Prof. dr Dragica Radosav

SOFTVERSKO INŽENJERSTVO II

BIBLIOTEKA 96 2005 UDŽBENICI

Univerzitet u Novom Sadu
Tehnički fakultet "Mihajlo Pupin", Zrenjanin
Zrenjanin, 2005

Prof.dr Dragica Radosav

SOFTVERSKO INŽENJERSTVO II

- *Troslojna arhitektura*
- *Distribuirani sistemi*
- *CORBA*
- *WEB servisi*
- *XML*
- *Data mining*

Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“,
Zrenjanin, 2005.godine

Predgovor

Knjiga ***Softversko inženjerstvo II*** se sastoji iz pet poglavlja, koja obrađuju šest aktuelnih tema, a to su:

- Troslojna arhitektura
- Distribuirani sistemi
- CORBA
- WEB servisi
- XML
- Data mining

Oblasti su izabrane u saglasnosti sa sadržajem predmeta Softversko inženjerstvo III, na fakultetima koji participiraju na Projektima:

- ***C.D.P. + WUS Austria,*** **za šk.2004/2005.** godinu, Prof.dr Dragica Radosav je rukovodilac Projekta za unapređenje curriculum-a iz Softverskog inženjeringa u BiH.
- TEMPUS JEP PROJEKTA 16110-2001.

Autor je u prethodnom periodu aktivno radio, na izučavanju novih oblasti, kao mentor na poslediplomskim studijama. Sa svojim poslediplomcima: Šarović Vladimirom, Tanjom Sekulić, Višnjom Jovanović, Sonjom Stanković autor je koncipirao radove koji obrađuju teme iz ovog rukopisa, napravio odabir korišćene literature i vodio izradu seminarskih radova, koji su jednim svojim delom, uz odgovarajuće korekcije, prezentovani u ovom rukopisu.

Poglavlje Data mining je dodatno uključeno zbog svoje sve veće primene kod nas i činjenice da se na Tehničkom fakultetu proučava uloga XML-a u Data mining-u, sa posebnim osvrtom na DL.

Autor se nada da je tekst ove knjige uredio na način da bude interesantan za čitaoce koji žele da se bave razvojem određenih oblasti iz domena Softverskog inženjerstva.

Autor

TROSLOJNA ARHITEKTURA / DISTRIBIURANI SISTEMI[1]

[1] *Seminarski rad Šarović Vladimira, urađen pod mentorstvom Prof.dr Dragice Radosav, u okviru predmeta Programiranje i softverski inženjering na postdiplomskom studiju na FIT-u, Univerziteta »Džemal Bijedić« u Mostaru, BiH*

SADRŽAJ:

1. UVOD

Razvoj Web-a tokom protekle decenije doveo je do odgovarajućeg razvoja usluga dostupnih na Web-u. Mnoge od tih usluga su Web lokacije koje funkcionišu pomoću podataka smeštenih u bazama podataka. Primeri baza podataka na Web-u obuhvataju usluge novinskih agencija koje pružaju pristup do velike količine uskladištenih podataka, zatim programe za elektronsko poslovanje (engl. *e-commerce*) kao što su prodavnice na Web-u i proizvode za podršku direktnoj komunikaciji između dva poslovna subjekta (engl. *business-to-business*, B2B).

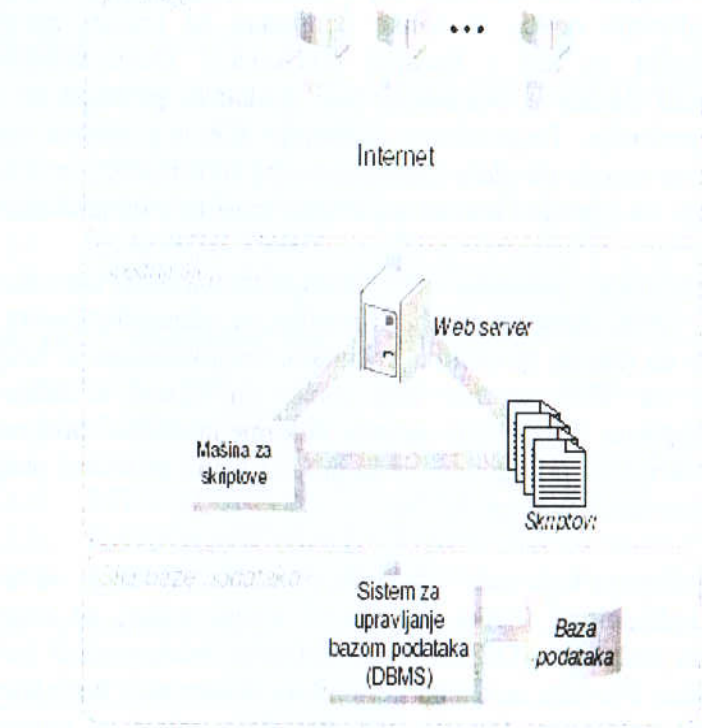
Aplikacije koje rade s bazama podataka postoje preko 30 godina i mnoge su napravljene primjenom mrežnih tehnologija poznatih davno pre nego što je Web uopšte postojao. Takozvani *point-of-service* sistemi koji omogućavaju podizanje novca iz banke očigledan su primer prvobitnih umreženih aplikacija za rad s bazama podataka.U ekspo-ziturama su instalirani terminali banke, a centralnoj bazi podataka pristupa se preko mreže za široko područje. Te prvobitne aplikacije bile su pogodne samo za organizacije koje su mogle da plate specijalizovanu terminalsku opremu i, u nekim slučajevima, da izgrade i koriste sopstvenu mrežnu infrastrukturu.

Web se realizuje jeftinim, lako dostupnim načinom umrežavanja. Postoji izuzetno veliki broj njegovih korisnika sa stanardizovanim Web čitačima koji rade na raznim modelima računara. Projektantima je besplatno dostupan softver za Web servere koji može da ispuni i zahteve za izvršavanjem programa i za učitavanjem dokumenata.Više programskih jezika za pisanje skriptova prilagođeno je ili projektovano za razvoj programa koji koriste Web servere i Web protokole.

Većina aplikacija koje rade s bazama podataka povezuje se na Web preko tri sloja aplikacione logike. U osnovi svega nalazi se sistem za upravljanje bazom podataka - DBMS (engl. *database management system*) i sama baza podataka. Na vrhu se nalazi korisnikov (klijentski) Web čitač koji služi kao interfejs aplikacije. Između ova dva sloja nalazi se najveći dio logike aplikacije, koja se obično realizuje u obliku skriptova koji komuniciraju sa DBMS-om na strani Web servera i koji mogu da kodiraju i generiraju HTML kod potreban za prezentaciju podataka u korisnikovom čitaču WEB-a.

2. TROSLOJNA ARHITEKTURA

Na slici 1 prikazana je web aplikacija napravljena po modelu troslojne arhitekture. Na najnižem nivou aplikacije nalazi se *sloj baze podataka*. On se sastoji od *sistema za upravljanje bazom podataka*, a njegova uloga je upravljanje bazom čije podatke korisnici unose, brišu, menjaju ili pretražuju. Iznad sloja baze podataka nalazi se složeni srednji *sloj* koji sadrži najveći dio logike aplikacije i prenosi podatke između preostala dva sloja. Na vrhu se nalazi *klijentski sloj*, obično Web čitač koji komunicira sa aplikacijom.



Slika 1: Arhitektura troslojnog modela WEB aplikacije koja radi s bazom podataka.

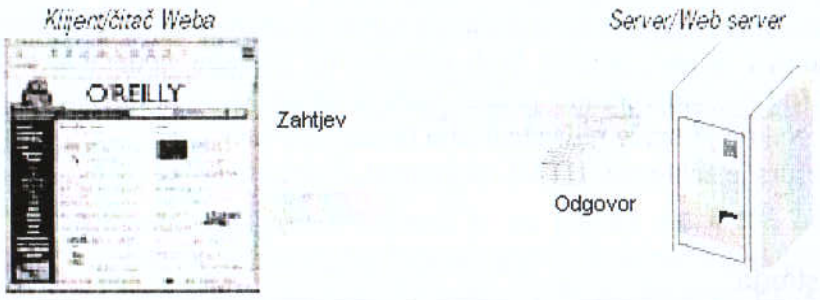
Formalan način opisivanja većine Web aplikacija kao troslojne arhitekture skriva činjenicu da takve aplikacije moraju da povežu različite protokole i softver. Kada koristimo reč Web prvenstveno mislimo na tri glavna, zasebna standarda i na njima zasnovane alate: jezik za označavanje hiperteksta (engl. Hypertext Markup Lan-guage,HTML),protokol za prenos hiperteksta (engl.Hypertext Transfer Protocol,HTTP) i paket mrežnih

protokola TCP/IP. HTML je pogodan za strukturiranje i prikazivanje podataka u Web čitačima. TCP/IP je efikasan mrežni protokol koji prenosi podatke između aplikacija na Internetu i ima vrlo malo značaja za projektante Web aplikacija. Problem u izradi dinamičkih Web aplikacija predstavlja komuniciranje pomoću HTTP protokola s klasičnim bazama podataka koje rade na Web-u. Tu je neophodna složena aplikaciona logika.

Protokol za prenos hiperteksta (HTTP)

Troslojna arhitektura obezbeđuje konceptualni temelj za izradu Web aplikacija koje pristupaju bazama podataka. Sam Web obezbeđuje protokole i mrežu koji povezuju klijentski i srednji sloj aplikacije, tj. obezbeđuje vezu između čitača Weba i Web servera. HTTP je komponenta koja povezuje sve slojeve u troslojnoj arhitekturi. Web čitači šalju zahteve za resursima pomoću HTTP protokola, dok Web serveri šalju odgovore na te zahteve. HTTP omogućava prosleđivanje i deljenje resursa na Webu. Posmatrajući iz perspektive mreže, HTTP predstavlja protokol aplikacionog sloja koji se nalazi neposredno iznad paketa TCP/IP mrežnih protokola. Većina Web servera i Web čitača komunicira pomoću tekuće verzije protokola, HTTP/1.1. Neki čitači i serveri koriste prethodnu verziju, HTTP/1.0 ali većina HTTP/1.1 softvera kompatibilna je s prethodnom verzijom HTTP/1.0. Komunikacije pomoću HTTP-a dominiraju mrežnim saobraćajem na Internetu. Tokom 1997. godine, pomoću HTTP-a se odvijalo oko 75% celokupnog saobraćaja. Smatra se da je ovaj procenat sada još veći s obzirom na porast broja i popularnosti aplikacija zasnovanih na HTTP-u (npr. usluge besplatne e-pošte).

HTTP je konceptualno jednostavan: klijentski čitač Web-a šalje Web serveru zahtev za određenim resursom, a Web server potom šalje čitaču odgovor. HTTP odgovor prenosi zahtevani resurs – HTML dokument, sliku ili izlazne podatke nekog programa – natrag do čitača Web-a kao ' koristan tovar". Jednostavan model zahtev–odgovor prikazan je na slici 2.



Slika 2: Čitač Web-a šalje zahtev, a Web server šalje odgovor u kome je sadržan zahtevani resurs

HTTP zahtev se sastoji od tekstualnog opisa resursa i dodatnih informacija koje se zovu zaglavlja (engl. headers). Primer HTTP zahteva:

GET /index.html HTTP/1.0
From: hugh@computer.org (Hugh Williams)
User-agent: Hugh-fake-browser/version-1.0 ,
Accept: text/plain, text/html

U ovom primeru se metodom GET zahteva HTML stranica index.html pomoću HTTP/1.0 verzije protokola.Tri dodatna reda zaglavlja identifikuju korisnika, čitač Web-a, i definišu tipove podataka koje čitač može da prihvati. Zahtev obično šalje čitač Web-a i on može da sadrži i druga zaglavlja.

HTTP odgovor sadrži kôd odgovora i poruku, dodatna zaglavlja, a najčešće i zahtevani resurs. Sledi primer odgovora na zahtev za Web dokumentom index.html:

HTTP/1.0 200 OK
Date: Sat, 21 Jul 2002 03:44:25 GMT
Server: Apache/1.3.20
Content-type: text/html
Content-lenght: 88
Last-modified: Fri, 1 Feb 2002 03:40:03 GMT
<html><head>
<title>Test stranica</title></head>
<body>
<h1>Ovo radi!</h1>
</body></html>

U prvom redu odgovora server obaveštava da prihvata upotrebu HTTP/1.0 protokola i potvrđuje da je uspešno primio zahtev ispisivanjem koda odgovora 200 i poruke OK; drugi čest odgovor je 404 Not Found (zahtevani resurs nije nađen). U pet redova dodatnih zaglavlja ovog primera prikazani su tekući datum i vreme, softver Web servera, tip poslatih podataka, dužina odgovora (u bajtovima), kao i podaci kada je zahtevani resurs poslednji put izmenjen. Nakon praznog reda sledi sam resurs. U ovom primeru resurs je zahtevani HTML dokument, *index.html*.

Čuvanje stanja

Klasične aplikacije koje rade s bazama podataka čuvaju stanje: korisnici se prvo prijavljuju, zatim izvršavaju odgovarajuće transakcije i na

kraju, kad obave posao, odjavljuju se. Na primer, aplikacija za banku zahteva da se operater na šalteru prvo prijavi, zatim da koristi aplikaciju pomoću niza menija dok opslužuje zahteve tekućeg komitenta, da bi se na kraju dana odjavio. Aplikacija za banku ima više stanja: kada se blagajnik prijavi, on može da koristi aplikaciju na strukturiran način, pomoću menija. Kada se blagajnik odjavi, on ne može više da koristi aplikaciju. HTTP ne čuva stanje.Kad kažemo da se stanje ne čuva, to znači da je svaka interakcija između čitača Weba i Web servera nezavisna od bilo koje druge interakcije. Svaki HTTP zahtev čitača Web-a sadrži identične informacije u zaglavlju kao što su identifikacioni podaci korisnika, tipovi stranica koje čitač može da prihvati i instrukcije za formatiranje odgovora. Činjenica da se stanje ne čuva ima i svojih prednosti: najvažnija je ušteda resursa pošto nema potrebe da se na Web serveru održavaju informacije pomoću kojih bi se pratile aktivnosti korisnika, kao i fleksibilnost koja korisnicima omogućava da se kreću između nepovezanih stranica ili resursa. Pošto HTTP ne čuva tekuće stanje, vrlo je teško projektovati Web aplikacije koje bi čuvale tekuće stanje. Neophodan je način za održavanje stanja u HTTP-u tako da podaci protiču i može da se nametne određena struktura pri upotrebi aplikacije. Često rešenje je razmena žetona (engl.token) između čitača Web-a i Web servera pomoću kojih se na jedinstven način identifikuje korisnik i njegova sesija. Svaki put kada čitač uputi zahtev za određenim resursom, on šalje žeton, i svaki put kada se Web server odazove, on vraća žeton čitaču Web-a. Softver u srednjem sloju pomoću žetona obnavlja podatke o korisniku od njegovog prethodnog zahteva (npr. kom je meniju u aplikaciji poslednji put pristupljeno). Razmena žetona omogućava da se aplikaciji dodaju strukture za koje je neophodno čuvanje stanja, kao što su meniji, koraci i praćenje procesa rada

Lagani klijenti

Čitači Weba su vrlo lagani klijenti: vrlo mali dio logike aplikacije je uključen u klijentski sloj. Čitač samo šalje HTTP zahteve za resursima i prikazuje odgovore, koji mahom sadrže HTML dokumente. Troslojni model znači da ne morate da pravite, instalirate ili podešavate klijentski sloj. Svaki posetilac koji ima čitač Web-a može da koristi Web aplikaciju koja čita podatke iz baze, obično bez potrebe da instalira dodatni softver, da koristi specifičan operativni sistem ili određenu hardversku platformu. To znači da aplikacija može da opsluži bilo koji broj različitih, geografski udaljenih korisnika.

Alternativa laganom klijentu je na primer namenski napisan Java aplet. Java aplet je primer težeg klijenta koji se ipak može uklopiti u troslojni model: korisnik preuzima aplet i izvršava više aplikacione logike na svojoj platformi nego u slučaju laganog klijenta. Aplet i dalje komunicira sa srednjim slojem koji, pak predstavlja interfejs ka sloju baze podataka. U

ovom slučaju prednost je prilagođenost određenoj nameni: umesto da se koristi generički čitač, koristi se namensko rešenje koje eliminiše mnoge probleme oko čuvanja stanja, bezbednosti i pomanjkanja fleksibilnosti Web-a. Aplet uopšte ne mora da koristi HTTP protokol za komuniciranje sa logikom aplikacije u srednjem sloju. Težak klijent je deo klasičnog dvoslojnog rešenja, koje je takođe poznato pod nazivom klijent/server arhitektura. Većina klasičnih aplikacija koje pristupaju bazama podataka (poput onih u bankama) ima samo dva sloja. Klijentski sloj sadrži najveći dio logike aplikacije, dok je serverski sloj, zapravo, sam DBMS. Prednost ovakve arhitekture je to što mogu da se projektuju namenska rješenja koja potpuno zadovoljavaju specifične aplikacione zahteve, bez ikakvih kompromisa. Mane su nedostatak fleksibilnosti po pitanju hardvera i operativnog sistema, kao i potreba da se svakom korisniku instalira softver.

2.1 KLIJENTSKI SLOJ

Klijentski sloj u modelu troslojne arhitekture obično je čitač Web-a. Čitač Web-a obrađuje, prikazuje HTML resurse, šalje HTTP zahteve za resursima i obrađuje HTTP odgovore. Kao što je već pomenuto, upotreba čitača Web-a kao tankog klijentskog sloja pruža značajne prednosti, među kojima su jednostavan razvoj i podrška za širok opseg različitih platformi. Na raspolaganju je veliki broj čitača Web-a i svaki od njih ima drugačije osobine. Dva najpopularnija čitača zasnovana na okruženju sa prozorima jesu Netscape i Internet Explorer. Zajedničke odlike ova dva čitača su:

- Svi čitači Web-a su HTTP klijenti koji šalju zahteve i prikazuju odgovore dobijene od Web servera (obično u nekom grafičkom okruženju).
- Svi čitači tumače sadržaj stranica sa HTML kodom kada prikazuju stranicu, tj.oni korisnicima prikazuju zaglavlja, slike, hiperveze itd.
- Neki čitači prikazuju slike, reprodukuju filmove i zvuk i vizuelizuju druge tipove objekata.
- Mnogi čitači mogu da izvršavaju JavaScript kôd ugrađen u HTML stranice. Java Script se koristi za, na primer, proveru ispravnosti podataka u HTML obrascu <form> ili promenu izgleda i sadržaja stranice u zavisnosti od korisnikovih akcija.
- Neki čitači Web-a mogu da pokreću komponente razvijene pomoću programskih jezika Java i ActiveX. Te komponente često daju dodatne animacije, alatke koje nije moguće izraditi pomoću HTML-a ili, druge, još složenije mogućnosti.
- Nekoliko čitača može da primenjuje kaskadne liste stilova (engl. Cascading Style Sheets, CSS) na HTML stranice kako bi upravljali prikazivanjem HTML elemenata

Postoje manje (a nekad i ne baš tako male) razlike u načinima na koje pojedini čitači Web-a prikazuju HTML stranice. Lynux je, na primer, čitač koji prikazuje isključivo tekst a ne prikazuje slike, niti izvršava JavaScript kôd. MultiWeb je čitač koji tekst na stranici reprodukuje kao zvuk (izgovorene reči), što omogućava pristup Webu osobama sa oštećenim vidom. Postoje mnoge manje razlike u podršci za CSS i mogućnosti HTML standarda, HTML 4.

Čitači Web-a su najočigledniji primer korisničkog agenta, softverskog klijenta koji zahteva resurse od Web servera.U druge korisničke agente spadaju Web pauci (automatizovani softver koji krstari Webom i pronalazi Web stranice) i posredničke ostave (keš), softverski sistemi koji učitavaju Web stranice i lokalno ih skladište kako bi one bile na raspolaganju drugim korisničkim agentima.

2.2 SREDNJI SLOJ

U većini troslojnih sistema baza podataka na Web-u, najveći dio logike aplikacije nalazi se u srednjem sloju. Klijentski sloj prikazuje podatke korisniku i prihvata podatke koje on unosi; sloj baze podataka skladišti i učitava podatke. Srednji sloj obuhvata većinu preostalih zadataka pomoću kojih se objedinjuju ostali slojevi: upravlja strukturom i sadržajem podataka koji se prikazuju korisniku i obrađuje podatke dobijene od korisnika pretvarajući ih u upite za učitavanje ili upisivanje u bazu podataka. Takođe, ovaj sloj omogućava upravljanje stanjem uz upotrebu HTTP protokola. Logika aplikacije ugrađena u srednji sloj integriše Web sa sistemom za upravljanje bazom podataka.

Kao što se može vidjeti na slici 1, srednji sloj se sastoji od web servera, programskih jezika za pisanje skriptova i mašina za izvršavanje skriptova. Web server obrađuje HTTP zahteve i formuliše odgovore. U slučaju Web aplikacija, ti zahtevi su obično upućeni programima koji komuniciraju sa osnovnim sistemom za upravljanje bazom podataka. a za pisanje skriptova koji rade u srednjem sloju koristimo programski jezik PHP. Budući da je PHP nastao kao projekat otvorenog koda iza koga stoji Apache Software Foundation, ne iznenađuje što je to najpopularniji dodatni modul za Apache HTTP server, sa oko 40% Apache HTTP servera koji podržavaju PHP.

Web serveri se često nazivaju i HTTP serverima. Izraz "HTTP server" je dobar sažetak funkcije koju obavljaju: njihov osnovni zadatak je da otkrivaju pristizanje HTTP zahteva iz mreže, primaju HTTP zahteve koje

šalju korisnički agenti (obično čitači Web-a), obrađuju te zahteve i vraćaju HTTP odgovore koji sadrže tražene resurse.

U suštini postoje dva tipa zahteva koji se šalju Web serveru: prvi, kada se zahteva slanje određene datoteke (obično statičke HTML Web stranice ili slike) i drugi, kada se zahteva pokretanje programa i slanje izlaznih podataka tog programa korisničkom agentu.

2.3 SLOJ BAZE PODATAKA

Sloj baze podataka je osnova Web aplikacija koje rade s bazama podataka. Razumevanje sistemskih zahteva, odabir softvera za sloj baze podataka, projektovanje baza podataka i izrada sloja prvi su koraci uspešnog projektovanja aplikacija.

U aplikaciji sa troslojnom arhitekturom, sloj baze podataka zadužen je za upravljanje podacima. Upravljanje podacima obično podrazumeva skladištenje i učitavanje podataka, kao i upravljanje postupkom ažuriranja, pri čemu više procesa iz srednjeg sloja može simultano, tj. istovremeno da pristupa, zatim, zaštitu podataka, očuvanje njihovog integriteta i obezbeđivanje usluga za podršku, poput izrade rezervnih kopija. U mnogim bazama podataka za Web sve te poslove obavlja sistem RDBMS i podaci smešteni u relacionoj bazi podataka. Upravljanje relacionim podacima u trećem sloju zahteva složeni RDBMS softver. Na sreću, većina RDBMS-ova je projektovana tako da se složenost softvera ne primećuje.

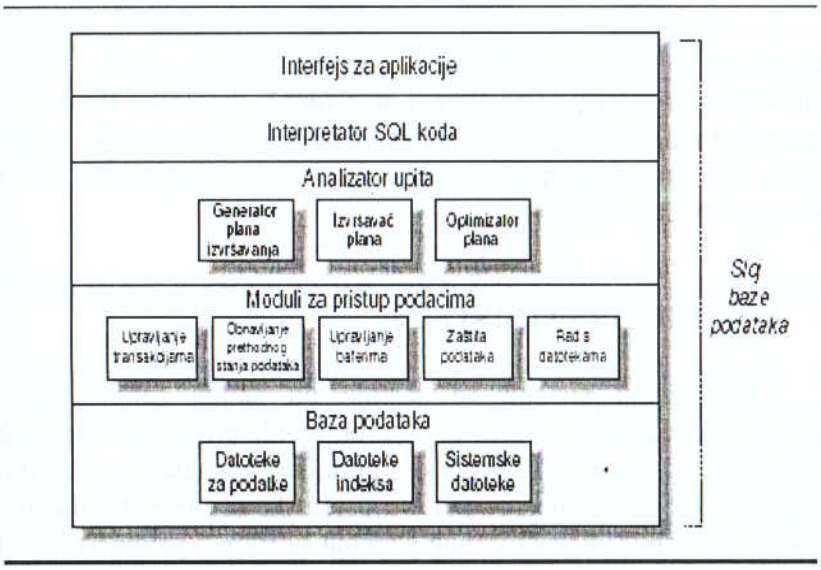
Da bi se efikasno koristio DBMS, potrebno je znati projektovanje baze podataka formulisanje komandi i upita za DBMS. U većini DBMS-ova jezik za upite je SQL. Većini korisnika nije potrebno da razumeju unutrašnju arhitekturu DBMS-a. Poput rasprava o tome koji programski jezik treba koristiti za skriptove u srednjem sloju, raspravlja se i o tome koji je DBMS najprikladniji za određenu aplikaciju. MySQL ima potpuno zasluženu dobru reputaciju o brzini i posebno je dobro projektovan za aplikacije gde se podaci mnogo češće učitavaju nego što se ažuriraju i za slučajeve gde su manja, jednostavnija ažuriranja uobičajena vrsta izmena.To su tipične karakteristike većine aplikacija koje rade s bazama podataka. Osim toga, poput PHP-a i Apachea, MySQL spada u softver otvorenog koda.

