, lipanj 1998, pp. 107-109.

[44] P. Goodwin, "COM, ActiveX Controls, and Microsoft Windows CE", *u [153]*, SAD, ožujak 1998.

[45] A. Grimshaw, A. Ferrari, F. Knabe, M. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale", *Computer*, IEEE, svibanj 1999, pp. 29-37.

[46] R. Guerraoui, E. Fayad, "Object-Oriented Abstractions for Distributed Programming", *Communications*, ACM, kolovoz 1999, pp. 125-127.

[47] R. Guerraoui, E. Fayad, "OO Distributed Programming Is Not Distributed OO Programing", *Communications*, ACM, travanj 1999, pp. 101-104.

[48] P. Hallam, D. Blakeley, "Universal providers", *Developer Network Journal*, Microsoft Corp., SAD, rujan/listopad 1998, pp. 30-33.

[49] J. Han, L. V. S. Lakshmanan, R. T. Ng, "Constraint-Based, Multidimensional Data Mining", *Computer*, IEEE, kolovoz 1999, pp. 46-50.

[50] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, P. J. Haas, "Interactive Data Analysis: The Control Project", *Computer*, IEEE, kolovoz 1999, pp. 51-59.

[51] J. R. Herkert, M. Loui & L. G. Salmon, C. Whitbeck, K. Miller, T. E. Bell, L. E. Harris, D. Gotterbarn, P. Hoon, J. Kesan, B. O'Connell, W. Sweet, "The Ethics of Intellectual Property and the New Information Technologies", *Spectrum*, IEEE, SAD, kolovoz 1999, pp. 29-37.

[52] L. Hightower, "The Web Report Database Reporting Tool", *Dr. Dobb's Journal*, SAD, listopad 1998, pp. 90-95.

[53] S. Hillier, D. Mezick, *Programming Active Server Pages*, Microsoft Press, SAD, studeni 1997.

[54] S. Hoag, "Together At Last – Microsoft Transaction Server and Visual Basic 6.0", *msdn news*, Microsoft Corp., SAD, rujan/listopad 1998, pp. 1, 8-9.

[55] D. Houlding, "A CORBA Bean Framefork: Encapsulating compleity and facilitating development", *Dr. Dobb's Journal*, SAD, studeni 1998, pp. 34-40.

[56] T. P. Hughes, J. R. Sheehan, "What Has Influenced Computing Innovation", *Computer*, IEEE, veljača 1999, pp. 33-43.

[57] M. N. Huhns, A. K. Malhotra, "Negotiating for Goods and Services", *Internet Computing*, IEEE, srpanj/kolovoz 1999, pp. 97-99.

[58] S. Isaacs, *Inside Dynamic HTML*, Microsoft Press, SAD, listopad 1997.

[59] J. Isaak, "IEEE STANDARD 2001: Web Page Engineering for Intranets and Extranets", *Internet Computing*, IEEE, ožujak/travanj 1999, pp. 94-95.

[60] J. Jamison, R. Nicklas, G. Miller, K. Thompson, R. Wilder, L. Cunningham, C. Song, "vBNS: not your father's Internet", *Spectrum*, IEEE, srpanj 1998, pp. 38-46.

[61] W. C. Janssen, "A "Next Generation" Architecture for HTTP", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 69-73.

[62] D. Jutla, P. Bodorik, C. Hajnal, C. Davis, "Making Business Sense of Electronic Commenrce", *Computer*, IEEE, ožujak 1999, pp. 67-75.

[63] D. Kiely, "Are Components the Future of Software?", *Computer*, IEEE, veljača 1998, pp. 10-11.

[64] W. Kim, J. Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", *Computer*, IEEE, prosinac 1991, pp. 12-18.

[65] M. Kirtland, "The COM+ Programming Model Makes it Easy to Write Components in Any Language", *u [153]*, SAD, 1997.

[66] S. Klein, "Designing for Customer Interaction on the Web", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 32-35.

[67] K. Kontovasilis, G. Kormentzas, A. Kourtis, N. Mitrou, J. Soldatos, E. Vayias, "A Web-Based System for Providing Information About Prototype ATM Platforms, Associated Research Projects and Related Trials", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 197-203.

[68] G. Koprowski, "Emerging Uncertainty Over IPv6", *Computer*, IEEE, studeni 1998, pp. 16-17, 20.