Postoje i druga, nerelaciona DBMS softverska rešenja za skladištenje podataka u sloju baze podataka. Tu spadaju mašine za pretraživanje, sistemi za upravljanje dokumentima i jednostavniji mrežni servisi, poput softvera za elektronsku poštu.

2.3.1 SISTEMI ZA UPRAVLJANJE BAZAMA PODATAKA

Sistem za upravljanje bazama podataka skladišti podatke, pretražuje ih i upravlja njima. Baza podataka je skup povezanih podataka. Uskladišteni podaci mogu se sastojati od svega nekoliko ulaznih podataka ili redova koji čine jednostavan adresar sa imenima, adresama i telefonskim brojevima. Nasuprot tome, baza podataka može da sadrži i milione zapisa koji opisuju katalog, kupovine, porudžbine i platni spisak velike kompanije.

Kada se kaže sistem za upravljanje bazom podataka (DBMS), uglavnom se misli na relacioni DBMS tj. RDBMS. Relacione baze podataka skladište veze između podataka i upravljaju njima. Na primer, kupci poručuju proizvode, porudžbine kupaca sadrže artikle iz proizvodne linije.



Slika 3: Struktura tipičnog DBMS-a

DBMS se sastoji od nekoliko komponenata:

Interfejs za aplikacije
Biblioteke za komunikaciju sa DBMS-om. Većina DBMS-ova ima jednostavan prevodilac u obliku komandne linije koji često koristi te biblioteke za prenošenje zahteva
unetih sa tastature do DBMS-a i za prikazivanje odgovora. U Web aplikacijama zasnovanim na korištenju baza podataka, prevodilac u obliku komandne linije obično je zamenjen funkcijskom bibliotekom koja je deo jezika za skriptove srednjeg sloja.

Interpretator SQL koda
Sintaksni analizator proverava sintaksu prosleđenih upita i prevodi ih u interni prikaz.

Analizator upita
Generiše razne planove izvršavanja upita na osnovu statistike baze podataka i njenih svojstava, odabire jedan od planova i prevodi ga u akcije koje izvršava na niskom nivou.

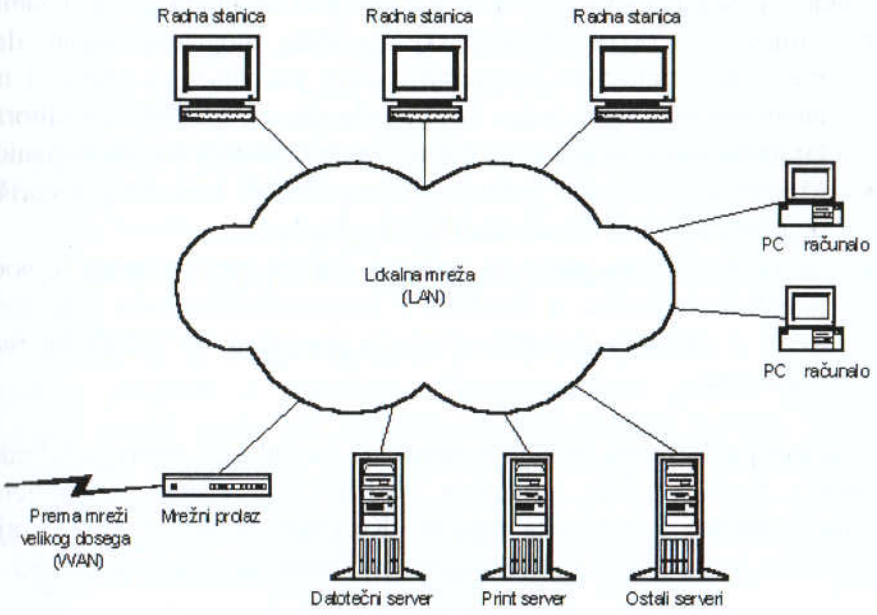
Pristup podacima
U module koji upravljaju pristupom podacima uskladištenim na disku spadaju modul za upravljanje transakcijama, modul za upravljanje postupkom obnavljanja stanja, modul za upravljanje baferom glavne memorije, modul za zaštitu podataka i modul za upravljanje pristupom datotekama.

Baza podataka
To su, zapravo, sami podaci fizički smešteni u datoteke. Podaci pored toga obuhvataju i indeksne datoteke za brz pristup, kao i statističke podatke o bazi podataka i sistemu, koji se prvenstveno koriste za generisanje i optimizovanje planova za izvršavanje upita.

3. DISTRIBUIRANI SISTEMI

Distribuirani sistemi se sastoje od skupa samostalnih računara povezanih računarskim mrežama i opremljenih programskom podrškom za distribuirane sisteme. Programska podrška omogućava računarima da koordiniraju svoje aktivnosti i da dele sistemske resurse - hardware, software i podatke. Korisnici distribuiranog sistema bi trebali primećivati samo jedan integrirani sistem iako on može biti implementiran pomoću mnogo različitih računara na različitim lokacijama.

Razvoj distribuiranih sistema je sledio pojavu brzih lokalnih mreža početkom sedamdesetih godina. Pojavom jakih PC računara, radnih stanica i servera došlo je do napuštanja centralizovanih i višekorisničkih računara, čemu je pridonio i razvoj distribuirane programske podrške, kao i distribuiranih aplikacija.



Slika 4: Jednostavan distribuirani sistem

3.1 OSNOVNE KARAKTERISTIKE DISTRIBUIRANIH SISTEMA

Postoji šest osnovnih karakteristika distribuiranih sistema. To su: deljenje resursa (resource sharing), otvorenost (openess), istovremenost (*concurrency*), skalabilnost (*scalability*), otpornost na pogreške (*fault tolerance*) te transparentnost (*transparency*). Niti jedna od tih karakteristika nije posledica distribuiranih sistema. Distribuirani sistemi moraju biti pažljivo planirani i izvedeni na taj način da osiguraju svaku od tih karakteristika.

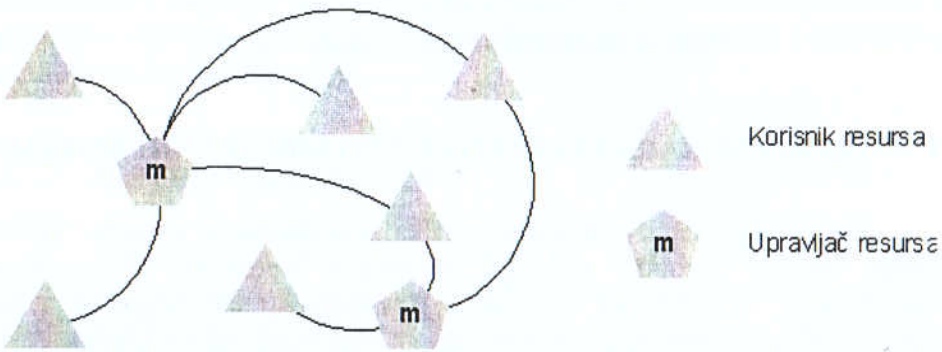
3.1.1. DELJENJE RESURSA

Iako je pojam resursa apstraktan, on najbolje opisuje što je sve moguće deliti u distribuiranim sistemima. Prednosti deljenja resursa kao što su baza podataka, programi, dokumentacija i ostale informacije su se prvi puta pokazale pojavom višekorisničkih sistema (*multi-user*) te sistema sa podelom vremena (*time sharing*) početkom 1960-ih, te pojavom višekorisničkih UNIX operativnih sistema 1970-ih godina.

Oprema kao što su štampači, diskovi velikog kapaciteta i ostali uređaji mogu biti deljeni zbog lakoće upotrebe i pristupačnosti. Deljenje podataka je jedan od osnovnih zahteva u mnogim računarskim primenama.

- Timovi za razvoj programske podrške mogu pristupati dosada razvijenim delovima programa i deliti iste alate za razvoj i na taj način koristiti samo jednu kopiju prevodioca, biblioteka i editora. Na taj način nova verzija alata postaje odmah dostupna svim korisnicima.
- Mnoge komercijalne aplikacije omogućavaju korisnicima korištenje objekata u jednoj zajedničkoj bazi podataka
- Najperspektivnije područje primene distribuiranih sistema je podrška grupama korisnika u timskom i kooperativnom radu koje uveliko ovise o deljenju podataka između programa na različitim radnim stanicama

Izraz **upravljač resursa** opisuje programski modul koji upravlja određenim resursom. Sledeća slika prikazuje distribuirani sistem sastavljen od upravljača resursa i korisnika resursa. Korisnici resursa komuniciraju sa upravljačima resursa da bi dobili mogućnost korištenja deljenog resursa.

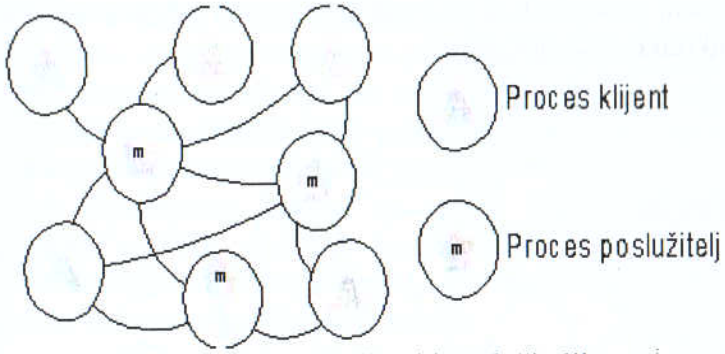


Slika 5: Upravljači i korisnici resursa

Iz takvog razmatranja razvijaju se dva osnovna modela: klijent-poslužitelj model i model zasnovan na objektima.

Klijent-poslužitelj model

To je najviše primenjivan model distribuiranih sistema. Sastoji se od procesa poslužitelja koji su upravljači resursa određenog tipa i od procesa klijenta koji za izvršavanje svog zadatka zahtevaju pristup deljenim resursima. Sami poslužitelji mogu trebati neke od deljenih resursa, i neki poslužitelji mogu istovremeno biti klijenti nekim drugim poslužiteljima.



Slika 6: Procesi klijenti i poslužitelji

Proces se u ovom kontekstu shvata jednako kao u operacijskim sistemima: program u izvođenju. Pojednostavljeni pogled na klijent-poslužitelj model može biti centralizovani poslužitelj resursa. Međutim, centralizovano posluživanje je neželjeno u distribuiranim sistemima. Zbog toga se pravi razlika između poslužitelja (*servers*) i usluga koje oni pružaju (*services*). Usluga je apstraktni entitet koji se može pružati preko nekoliko poslužitelja na različitim računarima koja sarađuju preko mreže.

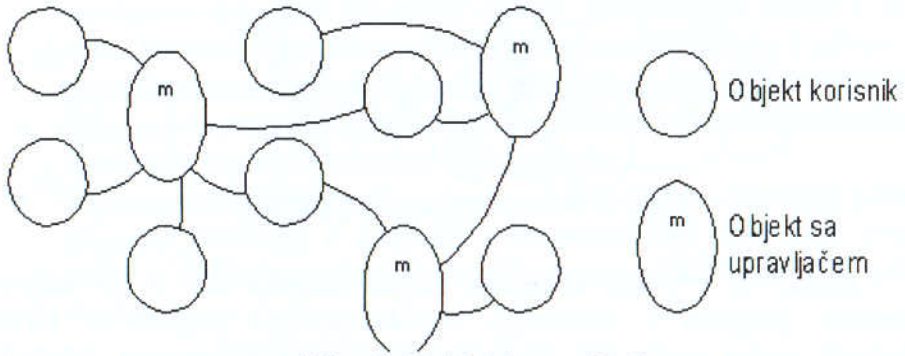
Klijent-poslužitelj model je uspešno primenjen kod deljenja različitih resursa: elektronske pošte, mrežnih novosti, kod deljenja datoteka, sinhronizacija satova računara u mreži, deljenja prostora na diskovima i drugim. Međutim, nije moguće sve resurse deliti na taj način. Neki resursi moraju ostati lokalni za svaki računar: radna memorija (RAM), centralna upravljačka jedinca (CPU) i mrežno sučelje se smatraju najmanjim skupom resursa koji moraju ostati lokalni. Ti se ključni resursi mogu deliti samo među procesima koji se izvode na istom računalu.

Klijent-poslužitelj model ne zadovoljava sve uslove - neke primene zahtevaju čvršću povezanost između klijenta nego što je to moguće u klijent-poslužitelj modelu, međutim, on je pogodan za veliki broj primena i kao baza za opšte distribuirane sisteme.

Model zasnovan na objektima

Ovaj model je sličan tradicionalnom objektnom modelu koji se koristi kod programiranja. On svaki izvršni deo programa posmatra kao objekt koji poseduje sistem za razmenu poruka pomoću kojeg se pristupa do mogućnosti objekta. U objektnom distribuiranom modelu, svaki deljeni resurs posmatra se kao objekt. Objekti su jednoznačno određeni i mogu menjati položaj na mreži bez menjanja identiteta. Kada program pokaže potrebu za resursom, on šalje poruku koja sadrži zahtev za objektom. Poruka se prosleđuje

odgovarajućoj proceduri ili procesu koji izvodi zahtevanu operaciju i šalje odgovor ukoliko je to potrebno.



Slika 7: Objekti i upravljači

Slika 7 pokazuje privlačnu jednostavnost takvog modela. On omogućava na ujednačen način pristup **svim** deljivim resursima od strane objekata korisnika. Kao i kod klijent-poslužitelj modela, objekti mogu biti istovremeno i upravljači i korisnici. Dok je kod klijent-poslužitelj modela način imenovanja zavisio o poslužitelju koji je pružao uslugu, kod objektnog modela je imenovanje svih resursa uvek ostvareno na isti način.

Kod primene objektnog modela pojavljuju se određeni problemi. On zahteva da se upravljači objekata nalaze na istom mestu kao i objekti kojima oni upravljaju zbog toga što oni poseduju reprezentaciju stanja upravljanog objekta. To je jednostavno kod sistema kod kojih se objekti ne mogu pomicati i taj pristup je primenjen u većini današnjih objektnih distribuiranih sistema. U eksperimentalnoj fazi nalaze se primene koje omogućavaju nezavisno razmeštanje objekata od njihovih upravljača.

3.1.2. OTVORENOST

Otvorenost je karakteristika operativnih sistema koja definiše mogućnost proširivanja na različite načine. Sistem može biti otvoren na sklopovskom nivou - npr. za dodavanje dodatnih vanjskih uređaja, memorije ili komunikacijskih uređaja, ili na programskom nivou - za dodavanje novih funkcija operativnom sistemu, novih komunikacijskih protokola ili servisa za deljenje resursa. Otvorenost distribuiranih sistema se većinom odnosi na mogućnost dodavanja servisa za deljenje resursa bez uznemirivanja ili udvajanja postojećih resursa.

Otvorenost se dakle, može svesti na:

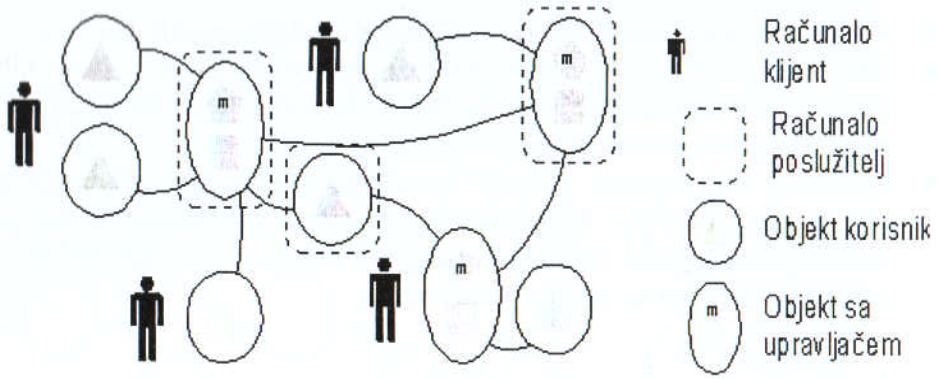
- Karakteristiku otvorenih sistema da svoja sučelja publiciraju.
- Otvoreni distribuirani sistemi se baziraju na pružanju ujednačenih mehanizama komunikacije među procesima i publikovanju sučelja da bi se omogućio pristup do deljenih resursa.
- Otvoreni distribuirani sistemi se mogu graditi od sklopova i programske podrške različitih proizvođača. Međutim, da bi se korisnici zaštitili od neusklađenosti, svi delovi se moraju brižljivo testirati s obzirom na publikovana sučelja.

3.1.3. Istovremenost

Kada na istom računaru postoji nekoliko procesa, oni se odvijaju istovremeno. Ako je računar opremljen samo sa jednom centralnom upravljačkom jedinicom, onda se to postiže naizmeničnim izvršavanjem procesa. Ukoliko računar ima N centralnih upravljačkih jedinica, do N procesa se može izvršavati paralelno što takođe rezultira N-terostrukim poboljšanjem.

U distribuiranom sistemu se nalazi više računara, svaki sa jednom ili više centralnih upravljačkih jedinica. Ako se radi o M računara, onda se istovremeno može izvršavati M procesa. Istovremenost se postiže u dva slučaja (kao što je to prikazano na slici 8):

1. Kada više korisnika koriste istovremeno program
2. Kada više procesa poslužitelja istovremeno poslužuju zahteve sa više procesa korisnika.



Slika 8: Istovremenost kod distribuiranih sistema

Istovremenost i paralelnost se, dakle, pojavljuju prirodno kod distribuiranih operativnih sistema kao posledica smeštanja procesa poslužitelja na različita računara. Takva podela procesa rezultira paralelnošću izvršavanja na različitim računarima. Istovremeni pristup i menjanje deljenih resursa mora biti usklađeno.

3.1.4. SKALABILNOST

Distribuirani sistemi moraju funkcionisati efikasno bez obzira na veličinu. Najmanji praktičan distribuirani sistem sastoji se od dve radne stanice i datotečnog poslužitelja. Distribuirani sistemi sagrađeni oko lokalne mreže mogu imati više stotina radnih stanica i više poslužitelja različitih namena. Više lokalnih mreža može biti međusobno povezano tako da više hiljada računara zajedno čine jedan distribuiran sistem koji omogućava deljenje resursa među njima.

Potreba za skalabilnošću nije samo problem opreme ili mrežnih karakteristika. Taj problem prožima sve delove distribuiranih sistema. Za razliku od centralizovanih sistema gde su resursi kao što su memorija, upravljačke jedinice i ulazno-izlazni kanali ograničeni i ne mogu se dodavati proizvoljno, distribuirani sistemi omogućavaju dodavanje takvih resursa bez fiksnih ograničenja. Međutim, ukoliko sistem nije bio planiran sa skalabilnošću u vidu, mogu se pojaviti ograničenja.

3.1.5. OTPORNOST NA POGREŠKE

Transparentnost se definiše kao skrivanje pojedinih komponenti distribuiranog sistema od krajnjeg korisnika i programera aplikacija na način da se sistem shvata kao celina, a ne kao skup nezavisnih delova.

Deljenje sistema na delove je osnovna karakteristika distribuiranih sistema. Posledice toga uključuju potrebu za komunikacijom među delovima i potrebu za upravljanjem i integracijom delova. Podela omogućava stvarni paralelizam u izvođenju programa, prikrivanje ispada opreme i oporavljanje od pogrešaka bez uznemirivanja celog sistema, izolaciju i kontrolu komunikacijskih kanala kao meru zaštite sadržaja kod prenosa, te rast ili smanjivanje sistema dodavanjem ili oduzimanjem komponenti.

ANSA Reference Manual i *International Standards Organization's Reference Model for Open Distributed Processing (RM-ODP)* određuju osam tipova transparentnosti. To su i osnovni motivi za izgradnju distribuiranih sistema.

Transparentnost pristupa - omogućava da se lokalni i udaljeni resursi predstavljaju na jednak način

Transparentnost pozicije - omogućava da informacije o objektima ne zavise od fizičkog smeštaja

Transparentnost istovremenosti - pruža mogućnost da više procesa istovremeno pristupa podacima bez problema koji se pri tome mogu pojaviti

Transparentnost udvajanjem - služi da se poveća pouzdanost i kvalitet pristupa udvajanjem objekata

Transparentnost na pogreške - omogućava izvršavanje korisničkih zadataka bez obzira na ispade opreme

Transparentnost na promjene lokacije sadržaja - bez uticaja na korisnike

Transparentnost performansi - omogućava prilagođavanje sistema za veća opterećenja bez uznemirivanja korisnika

Transparentnost na rast - omogućuje rast bez promene strukture sistema ili algoritama

Dve najvažnije transparentnosti su transparentnost na pristup i transparentnost pozicije. Njihova prisutnost ili nedostatak najjače utiču na

upotrebu distribuiranih resursa. Jednim imenom se nazivaju mrežna transparentnost.

4. OBLIKOVANJE DISTRIBUIRANIH SISTEMA

Osnovi ciljevi oblikovanja distribuiranih sistema su sledeći:

- performanse
- pouzdanost
- skalabilnost
- ujednačenost (consistency)
- sigurnost

Iako oblikovanje distribuiranih sistema zahteva razmatranje i ostalih problema nevezanih sa distribucijom, mi ćemo se ograničiti na sledeće:

- Imenovanje - imena bi trebala biti nezavisna od lokacije
- Komunikacije - potrebno je optimalno koristiti komunikacione resurse
- Struktura programske podrške - mora osigurati otvorenost, što se postiže definisanjem sučelja
- Raspoređivanje opterećenja - postizanje dobrih svojstava sistema da bi se postigli najbolji rezultati bez obzira na opterećenje sistema
- Ujednačenost sistema - problem ujednačenosti je jedan od najvećih problema distribuiranih sistema.

4.1 IMENOVANJE

Proces koji zahteva resurs sa kojim upravlja mora znati njegovo ime ili identifikator. Ime označava naziv koji ima značenje za korisnika ili sistem, dok identifikator označava naziv koji ima smisla **samo** za sistem.

Kažemo da je ime *određeno (resolved)*, kada je pretvoreno u oblik pogodan za pokretanje akcije nad resursom ili objektom na koji se ime odnosi. U distribuiranim sistemima određeno ime (*resolved name*) je najčešće **komunikacijski identifikator** zajedno sa ostalim atributima koji su korisni za uspostavljanje veze.
Imenovanje uključuje nekoliko bitnih odluka:

- Određivanje prikladnog prostora imena (*name space*) za svaki tip resursa. Prostor imena može biti beskonačan ili konačan, strukturirani ili jednostavan. Resursi istog tipa moraju imati različita imena, bez obzira na njihovu lokaciju. Kod objektnog sistema, svi objekti dele isti prostor imena.

- Iz imena se mora moći odrediti komunikacijski identifikator.

Imena se često određuju u zavisnosti od konteksta. Zbog toga je potrebno navesti kako ime, tako i kontekst za to ime. Treba izbegavati uključivanje mrežne adrese ili druge lokacijske oznake u ime resursa. Određivanje odgovarajućeg sistema imena je veoma važno za distribuirane sisteme.

4.2 KOMUNIKACIJE

Distribuirani sistemi su sastavljeni od delova koji su fizički i logički odvojeni i koji moraju komunicirati da bi mogli međusobno delovati.

Komunikacija između dva procesa uključuje slanje i primanje što ima za posledicu:

a) prenos podataka iz procesa koji šalje procesu koji prima *i*
b) kod nekih komunikacija *sinkronizaciju* slanja sa primanjem, tako da je proces koji šalje ili prima privremeno zaustavljen dok ne izvrši do kraja svoju akciju.

Kod slučaja (a) oba procesa dele isti komunikacijski kanal, dok je ponašanje opisano u slučaju (b) karakteristično za svako komuniciranje.

Osnovni način realizacije su šalji (*send*) i primi (*receive*) delovi koji zajedno čine akcije za **prenos poruke** između dva procesa. Akcijom prenosa poruke se određeni podaci (poruka) koju stvara proces koji šalje, prenose korištenjem određenog komunikacijskog mehanizma (kanala ili porta), do procesa koji podatke prima. Taj mehanizam može biti **sinhroni** (engleski izraz je *blocking*) što znači da pošiljalac čeka dok primalac ne primi poruku ili **asinhroni** (*non-blocking*) kod kojeg se poruka stavlja u niz poruka koje čekaju na primanje, dok proces koji šalje može nastaviti svoje izvršavanje.

Dva osnovna načina komuniciranja su **klijent-poslužitelj komuniciranje** između dva procesa i **grupno slanje** (*group multicast*) za komuniciranje između grupe procesa koji međusobno sarađuju.

4.2.1. KLIJENT-POSLUŽITELJ KOMUNICIRANJE

Klijent-poslužitelj komuniciranje je orijentisano prema pružanju usluge. Razmena podataka se sastoji od:

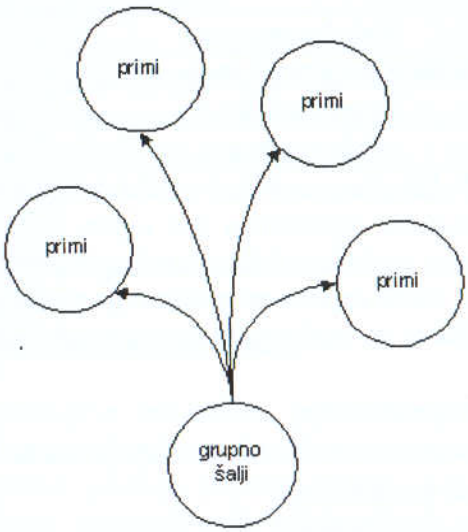
1. proces klijent šalje zahtev procesu poslužitelju
2. obrada zahteva na poslužitelju
3. prenos odgovora klijentu

Slika 9: Klijent poslužitelj komuniciranje

Takav način komuniciranja uključuje prenos dve poruke i sinhronizaciju klijenta i poslužitelja. Iako se ovaj način komunikacije može implementirati korištenjem osnovnih operacija šalji i primi, najčešće se upotrebljava iz viših programskih jezika preko **RPC**-a (*remote procedure calling*) sučelja koje skriva komunikaciju od korisnika.

Grupno slanje

Kod grupnog komuniciranja procesi razmenjuju poruke na taj način da svaka poruka ide svim procesima u grupi, a ne samo jednom. Jednom šalji pozivu odgovara više primi poziva, po jedan na svakom članu grupe. Takvo slanje naziva se **multicast**.



Slika 10:Grupno slanje

Grupno slanje ima sledeće dobre strane:

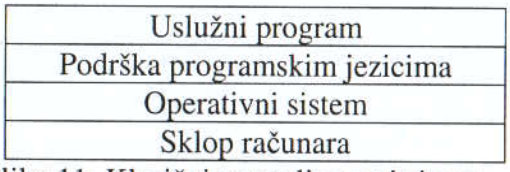
- nezavisno je od lokacije objekta kojem je upućena poruka - poruka se šalje svim objektima, dok odgovara samo onaj kojem je namenjena
- otpornost na pogreške - poruka može biti poslana istovremeno na više poslužitelja, tako da na nju odgovara jedan ili više od njih. Kvar jednoga od poslužitelja klijent ne primećuje.
- istovremeno usklađivanje svih članova grupe - jedan poslužitelj može poslati vreme grupnim slanjem tako da se svi ostali poslužitelji i klijenti sinhroniziraju na to vreme.

4.2.2. STRUKTURA PROGRAMSKE PODRŠKE

Kod centralizovanih sistema glavna komponenta je operativni sistem. On upravlja svim resursima i pruža usluge uslužnim programima i korisnicima koje uključuju:

- osnovno upravljanje resursima
  - dodeljivanje i zaštita memorije
  - stvaranje i određivanje redosleda izvršavanja procesa
  - upravljanje spoljnim uređajima
- usluge programima i korisnicima
  - provera korisnika i kontrolu pristupa
  - upravljanje datotekama i pravima pristupa
  - usluge sata

Sve te servise obavlja *kernel* operativnog sistema kod centralizovanog sistema.



Slika 11: Klasični centralizovani sistem

Distribuirani sistemi pružaju te i druge usluge uslužnim programima na taj način da dodavanje novih servisa ne predstavlja problem. Da bi to bilo moguće, *kernel* se ograničava na pružanje samo osnovnih usluga upravljanja resursima:

- dodeljivanje i zaštitu memorije
- stvaranje i određivanje redosleda izvršavanja procesa
- komunikaciju među procesima, i
- upravljanje spoljnim uređajima.

4.2.3. RASPOREĐIVANJE OPTEREĆENJA

Kod klasičnih centralizovanih sistema, svi resursi centralne jedinice i memorije su na raspolaganju operativnom sistemu u skladu sa trenutnim opterećenjem. Kod najjednostavnijih distribuiranih sistema resursi centralne jedinice i memorije su ograničeni najvećim raspoloživim resursom na jednoj od radnih stanica. Takav jednostavan model naziva se **radna stanica-poslužitelj** model. Smeštanje tih resursa "blizu korisniku" (na njegovu radnu stanicu) ima mnoge prednosti naročito kod interaktivnih primena i to je glavna prednost modela radna stanica-poslužitelj. Međutim, iz toga slede i ograničenja: model ne koristi optimalno resurse niti omogućava korisniku sa velikim zahtevima pristup do dodatnih resursa.

Zbog toga su se razvile dve modifikacije tog modela. **Procesorski pool** je prva od njih, i uključuje dinamičko dodeljivanje procesorskih resursa korisnicima. Druga varijanta je **upotreba neiskorištenih radnih stanica** koja koristi stanice koje se trenutno ne upotrebljavaju. Jedna od opcija su i **višeprocesorski sistemi sa deljenom memorijom** koji se mogu uspešno primeniti u oba slučaja, naročito kod velikih opterećenja koja se onda mogu podeliti na više procesora.

Procesorski pool

Kod procesorskog pool-a procesori se dodeljuju procesima za vreme celog njihovog izvršavanja, što rezultira deljenjem "procesora po procesu".

Korisnik sa više procesa može iskoristiti više procesorske snage nego što jedna radna stanica može ponuditi.

Procesorski pool sastoji se od skupine jeftinih računara, od kojih se svaki sastoji samo od procesora, memorije i mrežnog sklopa. Svaki procesor u pool-u ima svoj priključak na mrežu, kao i radne stanice te serveri. Procesori su smešteni na ploče, koje se montiraju u za to predviđene ormare (*rack*-ove). Procesori mogu biti različiti po arhitekturi da bi se omogućilo izvršavanje različite programske podrške.

Upotreba neiskorištenih radnih stanica

Druga zanimljiva mogućnost je korištenje radnih stanica koje su neiskorištene ili malo iskorištene kao dodatnih izvora procesorskih resursa, slično procesorskom pool-u. Primeri takvih sistema su korištenje crva (*worms*) koji putuju mrežom, traže neiskorištene stanice i tamo obavljaju svoje poslove, i premeštanje procesa (*process migration*) koje omogućava da se procesi vrate svom "izvorištu" u slučaju da se stanica počne više iskorištavati.

4.2.4. UJEDNAČENOST SISTEMA

Ujednačenost (*consistency*) je važan problem kod distribuiranih sistema zbog deljenja resursa i istovremenosti.

Ujednačenost kod obnavljanja sadržaja

Ovaj se problem javlja kada nekoliko procesa pristupa i menja podatke istovremeno. Menjanje podataka ne može biti trenutno, ali bi trebalo biti atomarna operacija, tako da svim ostalim procesima izgleda kao da je trenutno. Problem se nije pojavio sa distribuiranim sistemima, već postoji svugde gdje se dele podaci.

Ujednačenost kod udvajanja

Kada se podaci sa jednog izvora udvoje na više različitih lokacija javlja se problem ukoliko se oni i promene na jednoj od lokacija. Sve ostale lokacije moraju biti obaveštene o takvoj promeni, da bi sve kopije podataka bile identične.

Ujednačenost pričuvne memorije (cache-a)

Problem ujednačenosti pričuvne memorije je specijalni slučaj problema kod udvajanja podataka i rešava se na slične načine.

Ujednačenost kod neotklonjivih grešaka

Kada kod klasičnog sistema dođe do greške koja prekida izvršavanje obrade, svi procesi se zaustavljaju. Međutim, kod distribuiranih sistema verojatno je da će kod ispada jedne od komponenti sistema ostali procesi biti u različitim tačkama izvršavanja. Zbog toga je potrebno vratiti se na zadnje poznato i ispravno stanje (*roll back*) da bi se obrada mogla nastaviti.

Ujednačenost sata

Mnogi od algoritama distribuiranih sistema ovise o vremenskim oznakama (*time stamps*), tako da je ujednačenost satova kod distribuiranih sistema veoma bitna.

Ujednačenost korisničkog sučelja

Korisničko sučelje bi trebalo reagovati na komande korisnika dovoljno brzo da omogući praćenje promena koje se dešavaju radom korisnika. Vreme kašnjenja se sastoji od:

- vremena potrebnog da se korisnički unos primi, obradi, da se izračunaju potrebne promene na ekranu i
- vremena potrebnog da se promene prenesu do korisničkog sistema prozora i da se obnovi slika na ekranu.

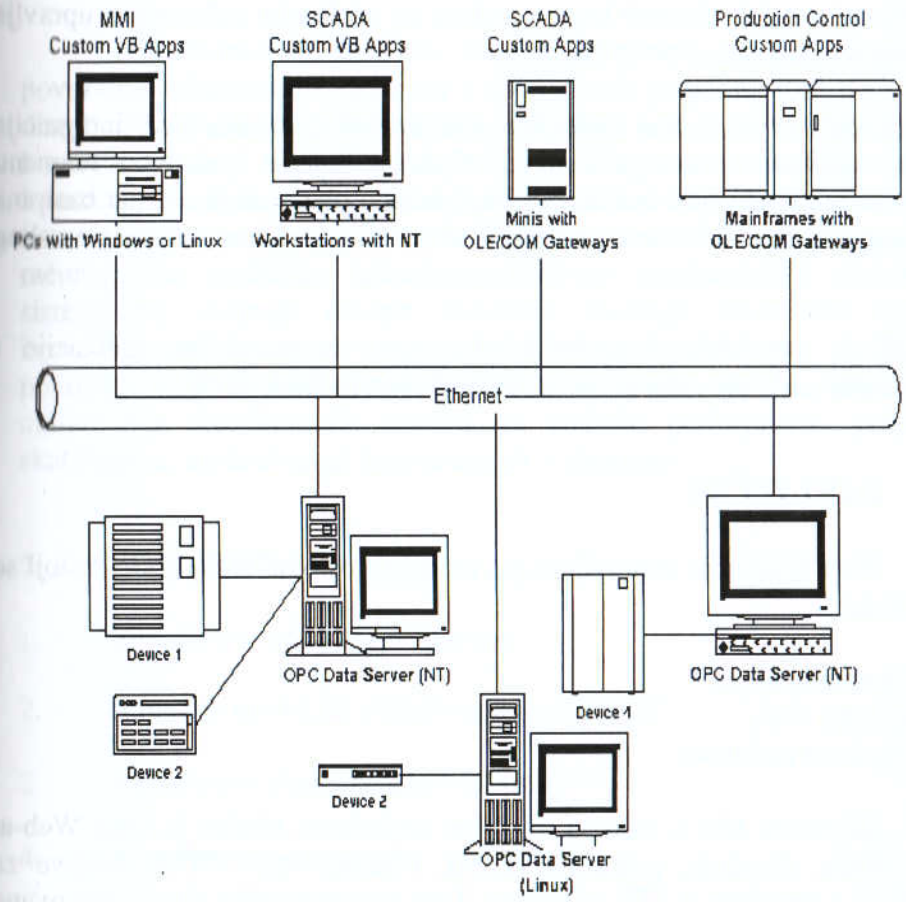
Po ergonomskim studijama, to vreme bi trebalo biti manje od 0.1 sekunde.

5. OPC STANDARD

OPC se definiše kao otvoren industrijski standard baziran na tehnologijama OLE, DCOM (COM) i ActiveX, koji obezbeđuje operabilnost između različitih uređaja, sistema za automatizaciju, kontrolu, vizuelizaciju i poslovnih sistema. Prva verzija OPC standarda V1.0 je objavljena avgusta 1996. godine. Tokom 1997. godine vršene su korekcije na standardu i pojavila se verzija OPC V1.0A. Krajem 1998. godine se pojavila verzija V2.0 sa značajnim izmenama. Standard je podržan od strane najvećih svetskih kompanija koje se bave izradom PLC-a i softvera za vizuelizaciju procesa.

OPC specifikacija definiše skup interfejsa koji se lako implementiraju primenom objektno orijentisanog programiranja i omogućava laku manipulaciju tim objektima. Prenos podataka se vrši pomoću DCOM tehnologije. Softver pomoću koga korisnik upravlja procesom (MMI, Man-Machine Interface), upravljački softver ili softver za akviziciju podataka (SCADA) može obrađivati ili prikupljati podatke sa različitih računara u mreži. Specifikacija definiše standardne mehanizme za pristupanje podacima na serveru po nazivu. Projektanti koji razvijaju hardver i softver mogu jednostavno da razmenjuju informacije pomoću širokog spektra sistemskih

aplikacija, u koji se ubrajaju distribuirana kontrola sistema (DCS), SCADA sistemi, PLC (Programmable Logical Controler) kao i razni inteligentni uređaji, povezani preko računarske mreže. OPC je baziran na tehnologijama OLE, ActiveX, COM (Component Object Model) i DCOM (Distributed Component Object Model) i dostupan je na 32-bitnom operativnom sistemu Microsoft Windows (9X, ME, NT, 2000). Pomoću DCOM tehnologije mogu se razmenjivati podaci (objekti) i sa drugim operativnim sistemima kao na primer sa Unix-om ili Linux-om. OPC definiše standardni skup interfejsa, osobina i metoda za procesnu kontrolu i automatske softverske aplikacije



Slika 12: Povezivanje različitih računarskih sistema u jedinstven sistem

Na slici 12 je prikazano više računarskih sistema povezanih u mrežu koji razmenjuju podatke primenom OPC standarda. Na slici se vide dva OPC servera koji su povezani na merno upravljačke uređaje (jedan od servera radi pod Windows NT operativnim sistemom, drugi pod Linux-om) treći OPC

server predstavlja server za baze podataka. Ovi serveri su povezani na računare na kojima se nalaze nadzorni, upravljački ili poslovno informacioni sistemi.

COM (Component Object Model) obezbeđuje interfejs i komunikaciju između komponenti sistema. Preko COM-a, aplikacija može koristiti osobine objekta bilo koje druge aplikacije. COM predstavlja jezgro tehnologija kao što su DCOM, ActiveX i OLE.

DCOM (Distributed Component Object Model) predstavlja proširenje COM-a koje omogućava rad sa objektima koji se nalaze na drugim računarima, preko računarske mreže. To je protokol koji omogućava da se softverskim komponentama (objektima) koje se nalaze na udaljenim računarima upravlja na sličan način kao sa lokalnim.

OLE (Object Linking and Embedding) se koristi da se obezbedi integracija između aplikacija i omogući razvoj objekata koji se koriste za razmenu informacija između više aplikacija. OLE takođe obezbeđuje rešenja bazirana na komponentama. Softverske komponente su nezavisne od programskog jezika.

ActiveX je otvorena, integrisana platforma za portabilne aplikacije i interaktivne sadržaje namenjene za WWW (World Wide Web).

6. ZAKLJUČAK

Web aplikacija napravljena po modelu troslojne arhitekture sastoji se iz tri dela:

- klijentskog sloja,
- srednjeg sloja,
- sloja baze podataka.

Klijentski sloj u modelu troslojne arhitekture obično je čitač Web-a. Čitač Weba obrađuje, prikazuje HTML resurse, šalje HTTP zahteve za resursima i obrađuje HTTP odgovore. Dva najpopularnija čitača zasnovana na okruženju sa prozorima jesu Netscape i Internet Explorer. U većini troslojnih sistema baza podataka na Web-u, najveći dio logike aplikacije nalazi se u srednjem sloju. Klijentski sloj prikazuje podatke korisniku i prihvata podatke koje on unosi; sloj baze podataka skladišti i učitava podatke.

Srednji sloj obuhvata većinu preostalih zadataka pomoću kojih se objedinjuju ostali slojevi: upravlja strukturom i sadržajem podataka koji se

prikazuju korisniku i obrađuje podatke dobijene od korisnika pretvarajući ih u upite za učitavanje ili upisivanje u bazu podataka.

Sloj baze podataka je osnova Web aplikacija koje rade s bazama podataka. Razumevanje sistemskih zahteva, odabir softvera za sloj baze podataka, projektovanje baza podataka i izrada sloja prvi su koraci uspešnog projektovanja aplikacija. U aplikaciji sa troslojnom arhitekturom, sloj baze podataka zadužen je za upravljanje podacima. Upravljanje podacima obično podrazumeva skladištenje i učitavanje podataka, kao i upravljanje postupkom ažuriranja, pri čemu više procesa iz srednjeg sloja može simultano, tj. istovremeno da pristupa, zatim, zaštitu podataka, očuvanje njihovog integriteta i obezbeđivanje usluga za podršku, poput izrade rezervnih kopija.

Distribuirani sistemi se sastoje od skupa samostalnih računara povezanih računarskim mrežama i opremljenih programskom podrškom za distribuirane sisteme. Programska podrška omogućava računarima da koordiniraju svoje aktivnosti i da dele sistemske resurse - hardware, software i podatke. Korisnici distribuiranog sistema bi trebali primećivati samo jedan integrisani sistem iako on može biti implementiran pomoću mnogo različitih računara na različitim lokacijama.Osnovne karakteristike distribuiranih sistema su: deljenje resursa (resource sharing), otvorenost (openess), istovremenost (*concurrency*), skalabilnost (*scalability*), otpornost na pogreške (*fault tolerance*) te transparentnost (*transparency*). Osnovi ciljevi oblikovanja distribuiranih sistema su sledeći: performanse, pouzdanost, skalabilnost, ujednačenost (consistency) i sigurnost.

7. LITERATURA

1. http://www.rot13.org/~dpavlin/
2. http://ftb.uni-bk.ac.yu/informacionisistemi/
3. http://www.student.foi.hr/~mkuster/
4. http://www.osa.co.yu/
5. http://svezaweb.dzaba.com/Razvoj/disap/index.htm

ČORBA[1]

[1] *Seminarski rad Tanje Sekulić, realizovan pod mentorstvom Prof.dr Dragice Radosav, u okviru predmeta Informatika na poslediplomskim studijama Tehničkog fakulteta u Zrenjaninu*

SADRŽAJ:

1. UVOD

Posmatrajući razvoj kompjuterskih nauka, stiče se utisak da stari principi polako ustupaju mesto jednoj novoj metodologiji, a to je objektna orjentacija. Ona sa sobom donosi dosta toga novog, sasvim drugačijeg od postojećeg, ali se polako dokazuje i prerasta u jednu moćnu alatku za dalji razvoj svih aspekata kompjuterskih nauka.

U ovom poglavlju biće reči o CORBA-i, jednoj od najpopularnijih i najboljih specifikacija za rad sa distribuiranim objektnim sistemima. Predstaviće se izumitelji CORBA specifikacije i ideje kojima su se oni vodili u kreiranju ove specifikacije, daće se pregled najvažnijih komponenata arhitekture CORBA-e, razvoj CORBA specifikacije kroz njene verzije. U petom poglavlju pažnja je posvećena distribuiranim i usađenim sistemima koji rade u realnom vremenu, dok šesto poglavlje daje pregled smernica daljeg razvoja CORBA specifikacija. U Prilogu je data lista svih skraćenica koje su korišćene u radu, njihov puni naziv na engleskom jeziku i prevod na srpski jezik.

2. GRUPA ZA OBJEKTNO UPRAVLJANJE I ARHITEKTURA OBJEKTNOG UPRAVLJANJA

2.1 GRUPA ZA OBJEKTNO UPRAVLJANJE

Posmatrajući karakteristike mreža poput Interneta, World Wide Web-a, raznih korporativnih intranetova, može se uočiti da sve one imaju jednu zajedničku osobinu, a to je heterogenost. Na primer, jedan korporativni intranet, može da bude sačinjen od UNIX radnih stanica i servera, PC sistema koji rade pod raznim verzijama Microsoft-ovog Windows-a, IBM-ovog OS/2, Apple Mackintosh-a,... Mrežni protokoli pod kojima bi ovakav intranet radio, opet mogu da se razlikuju. Samo neki od njih su: TCP/IP protokol, razne vrste udaljenih poziva procedura (RPC),... Međutim, ono što je fundamentalno jeste da se mora omogućiti razmena informacija i sadržaja i kod mreža koje nisu istog tipa i ne koriste iste operativne sisteme i softvere.

Grupa za objektno upravljanje (**Object Management Group-OMG**), je neprofitabilni konzorcijum formiran aprila 1989. godine u cilju razvoja, usvajanja i promovisanja standarda za razvoj aplikacija koje bi radile u distribuiranim heterogenim i objektnim okolinama. Iako je OMG u početku brojala trinaest kompanija osnivača, do danas je prerasla u grupu od preko 800 članova i korisnika. OMG je osnovana u SAD-u, ali sada već ima predstavništva i u Velikoj Britaniji, Nemačkoj, Australiji, Japanu i Indiji.

Glavna misija OMG-a je da razvije jedistvenu ahitekturu za interaciju distribuiranih aplikacija u objektno orjentisanom okruženju, koja će garantovati sledeće:

- interoperabilnost i prenosivost,
- ponovno korišćenje komponenata,
- mogućnost formiranja baze za komercijalne softvere.

Gledajući pojedinačne komponente, OMG je ustanovila i standard za komponente koji obuhvata:

- jedinstvenu terminologiju za objektnu orjentaciju,
- zajednički apstraktni okvir,
- zajedničke interfejse i protokole.

Snaga i značaj OMG-a, su i u tome što ova grupa usvaja nove standarde na osnovu zahteva za predlozima **(Request For Proposals – RFP)**, koji ustvari predstavljaju sveže ideje i tehnologije, a predlažu ih sami članovi OMG-a. Time se postiže neprekidan razvoj standarda za distribuirane objektne sisteme.

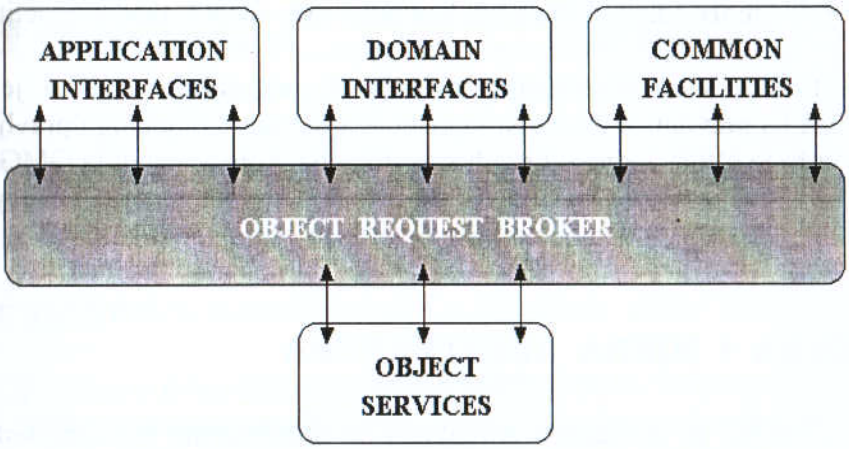
2.2 ARHITEKTURA OBJEKTNOG UPRAVLJANJA

Arhitektura objektnog upravljanja **(Object Management Architecture – OMA)**, predstavlja pokušaj, da se na visokom nivou apstrakcije, definišu različiti načini za opisivanje i formiranje distribuiranih objektnih sistema. OMA je sastavljena iz objektnog i referentnog modela. Objektni model definiše kako se mogu opisati objekti koji su distribuirani u heterogenoj okolini, dok referentni model daje karakteristike mogućih interakcija između tih objekata. OMG-ov RFP proces se koristi za usvajanje specifikacija tehnologija koje se uklapaju u objektni i referentni model i koje sarađuju sa već usvojenim tehnologijama. Spajanjem ovakvih tehnologija sa OMA, omogućen je konstantan razvoj distribuiranih objektnih sistema u heterogenim okruženjima.

U OMA objektnom modelu, objekat je enkapsulirani entitet sa jedinstvenim identitetom koji se ne može menjati, i čijim operacijama sa može pristupiti samo preko dobro definisanih interfejsa. Klijent izdaje zahtev objektu, za izvršenje određene operacije. Implementacija i lokacija svakog objekta ostaju skrivene za klijenta koji upućuje zahtev.

Na slici 1 mogu se videti komponente referentnog modela OMA. Centralni deo ovog modela je **ORB (Object Request Broker)**, koji je odgovoran za sprovođenje komunikacije između klijenata i objekata. Pored ORB-a, ovaj model sadrži još četiri komponente: interfejse aplikacije **(Application Interfaces)**, domenske interfejse **(Domain Interfaces)**,

zajedničke strukture **(Common Facilities)** i objektne servise **(Object Services)**.



Slika 1: Arhitektura OMG –ovog referentnog modela

Objektni servisi (Object Services) predstavljaju interfejse koji ne zavise od domena i koje koriste mnogi distribuirani objektni programi. Primer za to bi bio servis koji je zadužen za otkrivanje drugih raspoloživih servisa, on je uvek neophodan, bez obzira na domen aplikacije. Dva najpoznatija objektna servisa koji ispunjavaju ovu ulogu su:

- servis koji omogućava klijentima da pronađu traženi objekat na osnovu njegovog imena (The Naming Service)
- servis koji omogućava klijentima da pronađu traženi objekat na osnovu njegovih osobina (The Trading Service)

Takođe postoje i specifikacije objektnih servisa koji su zaduženi za životni ciklus upravljanja, sigurnost, transakcije,...