[69] J. Korpela, "Lurching Toward Babel: HTML, CSS, and XML", *Computer*, IEEE, srpanj 1998, pp. 103-106.

[70] B. Krämer, M. Papazoglou, H. W. Schmidt, *Information Systems Interoperability*, Research Studies Press Ltd., Velika Britanija, 1998.

[71] D. Krieger, and R. M. Adler, "The Emergence of Distributed Component Platforms", *Computer*, IEEE, ožujak 1998, pp. 43-53.

[72] D. J. Kruglinski, G. Shepherd, S. Wingo, *Programming Microsoft Visual C++ (5th ed.)*, Microsoft Press, SAD, kolovoz 1998.

[73] C. Laird, K. Soraiz, "GET a GRIP on SCRIPTS", *BYTE*, lipanj 1998, pp. 89-96.

[74] K. Lassesen, "ADODB – ActiveX Data Objects 2.1", *msdn news*, Microsoft Corp., SAD, ožujak/travanj 1999, pp. 8-9.

[75] K. Lassesen, "An Extended Map of the Active Server Page and Scripting Objects", *msdn news*, Microsoft Corp., SAD, rujan/listopad 1998, pp. 6-7.

[76] G. Lawton, "Multicasting: Will it Transform the Internet?", *Computer*, IEEE, srpanj 1998, pp. 13-15.

[77] J. Lewallen, "What's New in ADO 2.0", *u [153]*, SAD, srpanj 1998.

[78] T. Lewis, "Mainframes Are Dead, Long Live Mainframes", *Computer*, IEEE, kolovoz 1999, pp. 104, 102-103.

[79] S. M. Lewandowski, "Frameworks for Component-Based Client/Server Computing", *ACM Computing Surveys*, Vol. 30, No. 1, ožujak 1998.

[80] U. Lindqvist, E. Jonsson, "A Map of Security Risks Associated with Using COTS", *Computer*, IEEE, lipanj 1998, pp. 60-66.

[81] Rob Macdonald, "ADO: Learn to Love It", *u [153]*, SAD, 1998.

[82] F. Manola, "Technologies for a Web Object Model", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 38-47.

[83] S. McGrath, "Rendering XML Documents Using XSL", *Dr. Dobb's Journal*, SAD, srpanj 1998, pp. 82-85.

[84] A. McKay, "Components and Your Business", *Sybase Inc.*, SAD, lipanj 1998.

[85] A. McKay, "Moving Client/Server to the Web", *Sybase Inc.*, SAD, 1998.

[86] B. Meyer, "On To Components", *Computer*, IEEE, siječanj 1999, pp. 139-140.

[87] B. Meyer, "The Role of Object-Oriented Metrics", *Computer*, IEEE, studeni 1998, pp. 123-125.

[88] B. Meyer, C. Mingins, H. Schmidt, "Providing Trusted Components to the Industry", *Computer*, IEEE, svibanj 1998, pp. 104-105.

[89] T. Mohorič, "Database reverse engineering: Relational to ER model translation", *8th Electrotechnical and Computer Science Conference – ERK'99*, Portorož, Slovenija, rujan 1999, pp. 85-88.

[90] L. Monson, "The WIDL Specification: Programmatic interfaces to the Web", *Dr. Dobb's Journal*, SAD, studeni 1998, pp. 92-95.

[91] B. Morgan, "Building Distributed Applications with Java and CORBA", *Dr. Dobb's Journal*, SAD, travanj 1998, pp. 94-99.

[92] D. Moss, "Windows Terminal Server", *Developer Network Journal*, Microsoft Corp., SAD, srpanj/kolovoz 1998, pp. 31-35.

[93] E. Murray, "Using ActiveX Controls in Web Pages", *msdn news*, Microsoft Corp., SAD, rujan/listopad 1998, pp. 4-5.

[94] T. Neild, N. Jukic, "Heterogeneous Database Integration Strategies", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 311-316.

[95] P. Neumann, "Internet Security – Beyond Cryptography", *Internet Computing*, IEEE, ožujak/travanj 1999, pp. 100.

[96] K. Nichols, "The Age of Software Patents", *Computer*, IEEE, travanj 1999, pp. 25-31.

[97] M. Nicholson, "Inside Windows DNA", *Developer Network Journal*, Microsoft Corp., SAD, srpanj/kolovoz 1998, pp. 20-29.

[98] M. Nicholson, "On the road to COM Plus", *Developer Network Journal*, Microsoft Corp., SAD, studeni 1997, pp. 12-13.