Zajedničke strukture (Common Facilities) su takođe interfejsi koji su horizontalno orjentisani poput objektnih servisa, ali za razliku od njih, oni su orjentisani ka aplikacijama krajnjih korisnika. Primer za zajedničke strukture bi bila struktura komponente distribuiranog dokumenta (Distributed Document Component Facility- DDCF), koja omogućava prezentaciju i razmenu objekata koji su bazirani na modelu dokumenta.

Domenski interfejsi (Domain Interfaces) igraju ulogu koja je slična onoj kod objektnih servisa i zajedničkih struktura, ali su ovi interfejsi orjentisani ka domenima specifičnih aplikacija. Njih može biti i više, jer tako prikaziju mogućnost egzistencije više različitih domena neke aplikacije.

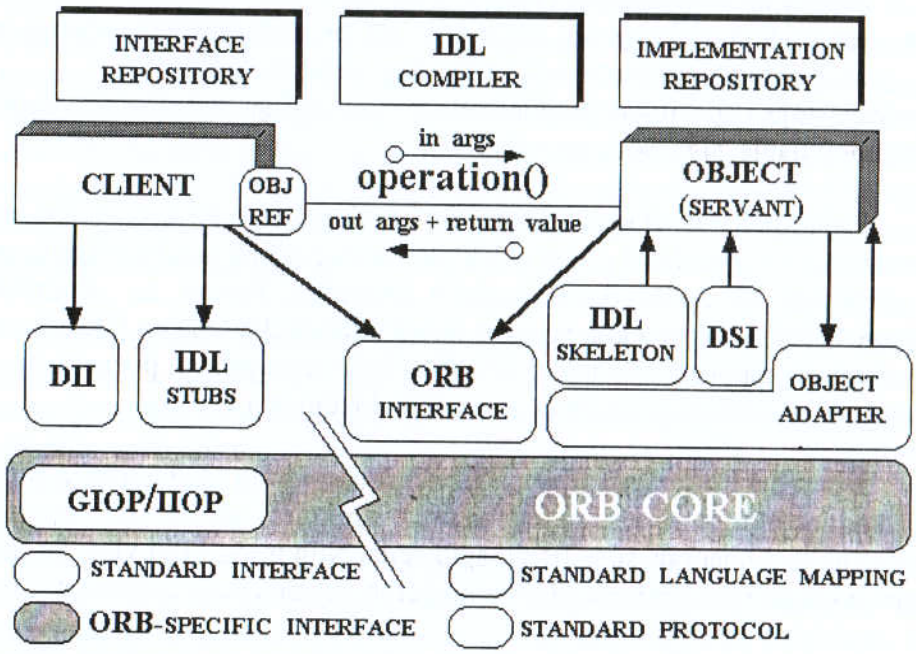
Interfejsi aplikacije (Application Interfaces) su interfejsi koji su razvijeni za specifične aplikacije. Ovi interfejsi nisu standardizovani, jer su

specifični za svaku aplikaciju, ali i zbog toga što OMG ne razvija aplikacije već samo standarde. Međutim, ako se nekada desi da se neki od ovih servisa toliko često koristi i da počinje da ne zavisi od domena aplikacije, tada se takav servis može smatrati budućim kandidatom za OMG standardizaciju.

Tokom svog dosadašnjeg postojanja, sva pažnja OMG-a je bila usmerena ka razvoju centralne komponente arhitekture objektog opravljanja-OMA, a to je ORB. Jedna od prvih specifikacija, koju je usvojila OMG, bila je CORBA (Common Object Request Broker Architecture). CORBA, što ćemo videti kasnije, u detalje definiše interfejse i karakteristike ORB-a.

3. CORBA I NJENA ARHITEKTURA

CORBA je standardna arhitektura za distribuirane objektne sisteme, koja je standardizovana od strane OMG-a. Ona omogućava da distribuirani i heterogeni objekti zajedno funkcionišu i komuniciraju. Pri tome, vrsta računara, programskog jezika, proizvođača, mreže, uopšte nisu relevantni, ako su programi bazirani na CORBA specifikaciji. Svi oni među sobom mogu da sarađuju i razmenjuju informacije. CORBA se može pronaći u pozadini najvećih svetskih web-sajtova, a posebno je korisna u radu na serverima koji opslužuju veliki broj klijenata i od kojih se zahteva velika pouzdanost. Slika 2 daje pregled arhitekture CORBA-e, sa svim njenim komponentama.



LEGENDA:

Slika 2: CORBA ORB arhitektura

U odeljcima koji slede biće detaljno opisane sve komponente CORBA-e sa slike.

3.1 ORB JEZGRO

ORB jezgro je na slici označeno sa **ORB CORE**. Šta je prava svrha ORB-a? ORB je centralni deo CORBA-e, zadužen za komunikaciju klijenata i objekata. Kada klijent uputi neki zahtev, ORB ga isporučuje objektu koji taj zahtev može da ispuni i vraća odgovor klijentu, ako ga ima. Objekat kome klijent želi da uputi zahtev preko ORB-a se zove *ciljni objekat*. Glavna odlika ORB-a je da on uspeva da komunikaciju između klijenta i objekta učini nevidljivom. Generalno, ORB skriva sledeće:

- **lokaciju objekta** – klijent ne zna gde se nalazi ciljni objekat. Objekat se može nalaziti u drugom procesu na drugom računaru , može biti deo istog procesa, ili biti na istom računaru ali u drugom procesu,...
- **implementaciju objekta** – klijent ne poznaje implementaciju objekta, ne zna u kom je programskom jeziku pisan, pod kojim operativnim sistemom radi ili koji hardver koristi.
- **izvršno stanje objekta** – kada klijent uputi zahtev objektu, on ne zna da li je objekat aktivan ili se nalazi u stanju pripremnom za izvršenje zahteva klijenata. Ako je potrebno ORB skriveno od korisnika aktivira objekat i pre nego što mu uputi zahtev od klijenta.
- **mehanizmi komunikacije objekta** – klijent ne može da vidi koje mehanizme komunikacije koristi ORB kada poziva ciljni objekat (npr. TCP/IP, poziv lokalne metode,...) i vraća odgovor korisniku.

Da bi klijent uputio zahtev objektu, on mora koristiti referencu tog objekta. Svaki put kada se kreira CORBA objekat, kreira se i njegova referenca. Ona postoji dok postoji i objekat na koji ona ukazuje i jedinstvena je u smislu da uvek ukazuje na objekat za koji je i kreirana. Klijenti ne mogu menjati referencu objekta. Jedino ORB zna šta se nalazi unutar reference objekta. Objektne reference imaju standardizovane formate od strane OMG-a. ORB koristi reference objekta da identifikuje i locira objekte, tako da može direktno da im pristupi. Objektne reference se mogu sačuvati ako se uputi zahtev ORB-u da ih konvertuje u stringove, i kao takve klijent ih može čuvati kao podatke. Kasnije, klijent može zahtevati od ORB-a da stringove prebaci

nazad u objektne reference i onda ih koristiti za upućivanje novih zahteva.Ovaj princip se može koristiti za održavanje konstantnih veza između objekata i aplikacija koje ih koriste.

3.2 JEZIK ZA DEFINICIJU INTERFEJSA

Da bi klijent uopšte mogao uputi zahtev nekom objektu, on mora znati koje tipove operacija taj objekat podržava. Interfejs objekta specifikuje operacije i tipove koje objekat podržava i time definiše zahteve koji se mogu uputiti objektu. Interfejsi objekata su definisani OMG-ovim jezikom za definiciju interfejsa (**OMG Interface Definition Language – OMG IDL**).

Važna odlika OMG IDL-a je nezavisnost od jezika. Pošto je OMG IDL deklarativni, a ne programski jezik, on podržava definisanje interfejsa odvojeno od implementacije objekta. To omogućava komunikaciju objekata koji su konstruisani uz pomoć različitih programskih jezika. Nezavisnost interfejsa od jezika je posebno važna u heterogenim okruženjima, pošto svi programski jezici nisu svugde dostupni ili ih ne podržavaju sve platforme.

Još jedna od bitnih osobina OMG IDL-a jeste nasleđivanje interfejsa. Naime, novi interfejsi mogu nasleđivati osobine, od jednog ili više, već postojećih interfejsa. Time se omogućava višestruko korišćenje postojećih interfejsa kada se definišu novi servisi. Nasleđivanje interfejsa je veoma važno za CORBA-u. Ono omogućava da sistem bude otvoren za ekstenziju, ali da istovremeno bude i zatvoren za modifikaciju. Ova pojava se naziva "*princip otvoreno – zatvoreno*". Pošto izvedeni interfejs nasleđuje sve operacije od baznih interfejsa, tada objekti koji podržavaju izvedeni interfejs, moraju takođe podržavati i sve nasleđene operacije. Ovo omogućava da se objektne reference za izvedene interfejse zamene objektnim referencama baznih interfejsa, kad god je to moguće.

OMG IDL sistem tipova je dovoljan za većinu distribuiranih aplikacija, a u isto vreme je minimalan i takav će i ostati. Održavanjem OMG IDL-a što jednostavnijim, postiže se to da se on može koristiti kod više programskih jezika, nego što bi mogao ako bi sadržao specijalne tipove koji se ne mogu realizovati u nekim popularnim programskim jezicima.

Uzimajući u obzir neizbežnu heterogenost distribuiranih objektnih sistema, jednostavnost OMG IDL-a je presudna za uspeh CORBA-e kao integracione tehnologije.

3.2.1 PRESLIKAVANJE JEZIKA

OMG IDL je samo deklarativni jezik i kao takav ne nudi mogućnost za kontrolne konstrukte, niti se koristi za implementaciju distribuiranih aplikacija. Umesto toga, preslikavanje jezika utvrđuje kako se OMG IDL obeležja preslikavaju u komponente koje su pisane u nekim programskim jezicima. Do sada, OMG je standardizovao preslikavanja za jezike: C, C++, Smalltalk, Ada, Java, COBOL, Lisp i još neke druge.

Jedan od važnih aspekata OMG IDL preslikavanja jezika je kako on preslikava ORB interfejse i druge pseudo objekte koji se javljaju u CORBA specifikaciji. Pseudo objekti nisu pravi CORBA objekti, već su to ORB interfejsi koji omogućavaju aplikacijama da manipulišu ORB-om kao što manipulišu sa običnim objektima.

OMG IDL preslikavanje jezika predstavlja mesto gde se apstrakcije i koncepti specifikovani u CORBA –i sreću sa "realnim svetom" implementacije. Ono se ne može zanemariti, jer loše ili nepotpuno preslikavanje jezika, za određeni programski jezik može rezultovati nemogućnošću programera da efektivno iskoristi CORBA tehnologiju u datom programskom jeziku.

3.3 NOSILAC INTERFEJSA

Svaka aplikacija bazirana na CORBA-i tokom svog izvršavanja traži pristup OMG IDL sistemu tipova. Ovo je neophodno jer aplikacija mora znati tipove vrednosti koje se prenose kao argumenti nekog zahteva. Takođe, aplikacija mora poznavati i tipove interfejsa koje podržavaju objekti koji se koriste.

Mnoge aplikacije zahtevaju samo statičko znanje o OMG IDL sistemu tipova. To je usled toga što se OMG IDL specifikacija kompajlira ili prevodi u kôd programskog jezika koji aplikacija koristi po pravilima prevođenja koja važe za taj programski jezik. Tako generisan kôd se zatim ugrađuje direktno u aplikaciju. Ovim pristupom se postiže to da se znanje aplikacije o OMG IDL sistemu tipova fiksira pri ugradnji u samu aplikaciju. Međutim, ako se ikada promeni sistem tipova za ostatak distribuiranog objektnog sistema, tako da bude nekompatibilan sa aplikacijom u koju je ugrađen, onda se moraju vršiti promene u samoj aplikaciji.

Postoje i aplikacije kod kojih statičko znanje o OMG IDL sistemu tipova nije praktično. To su aplikacije koje dozvoljávaju aplikacijama iz nekih drugih objektnih sistema (npr. Microsoft-ovog COM-a) da pristupaju CORBA objektima. Bilo bi prilično nezgodno sa stanovišta održavanja, ali i sa stanovišta upravljanja, menjati aplikaciju, svaki put kada se pojavi neki novi OMG IDL tip interfejsa.

Nosilac interfejsa **(Interface Repository – IR)** kod CORBA-e omogućava da se OMG IDL sistemu tipova pristupa i programski obrađuje u toku samog rada. IR se može posmatrati i kao CORBA objekat, i njegove operacije se mogu pozivati kao operacije kod bilo kog drugog objekta. Koristeći IR interfejs, aplikacije mogu preći preko čitave hijerarhije OMG IDL informacija.

Pošto IR dopušta aplikacijama da programski ispituju tipove informacija u toku samog rada, njegova prava korist se sastoji u podršci dinamičkog poziva kod CORBA-e. O tome će biti reči u odeljcima koji slede.

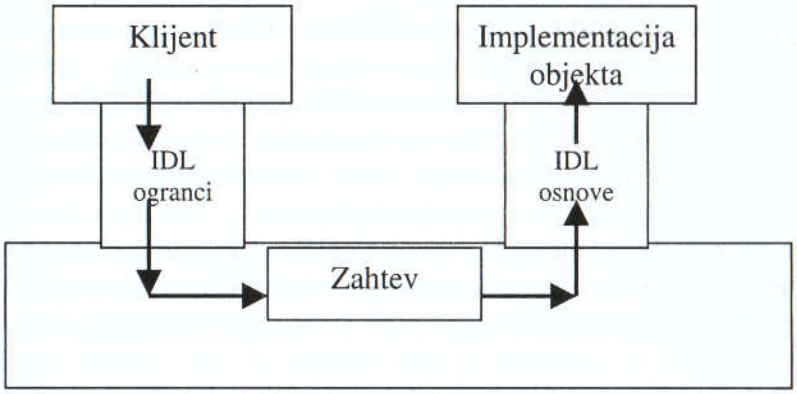
3.4 OGRANCI I OSNOVE

Ogranci **(Stubs)** i osnove **(Skeletons)** su mehanizmi koji se nalaze sa strane klijenta i servera, respektivno, a njih generišu OMG IDL jezički kompajleri i prevodioci. Ogranak je mehanizam koji efektivno kreira i izdaje zahteve, a sve u službi klijenta, dok osnove služe za isporuku tih zahteva implementaciji objekata CORBA-e. Pošto se ogranci i osnove direktno prevode iz OMG IDL specifikacija, oni su u opštem slučaju određeni interfejsom.

Raspoređivanje putem ogranaka i osnova se često naziva *statički poziv*. OMG IDL ogranci i osnove su ugrađeni direktno u klijentovu aplikaciju i implementaciju objekta. Oni zbog toga u sebi unapred sadrže potrebna znanja o OMG IDL interfejsima objekata CORBA-e koji su pozvani.

Kada zahtev stigne do ciljnog objekta, ORB server i osnova kooperiraju u razvrstavanju zahteva (prebacuju ga iz prenosne forme u formu programskog jezika) i prosleđuju ga objektu. Jednom, kada objekat ispuni traženi zahtev, odgovor se šalje istim putem kojim je stigao i zahtev: preko osnove do ORB servera, zatim preko veza i klijentovog ORB-a i ogranaka, dok konačno ne stigne do klijentove aplikacije.

Ovo se može videti na slici 3.



Slika 3: Put zahteva od klijenta do implementacije objekta

Iz gore navedenog, može se zaključiti da ogranci i osnove igraju važnu ulogu u povezivanju sveta programskih jezika sa ORB-om. U tom smislu, oni čine neku formu adaptera, jer ogranci adaptiraju programski jezik pozivne funkcije u zahtev koji ORB dalje prosleđuje, a osnove prevode mehanizam prosleđivanja zahteva u poziv metode koja odgovara formi implementacije objekta.

3.5 DINAMIČKI POZIV I PROSLEĐIVANJE

Pored statičkog poziva uz pomoć ogranaka i osnova, CORBA podržava i dva interfejsa za dinamički poziv:

- Interfejs dinamičkog poziva **(Dynamic Invocation Interface – DII)** koji podržava dinamički poziv klijentovog zahteva
- Interfejs dinamičke osnove **(Dynamic Skeleton Interface – DSI)** koji omogućava dinamičko prosleđivanje do objekata

DII i DSI se mogu posmatrati kao generički ogranak i generička osnova. Svaki od njih je interfejs koji je obezbeđen direktno od strane ORB-a i nijedan od njih ne zavisi od OMG IDL interfejsa objekata koji su pozvani.

3.5.1 INTERFEJS DINAMIČKOG POZIVA

Koristeći DII, klijentova aplikacija može izdati zahtev bilo kom objektu, bez potrebe za saznanjem o vremenu kompajliranja objektovih interfejsa. Pretpostavimo da imamo neku aplikaciju. Kada ta aplikacija primi poziv od nekog objekta, ona mora taj poziv prebaciti u zahtev koji onda isporučuje određenom CORBA objektu. Bilo bi prilično nezgodno rekompajlirati aplikaciju u cilju uključenja novih statičkih ogranaka, svaki

Nosilac interfejsa **(Interface Repository – IR)** kod CORBA-e omogućava da se OMG IDL sistemu tipova pristupa i programski obrađuje u toku samog rada. IR se može posmatrati i kao CORBA objekat, i njegove operacije se mogu pozivati kao operacije kod bilo kog drugog objekta. Koristeći IR interfejs, aplikacije mogu preći preko čitave hijerarhije OMG IDL informacija.

Pošto IR dopušta aplikacijama da programski ispituju tipove informacija u toku samog rada, njegova prava korist se sastoji u podršci dinamičkog poziva kod CORBA-e. O tome će biti reči u odeljcima koji slede.

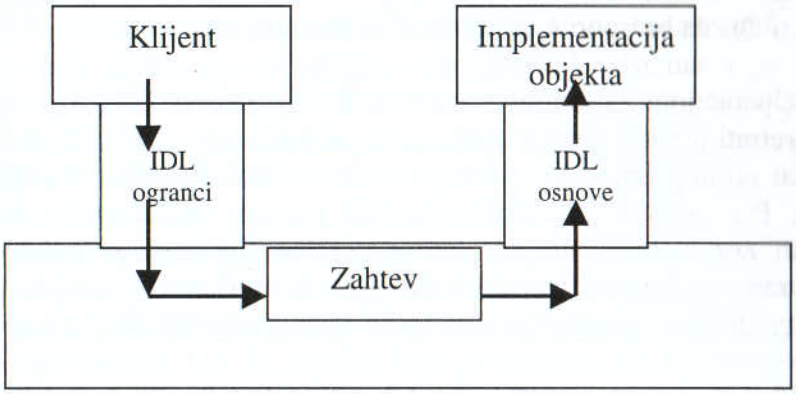
3.4 OGRANCI I OSNOVE

Ogranci **(Stubs)** i osnove **(Skeletons)** su mehanizmi koji se nalaze sa strane klijenta i servera, respektivno, a njih generišu OMG IDL jezički kompajleri i prevodioci. Ogranak je mehanizam koji efektivno kreira i izdaje zahteve, a sve u službi klijenta, dok osnove služe za isporuku tih zahteva implementaciji objekata CORBA-e. Pošto se ogranci i osnove direktno prevode iz OMG IDL specifikacija, oni su u opštem slučaju određeni interfejsom.

Raspoređivanje putem ogranaka i osnova se često naziva *statički poziv*. OMG IDL ogranci i osnove su ugrađeni direktno u klijentovu aplikaciju i implementaciju objekta. Oni zbog toga u sebi unapred sadrže potrebna znanja o OMG IDL interfejsima objekata CORBA-e koji su pozvani.

Kada zahtev stigne do ciljnog objekta, ORB server i osnova kooperiraju u razvrstavanju zahteva (prebacuju ga iz prenosne forme u formu programskog jezika) i prosleđuju ga objektu. Jednom, kada objekat ispuni traženi zahtev, odgovor se šalje istim putem kojim je stigao i zahtev: preko osnove do ORB servera, zatim preko veza i klijentovog ORB-a i ogranaka, dok konačno ne stigne do klijentove aplikacije.

Ovo se može videti na slici 3.



Slika 3: Put zahteva od klijenta do implementacije objekta

Iz gore navedenog, može se zaključiti da ogranci i osnove igraju važnu ulogu u povezivanju sveta programskih jezika sa ORB-om. U tom smislu, oni čine neku formu adaptera, jer ogranci adaptiraju programski jezik pozivne funkcije u zahtev koji ORB dalje prosleđuje, a osnove prevode mehanizam prosleđivanja zahteva u poziv metode koja odgovara formi implementacije objekta.

3.5 DINAMIČKI POZIV I PROSLEĐIVANJE

Pored statičkog poziva uz pomoć ogranaka i osnova, CORBA podržava i dva interfejsa za dinamički poziv:

- Interfejs dinamičkog poziva **(Dynamic Invocation Interface – DII)** koji podržava dinamički poziv klijentovog zahteva
- Interfejs dinamičke osnove **(Dynamic Skeleton Interface – DSI)** koji omogućava dinamičko prosleđivanje do objekata

DII i DSI se mogu posmatrati kao generički ogranak i generička osnova. Svaki od njih je interfejs koji je obezbeđen direktno od strane ORB-a i nijedan od njih ne zavisi od OMG IDL interfejsa objekata koji su pozvani.

3.5.1 INTERFEJS DINAMIČKOG POZIVA

Koristeći DII, klijentova aplikacija može izdati zahtev bilo kom objektu, bez potrebe za saznanjem o vremenu kompajliranja objektovih interfejsa. Pretpostavimo da imamo neku aplikaciju. Kada ta aplikacija primi poziv od nekog objekta, ona mora taj poziv prebaciti u zahtev koji onda isporučuje određenom CORBA objektu. Bilo bi prilično nezgodno rekompajlirati aplikaciju u cilju uključenja novih statičkih ogranaka, svaki

put kada se kreira novi CORBA objekat. Umesto toga, aplikacija može iskoristiti DII da bi isporučila zahtev CORBA objektu.

Operacijom za kreiranje zahteva, koju podržava CORBA, aplikacije mogu kreirati pseudo objekte zahteva. Upućivanjem te operacije na objektnu referencu ciljnog objekta, aplikacija može kreirati dinamički zahtev za taj objekat. Pre nego što zahtev može biti pozvan, potrebno je obezbediti vrednosti argumenata zahteva, što se postiže direktnim pozivom pseudo objekta zahteva. Jednom, kada se kreira pseudo objekat zahteva, i kada mu se dodaju vrednosti argumenta, on se može pozivati na bilo koji od sledeća tri načina:

- **Sinhroni poziv** – klijent poziva zahtev i zaustavlja rad dok čeka na odgovor. Iz perspektive klijenta, ovo je ekvivalentno sa RPC. Ovo je uobičajeni način za pozivanje koji se koristi kod CORBA aplikacija, jer je podržan statičkim ograncima.
- **Odloženi sinhroni poziv** – klijent poziva zahtev, nastavlja rad dok se zahtev prosleđuje i kasnije preuzima odgovor. Ovo je korisno ako klijent poziva više nezavisnih servisa. Pozivi se upućuju paralelno a odgovori se sakupljaju kako pristižu, što je mnogo povoljnije od serijskog pozivanja i zaustavljanja rada za svaki odgovor.
- **Jednosmerni poziv** – klijent poziva zahtev i nastavlja sa radom, odgovora nema. Ovo se nekada naziva "ispali i zaboravi" ("fire and forget"), jer je to jedini način na koji klijent može saopštiti da je neki zahtev (prethodno upućen) uspešno izvršen.
-

Iako DII pruža mnogo više fleksibilnosti od statičkih ogranaka, nekada njegovo korišćenje ima veću cenu. Uopšteno, kreiranje DII zahteva može prouzrokovati da ORB skriveno pristupa IR u cilju sakupljanja informacija o tipovima argumenata i povratnih vrednosti. Kako je IR i sam CORBA objekat, svaki skriveni IR zahtev koji isporuči ORB, može biti udaljeni poziv. Zbog toga, kreiranje i poziv jednog DII zahteva, nekada povlači za sobom više udaljenih poziva, što je skuplje od ekvivalentnog statičkog poziva. Statički pozivi ne impliciraju višestruke pristupe IR-u, pošto se oslanjaju na tipove informacija koji su već ugrađeni u aplikaciju.

3.5.2 INTERFEJS DINAMIČKE OSNOVE

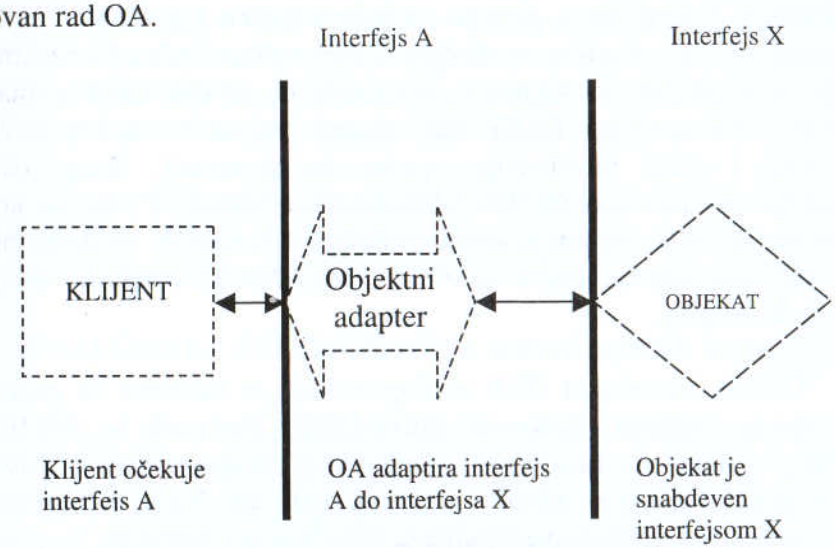
Analogno DII sa strane klijenta, sa strane servera stoji interfejs dinamičke osnove – DSI. Kao što DII klijentu omogućava pristup statičkim ograncima, DSI dozvoljava serverima da budu zapisani, bez da se osnove objekata koji su pozvani statički kompajliraju u program.

DSI je, za razliku od ostalih CORBA komponenata, dodat tek kasnije. Glavni razlog za to je bila podrška za implementaciju puteva za

komunikaciju ORB-a koji koriste različite protokole komunikacije. Zbog toga je DSI posebno koristan za uspostavljanje mostova između ORB-ova i aplikacija koje služe za premošćavanje CORBA sistema i servisa i implementacija koje ne rade pod CORBA-om.

3.6 OBJEKTNI ADAPTERI

Objektni adapteri (**Object Adapters – OA**) služe kao "lepak" između implementacije objekata CORBA-e i ORB-a. Uloga objektnog adaptera je da interfejs nekog objekta prilagodi interfejsu korisnika koji ga poziva. Jednostavnije rečeno, OA je objekat koji je umetnut da bi omogućio da klijent ne mora da zna kakav je interfejs objekta koji poziva. Na slici 4 je ilustrovan rad OA.



Slika 4: Uloga objektnih adaptera

OA predstavljaju dodatni napor da se ORB učini što jednostavnijim. Uloge OA su sledeće:

- **Registracija objekta** – OA podržavaju operacije koje omogućavaju da se entiteti programskih jezika registruju kao implementacije CORBA objekata
- **Generisanje objektne reference** - OA generišu objektne reference za CORBA objekte
- **Aktiviranje procesa servera** – ako je potrebno, OA startuju procese servera u kojima se mogu aktivirati objekti
- **Aktiviranje objekata** – OA aktiviraju objekte koji već nisu aktivni kada do njih pristigne zahtev
- **Poziv objekta** – OA prosleđuju zahteve registrovanim objektima

Bez OA, sposobnost CORBA-e da podržava različite stilove implementacija objekata bi bila ozbiljno ugrožena. Nedostatak OA bi značio da bi se implementacije objekata direktno povezivale sa ORB-om u cilju primanja zahteva. Kada bi skup interfejsa za poziv objekata bio mali, onda bi se mogle podržati samo neke implementacije objekata. Ako bi taj skup proširili, tada bi on bio preobiman i doveo bi do nepotrebnog proširenja i komplikovanja i samog ORB-a. Zbog svega toga, CORBA podržava višestruke OA. Za svaki programski jezik je potreban jedan objektni adapter.

3.7 ORB MEĐUPROTOKOLI

ORB međuprotokoli su novijeg datuma što se tiče CORBA specifikacije. Nastali su iz potrebe za operabilnošću između različitih ORB produkata. Ovi protokoli se realizuju na dva načina. Jedan je direktna veza između dva ORB-a, a moguća je u slučaju da ta dva ORB-a imaju iste objektne reference, isti OMG IDL sistem tipova, a možda da su im zajedničke i neke informacije vezane za sigurnost. Drugi način je uspostavljanje veza između dva ORB-a preko "mostova", što se koristi u slučaju da ti ORB-ovi imaju različite domene, a moraju da komuniciraju. Most tada uspostavlja mapu za ORB specifične informacije, od jednog domena do drugog.

Glavna arhitektura ORB međuprotokola je bazirana na generalnom ORB međuprotokolu **(General Inter-ORB Protocol – GIOP)** koji specifikuje sintaksu transfera i standardnog skupa formata poruka za međuprotokole ORB-a. GIOP je napravljen da bude jednostavan za implementaciju, a da pri tome ispunjava određene performanse.

Internet ORB međuprotokol **(Internet Inter-ORB Protocol –IIOP)** daje specifikaciju kako se formira GIOP na bazi TCP/IP protokola. Veza između GIOP-a i IIOP-a podseća na vezu između implementacije objekta i definicije njegovog OMG IDL interfejsa. GIOP daje specifikaciju protokola, kao što OMG IDL interfejs definiše protokol između objekta i njegovih klijenata, dok IIOP određuje kako se može implementirati GIOP koristeći TCP/IP , kao što implementacija objekta određuje realizaciju interfejsa objekta.

Arhitektura ORB međuprotokola predviđa i specifične ORB protokole koji zavise od vrste okruženja **(Environment-specific inter-ORB protocols – ESIOPs)**. Ovi protokoli omogućavaju kreiranje ORB protokola za specifične situacije gde su neke infrastrukture distribuiranih objektnih sistema već u upotrebi.

Za operabilnost između ORB-a neophodni su i standardni formati objektnih referenci. Dok su objektne reference skrivene od same aplikacije, ORB koristi sadržaj tih objektnih referenci u cilju utvrđivanja kako da najbolje uputi zahtev ka objektima. CORBA daje specifikaciju za standardni format objektnih referenci, koji se naziva *međuoperabilna objektna referenca* **(Interoperabile object reference – IOR)**. IOR čuva informacije koje su neophodne za lociranje i komunikaciju sa objektom i to preko jednog ili više protokola.

3.8 OBJEKAT, KLIJENT I SLUGA

Objekat **(Object)** je CORBA programski entitet koji ima svoj identitet, interfejs i implementaciju, koja je poznata pod nazivom *sluga* **(Servant)**. Sluga je implementacija entiteta programskog jezika koja definiše operacije koje podržava CORBA IDL interfejs. Sluge mogu biti pisane u različitim programskim jezicima, kao što su: C, C++, Java, Smalltalk i Ada.

Klijent **(Client)** je programski entitet koji poziva operaciju iz implementacije objekta. Pristupanje servisima udaljenog objekta je naravno skriveno od klijenta. To omogućavaju ostale komponente CORBA-e.

4. RAZVOJ CORBA SPECIFIKACIJA KROZ NJENE VERZIJE

Još od svog pojavljivanja 1991. godine, CORBA specifikacija je služila kao baza za veliki broj distribuiranih sistema. Iako je CORBA po svojoj specifikaciji fleksibilna i primenljiva u različitim okruženjima, morala je pretrpeti određene izmene u cilju poboljšanja svojih performansi, da bi i dalje predstavljala jednu od važnijih specifikacija za distribuirane objektne sisteme. Do sada, OMG je izbacio na tržiste nekoliko verzija CORBA-e, počev od verzije 1.0 do najnovije verzije 3.0. U daljem tekstu biće dat pregled najvažnijih odlika svake od verzija CORBA-e.

4.1 CORBA 1.0

Prva verzija CORBA-e je CORBA 1.0 koja se pojavila 1991. godine. Ovu verziju karakteriše nedostatak protokola za komunikaciju između aplikacija. Ovo je nadoknađivano time što je svaki ORB koristio svoj protokol za komunikaciju. Međutim ovo nije predstavljalo problem za prve korisnike CORBA-e, jer su u to vreme aplikacije koje su koristile CORBA 1.0 specifikaciju bile dovoljno male, da je svaka mogla da se osloni na servise samo jednog ORB-a.

Daljim razvojem, aplikacije su postajale sve veće i sve distribuiranije, i postajalo je neophodno da komuniciraju sa aplikacijama koje koriste i neke druge ORB-ove. Ovo je nadoknađeno u verzijama CORBA-e koje su usledile.

4.2 CORBA 2.0

Sredinom 1995. godine, pojavila se CORBA 2.0 specifikacija. Ona je donela dosta novina, od kojih su najbitnije sledeće:

- ORB jezgro
- OMG IDL
- Nosilac interfejsa
- Preslikavanje jezika
- Ogranci i osnove
- Dinamički poziv i prosleđivanje
- Objektni adapteri
- ORB međuprotokoli (GIOP i IIOP)

Svaka od ovih komponenata je doprinela povećanju iskorišćenja i primenljivosti CORBA specifikacija na probleme vezane za distribuirane objektne sisteme.

4.3 CORBA 3.0

CORBA 3.0 se nadovezije na prethodne CORBA specifikacije i nudi rešenja za neke probleme koji su uočeni u prethodnim verzijama. Ova verzija CORBA-e će biti opremljena sa tri nova veoma bitna obeležja:

- Prenosivi objektni adapter **(Portable Object Adapter – POA)**
- CORBA prosleđivanje poruka **(CORBA Messaging)**
- Objekti po vrednosti **(Objects By Value)**

U daljem tekstu će biti dat opis svakog od ovih obeležja.

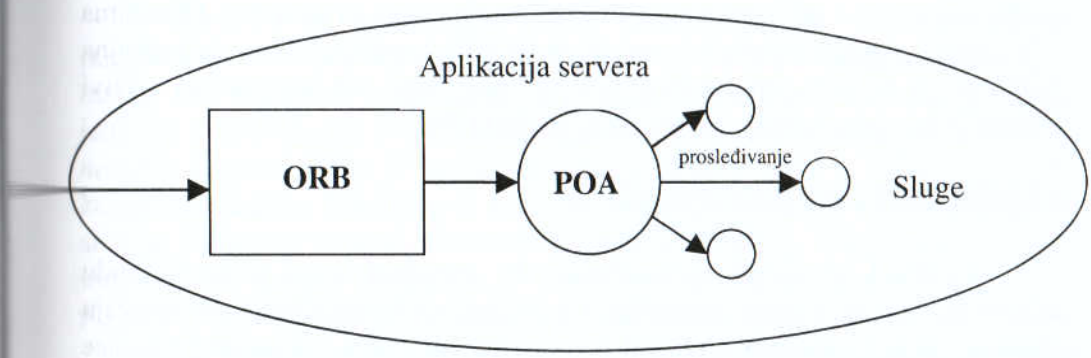
4.3.1 PRENOSIVI OBJEKTNI ADAPTER

Objektni adapteri u CORBA-i posreduju između CORBA objekata i implementacija programskih jezika, poznatih pod imenom sluge. Još od prvih verzija, pa do verzije CORBA 2.1, jedini standardni objektni adapter definisan od strane OMG-a je bio osnovni objektni adapter **(Basic Object Adapter – BOA)**. Kada je BOA specifikovan, izgledalo je da će on zadovoljavati većinu zahteva vezanih za implementaciju objekata, međutim pokazalo se da nije tako. U pokušaju da BOA podrži što više programskih jezika, njegova specifikacija je u određenim oblastima bila poprilično nejasna. Ovo je rezultovalo velikim problemima vezanim za prenosivost

BOA implementacija, jer je svaki isporučilac ORB-a problem rešavao na svoj način.

Krajem 1997. godine, OMG izbacuje CORBA 2.2 specifikaciju, koja ima novi objektni adapter – prenosivi objektni adapter. Glavna svrha POA, je da posreduje između ORB-a i aplikativnog servera, i da omogući njihovu prenosivost.

Kako radi POA? Prvo klijent poziva zahtev koristeći objektnu referencu koja ukazuje na ciljni objekat. Taj zahtev prima ORB servera. Deo tog zahteva sadrži identifikator koji se čuva u objektnoj referenci i naziva se *objektni ključ*. On jedinstveno identifikuje ciljni objekat u aplikaciji servera. Kako jedna aplikacija može imati više POA, objektni ključ omogućava ORB-u da uputi zahtev ka POA koji je domaćin ciljnom objektu. POA tada koristi deo objektnog ključa koji se naziva *identifikacija objekta* (**Object ID**), i služi da uspostavi vezu između ciljnog objekta i sluge programskog jezika. Sluge izvršavaju zahtev i isporučuju rezultate nazad u POA, POA ih sprovodi nazad u ORB, a ORB ih upućuje klijentu. Na slici 5 se može videti tok zahteva u serveru koji je baziran na POA.



Slika 5: Tok zahteva u serverima baziranim na POA

Zahtev
, POA radi kao prevodilac između objektnih referenci, identifikacije objekta i sluga, u cilju isporučivanja zahteva od CORBA objekata do sluga programskih jezika.

4.3.2 CORBA PROSLEĐIVANJE PORUKA

Kao što je navedeno u prethodnim odeljcima, CORBA omogućuje tri različita modela za pozivanje zahteva: sinhroni, odloženi sinhroni i jednosmerni poziv. Međutim, u proteklih par godina, nedostatak specifikacije za asinhrono prosleđivanje poruka u CORBA-i, smatran je za jedan od većih propusta. Asinhrono prosleđivanje poruka se, široko gledano, smatra za visoko kotirano rešenje za upotrebu u distribuiranim objektnim sistemima. To je stoga što se smatra da nisu svi sistemi u svakom trenutku kvalitetno povezani. Nekada se dešava da se veze prekidaju, posebno u velikim sistemima i tada je sinhrono prosleđivanje poruka praktično nemoguće, jer ono zahteva usku povezanost klijenta i servera.

Specifikacija za CORBA prosleđivanje poruka očuvava tri, već postojeća modela, ali dodaje i dva nova asinhrona modela za prosleđivanje poruka. To su:

- **Uzvraćeni poziv (Callback)** – klijent sa svakim pozivom zahteva šalje i dodatni parametar objektne reference, tako da kada stigne odgovor, ORB koristi tu objektnu referencu da bi isporučio odgovor aplikaciji.
- **Pozivanje (Polling)** – klijent poziva operaciju koja odmah vraća tip vrednosti koji se može koristiti za pozivanje ili čekanje na odgovor.

Uzvraćeni poziv i pozivanje su asinhroni modeli koji su dostupni klijentima koji koriste statičke ogranke generisane iz IDL interfejsa. Ovo je značajna prednost za većinu programskih jezika zbog toga što su statički pozivi prirodniji kao programski model, nego što je to DII.

4.3.3 OBJEKTI PO VREDNOSTI

CORBA je oduvek omogućavala da određeni tipovi podataka budu prihvaćeni kao argumenti operacija. Da bi omogućila objektima da pozivaju operacije drugih objekata, CORBA takođe dozvoljava da i objektne reference budu prihvaćene za argumente. Prosleđivanje objektne reference udaljenoj aplikaciji, znači da objekat ne menja svoje mesto, nego se svaki put kada aplikacija pozove objekat, zahtev preko mreže pomera ka objektu, umesto da se pomera sam objekat. Prosleđivanje objekata po vrednosti je nedavno dodato CORBA specifikaciji. Usvajanje ovog principa podrazumeva uvođenje tzv. tipa vrednosti (**valuetype**) u OMG IDL. Tip vrednosti podržava i podatke i operacije, može se izvoditi iz dugih tipova vrednosti, i može

Kada se tip vrednosti prosledi kao argument nekoj udaljenoj operaciji, on se kreira kao kopija u adresnom prostoru primaoca. Identitet kopije tipa vrednosti je potpuno različit od originala tako da operacije na jednom, nemaju uticaja na drugog. Zbog semantike kopije, sve operacije koje se

pozivaju na tip vrednosti su uvek lokalne u odnosu na proces u kojem on egzistira. Ovo znači da, za razliku od CORBA objekata, pozivi tipa vrednosti nikada ne podrazumevaju prenos zahteva i odgovora preko mreže.

Nedostatak ove specifikacije je u tome što je često previše komplikovana, a u isto vreme i nepotpuna.

5. DISTRIBUIRANI I USAĐENI SISTEMI KOJI RADE U REALNOM VREMENU

Tokom prošle dekade, pojavile su se specifikacije kao što su CORBA, DCOM, Java RMI i SOAP/.NET koje doprinose smanjivanju kompleksnosti distribuiranih objektnih sistema. Pored toga, one imaju značajnu ulogu i u razvoju distribuiranih i usađenih sistema koji rade u realnom vremenu **(Distributed, real-time and embedded (DRE) systems)**. DRE sistemi su teži za razvijanje i održavanje od ostalih sistema, zbog višestrukih ograničenja nad različitim dimenzijama svojih performansi. Neki od razloga za to su potreba za nametanjem ograničenja za *kvalitet servisa* **(Quality of Service – QoS)** i to što uglavnom svi usađeni sistemi rade pod ograničenjima vezanim za memoriju, procesore i napajanje.

Pored svih ograničenja, DRE sistemi postaju sve prisutniji i raznovrsniji u svojoj nameni. Primeri DRE sistema su: telekomunikacione mreže, tele-medicina, automatizacija procesa, odbrambene aplikacije,... Ipak jedno je zajedničko za sve DRE sisteme: *tačan ogovor isporučen prekasno je netačan odgovor*. Ovo je važno jer postoje DRE sistemi u kojima računari kontrolišu fizičke, hemijske ili biološke procese i uređaje. Primeri za takve sisteme uključuju avione, automobile, mobilne telefone, nuklearne reaktore, monitore za nadzor pacijenata,... Zbog toga je isporučivanje odgovora u realnom vremenu jedno od najjačih ograničenja za DRE sisteme.

Da bi odgovorio izazovima koje su postavili DRE sistemi, OMG je standardizovao CORBA specifikaciju koja radi u realnom vremenu **(Real-time CORBA)**. CORBA koja radi u realnom vremenu može pojednostaviti mnoge izazove koje postavljaju DRE aplikacije. Ona je dizajnirana za aplikacije koje postavljaju oštre uslove za rad u realnom vremenu, kao što su sistemi u avijaciji, ali i za aplikacije sa nešto mekšim ograničenjima, kao što je na primer procesiranje poziva u telekomunikaciji.

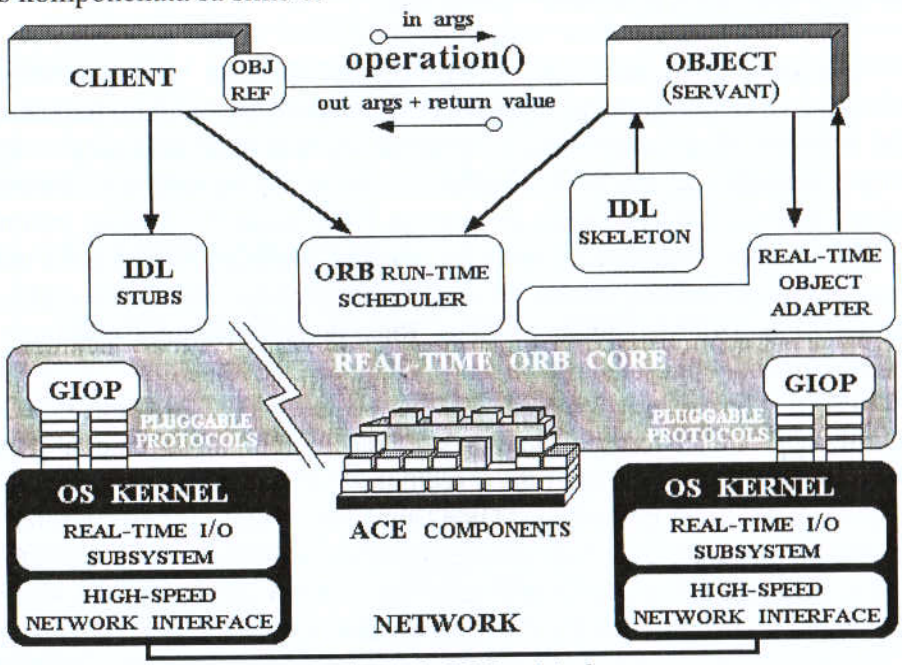
Postoje dve CORBA specifikacije za rad u realnom vremenu. Jedna je takozvani **TAO**, razvijen za C++, a druga specifikacija je poznata pod imenom **ZEN** i razvijena je za programski jezik Java.

5.1 TAO

Istraživači na univerzitetima iz Vašingtona, Sent Luisa, Irvina i sa Kalifornijskog univerziteta su u proteklih nekoliko godina sarađivali sa stotinama kompanija širom sveta u cilju prevazilaženja problema koji su se pojavljivali u razvoju DRE sistema. Kao rezultat tog istraživanja, pojavio se TAO, koji predstavlja ORB za CORBA-u koja radi u realnom vremenu i implementiran je u programskom jeziku C++. U široku upotrebu je ušao 1997. godine i od tada je nastavio i dalje da se razvija. TAO uglavnom zadovoljava visoke zahteve koje postavljaju DRE sistemi, a saglasan je sa većinom CORBA 3.0 specifikacija.

5.1.1 TAO, NJEGOV DIZAJN I NJEGOVE MOGUĆNOSTI

Istraživanja vezana za TAO su fokusirana na optimizaciji iskorišćenja ORB-a u cilju zadovoljenja QoS zahteva neke aplikacije. Komponente najnovije TAO verzije su prikazane na slici 6. U daljem tekstu sledi kratak opis komponenata sa slike 6.



Slika 6: TAO ORB arhitektura

LEGENDA:

CLIENT – klijent

PLUGGABLE PROTOCOLS – protokoli koji se uključuju

OBJECT (SERVANT) – objekat (sluga)

REAL-TIME I/O SUBSYSTEM – I/O podsistem u realnom

IDL STUBS – IDL ogranci vremenu

IDL SKELETON – IDL osnova

HIGH-SPEED NETWORK INTERFACE – mrežni

ORB RUN-TIME SCHEDULER – ORB program planer interfejs

REAL TIME OBJECT ADAPTER – objektni adapter u velike brzine realnom vremenu

NETWORK - mreža

REAL TIME ORB CORE – jezgro ORB-a u realnom vremenu

IDL prevodioc (IDL Compiler) – IDL prevodioc za TAO generiše prevedene ogranke i osnove koji podržavaju C++. Ovde je implementirano najnovije CORBA 3.0 preslikavanje jezika za C++, uključujući i najnovije odlike POA. IDL prevodioc podržava i specifikaciju za objekte po vrednosti, i specifikaciju za CORBA prosleđivanje poruka. Šta više, IDL prevodioc za TAO generiše kôd za asinhrone pozive klijenta i servera.

ORB jezgro (ORB Core) – TAO ORB jezgro obezbeđuje efikasnu i opsežnu infrastrukturu za aplikacije koje imaju visoke performanse, a rade u realnom vremenu. Takođe obezbeđuje i mehanizme za sinhronu i asinhronu komunikaciju, po principu dvosmernog ili jednosmernog poziva metode. Podržava i paralelni tok u realnom vremenu za strategije za prosleđivanje.

ORB međuprotokoli (Inter-ORB Protocols) – TAO u sebi sadrži protokol u koji je implementiran CORBA 3.0 Generalni/Internet ORB međuprotokol (GIOP/IIOP). Takođe podržava i statičke i dinamičke modele, SII/SSI i DII/DSI, respektivno.

POA – obezbeđuje konstantan nadzor sluga koji su bazirani na objektnom ključu i imenima operacija koja su uključena u CORBA zahteve, bez obzira na broj objekata, operacija ili ugnježdenih POA.

Nosioci implementacije i interfejsa (Implementation and Interface Repositories) – TAO nosilac implementacije automatski lansira servere kao odgovor na klijentove zahteve. Takođe, u TAO je uključen nosilac interfejsa koji u toku rada može pružati informacije o IDL interfejsima.

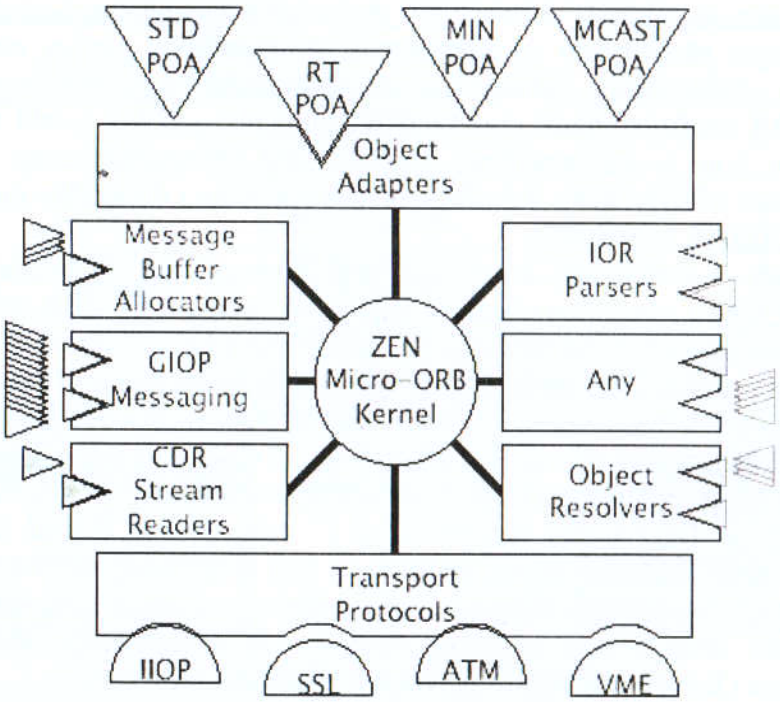
Iako je TAO široko rasprostranjen i dokazan kao moćan alat za razvoj pouzdanih sistema visokih performansi koji rade u realnom vremenu, i on ima svojih nedostataka. Njegovi glavni nedostaci su bogat skup obeležja koji nekada zahteva više memorije nego što je neki uređaji imaju na raspolaganju i kompleksnost preslikavanja jezika CORBA-e za C++.