[99] M. Nicholson, "Understanding Software Components", *Developer Network Journal*, Microsoft Corp., SAD, rujan 1997, pp. 20-25.

[100] J. Noll, W. Scacchi, "Integrating Diverse Information Repositories: A Distributed Hypertext Approach", *Computer*, IEEE, prosinac 1991, pp. 38-45.

[101] R. Oppliger, "Security at the Internet Layer", *Computer*, IEEE, rujan 1998, pp. 43-47.

[102] T. Prenđa, M. Žagar, D. Trupec, "RASIP Virtual Shop", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 453-457.

[103] C. Pu, A. Leff, S. W. F. Chen, "Heterogeneous and Autonomous Transaction Processing", *Computer*, IEEE, prosinac 1991, pp. 64-72.

[104] A. Puder, "The MICO CORBA Compliant System: A freely available CORBA implementation", *Dr. Dobb's Journal*, SAD, studeni 1998, pp. 44-50.

[105] S. Ram, "Heterogeneous Distributed Database Systems", *Computer*, IEEE, prosinac 1991, pp. 7-11.

[106] B. Reinwald, H. Pirahesh, "SQL Open Heterogeneous Data Access", *Proc. SIGMOD'98 – ACM SIGMOD Int'l Conf. on Management of Data*, Seattle - WA, SAD, lipanj 1998, pp. 506-507.

[107] D. Rogerson, *Inside COM*, Microsoft Press, SAD, prosinac 1996.

[108] D. Rogerson, "MFC/COM Objects 8: Revisiting Multiple Inheritance Without MFC", *u [153]*, SAD, rujan 1995.

[109] P. Rousselle, "Dynamic Distributed Systems in Java", *Dr. Dobb's Journal*, SAD, travanj 1998, pp. 88-92.

[110] A. D. Rubin, D. E. Geer Jr., "A Survey of Web Security", *Computer*, IEEE, rujan 1998, pp. 34-41.

[111] M. Rusinkiewicz, A. Sheth, G. Karabatis, "Specifying Interdatabase Dependencies in a Multidatabase Environment", *Computer*, IEEE, prosinac 1991, pp. 46-53.

[112] G. Salamunićcar, V. Glavinić, "An OLE DB Based Web Interface to Databases", *8th Electrotechnical and Computer Science Conference – ERK'99*, Portorož, Slovenija, rujan 1999, pp. 101-104.

[113] G. Salamunićcar, V. Glavinić, "On Insuring Interoperability of OLE DB Based Data Access", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 317-324.

[114] B. Schatz, W. Mischo, T. Cole, A. Bishop, S. Harum, E. Johnson, L. Neumann, H. Chen, D. Hg, "Federated Search of Scientific Literature", *Computer*, IEEE, veljača 1999, pp. 51-59.

[115] B. Schneier, "Cryptographic Design Vulnerabilities", *Computer*, IEEE, rujan 1998, pp. 29-33.

[116] B. Schneier, "Cryptography: The Importance of Not Being Different", *Computer*, IEEE, ožujak 1999, pp. 108-109 & 112.

[117] G. Shepherd, S. Wingo, "ATL and Connection Points", *Dr. Dobb's Journal*, SAD, lipanj 1998, pp. 117-119.

[118] J. Siegel, "A Preview of CORBA 3", *Computer*, IEEE, svibanj 1999, pp. 114-116.

[119] M. P. Singh, "Value-Oriented Electronic Commerce", *Internet Computing*, IEEE, svibanj/lipanj 1999, pp. 6-7.

[120] S. N. Singh, N. P. Dalal, "Web Home Pages as Advertisements", *Communications*, ACM, kolovoz 1999, pp. 91-98.

[121] D. A. Solomon, "The Windows NT Kernel Architecture", *Computer*, IEEE, listopad 1998, pp. 40-47.

[122] M. Sood, "JDBC Drivers and Web Security", *Dr. Dobb's Journal*, SAD, srpanj 1998, pp. 90-95.

[123] N. Soparkar, H. F. Korth, A. Silberschatz, "Failure-Resilient Transaction Management in Multidatabases", *Computer*, IEEE, prosinac 1991, pp. 28-36.

[124] R. Soukup, *Inside Microsoft SQL Server 6.5*, Microsoft Press, SAD, studeni 1997.