5.2 ZEN

ZEN je ORB za CORBA-u koja radi u realnom vremenu koji je implementiran u programskom jeziku Java. Razvijen je kao alternativni alat sa pojednostavljenim programskim modelom za DRE aplikacije. ZEN preuzima sve dobre osobine koje je imao TAO, ali ide i korak dalje i poboljšava neke njegove performanse. Međutim, ključna razlika između ZEN-a i ranijih CORBA ORB specifikacija, jeste to što se struktura ZEN-a oslanja na koncepciju *virtuelne komponente*. Ovakva struktura omogućava ZEN-u da smanji memorijski otisak, tako što isključuje iz aplikacije opcione ili retko korišćene komponente. Ranije verzije ORB-a su bile *monolitičke*, tj. čuvale su u sebi kodove koji podržavaju sve moguće događaje, obeležja i varijante, koje su date u CORBA specifikaciji.

5.2.1 ZEN ARHITEKTURA

Arhitektura ZEN-a je bazirana na konceptu *slojevitog uključivanja*, gde se određene komponente mogu po potrebi "uključiti" ili "isključiti" iz konfiguracije date specifikacije. Na osnovu ranijih iskustava sa TAO-om, formirana je arhitektura ZEN-a koja je prikazana na slici.



Slika 6: Mikro ORB ZEN arhitektura

LEGENDA:

STD POA – standardni POA
Any – bilo koji modeli rukovanja
RT POA – POA u realnom vremenu
Object Resolvers – objektni sistemi za
MIN POA – minimalni POA rešavanje
MCAST POA – POA koji ima više formi
IIOP – ĬIOP protokol
Message Buffer Alocators – alokatori bafera za poruke
SSL – Secure Socket Layer protokol
GIOP Messaging – GIOP prosleđivanje poruka
ATM – Asynhronous Transfer Mode
CDR Stream Readers – čitači CDR toka podataka
VME – Versa Model Europe
IOR Parsers – parseri međuoperabilnih objektnih referenci

Ahitektura ZEN-a prikazuje sasvim drugačiju organizaciju ORB-a. Za razliku od monolitičkog ORB dizajna, što je bilo svojstveno za TAO, ZEN uvodi mikro-ORB arhitekturu, koja donosi smanjenje memorijskog otiska i poboljšanje performansi. Sa slike se može videti da je osam servisa jezgra ORB-a (objektni adapteri, alokatori bafera za poruke, GIOP prosleđivanje poruka, čitači/pisači CDR toka podataka, transportni protokoli, objektni sistemi za rešavanje, parseri interoperabilnih objektnih referenci i ostali modeli rukovanja) izbačeno iz ORB-a radi njegovog pojednostavljivanja. Preostali deo ORB-a se naziva *jezgro* ZEN-a **(ZEN kernel).** Zbog koncepta slojevitog uključivanja, svaki servis se učitava tek onda kada se pojavi potreba za njegovom upotrebom. Pri tome se svaki ORB servis može dekomponovati na manje komponente, koje se opet po potrebi mogu uključivati u ORB. Ovakva arhitektura čini ZEN moćnom platformom i za istraživanja, jer se različite implementacije ORB komponenata mogu uključiti u ORB, što omogućava da se utvrdi njihova korisnost.

6. RAZVOJ CORBA-e U BUDUĆNOSTI

Glavne smernice razvoja CORBA-e određuju sami zahtevi za predlozima – RFP, koje OMG konstantno postavlja svojim članovima i korisnicima njegovih specifikacija. Glavne odrednice budućeg razvoja CORBA-e se grupišu oko nekoliko oblasti, gde se ona najčešće i koristi. To su:

- poslovanje
- elektronsko poslovanje
- finansije
- proizvodnja
- medicina
- telekomunikacije
- transport

Za sve ove oblasti, OMG je formirao grupe koje izdaju RFP, iste obrađuju i daju predloge za nove CORBA specifikacije ili za poboljšanje već postojećih. Rezultat svega jesu pojavljivanja novih CORBA verzija, i konstantan razvoj same CORBA-e.

Pored konstantnog razvoja CORBA arhitekture, ona se stalno suočava i sa drugim izazovima. Najvažniji od tih izazova jeste sama konkurencija, jer CORBA nije jedina specifikacija za distribuirane objektne sisteme. Najveći konkurent CORBA-i dolazi od strane druge specifikacije za distribuirane objektne sisteme koja je produkat jedne kompanije koja ima najdominantnije mesto u industriji. Specifikacija o kojoj govorimo je DCOM, a kompanija naravno – Microsoft. DCOM se jako dobro kotira kao specifikacija za DOC, i OMG mora uložiti značajan napor u budućnosti, da bi CORBA mogla da isprati razvoj DCOM-a. Jedan od načina za održavanjem ovih specifikacija aktuelnim jeste i njihovo udruživanje, tj. sada se radi na formiranju zajedničkih protokola, koji bi trebalo da omoguće nesmetanu komunikaciju između CORBA-e i DCOM-a.

Sa druge strane, radi se i na integraciji CORBA-e sa Java-om, tj. na formiranju specifikacija koje će biti sastavljene od najboljih CORBA/EJB komponenata. Od Java-e bi bio preuzet njen RMI, tako da bi novi CORBA protokoli, radili po principu Java-inog RMI-a.

I na kraju, vredelo bi spomenuti razvoj CORBA-e koji bi išao u pravcu DRE sistema. Kako ovi sistemi postaju sve popularniji, i kako nalaze sve veću primenu u svim granama industrije, medicine, poslovanja, telekomunikacija, itd., može se očekivati da u budućnosti ovo postane jedan od glavnih pravaca za dalji razvoj CORBA specifikacija.

7. ZAKLJUČAK

Analizirajući različite vrste softvera, modela, specifikacija, koje su se pojavile u skorije vreme, može se zaključiti da objektna orjentacija sve više uzima maha. Sa druge trane, razvojem novih tehnologija i sve većom potrebom za informacijama u tzv. realnom vremenu, nameće se ideja da komponente jednog sistema ne moraju biti fizički prisutne na jednom mestu, nego mogu biti praktično "rasute" po različitim sistemima, mrežama,... Distribuirani objektni sistemi pružaju upravo tu mogućnost rada na daleko, tj. korisnik može raditi i koristiti neke objekte pod prividom da se oni nalaze na jednom istom mestu, dok oni zapravo na tom mestu uopšte i ne moraju biti prisutni. Ovakvim rezonovanjem se bitno rasterećuju pojedinačni sistemi, a nije zanemarljivo ni smanjenje troškova koji bi bili daleko veći kada bi korisnici morali doslovno da idu od mesta do mesta u potrazi za određenim objektima.

Tako stižemo i do CORBA-e, koja je do sada već prešla dug put u svom razvoju. OMG koji je i izdao CORBA specifikaciju je od male grupe istomišljenika izrastao u ogroman konzorcijum koji danas broji više od 800 članova, sa tendencijom daljeg porasta. Kao takav, OMG danas zauzima značajnu i visoko kotiranu poziciju na tržištu. CORBA standard se neprekidno poboljšava i usavršava, u cilju pružanja sve većih mogućnosti njenim korisnicima. Od svojih skromnih početaka, možemo reći da je CORBA danas izrasla u dokazanu i moćnu platformu za rad sa distribuiranim objektnim sistemima, koja podržava rad sa velikim brojem operativnih sistema i programskim jezicima različitih proizvođača.

Uprkos tome što se po svojim mogućnostima trenutno nalazi u samom vrhu, CORBA se i dalje razvija. Trenutno se rade njena proširenja vezana za DRE sisteme, koji će CORBA-u učiniti još upotrebljivijom, ako je to uopšte moguće.

OMG tek sada ima velike planove vezane za CORBA-u, i u budućnosti će sve biti podređeno njihovoj realizaciji. Ne samo što će CORBA postati još moćnija i robusnija, što se tiče razvoja distribuiranih objektnih sistema, nego će i sve više biti integrisana u razvoj distribuiranih aplikacija. Gledano sa stanovišta razvoja CORBA-e, budućnost nam tek donosi mnogo novih i uzbudljivih dostignuća.

8. LITERATURA

1. Klefstad, R., Krishna, S.A., Schmidt C.D. : Design and Performance of a Modular Portable Object Adapter for Distributed, Real-Time, and Embedded CORBA Applications, Electrical and Computer Engineering Dept., University of California, Irvine, USA
www.cs.wustl.edu/~schmidt/PDF/POA-02.pdf

2. Vinoski, S. : CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Communications Magazine, Vol. 35, No.2, February 1997.
www.iona.com/hyplan/vinoski/ieee.pdf

3. Vinoski, S. : New Features for CORBA 3.0, Commmunications of the ACM, Vol. 41, No. 10, october 1998.
www.iona.com/hyplan/vinoski/cacm.pdf

4. Krishna, S.A., Schmidt C.D., Klefstad, R., Corsaro, A. : Real-time CORBA Middleware, Middleware for Communications, Edited bz Qusay H. Mahmoud, John Wiley & Sons, Ltd, 2001.
www.cs.wustl.edu/~schmidt/PDF/Real-**time**-Middleware.pdf

5. Vinoski, S. : Distributed Object Computing With CORBA, C++ Report magazine, July/August 1993.
www.cs.wustl.edu/~schmidt/**corba**.html

6. Schmidt C.D., Vinoski, S. : Object Interconnections – Modeling Distributed Object Applications (Column 2), SIGS C++ Report magazine, February 1995. www.iona.com/hyplan/**vinoski**/

7. Schmidt C.D., Vinoski, S. : Object Interconnections –Introduction to Distributed Object Computing (Column 1), C++ Report magazine, January1995. www.iona.com/hyplan/**vinoski**/

9. PRILOG

U ovom prilogu dat je spisak svih skraćenica koje su korišćene u radu, njihov pun naziv na engleskom jeziku i prevod na srpski jezik (ako je adekvatan).

SKRAĆENICA	PUN NAZIV	SRPSKI JEZIK
ATM	Asynhronous Transfer Mode	Mod asinhronog transfera
BOA	Basic Object Adapter	Osnovni objektni adapter
CDR	Character Data Representation	Reprezentacija podataka tipa karaktera
COM	Component Object Model	Komponentni objektni model
CORBA	Common Object Request Broker Architecture	Zajednička ORB arhitektura
DCOM	Distributed Component Object Model	Distribuirani komponentni objektni model
DDCF	Distributed Document Component Facility	Komponentno odeljenje za distribuirane dokumente
DII	Dynamic Invocation Interface	Interfejs dinamičkog poziva
DOC	Distributed Object Computing	Distribuirani objektni sistemi
DRE	Distributed Real-Time Embedded	Distribuirani i usađeni koji rade u realnom vremenu
DSI	Dynamic Skeleton Interface	Interfejs dinamičke osnove
EJB	Enterprise Java Beans	-
ESIOP	Environment-specific inter-ORB protocols	ORB međuprotokoli specifični za okoline
GIOP	General Inter-ORB Protocol	Generalni ORB međuprotokol

IDL	Interface Definition Language	Jezik za definiciju interfejsa
IIOP	Internet Inter-ORB Protocol	Internet ORB međuprotokol
IOR	Interoperabile object reference	Međupoerabilna objektna referenca
IR	Interface Repository	Nosilac interfejsa
MCAST POA	Multicast Portable Object Adapter	Prenosivi objektni adapter sa više uloga
MIN POA	Minimal Portable Object Adapter	Minimalni prenosivi objektni adapter
OA	Object Adapter	Objektni adapter
OMA	Object Management Architecture	Arhitektura objektnog upravljanja
OMG	Object Management Group	Grupa za objektno upravljanje
OO	Object Orientation	Objektna orjentacija
ORB	Object Request Broker	-
POA	Portable Object Adapter	Prenosivi objektni adapter
QoS	Quality of Service	Kvalitet servisa
RFP	Request For Proposals	Zahtev za predlozima
RMI	Remote Method Invocation	Poziv udaljene metode
RPC	Remote Procedure Call	Poziv udaljene procedure
RT POA	Real-Time Portable object Adapter	Prenosivi objektni adapter u realnom vremenu
SII	Static Invocation Interface	Interfejs statičkog poziva
SOAP	Simple Object Access Protocol	Protokol za jednostavan pristup objektima
SSI	Static Skeleton Interface	Interfejs statičke osnove

SSL	Secure Socket Layer protokol	
STD POA	Standard Portable Object Adapter	Standardni prenosivi objektni adapter
TCP/IP	Transfer Control Protocol/Internet Protocol	Protokol za kontrolu transfera/Internet protokol
VME	Versa Model Europe	

WEB SERVISI[1]

[1]Materijal je preuzet sa *www.svezaweb.dzaba.com*

SADRŽAJ:

1. WEB SERVISI

Oni predstavljaju osnovne gradivne blokove budućih informacionih sistema, a u suštini su aplikacije koje su raspložive na mreži i koje mogu da urade ono što vama u tom trenutku treba. Drugim rečima, to su resursi koji se adresiraju primenom URL-a koji vraća informaciju korisniku koji želi da je koristi. Glavni komunikacioni protokol je SOAP tj. XML preko HTTP-a.

Osnovni pokretač ovih promena je XML, koji kroz svoju jednostavnost omogućuje praktičnu nezavisnost aplikacija i sistema jer je razumljiv i za čoveka i za mašinu.

Web Servisi se objavljuju na jedinstvenoj lokaciji i nude se kao usluge. Kako bi usluga bila kompletna, nudi se i potpuna specifikacija interfejsa, poslovnih zahteva, kvaliteta servisa, pravnih i finansijskih uslova korišćenja itd.

Ukoliko posedujete već napisane aplikacije koje nude stabilna i proverena rešenja, jednostavno možete objaviti aplikaciju kao uslugu korisnicima širom naše planete, zbog te specifičnosti ova tehnologija se brzo raširila.

Primera radi pogledajte sledeće lokacije :

IBM Web Services Browser
http://apps.alphaworks.ibm.com/browser/ URL

Microsoft VS Web Service Search Categorization
http://uddi.microsoft.com/search/ URL

Google's Web Service - access the Google search engine
http://www.google.com/apis/ URL

Amazon's Web Service - access Amazon's product information
http://associates.amazon.com/exec/
panama/associates/ntg/browse/-/56763 URL

1.1. SET PROTOKOLA WEB SERVISA

Set Web servis protokola stalno se razvija i koristi se da definiše, otkriva i implementira Web servise. Osnova ovih protokola leži u sledeća četiri nivoa :

Service Transport: Ovaj nivo je odgovoran za prenos poruka između aplikacija i u njega su trenutno uključeni HTTP, SMTP, FTP, i novi protokoli kao što je BEEP - Blocks Extensible Exchange Protocol.

XML Messaging: Ovaj nivo je odgovoran za razumevanje poruka za razmenjivanje koje se implementiraju u XML formatu i trenutno se koristi XML-RPC (XML – Remote Procedure Call) i SOAP (Simple Object Access Protocol).

Service Description: Ovaj nivo definiše javni interfejs za određeni Web servis, i trenutno se opisuje kroz WSDL (Web Service Description Language).

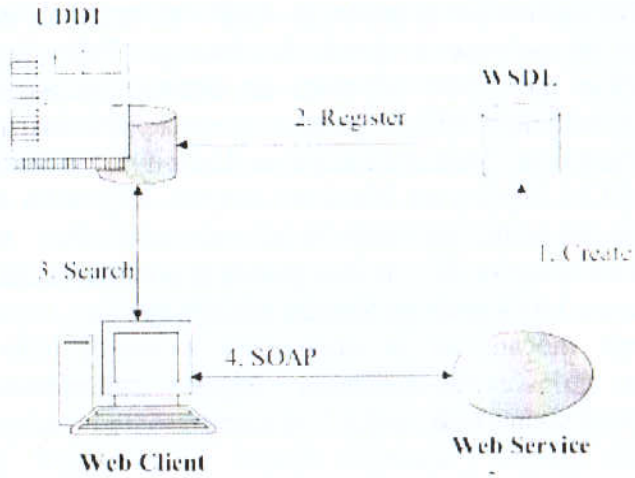
Service Discovery: Ovaj nivo je odgovoran za centralizovanje servisa u zajednički i jedinstveni registar koji obezbeđuje jednostavno objavljivanje i pronalaženje servisa. Otkrivanje servisa trenutno se obrađuje kroz UDDI (Universal Description, Discovery, and Integration).

Pored pomenutih, set sadrži nove protokole koji se još razvijaju, uključujući WSFL (Web Services Flow Language), SOAP-DSIG (SOAP Security Extensions: Digital Signature) i USML (UDDI Search Markup Language).

Nije potrebno da razumete kompletan set protokola da biste radili sa Web servisima. Ukoliko znate osnove HTTP-a, dovoljno je da započnete sa XML Messaging nivoom.

1.2. KAKO DA NAPRAVITE WEB SERVIS?

Web servisi mogu se kreirati od već gotovih aplikacija ili započeti od nule, i pri tome možete programske jezike, gotove komponente i platforme koristiti po svom izboru.



Slika 1.

Pošto se usluga osmisli i kreira potrebno je :

1. Opisati servis pomoću WSDL jezika.
WSDL je zasnovan na XML i predstavlja opis interfejsa Web servisa koji treba da sadrži:

- Opis interakcije koju servis nudi
- Opis argumenata i rezultata koji su uključeni u interakciju
- Adresu za lociranje servisa
- Komunikacioni protokol
- Format podataka koji se koristi u porukama
- Definisanje interfejsa je zapravo ugovor između servera i klijenta

2. Registrovati servis na UDDI registru, Internet lokaciji predviđenoj za objave servisa, i time omogućiti i svom softveru da koristi tuđe usluge, i zainteresovanim korisnicima da pronađu vaš servis.
UDDI predstavlja centralizovanu lokaciju koja obezbeđuje mehanizam za registrovanje i pronalaženje servisa. Koristi SOAP za komunikaciju i omogućuje klijentima da pronađu servis i serveru da ga objavi.

3. Omogućiti pristup svom web servisu posredstvom SOAP-a, (Simple Object Access Protocol), koji koristi XML jezik za specificiranje pozivnih parametara i rezultata rada servisa.

SOAP je kompletno kreiran na postojećim, proverenim i široko prihvaćenim tehnologijama kao što su HTTP i XML. SOAP koristi XML za prenos podataka između aplikacija, a pošto je XML univerzalni standard, sve platforme mogu da pristupe i obrade informaciju. Pošto koristi HTTP, jednostavno prolazi kroz port 80, tako da firewall-ovi ne predstavljaju problem. Pristup različitim aplikacijama na raznim platformama sa SOAP-om postaje jednostavan, Java aplikacija na Unix-u jednostavno može da poziva metode COM objekta na Windows serveru. Klijentska aplikacija na iMac-u pristupa objektu na mainframe računaru. Sve ovo postaje transparentno i ne zahteva bilo kakvu posebnu administraciju. SOAP ne pokušava da zameni bilo koji drugi distribuirani sistem.
SOAP je manje moćan, ali je impresivan u svojoj jednostavnosti i proširivosti, što je veoma korisno s aspekta potrebe za što širim prihvatanjem, i od strane kompanija, i od strane programera.

Brzina razvoja Interneta dovela je do pojave mnogih tehnologija i standarda koji su istom tom brzinom i nestali. Sa današnje tačke gledišta postoje samo dva prilaza i to kroz J2EE (Java 2 Platform Enterprise Edition) okruženje i Microsoft-ov .NET skup tehnologija.

1.3. J2EE I .NET TEHNOLOGIJE

Obe platforme su korisne i vode ka istoj destinaciji, odabir se vrši u odnosu na već postojeću bazu znanja, postojeće informacione sisteme, veze sa partnerima i korisnicima. Ka pravoj odluci uvek vode odgovori na ova pitanja, a ne minorne razlike u funkcionalnosti.

Argumenti za podršku obema platformama:

- Za obe platforme morate podučiti razvojni tim (Java podučavanje za J2EE, objektno orijentisano podučavanje za .NET)
- Možete kreirati web servise koristeći obe platforme.
- Obe platforme su sa sistemskog pogleda jeftine kao što su jBoss/Linux/Cobalt za J2EE, ili Windows/Win32 hardware za .NET
- Skalabilnost obeju platformi teoretski je neograničena

2. WEB ARHITEKTURA[2]

(*Web Services*), omogućavaju jednostavnu komunikaciju između različitih sistema, platformi i aplikacija, i predstavljaju nove gradivne elemente budućih informacionih sistema za Internet. Mogu se jednostavno izgraditi primenom različitih programskih jezika.

Dvoslojna arhitektura u velikim organizacijama suočava se sa problemom održavanja i proširivanja informacionog sistema novim uslugama. Svaka izmena na aplikaciji iziskuje previše resursa. U okviru ovog dela opisaće se primer troslojne arhitekture sa upotrebom Web servisa za komunikaciju između klijenata i servera, iz perspektive kreatora.

Kreiraće se jednostavna troslojna, distribuirana aplikacija, kako bi se obezbedila čista podela između logičkih celina radi jednostavnog održavanja, prilagođavanja i nadogradnje sistema novim funkcijama.

Aplikacija omogućava pregled i manipulaciju podacima iz MS SQL baze podataka preko klijenata desktop i Web klijenata pod Windows i Linux platformom. Komunikacija između klijenata i servera odvijaće se preko jednog Web servisa, korišćenjem HTTP-a i XML-a.

Aplikacija koristi Web servis, kako bi smo demonstrirali jednostavnost i najbolju praksu za razvoj i dizajn rešenja budućih informacionih sistema. Ova demo aplikacija može da se koristi kao šablon za kreiranje drugih aplikacija zasnovanih na Web servisima, razmatrajući nove ciljeve i tehnologije implementacije.

Aplikacija treba da se zasniva na otvorenim standardima kako bi se omogućilo jednostavno proširivanje i prilagođavanje novim zahtevima i potreba tj. sistem treba da se odlikuje sa :

- Modularnim dizajnom
- Fleksibilnom, skalabilnom i sigurnom arhitekturom
- Brzom i pouzdanom komunikacijom

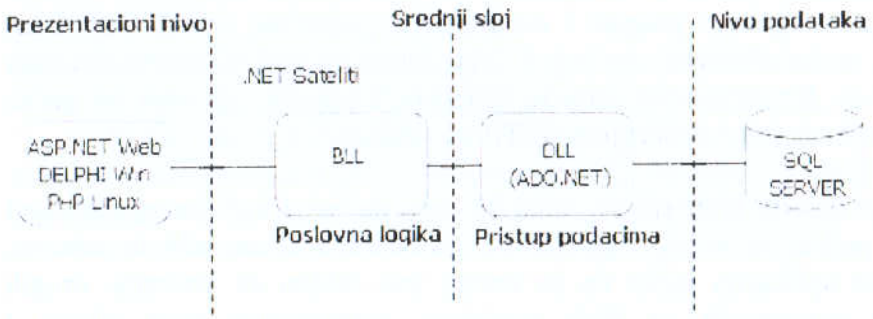
Sagledavši prethodne činjenice opredeljenje je za sledeće standarde i tehnologije :
Microsoft .NET (ASP.NET, ADO.NET, C#)
XML, XSLT i XmlSchemas (Xml šeme)
Delphi, PHP
SOAP (Simple Object Access Protocol) i WebServices (Web servisi)

---

[2] Materijal je preuzet sa *www.svezaweb.dzaba.com*

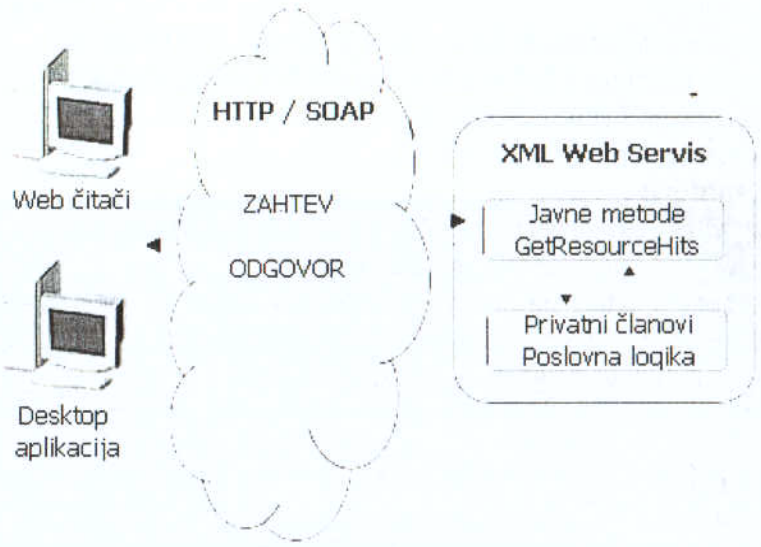
2.1. TROSLOJNA ARHITEKTURA APLIKACIJE

Sloj podataka
Potrebni podaci smešteni su na SQL SERVER bazi podataka zajedno sa uskladištenim procedurama.
Srednji sloj
Poslovna logika i pristup podacima obezbeđeni su preko satelita tj. nezavisnih kompajliranih komponenti (Microsoft.Net Assembly) i XML-a radi jednostvnog upravljanja, komunikacije i integracije različitih tipova i izvora podataka.
Prezentacioni sloj
Kao klijente možemo posedovati razne PC Desktop aplikacije, Web aplikacije, Mobilne aplikacije itd. u zavisnosti od tehnologija koje koristimo za prezentacioni sloj, kao što su ASP.NET, MOBILE.NET, PHP, DELPHI itd.



Slika 2.

Komunikacija između klijenta i sistema odvija se preko HTTP-a ili HTTPS-a, zavisno od potrebnog nivoa sigurnosti. Što znači da se klijenti konektuju na server preko standardnog porta za Web,a podaci putuju kao XML set podataka uz korišćenje SOAP-a.



Slika 3.

Simple Object Access Protocol (SOAP) definiše protokol za razmenu informacije. Glavni deo SOAP specifikacije definiše skup pravila za upotrebu XML za reprezentaciju podataka. Drugi delovi SOAP specifikacije definišu prošrivi format poruka, konvencije za udaljeno pozivanje metoda i povezivanje sa HTTP protokolom.

Prednosti ovakog pogleda na sistem su:

- Ovakva izolacija omogućava veću upotrebljivost napisanog programskog koda a olakšava održavanje i unapređivanje programerskog koda.

- Razvoj je olakšan pošto čista dekompozicija funkcionalnosti omogućava timsku saradnju i fokusiranje na različite delove aplikacije tokom implementacije.

- Poslovna logika i pristup podacima su centralizovani pa je obezbeđeno jednostavno održavanje.

Pre nego što krenemo u realizaciju aplikacije odredimo hardverske i softverske zahteve za aplikaciju.

Serverski zahtevi:

- Microsoft Windows® 2000 (i novije verzije .NET servera)
- Microsoft Internet Information Server 5 (i novije verzije)
- SQL Server 7 (i novije verzije)
- Microsoft .NET Framework 1.0

Klijentski zahtevi:

- Web čitač (*Browser*) ili instalirana Windows aplikacija
- Pristup Ineternet-u, intranetu ili extranetu

Jedan od mogućih scenarija kreiranja troslojnih aplikacija, je kreiranje sloja po sloja, može prvo da se realzuje šema podataka i sloj pristupa podacima, potom da se razvija poslovna pa prezentaciona logika.

3. ZAKLJUČAK

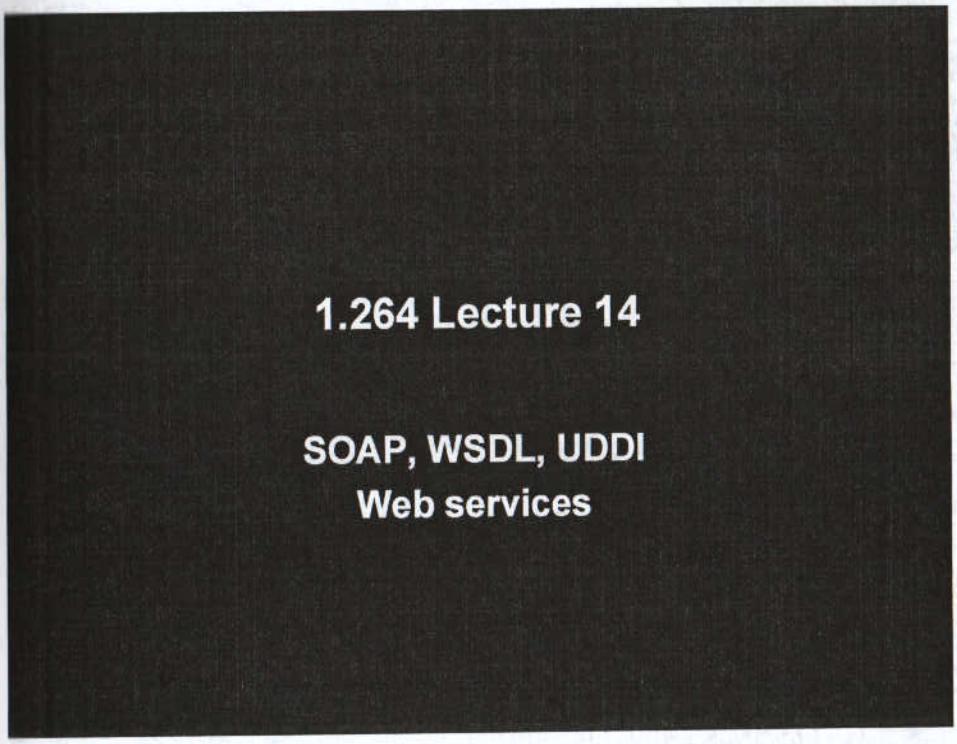
Web servisi čine novo „čedo" profesionalnog razvoja distribuiranih aplikacija. Rešavaju glavne probleme kod platformske zavisnosti i jezičke zavisnosti. Implementacija SOAP a samim tim Web servisa je moguća u skoro svakom programskom jeziku, na svim popularnim platformama. Postojeća distribuirana okruženja se proširuju za podršku SOAP-u.

Microsoft je, izdavši .NET Visual Studio, zajedno sa konceptom .NET runtime-a, napravio ozbiljan korak ka usvajanju SOAP-a za standard u razvoju distribuiranih aplikacija. Sun je najavio nešto slično u vidu svoje ONE (Open Network Environment) platofrme. IBM (WebSphere) i ostali industrijski giganti na ovom polju takođe najavljuju opsežnu SOAP podršku.

SOAP poseduje potencijale da kreira transparentni web za servise i aplikacije kojima se može pristupiti na zahtev od strane svakoga sa svake tačke, što će dovesti do eksplozivnog rasta novih servisa i prihoda od njih.

Veliki plus za SOAP i Web servise leži u podršci od strane velikih kompanija kao što su IBM i Microsoft kao i već postojećoj velikoj strukturi koja je izgrađena za WEB i HTTP. SOAP se i dalje razvija i na scenu će stupiti čitav niz novih protokola koji će pojednostaviti i olakšati primenu Web servisa u svim sferama poslovanja.

4. PRILOG 1[3]:



[3] *Prevod Lecture 14.. SOPA, WSDL,UDDI – Web services (PDF), MIT OpenCourseWare » Civil and Environmental Engineering » 1.264J Database, Internet, and Systems Integration Technologies, Fall 2002*

Front Page Demo

- File->New Web (must create on CEE server)
- Choose Web type
- Add navigation using Format->Shared Borders (frames)
  - Use top and left, include navigation buttons
  - Go to Navigation view, drag pages into workspace
- Save each page before viewing in browser
- Create structure, layout of Web site
  - Navigation window
- Organize files and folders
  - Folders window
- Import and export files
  - File->Import, don't copy
- Tests and repairs hyperlinks
  - Hyperlinks view, Tools->Recalculate Hyperlinks

Simple Object Access Protocol: SOAP

- Microsoft proposed the SOAP standard, having given up on ActiveX/COM for Internet applications
- SOAP is essentially XML and HTTP
  - Intent is to introduce no new technology for distributed computing beyond what is being used
  - SOAP adds some headers to HTTP; no other changes
  - Other middleware solutions are incorporating SOAP (CORBA mostly; COM (MS) moving to SOAP directly)
- Key element of SOAP:
  - New MIME type: text/xml
  - Agreed definitions of data types, mandatory values, etc.
- SOAP implements all the key features of legacy middleware (COM and CORBA) at a fraction of the complexity

SOAP

- URLs replace the repositories and registries of COM and CORBA
- A search engine can be used to dynamically locate remote objects unknown to the client application when designed
- HTTP POST and response replace application programming (in legacy CORBA and COM IDLs)
- SOAP (HTTP and XML) is all in text rather than binary, so it's much easier to interoperate across machines and debug it
- SOAP is sufficiently efficient for most machine-machine communication
  - Don't use it on a single machine: use native COM or Java mechanisms

UDDI

- Universal Description, Discovery and Integration
  - UDDI Business Registry (UBR) is global public directory of businesses and services
  - Users query UBR to discover Web services and get info to use them
  - White pages, yellow pages, green pages (services)
  - Uses SOAP to publish, edit, browse the UBR
  - WSDL (Web Services Description Language) is used to describe the services
- Information at Microsoft and IBM (see homework 5)
  - Brief demo

WSDL

- Contains:
  - Description and format of messages that can be passed in <types> and <message> tags
  - Direction of message passingin <portType>:
    - Request-only, request-response, response-only
  - Message encoding in <binding> element (literal, etc.)
  - Location where service is offered in <service> element

WSDL Example

```
<?xmlversion="1.0"?>
<definitions name="StockQuote"
    targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <schema targetNamespace=http://example.com/stockquote.xsd
                xmlns="http://www.w3.org/2000/10/XMLSchema">
                <element name="TradePriceRequest">
                    <complexType>
                        <all>
                        <element name="tickerSymbol" type="string"/>
                        </all>
                    </complexType>
                </element>
                <element name="TradePrice">
                    <complexType>
                        <all>
                        <element name="price" type="float"/>
                        </all>
                    </complexType>
                </element>
        </schema>
    </types>
```

WSDL, p.2

```
<message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
</message>
<portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
    </operation>
</portType>
<binding name="StockQuoteSoapBinding"  type="tns:StockQuotePortType">
    <soap:binding style="document"   transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
        <soap:operation   soapAction="http://example.com/GetLastTradePrice"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
<service name="StockQuoteService">   (etc.—defines URL for service)
```

SOAP, UDDI and WSDL

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
```

From UDDI

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <SOAP-ENV:Header>
        <t:Transaction xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
            5
        </t:Transaction>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>SUNW</symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

WSDL

XML, SOAP, UDDI, WSDL Summary

- XML is built into major Internet applications developed in the last 12-24 months
- XML is expected to be the dominant protocol for Internet commerce, supply chain, engineering collaboration and other cross-organization activities
- XML document type definitions (DTD, XSchema) are difficult to standardize but there are many efforts underway
  - RosettaNet, other exchanges
  - SOAP for the most general interoperability
- XML will replace legacy middleware (COM and CORBA) over time, and will probably be the dominant payload on the 'Net
  - See http://www.theserverside.com/resources/article.jsp?l=WebServices-Dev-Guide
- SOAP in wide use now; WSDL coming into use
- UDDI in early stages

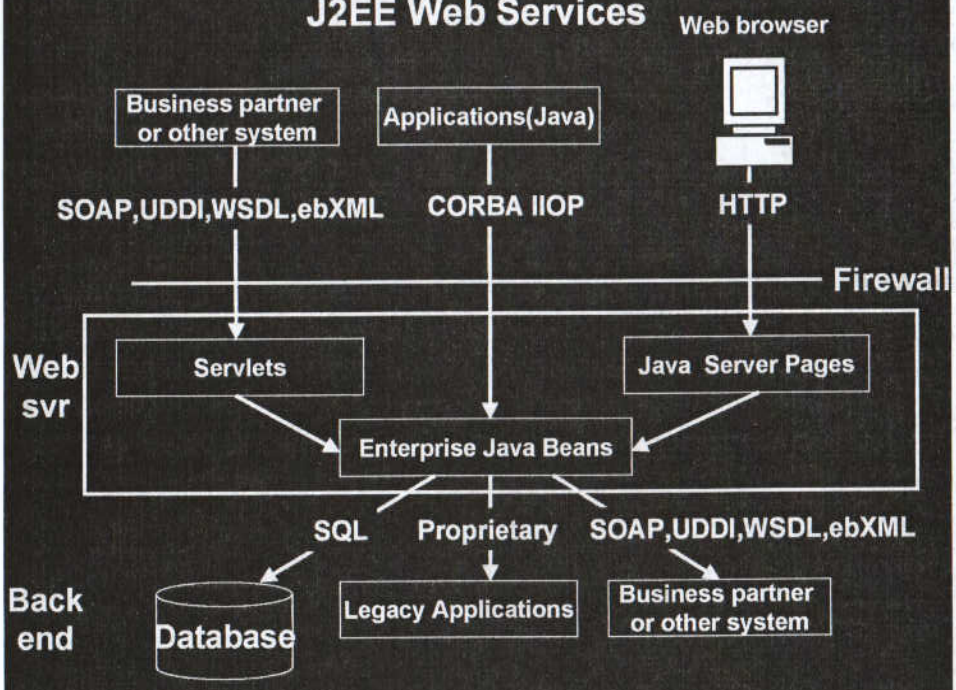
Web services

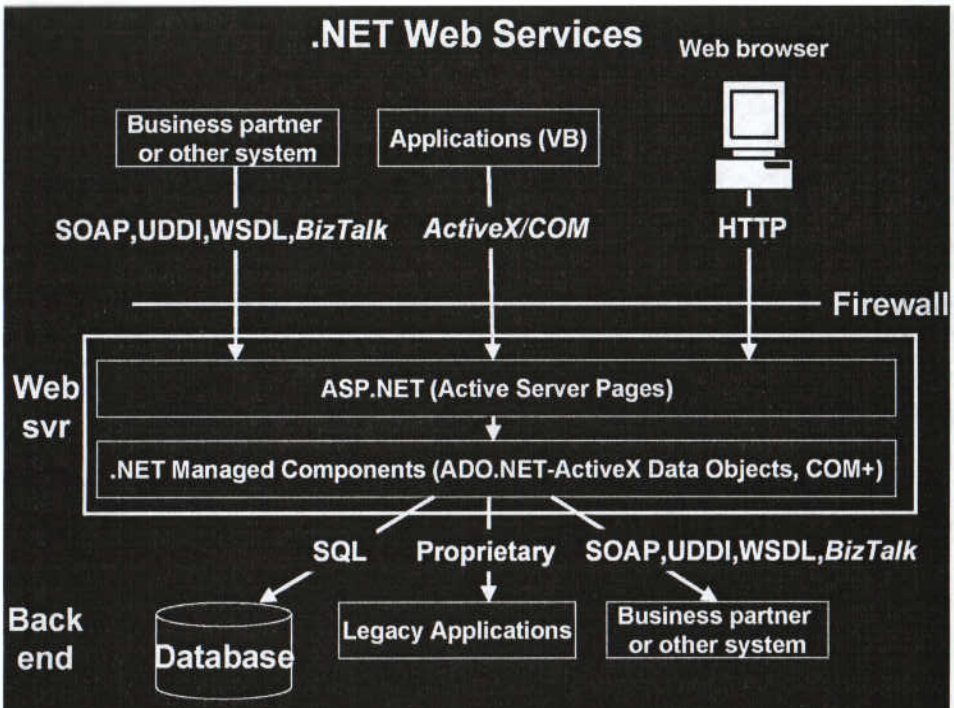
- XML allows heterogeneous environments to share information over the Web
- It now allows means to share process also.
- Keys are XML and HTTP/SOAP, each with wide acceptance.
- One way of building Web services:
  - Create and deploy Web service using programming language and Web server of your choice
  - Define Web service in WSDL
  - Register Web service in UDDI
  - User searches UDDI to find service and its parameters
  - User invokes service using SOAP, which has format for representing parameters and return values over HTTP

Web services, cont.

- This is ok for simple services
- Extended services need standards for common transactions (orders, catalogs, engineering design documents, etc.)
  - ebXML (eBusiness XML) is set of XML specifications
- Java and .NET both support SOAP, WDSL and UDDI
  - ebXML is evolving and will likely be supported by Java and .NET
- Java (J2EE) and .NET support Web services with tools for reliable, scalable, high-performance implementations
  - Implementation details vary; standards are the same
  - J2EE is a standard; many vendors build products
  - .NET is a product, built by Microsoft

J2EE Web Services





.NET and J2EE

Property	J2EE	.NET
Implementation	Standard	Product
Product vendor	Many (Sun, IBM, etc)	Microsoft
Web pages	Java Server Pages	Active Server Pages
Server components	Enterprise JavaBeans	.NET Components
Database access	JDBC and SQL/J	ADO.NET (on ODBC)
Middleware	SOAP, UDDI, WSDL	SOAP, UDDI, WSDL
Primary language	Java	C++, C#, VB
Portability	Across many vendors	Microsoft only

.NET and J2EE

- Differences between them are fairly small
- Some secondary differences:
  - ASP.NET is independent of client device, and user interfaces can work on PCs, handheld devices, etc. without change. JSP requires changes.
  - .NET Components are simpler than EJBs. If you don't need additional features, .NET is simpler. Otherwise you have to build the extra features yourself, which is more complex (such as nested transactions, state machines)
  - J2EE tools come from many vendors and don't interoperate as smoothly as .NET, which is all Microsoft
  - J2EE is more mature; .NET is new and a major change

J2EE Support for XML, UDDI, SOAP, WSDL

- JAXP: Java API for XML Parsing
  - Supports SOAP, UDDI, WSDL and ebXML
- JAX/RPC: Java API for XML Remote Procedure Calls
  - Supports program-to-program communication via SOAP
- JAXM: Java API for XML Messaging
  - Supports automated handling of XML messages using ebXML rules. Handles headers, errors, etc.
- ebXML: XML specifications, core components
  - E.g., dates, taxation, account, contract
  - See ebxml.org for more info
- JAXB: Java API for XML Binding
  - Converts XML to/from Java objects
- XSLT: XML Stylesheet Language Transformations
  - Converts from one XML document to another
- JCA: Java Connector Architecture
  - Includes adapters for legacy systems to/from Java, XML

.NET Support for XML, UDDI, SOAP, WSDL

- ASP.NET and ADO.NET offer roughly the same features for XML documents as J2EE
  - BizTalk rather than ebXML is the standard supported
    - BizTalk is Microsoft proprietary and appears to be losing the war
    - MS does not support ebXML
  - XSLT is supported
  - UDDI, WSDL, SOAP are supported directly
    - SOAP was proposed by Microsoft

XML

SADRŽAJ:

1. XML

XML (*eXtensible Markup Language*) predstavlja sintaksu za kreiranje "markserskih" jezika (*metadata*), jezika koji opisuju podatke. Nastao je iz SGML (*Standard Generalized Markup Language*) internacionalog standarda (ISO) za kreiranje i korišćenje formata dokumenata sa ciljem da razdvoji podatke od procesa. SGML predstavlja daleko kompleksniji "markerski" jezik, koji nije našao svoju upotrebu, pre svega na Internetu, zbog svoje složenosti.

Kada se XML pojavio 1998. godine izvršio je veliki uticaj kako na povezivanje među postojećim podacima tako i na dalji razvoj tehnologija. Mogućnosti ovog zapisa su se dosada posebno dobro iskoristile pri razvoju Bibliotečkih informacionih sistema, kao i Informacionih sistema državne uprave.

XML predstavlja podatke za opis podataka tj. sintaksu, u tekstualnom formatu. On je kreiran sa namerom da bude jednostavan za učenje, jevtin, brz i optimizovan za Internet. XML se naziva i *eXcellent Marketing Language* zbog sledećih karakteristika:

Univerzalni format podataka
XML omogućuje kreiranje sopstvenih formata podataka i njihovu razmenu preko postojećih mreža i aplikacija.

Integracija podataka
XML vrši jednostavnu integraciju podataka kod već postojećih aplikacija i platformi.

Prilagodiv
On je prilagodiv tj. razumljiv i za čoveka i za mašinu, primaoca i pošiljaoca, te predstavlja najupotrebljiviji standard za manipulaciju podataka i njihovu razmenu.

Svrha XML je da generiše sopstvene tagove, njihovo značenje i njihov prikaz. XML ne radi ništa, on samo nosi informacije okružene XML tagovima. Znači, XML definiše strukturu dokumenata. On menja način na koji *browser*-i prikazuju, organizuju i pretražuju informacije. XML se može shvatiti kao osiromašena verzija SGML-a i proširiv je za razliku od HTML.

XML uklanja granice HTML-ovog ograničenog skupa oznaka, dozvoljavajući ljudima koji se bave rezvojem da definišu neograničen broj oznaka za opis bilo kakavih podatak.

2. ISTORIJAT

Ideja o strukturiranju dokumenata bila je pokrenuta još 60-tih godina. Stukturiranje u standardizovani dokument ima za cilj da se olakša razmena i rukovanje podacima. Prvo je IBM je kreirao GML (*Generalized Markup Language*) za sopstvene potrebe. GML se koristio za izveštaje, knjige i druge dokumente iz izvornih datoteka. I druge organizacije su stvarale sopstvena rešenja strukturiranja informacija, ali ništa nije bilo napravljeno za opštu upotrebu.

Prva značajna standardizovana tehnologija strukturiranja informacija bio je SGML (*Standard Generalized Markup Language*), takođe iz IBM-a. SGML je dao način formatiranja i održavanja validnih dokumenata unutar IBM-a, a kasnije je proširen i prilagođen za upotrebu u raznim područjima industrije kao opšti standard.

Ipak, tek 1986. g. SGML je prihvaćen od ISO standarda. Iako velikih mogućnosti, SGML je vrlo složen i glomazan.

1989. g. Tim Berners-Lee i Andres Berglund, dva istraživača iz CERN-a, kreirali su jezik oznaka (*markup language*) za obeležavanje tehničkih dokumenata koji su se prenosili putem Interneta. Ovaj jezik se razvio kao pojednostavljena primjena SGML-a, nazvan je HTML, i postao standardni oblik prikaza informacija za WEB.

HTML je originalno bio zamišljen kao način predstavljanja statičkih informacija. Razvojem Web-a, dodajući interaktivnost u HTML, jezik oznaka za prezentaciju postao je razvijena softverska tehnologija. Deo HTML dokumenta danas čine različiti skriptni programi i Java apleti. Omogućeno je dinamičko serviranje informacija, ali one trebaju biti analizirane, sortirane, filtrirane i prilagođene korisniku.

3. XML FORMAT

Verovatno najočiglednije ograničenje HTML-a je njegov strogo definisan skup oznaka. 1996. g. članovi World Wide Web Consortiuma (*W3C*) su uočili tri značajne prednosti SGML-a u odnosu na HTML:

- proširivost,
- strukturiranost i
- validnost.

Okupili su tim SGML stručnjka koji su kreirali novi jezik oznaka s jezgrom SGML-a i jednostavnošću HTML-a. Nastao je eXtensible Markup Language (*XML*). Na sledećem primeru se lako uočava razlika:

HTML:
```
<p> Obrazovni računarski softver
<br> prof. dr. Dragica Radosav
<br> Obrazovni softver, Multimedija, Korisnički interfejs,
Provera znanja
```

XML:
```
<predmet>
    <naziv> Obrazovni računarski softver </naziv>
    <predavac> prof. dr. Dragica Radosav </predavac>
    <sadržaj> Obrazovni softver, Multimedija, Korisnički
     interfejs, Provera znanja </sadržaj>
</predmet>
```

Oba primera mogu biti jednako prikazana u pretraživaču, ali dok HTML govori samo kako podaci treba da se prikažu, XML nosi značenje podataka.

Da bi oblikovali prikaz podataka strukturiranih XML-om, potrebno je definisati stilski predložak, XSL (*eXstensible Stylesheet Language*) ili CSS (*Cascading Style Sheet*).

XSL definiše kako da se prikažu podaci u XML obliku, u HTML, pdf ili bilo kom drugom formatu. Različiti stilski predlošci se mogu koristiti za prikaz istih podataka na različite načine, za različite korisnike.

S XML-om i ne-programeri dobijaju mogućnost da kreiraju svoje podatake, oblikuju ih i upravljaju nad njima, jer se radi o običnom tekstualnom zapisu.

4. NAMENA [1]

XML je rešenje za situacije sa sledećim karakteristikama:

1. Informacije su složenog oblika. Npr. karton pacijenata u doktorskoj ordinaciji može sadržati zapise o nekoliko poseta doktoru, do nekoliko stotina poseta.
2. Pojedina polja su velika i složena, a kako u bazi podataka svako polje mora biti jednake veličine imali bi veliki gubitak memorijskog prostora. Primer za takvo polje u kartonu pacijenta je zapis doktora o poseti, koji može biti u nekoliko reči, pa do cele stranice.
3. Brzina pretraživanja nije važna. Baze podataka imaju optimizirane programe za pretraživanje, dok se XML datoteka pretražuje kao string.

[1] Materijal je preuzet sa www.svezaweb.dzaba.com

4. Tip podataka nije važan, jer XML datoteka sadrži samo nizove znakova. Prevođenje niza znakova u broj je vremenski zahtevno, pa ako nad podacima želimo da obavljamo zahtevne matematičke proračune, bolje je da aplikacija ima pristup već odgovarajućem obliku podataka.
5. Količina informacija je mala, ali postoji potreba za povećanjem. Ako se ima manje od 100.000 zapisa i frekvencija pristupa podacima nije veća od 1 u 10 sek, XML je odličan izbor, neće se primetiti razlika u brzini u odnosu na bazu podataka.

XML je pogodan da opiše strukturu, integriše protokole između aplikacija, da razmenjuje podatke. XML je skup pravila koja omogućavaju kreiranje tekstalnog formata koji opisuje strukturu podataka (kao što su adresari, konfiguracioni parametri, finasijske transakcije itd.)

XML opisuju podatke u tekstualnom formatu te omogućuje razmenu podataka nezavisno od sistema i formata podataka i predstavlja budućnost mrežnog programiranja. Veliku primenu ima u razmeni podataka, pogotov za komunikaciju *client-server* preko Interneta.

Pogledajmo gde je sve XML našao svoju primenu:

XML for Content Providers

Istoj informaciji može se pristupati i čitati na različitim jezicima. Različit prikaz istih podataka mogu se prezentovati različitim korisnicima. Svaki XML dokument može da sadrži opis gramatike ili sintakse kako bi se moglo proveriti i ispravnost sadržaja.

XML for Content and knowledge management

Pretraživanje, indeksiranje i lokacija podataka postaje jednostavnija pošto XML nosi informaciju o sadržaju, on je samo opisujući dokument. Trasformacija podataka iz XML omogućava prikaz na različite medije (Web, CD ROM, papir) bez nepotrebnih modifikacija i dupliranja sadržaja.

XML for Content Aggregation

XML obezbeđuje da se informacije sa različitih mesta integrišu na jednom mestu i da se one prikupljaju na osnovu akcija krajnjeg korisnika. XML na taj način obezbeđuje vezu B2C preko B2B veze.

XML for Electronic Document Interchange

XML omogućava kreiranje strukture za razmenu informacija kao i da objedini postojeće protokole i standarde.

XML and E-Commerce

XML obezbeđuje sintaksu da indetifikuje svaku informaciju potrebnu za kompletnu transakciju. Drugi spoj je poverenje, jer XML omogućuje da se informacija o učesnicima u transakciji nosi zajedno sa transakcijom. Da bi se pratila promena tržišta potrebno je mnogo manje vremena i novca sa XML-om.

XML for Design

Scalable Vector Graphic (SVG) predstavlja jezik za opis dvodimenzionalnih vektora pomoću XML-a. SVG-a omogućuje da svaki korisnik u realnom vremnu pristupa slici u bilo koje vrme i sa bilo kojim uređajem, sa bilo kog mesta.

XML omogućuje da strukturni podaci iz različitih izvora jednostavno kombinuju. XML dokument kao poruka je samobjašnjiv skup podataka, jer pored samih podataka koji su predmet poruke, XML dokument sadrži i meta podatke pomoću kojih se ti podaci mogu interpretirati. Programski agenti, mogu se koristiti da integrišu podatke u srednjem sloju servera iz baze za druge aplikacije. Ovi podaci mogu da se prenose klijentima ili drugim serverima za dalju agregaciju, procesiranje ili distribuciju.

Prvi Web browser koji omogućuje pregled XML -a je Internet Explorer 5.0+, a Netscape podržava XML specifikaciju od verzije 6 svog browsera.

Fleksibilnost XML-a omogućava da se opišu podaci sadržani u širokom krugu različitih aplikacija, od opisa web strane do polja baze podataka.

5. STANDARD [2]

Mogućnost da se podaci odvoje od procesa predstavlja ključ uspeha XML-a. XML je otvoreni standard, te omogućava da XML funkcioniše na bilo kojoj platformi sa bilo kojim programskim jezikom. Veliki broj programskih jezika omogućuje rad sa XML-om poput Jave, MS Visual Basica, MS Visual C++, Perl, Cobol i C#.

Pomoću XML mogu se napisati i novi jezici. WML (*Wireless Markup Language*), koristi se za kreiranje Internet aplikacija u mobilnim telefonima, i napisan je u XML-u.

XML je nezampamćenom brzinom postao standard, zbog svoje jednostavnosti.

XML 1.0 , je usvojen kao W3C Recommendation u februaru 1998. i predstavlja sintaksu definisanu po W3C specifikaciji.

Takođe W3C omogućava progrmiranje efikasinijim, kreirajući familiju tehnologija koje podržavaju XML kao što su:

XML Schema, takođe predstavlja XML dokument, omogućava modularnost jer kreira strukturu i opis samih XML dokumenta. Jednostavno kombinuje više različitih šema koje pokriva i sjedinjava strukturu dokumenta. Znači, nasleđuj se pravila sa drugih šema. Nastale su kao alternativna zamena DTD-a (*Document Type Definition*), pošto DTD nije bio XML dokument. XMl šeme obezbeđuju podršku za standardne tipove podataka kao što su broj,datum,... i omogućuju definisanje novih tipova.

Namespace , eleminiše konfuziju prilikom kombinovanja više šema u jednom XML dokumentu. On upućuje XML procesor da pronađe strukturna pravila (definisanih u šemi) koji se primenjuju na sam dokument. U slučaju kada u dokumentu koristimo dva eksterna dokumenta koji poseduju isti naziv a različito značenje elemenata koristimo namespace, kako bi ukazali na koje se elemente odnose i koje značnje poseduju. Namespace ukazuje na šemu koja sadrži informacije o dokumentu koji se koristi.

XSLT/xPath vrši transformaciju sadržaja XML dokumenta u bilo šta, najčešće HTML, omogućavajući na taj način razdvajanje podataka od prezentacije.

---

[2] Materijal je preuzet sa *www.svezaweb.dzaba.com*

6. OSNOVE XML SINTAKSE[3]

XML dokumenti se sastoje od znakovnih podataka (*character data*) i oznaka (*markup*). Osnovne komponente koje su definirane XML 1.0 specifikacijom su:

- oznaka elementa (*element tag*)
- naredba (*processing instruction*)
- DTD (*document type declaration*)
- entitet (*entity reference*)
- komentar

Ovo su delovi dokumenta koje treba razumeti XML procesor. Delovi koji se ne nalaze između oznaka su podaci namenjeni ljudskom razumevanju.

XML jednsostavno omogućuje kompjuteru da generiše, čita i proverava validnost podataka. XML je prihvatljiv za svakog čoveka.

XML I HTML koriste <,> za kreiranje elementa i atributa strukture. Znači, sve što je unutar < i > se podrzumeva kao element, koji se parsira (prevodu u odgovarajuću akciju) a sve između <> i </> se tretira kao običan tekst tj. sadržaj. Svaki element koji se otvori mora biti zatvoren. Elementi se ne mogu preklapati, jer u suprotnom došlo bi do zaustavljanja prikaza dokumenta za razliku od HTML browser-a, koji će jednostavno preskočiti tu grešku.

Primer_1 : Ispravno pisanje XML dokumenta

Ispravno	Ne ispravno	Ne ispravno
<RODITELJ> <DETE> Sadržaj.Sadržaj. </DETE> </RODITELJ>	<RODITELJ> <DETE> Sadržaj.Sadržaj. </RODITELJ> </DETE>	<DETE> <RODITELJ> Sadržaj.Sadržaj. </DETE> </RODITELJ>

Prvi tag u XML dokumentu se naziva "root" element, on je roditelj za sve ostale elemente. Svi XML dokumenti moraju da imaju "root" element, tj. glavni tag koji definiše sam XML dokument. Svi ostali tagovi moraju biti u okviru "root" elementa. Ostali elementi mogu da imaju svoju decu, deca moraju biti ispravno ugnježđeni sa svojim roditeljima, kao u prethodnom primeru.

[3] Materijal je preuzet sa www.svezaweb.dzaba.com

Nazivi elementa su CASE SENSITIVE tj. osetljivi su na mala i velika slova, tag <poruka> je raličit od taga <Poruka>.

Ako element nesadrži nikakav sadržaj mora početi sa < i završiti sa /> poput <EMPTY/>.

Primer_2: Jednostvan XML dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<!--trikovi last updated 2001-01-01-->
<trikovi>
 <css>CSS <quote>level 1</quote></css>
 <brovseri><quote>IE,NS</quote></brovseri>
 <uspesno/>
</trikovi>
```