[125] J. A. Stankovic, S.-H. Son, J. Hansson, "Miscoceptions About Real-Time Databases", *Computer*, IEEE, lipanj 1999, pp. 29-36.

[126] A. Stevens, "The Next Great Migration: From C++ to Standard C++", *Dr. Dobb's Journal*, SAD, rujan 1998, pp. 105-109.

[127] N. Talbert & J. McDermid, "The Cost of COTS", *Computer*, IEEE, lipanj 1998, pp. 46-52.

[128] J. Thomas, "Ecommerce in Europe – IIS, SET, Windows NT, Site Server - Join the party", *Developer Network Journal*, Microsoft Corp., SAD, ožujak/travanj 1999, pp. 36-40.

[129] J. Thomas, "OLAP for the masses", *Developer Network Journal*, Microsoft Corp., SAD, rujan/listopad 1998, pp. 38-39.

[130] N. Thompson, "MFC/COM Objects 4: Aggregation", *u [153]*, SAD, ožujak 1995.

[131] N. Thompson, "MFC/COM Objects 5: Using Multiple Inheritance", *u [153]*, SAD, ožujak 1995.

[132] M. Varga, "Conversion of Relational into Multidimensional Database Schema", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 331-338.

[133] B. Vaughn, "Much ADO about OLE DB", *Developer Network Journal*, Microsoft Corp., SAD, siječanj/veljača 1998, pp. 38-41.

[134] R. Vetter, "Web-Based Enterprise Computing", *Computer*, IEEE, svibanj 1999, pp. 112-113 & 116.

[135] J. M. Voas, "Certifying Off-the-Shelf Software Components", *Computer*, IEEE, lipanj 1998, pp. 53-59.

[136] J. M. Voas, "The Challenges of Using COTS Software in Component-Based Development", *Computer*, IEEE, lipanj 1998, pp. 44-45.

[137] M. G. Wales, "WIDL: Interface Definition for the Web", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 55-59.

[138] Y. M. Wang, P. Y. E. Chung, "Customization of distributed systems using COM", *IEEE Concurency*, srpanj/kolovoz 1998, pp. 8-12.

[139] Y. M. Wang, W. J. Lee, "COMERA: COM Extensible Remoting Architecture", *COOTS'98: 4th USENIX Conf. Object-Oriented Technologies and Systems*, SAD, travanj 1998, pp. 79-88.

[140] K. Watterson, "Enterprise Databases BATTLE ON", *BYTE*, lipanj 1998, pp. 97-99.

[141] F. Wild, "C++ Interfaces", *Dr. Dobb's Journal*, SAD, kolovoz 1998, pp. 18-22.

[142] S. Willliams, C. Kindel, "The Component Object Model: A Technical Overview", *u [153]*, SAD, listopad 1994.

[143] D. Willits, "Implementing ADO with Various Development Languages", *u [153]*, SAD, srpanj 1998.

[144] L. Wood, "PROGRAMMING THE WEB: The W3C DOM Specification", *Internet Computing*, IEEE, siječanj/veljača 1999, pp. 48-54.

[145] L. Zhang, J. Li, J. Pan, "An Intelligent Interface Agent for Web-based Information Retrieval", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 125-132.

[146] Q. Zhong, N. Edwards, "Security Control for COTS Components", *Computer*, IEEE, lipanj 1998, pp. 67-73.

[147] S. Zimmerman, "Extreme C++: Looking Forward to COM+", *u [153]*, SAD, 1998.

[148] *Annual Report of the Harvard Information Infrastructure Project for 1997-1998*, "http://www.ksg.harvard.edu/iip/HIIP-Report.pdf", 1998.

[149] *Automation Programmer's Reference*, Microsoft Press, SAD, veljača 1997.

[150] *Distributed Component Object Model Protocol (DCOM/1.0)*, Microsoft Corp., SAD, siječanj 1998.

[151] *Microsoft OLE DB 1.1*, Microsoft Press, SAD, travanj 1997.

[152] *Microsoft TerraServer*, Microsoft Corp., SAD, lipanj 1998.

[153] *MSDN Library April 1999*, Microsoft Corp., SAD, travanj 1999.

[154] *Sybase Adaptive Server Enterprise 11.9.2*, Sybase Inc., 1998.

[155] *The Common Object Request Broker: Architecture and Specification*, "ftp://ftp.omg.org/pub/docs/formal/98-07-01.pdf.gz", srpanj 1998.

[156] *The Component Object Model Specification*, Microsoft Corp., SAD, listopad 1995.

[157] *The JavaBeans Specification V 1.01*, Sun Microsystems Inc., srpanj 1997.

[158] *Using ADO with Visual Basic, VBScript, Visual C++, and Java*, u [153], SAD, 1998.

[159] *What Are the Objects in the ADO Object Model?*, u [153], SAD, ožujak 1998.

[160] "SQL Server 7 – Musqling in", *Developer Network Journal*, Microsoft Corp., SAD, studeni/prosinac 1998, pp. 20-23.

[161] "Conference Report – What's there to C in Amsterdam", *Developer Network Journal*, Microsoft Corp., SAD, ožujak/travanj 1999, pp. 16-18.

POPIS KRATICA

ACID – *Atomic, Consistent, Isolated, and Durable properties of transactions*

ADO – *ActiveX Data Objects*

API – *Application Programming Interface*

ASP – *Active Server Pages*

ATL – *Active Template Library*

ATM – *Asynchronous Transfer Mode*

CLSID – *Class Identifier*

COM – *Component Object Model*

CORBA – *Common Object Request Broker Architecture*

DCOM – *Distributed Component Object Mode*

DHTML – *Dynamic HyperText Markup Language*

DISPID – *Dispatch Identifier*

GIIC – *Global Information Infrastructure Commission*

HIIP – *Harvard Information Infrastructure Project*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IDL – *Interface Definition Language*

IIS – *Internet Information Server*

IITF – *Information Infrastructure Task Force*

INMARSAT – *International Mobile Satellite Organization*

INTELSAT – *International Telecommunications Satellite Organization*

IP – *Internet Protocol*

IPv4 – *Internet Protocol version 4 (sadašnja verzija)*

IPv6 – *Internet Protocol version 6*

IRC – *Internet Relay Chat*

ISAM – *Indexed Sequential Access Method*

ITU – *International Telecommunications Union*

JDBC – *Java Database Connectivity*

K – *Korisnik*

KR – *Korisnički Red*

LCID – *Local Identifier*

MFC – *Microsoft Foundation Class*

MIDL – *Microsoft Interface Definition Language compiler*

MTS – *Microsoft Transaction Server*

NDR – *Network Data Representation*

NSRC – *Network Startup Resource Center*

NTIA – *National Telecommunications and Information Administration*

ODBC – *Open Database Connectivity*

OECD – *Organisation for Economic Co-operation and Development*

OLAP – *On-Line Analytical Processing*

OLE – *Object Linking and Embedding*

OLE DB – *Object Linking and Embedding Database Connectivity*

ORB – *Object Request Broker*

PDF – *Portable Document Format*

PRTFIS – *Podaci Raznorodnog Tipa, Formata i Semantike*

PR – *Posrednički Red*

PTC – *Pacific Telecommunications Council*

RPC – *Remote Procedure Call*

SII – *Science Information Infrastructure*

SQL – *Structured Query Language*

SSL – *Security Socket Layer*

TCP – *Transmission-Control Protocol*

UUID – *Universally Unique Identifier*

WWW – *World Wide Web*

WIDL – *Web Interface Definition Language*

Windows DNA – *Windows Distributed interNet Application*

POPIS PRILOGA NA OPTIČKOM DISKU

X:\gsc_mscs.pdf	- magistarski rad u formatu PDF
X:\gsc_mscs.zip	- programske datoteke prototipa modela i mrežne knjižnice
X:\obrana.pdf	- obrana magistarskog rada u formatu PDF
X:\obrana.pps	- obrana magistarskog rada u formatu PPS
X:\sazetak.pdf	- sažetak magistarskog rada na hrvatskom jeziku u formatu PDF
X:\sazetak.txt	- sažetak magistarskog rada na hrvatskom jeziku u formatu TXT
X:\abstract.pdf	- sažetak magistarskog rada na engleskom jeziku u formatu PDF
X:\abstract.txt	- sažetak magistarskog rada na engleskom jeziku u formatu TXT
X:\zivopis.pdf	- životopis u formatu PDF
X:\zivopis.txt	- životopis u formatu TXT
X:\winzip70.exe	- instalacija WinZip 7.0
X:\ar405eng.exe	- instalacija Acrobat Reader 4.05
X:\DBMSs\ntserv.zip	- IBM DB2 Universal Database Version 5.2.0 Enterprise Edition
X:\DBMSs\o8nt.exe	- Oracle8 Release 8.0.3 Database for Windows NT
X:\DBMSs\805ship.tgz	- Oracle8 Release 8.0.5 Database for Linux
X:\DBMSs\sybase.zip	- Sybase Adaptive Server 11.5 Enterprise Database
X:\DBMSs\winnt.zip	- Informix Dynamic Server 7.30.TC3 Database
X:\DBMSs\teradata.zip	- NCR Teradata Version 02.00.01 Database

SAŽETAK

Naslov: Modeliranje pristupa kolekcijama raznorodnih podataka unutar informacijske infrastrukture

Sažetak: Jedna od značajnih usluga koju trebaju pružati informacijske infrastrukture, globalne računalno-informacijske mreže koje objedinjavaju elemente računalne mreže i korisnički usmjerene elemente raspodijeljenih aplikacija, pristup je podacima raznorodnog tipa, formata i semantike. U svrhu olakšanja i poboljšanja interakcije s raspodijeljenim informacijskim sustavom podržanim informacijskom infrastrukturom, kroz ovaj rad postavlja se i razrađuje model koji osigurava jednoobraznost pristupa njegovim podacima definicijom standardnog sučelja. Arhitektura modela spada u klasu trorednih Internet/intranet aplikacija: prvi (korisnički) red zadužen je za komunikaciju s krajnim korisnikom što među ostalim uključuje i grafičko sučelje s višemedijskom podrškom; drugi (posrednički) red vrši uslugu pristupa raspodijeljenim podacima te po potrebi konverziju istih kako bi se podržala jednoobraznost pristupa; treći red predstavljaju raznorodne kolekcije podataka raspodijeljenog informacijskog sustava koje je potrebno integrirati. Prototip sustava, koristeći prvenstveno tehnologiju OLE DB, implementiran je na operacijskom sustavu Windows NT Server 4.0 u skladu s arhitekturom Windows DNA, kroz primjenu objektno orijentirane (C++) i komponentno temeljene (Distributed COM) paradigme. Nedostatci ponovne iskoristivosti COM komponenata koji su se javili prilikom implementacije jezgre posredničkog reda, otklonjeni su proširivanjem COM mehanizama agregiranja. Paralelno s tim otklonjeni su i nedostatci vezani za ponovnu iskoristivost sučelja COM komponenata za prikazne WWW jezike. Kako bi se ilustrirao sam pristup, razvijena je jedna jednostavna mrežna knjižnica, te je ispitana u kombinaciji s danas najčešće korištenim komercijalnim sustavima poslovanja relacijskim bazama podataka uključujući: DB2, Oracle, SQL Server, Sybase, Access, Informix i NCR Teradata. Na kraju, kroz provjeru ispravnosti modela te kroz analizu njegove svojstvene funkcionalnosti, potvrđena je i ispravnost ovakvog pristupa.

Ključne riječi: informacijska infrastruktura, univerzalni pristup podacima, distribuirane aplikacije, zajednički rad, Distributed COM, OLE DB, relacijske baza podataka.

ABSTRACT

Title: Modeling of Access to Collections of Heterogeneous Data Within the Information Infrastructure

Abstract: One of the important services an information infrastructure should provide is to provide access to all kind of data, no matter of their type, format or semantics. To simplify and to improve interaction mechanisms with a distributed information system supported by the information infrastructure, a model is proposed that provides a uniform and universal access to diverse data originating in databases, as well as in other data sources. The architecture of the model can be classified as a Three-Tier Internet/intranet application: the first User Interface (UI) tier responsibility is the communication with the end-user what includes the graphical interface with possible multimedia support; its middle-tier – Universal Data Access (UDA) tier – being the one assigned the mediation task, thus insuring universal data access and providing necessary data format conversions to insure uniform access; the third Diverse Data Sources (DDS) tier consists of data sources available within the information infrastructure. The prototype of such a system, based mostly on OLE DB technology, is implemented on the Windows NT Server 4.0 according to Windows DNA principles; the framework for its implementation is found in the object oriented (C++) and component based (Distributed COM) paradigms. Problems related to reusability of COM components that appeared during UDA core sub-tier implementation were successfully solved introducing extension of COM aggregation. Problems related with reusability of COM interfaces for WWW languages are also successfully solved. A sample bookstore Web application is developed in order to illustrate the approach and is tested against the most promising/commercial DBMSs presently known, here including: DB2, Oracle, SQL Server, Sybase, Access, Informix and NCR Teradata. At the end, the model is evaluated and the correctness of such an approach confirmed.

Keywords: information infrastructure, universal data access, distributed applications, interoperability, Distributed COM, OLE DB, DBMS.

ŽIVOTOPIS

Goran Salamunićcar rođen je u Karlovcu, Republici Hrvatskoj, 6. srpnja 1972. Nakon završetka Karlovačke gimnazije, 1991. godine upisao je Elektrotehnički fakultet (sada Fakultet elektrotehnike i računarstva) Sveučilišta u Zagrebu. Za primjeran uspjeh na 2. godini studija dodijeljeno mu je priznanje "Josip Lončar". Također, dodijeljena mu je "Sveučilišna stipendija" za 4. godinu studija i "Stipendija Grada Zagreba" za 9. (posljednji) semestar studija. Zahvaljujući ovakvom uspjehu, odobren mu je završetak studija s naglaskom na znanstveno-istraživačkom radu na programu *Zasnivanje pogonske sabirnice za prilagodljive obradne sustave* [1][2]. S izvrsnim uspjehom je diplomirao 5. veljače 1996. na smjeru Industrijska elektronika. Zahvaljujući srednjoj ocjeni ispita od 4.68 te pokazanim znanstveno-istraživačkim rezultatima, dodijeljeno mu je oslobađanje od plaćanja za poslijediplomski znanstveni magistarski studij. Iste godine na Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave, upisuje poslijediplomski znanstveni magistarski studij, smjer Jezgra računarskih znanosti. Sve upisane ispite položio je s ocjenom izvrstan (5). Od početka 1998. aktivno sudjeluje i u znanstvenoistraživačkom projektu 036033 *Elementi arhitektura za regionalne informacijske infrastrukture* pod pokroviteljstvom Ministarstva znanosti i tehnologije Republike Hrvatske te Istarske Županije. Od 1. lipnja 1997. do 7. studenog 1999., zaposlen je u firmi PELSYS d.o.o., na razvoju programske podrške za Windows okruženje koristeći prvenstveno Visual C++ 5.0. Od 8. studenog 1999. je na odsluženju vojnog roka, gdje je nakon temeljne i specijalističke obuke zaposlen kao informatičar u Ministarstvu obrane, Glavnom stožeru Oružanih snaga Republike Hrvatske. Njegovi glavni akademski i profesionalni interesi su: računalne mreže i razvoj Internet aplikacija (OLE DB, ASP) [5][6], sustavi poslovanja relacijskim bazama podataka (DB2, Oracle, SQL Server, Sybase i Access), distribuirani sustavi, objektno orijentiran i komponentno temeljen (COM+) razvoj programske podrške, te programski jezici Visual C++ i VHDL [3][4]. Svoje slobodno vrijeme većinom provodi u aktivnostima udruženja IEEE, ACM i Mense kojih je i član.

Objavljeni radovi:

[1] V. Glavinić, G. Salamunićcar, "Architecture for a PROFIBUS Controller", *5th International Symposium on New Technologies – 5thSONT*, Poreč, Hrvatska, rujan 1995, pp. 210-215.

[2] V. Glavinić, G. Salamunićcar, "Case Study of a PROFIBUS FDL Layer Coprocessor", *41st Annual Conference – KoREMA'96*, Opatija, Hrvatska, rujan 1996, Vol. 4 pp. 71-74.

[3] G. Salamunićcar, "A Proposal for Data Modeling Extension to VHDL Using an Object-Oriented Approach", *1st IEEE International Workshop on Design, Test and Application – WDTA'98*, Dubrovnik, Hrvatska, lipanj 1998, pp. 141-144.

[4] G. Salamunićcar, "A Queuing Networks Library Extension for VHDL", *1st IEEE International Workshop on Design, Test and Application – WDTA'98*, Dubrovnik, Hrvatska, lipanj 1998, pp. 145-148.

[5] G. Salamunićcar, V. Glavinić, "On Insuring Interoperability of OLE DB Based Data Access", *21st International Conference on Information Technology Interfaces – ITI'99*, Pula, Hrvatska, lipanj 1999, pp. 317-324.

[6] G. Salamunićcar, V. Glavinić, "An OLE DB Based Web Interface to Databases", *8th Electrotechnical and Computer Science Conference – ERK'99*, Portorož, Slovenija, rujan 1999, pp. 101-104.