Po XML deklaraciji koju sprovodi W3C svaki XML dokument započinje sa: <?xml ...?>. Koji u suštini definiše verziju XML u kome je napisan. Deklaracija može da sadrži i encoding atribut za definisanje kodnog rasporeda i standalone deklaraciju koja govori da dokument zavisi od informacija iz eksternog izvora kao npr. eksterni DTD. XML deklaracija nema završni tag, zato što to nije deo samog XML dokumenta niti je XML element, pa ne treba da ima završni tag.

Sledeća linija tj. <trikovi> opisuje "root element" dokumenta tj. on govori da je ovaj dokument nosi informacije o trikovima. Ostale linije predstavljaju decu za element <trikovi> a suvišan prazan prostor "WHITE SPACE", XML preskače kao u HTML-u.

Prazni elementi poput u primeru: (<uspesno/>) jednostavno mapiraju gde se element nalazi. Prazni elementi mogu da budu zadati i u sledećoj sintaksi <uspesno></uspesno>.

Nema ništa specijalno u vezi XML, to je jednostavni tekstualni dokument kome su dodati XML tagovi. Programi koji mogu da rade sa običnim tekstom mogu i sa XML-om. U njima XML tagovi su vidljivi i neće biti obrađivani. Dok u aplikacijama koje rade sa XML-om, tagovi mogu da se obrađuju, i mogu da imaju značenje zavisno od prirode programa.

Najvažniji delovi XML dokumenta predstavljaju elementi i atributi:

Elementi

Elementi određuju prirodu sadržaja kojeg opkružuju. XML elementi nisu definisani, vi morate definisati sopstvene XML elemente. Pomoću njih se određuje strktura dokumenta i omogućava njihovo programiranje i vizualnu predstavu pomoću stilova. Neki elementi mogu biti prazni i nemoraju da poseduju sadržaj. Svaki element započinje sa početnim tagom <element>, i završava se sa krjanjim tagom, </element>.

Pogledajmo primer XML-a:	Ukoliko imamo program koji može da obradi elemente <od>, <za>, i <telo> iz XML dokumenta da proizvede izlaz kao:
<poruka id="1"> <od>Meni</od> <za>Njega</za> <telo>Poseti me opet!</telo> </poruka>	1.PORUKA Od: Mene Za: Njega Poseti me opet!

Jednostavnim dodavanjem novog elementa u XML neće narušiti njegovu funkcionalnost.

Program će i dalje biti u stanju da obrađuje elemente koje smo prethodno definisali i ako smo dodali nove elemente. XML JE PROŠIRIV!

Atributi

Atributi predstavljaju parove ime-vrednost koji se nalaze unutar početnog elementa, odmah posle imena elementa. Vrednosti atributa moraju biti uvek pod navodnicima. Najčešće se upotrebljavaju dvostruki navodnici, zato što nekad i same vrednosti atributa sadrže navodnike pa u tim slučajevima koristimo jednostruke navodnike. Primer:

```
<city name="NEW YORK 'BIG APPLE'">23.000</city>
```

Primer sadrži element sa nazivom city koji opet sadrži atribut sa nazivom name koji ima vrednost NEW YORK 'BIG APPLE'.

Atributi najčešće obezbeđuju informacije koje nisu deo podataka na primer atribut ID koji smo koristili u prethodnim primerima nije deo sadržaja elemenata ali programima koji obrađuju te elemente su mnogo važni.

Podaci mogu da budu smešteni i u atribute ili elemente. Nema neko generalno pravilo ali treba izbegavati da se koriste atributi za smeštanje

podataka. Naročito kada dodje do promene elemenata teško je menjati i atribute.

Evo nekoliko razloga za izbegavanja atributa:

- Atributi se teže proširuju i menjaju
- Atributi ne mogu da opišu strukturu
- Atributi se mnogo teže manipuliše
- Vrednosti atributa nije lako proveriti

Koristite elemente da opište podatke dok atribute samo kad treba ali samo kad treba da nose informaciju koja nije povezana sa samim podacima. Najčešći primer upotrebe atributa je korišćenje jednostavnog identifikator npr. ID koji nema veze sa podacima, ali omogućava lakši pristup elementima kao u HTML kada koristimo ID ili NAME atribute.

7. XML DOKUMENT[4]

Između elemenata i njihovog sadržaja postoji veza. Zamislimo da nam je potreban XML dokument koji će da opiše npr. sajamske manifestacije. Znam i meni je to strano! Ukoliko si uspeo da pročitaš dovde super je, a sad idemo dalje. Spisak sajmova za našu zemlju može da izgleda ovako:

SPISAK SAJMOVA U JUGOSLAVIJA ZA 2001.

Beogradski sajam
12.06 - 12.06
XX MEDJUNARODNI SAJAM VINA, OSTALIH PIĆA I
PREHRAMBRENO-POLJOPRIVREDNIH PROIZVODA - "MESVIPP"
.....
Novosadski sajam
12.09 - 12.09
XII MEDJUNARODNI SAJAM POLJOPRIVREDE
.....

Na osnovu toga možemo napraviti i XML naprimer ovako :

```
<fairs id="000004">
  <country>YU</country>
  <foryear>2001</foryear>

  <fairsrs id="0000001" name="Beogradski Sajam">
```

[4] Materijal je preuzet sa *www.svezaweb.dzaba.com*

```
    <fair id="000001">
     <fairnumber>XX</fairnumber>
     <fairname>MEDJUNARODNI SAJAM VINA, OSTALIH PIĆA I
PREHRAMBRENO-POLJOPRIVREDNIH PROIZVODA
       <fairacronym>MESVIPP<fairacronym>
     </fairname>
     <fairlink></fairlink>
     <fairdate>
      <datestart>12.06</datestart>
      <dateend>16.06</dateend>
     </fairdate>
    </fair>
  </fairorg>
  <fairorg id="0000001" name="Novosadski Sajam">
    <fair id="000001">
     <fairnumber>XII</fairnumber>
     <fairname>MEDJUNARODNI SAJAM POLJOPRIVREDE
      <fairacronym>MESP<fairacronym>
     </fairname>
     <fairlink></fairlink>
     <fairdate>
      <datestart>12.09</datestart>
      <dateend>16.09</dateend>
     </fairdate>
    </fair>
  </fairorg>
  ...

</fairs>
```

Root element je fairs, dok su country i fairorg deca od fairs i predstavljaju sestre (*siblings*) zato što imaju istog roditelja. Kao što vidimo iz prethodnog primera, elementi mogu da imaju različite sadržaje. Sam element može imati druge element ili da sadrži i sadržaj i elemente ili da bude prazan kao i da sadrži atribute.

Takođe u prethodnom primeru vidimo da element fairorg ima atribute id i name.

Prilikom imenovanja elemeta moramo se voditi sledećim pravilima:

- Ime može da sadrži slova, cifre i ostale znakove
- Imena nesmeju da započinju sa brojem ili drugim znakovima
- Imena netreba da započinju sa slovima xml (ili XML ili Xml ...)
- Imena nesmeju da sadrže prazan razmak

Prilikom kreiranja imena vodite se sledećim pravilima:

- Sve može da se koristi u nazivu elementa ali naziv treba da opiše sadržaj koji nosi. Primer: <fair_org>, <fair_name>.
- Izbegavajte znakove "-" i "." u imenima, jer podsecaju na kod u nekom programskom jeziku.
- Nazivi mogu da budu veliki koliko god želite, ali trudite se da smanjite na najmanju moguću meru umesto <fair_date_start> možete koristiti jednostavnije <date_start>.
- Znak ":" se ne koristi pošto je rezervisan za namespaces, za refernciranje šema pomoću kojih je opisan sam XML dokument.

XML dokumenta najčešće oslikavaju baze podataka, pa koristite nazive polja u bazi za nazive elemenata. Dobra prakasa je korišćenje pravila kod kreiranja imena u bazi podataka i kod XML dokumenata.

I naša slova mogu da se koriste poput ćčđšž ali ne bih preporučio, koristite engleski jezik za imenovanje XML elemenata.

Referenciranje Entiteta

U XML entiteti se koriste da reprezentuju specijalne znake, takođe se koriste da zamene često korišćene tekstove koji se ponavljaju. Naprimer znak, < , označava početni tag elementa ili krajnji, da bi se mogao isti taj znak koristi u dokumentu potrebno je alternativno rešenje za njihovu reperezentaciju a to su entiteti.

Svaki entitet mora imati jedinstveno ime. Definisanje sopstvenih enititeta možete pronaći na deklaraciji entiteta. Entiteti se koriste prostim referenciranjem na njihovo ime, oni započinju sa & a završavaju se sa ;.

Primer: lt enitet ubacuje znak < u dokument. Tako da string <element> može se predstaviti u XML dokumentu kao <element>.

Referenciranjem karaktera, u mogućnosti ste da ubacite Unicode karaktere u dokument pričemu se on ne unosi direktno preko tastature.

Postoje dve forme referenciranja karaktera: decimalna refernca, ℞, i

heksadecimalna referenca, ℞. I jedni i drugi referncirauju ka karakteru čiji je broj u Unicodu karakter setu: U+211E.

Komentari
Komentari su kao u HTML-u, počinju sa <!-- i završavaju sa -->. Oni sadrže sve podatke sem -- i postvljaju se na bilo kojem mestu u dokumentu. Komentari nisu deo tekstualnog sadržaja XML dokumenta.

8. LOKALNA MANIPLUACIJA[5]

Pošto se podaci dostave klijentu, mogu se lako obrađivati lokalno pa se svaka manipulacija sa podacima prebacuje kod klijenta. Čime se poboljšavaju performanse servera pošto nije potrebno skakati na server pri svakoj obradi podataka, pošto se svako obrada premesta kod klijenta. Na ovaj način stvoren je moćan mehanizan sa interakciju sa korisnikom u offline režimu.

DOM (*Document Object Model*) obezbeđuje interfejs za učitavanje, pristup i manipulaciju XML dokumentom. W3C DOM Level 1 Specification, sadrži standardni set osobina, metoda i događaja koje treba da sadrži objekat koji vrši manipulaciju sa XML-om. XML DOM omogućava jednostavan rad sa XML-om pomoću raznih programskih jezika.

MSXML parser
Micorosoft-ova implementacija DOM-a podržava W3C standard ali je i proširen kako bi se obezbedio lakši rad sa XML dokumentom. Microsfot isporučuje svoj XML DOM objekat u okviru svog XML parsera MSXML u Internet Explorer 5, MS Office 2000, Win 2000, Win XP ...
MSXML DOM obezbeđuje reprezentaciju kompletnog XML u memoriji sa strukturom drveta, omogućavajući slučajn pristup sadržaju celog XML dokumenta.

Parser je neutralan u odnosu na programske jezike koje podržava: JavaScript, VBScript, Perl, VB, Java, C++ ...

Parser omogućava skriptovanje i kod klijenta i na serveru u okviru ASP fajlova. MSXML parser je COM (*Component Object Model*) objekat, tako da se može korstiti u bilo kojoj aplikaciji koja koristi COM. Može mu se pristupiti kao *ActiveX* kontroli u okviru browser-a.

XML DOM se neprekidno unapređuje, pa na vašoj mašini možete naći mnogo verzija XML parsera tipa *xmldom.dll* na Vašem računaru. Ako koristite MSXML.DOMDocument onda koristite verziju 2.5 dok

[5] Materijal je preuzet sa www.svezaweb.dzaba.com

MSXML.DOMDocument30 omogućava pristup verziji 3. Kada instalirate svežu verziju MSXML parsera, potrebno je da startujete program xmlinst.exe kako bi se registrovala zadnja verzija parsera.

Ako koristite JavaScript u IE 5.0, možete kreirati XML objekat:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
```

Ako koristite VBScript:

```
set xmlDoc = CreateObject("Microsoft.XMLDOM")
```

Ukoliko koristite VBScript u okviru Active Server Page (ASP):

```
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
```

Kada MSXML parser učita XML dokument u DOM, on čita od početka do kraja kreirajući logičan model strukture i sadržaja XML dokumenta.

```
<script>
//Kreiranje instace na XML parser
var objDoc = new ActiveXObject("Microsoft.XMLDOM");

//Ansihrono ucitavanje
objDoc.async = false;

//Ucitavanje XML-a iz stringa u parser
objDoc.loadXML("<msg>ZDRAVO SVETE</msg>")

//Provera validnosti XML-a
if (objDoc.parseError.errorCode==0){
  //Pristup root elementu XML-a
  var root = objDoc.documentElement;

  //Prikaz vrednosti root elementa
  alert(root.text);
}
else {

 //Ukoliko je doslo do greske prikazi ih
```

```
alert("Greska:"+objDoc.parseError.reason);
}
</script>
```

Na sličan način može se učitati iz eksternog fajla XML:

```
<script>
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";

xmlDoc.load("myXML.xml");

// ..... procesiranje

</script>
```

Iz prethodna dva primera vidimo da se XML dokument može učitati na dva načina iz stringa i iz fajla refernciran URL-om.

async - Ova osobina omogućava setovanje pristupa XML dokumentu. Ako se setuje na *false* tada se ne može primeniti ni jedna akcija nad XML dokumentom dok se ne učita. Njena inicijalna vrednost je true.
Pri većim XML dokumentim potrebno je asihrono učitavanje pa se primenom onreadystatechange obrade dogadjaja može proveravati dali se dokument učitao ili ne primenom readyState osobine koja proverava status učitavanja.

Vrednosti koje vraća readyState osobina:
LOADING
XML se učitava

LOADED
XML je učitan i započeto je parsiranje

INTERACTIVE
Neki delovi su parsirani i objektni model XML-a je sada na raspolaganju

COMPLETED
Dokument je obrađen uspešno ili neuspešno

```
<script>
function handleReadyState()
{
if (xmlDoc.readyState == 4)
{
//... Procesiranje ....

//Pristup root elementu XML-a
var root = objDoc.documentElement;

//Prikaz vrednosti root elementa
alert(root.text);

//Prikaz naziva root elementa
alert(root.nodeName);

}
}

var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");

xmlDoc.load("myXML.xml");

xmlDoc.onreadystatechange = handleReadyState;

</script>
```

Validnost XML dokumenta provervamo pomoću parseError objekta koji ima sledeće osobine :

errorCode
Broj greške u decimalnom formatu, ukoliko je 0 nema greške
url
URL (Link) ka XML fajlu gde se desila greška
reason
Razlog greske
srcText
Pun tekst linije gde se desila greška
line
Broj linije gde se desila greška
linepos
Pozicija karaktera u liniji gde se greška javila
filepos
Absolutna pozicija karaktera u fajlu gde se greška javila

Kada porverimo validnost dokumenta pomoću parseError objekta možemo da manipulišemo podacima.

Osobine pomoću kojih možemo pročitati vrednost i naziv XML elementa su text i nodeName. Ove dve osbine rade na bilo kom elementu XML dokumenta.
Atributi nekog elementa se smeštaju u kolekciju - niz koji je indeksiran prema nazivu atributa.

```
aid = root.attributes.getNamedItem("id");
if ( aid != null ) resultId = aid.nodeValue;
```

Pristup elementima XML-a

```
<xml version="1.0"><root><one>1</one><two>2</two></root>

ele = root.childNodes;

for ( i = 0; i < ele.length; i++)
{
  result += ele.item().text;
}
```

Vrednost promenljive result je 12;

Postoji i alternativni metod za pristup elemntima, Microsoft je implementirao dva metoda:
selectNodes(query)
Vraca listu nodova refrenciranih pomocu atributa query.
selectSingleNode(query)
Vraca jedan nod referenciran pomocu atributa query

Query atribut sadrži putanju ka elementu u XML dokumentu, definisan po *W3C xPath* specifikaciji, ako je XML dokument glasio:

```
<xml version="1.0"><root><one>1</one><two>2</two></root>
```

Tada da bi pristupili elementu two u XML dokumentu koristimo upit: //root/two

```
result = root.selectSingleNode("two").text;
//ili
result = xmlDoc.selectSingleNode("//root/two").text;
```

Ukoliko koristite ASP.NET i C#, obrada XML-a na serveru isto koristi DOM:

```
using System.Xml;
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load("myfile.xml");
XmlNode rootNode = xmlDoc.DocumentElement;
XmlNodeList nodeList = rootNode.ChildNodes;
for (int i=0;i<nodeList.Count;i++)
{ ...procesiranje nodova... }
```

Data Islands

U Microsoft Internet Explorer 5.0 predstavljena su ostrva podataka koja omogućuju ubacivanje XML u HTML, W3C očekuje u sledećoj specifikaciji HTML da uključi ubacivanje XML u HTML dokument. XML tag je prisutan u DOM-u. Prisutupom root elementu pomoću DOM-a u XML tagu omogućeno je upravljanje o im XML-om.

```
<head>
<xml id="xmlID">
<XMLDATA>
<DATA>TEXT</DATA>
</XMLDATA>
</xml>
<script>
var objXml = xmlID.XMLDocument;
var data = objXml.documentElement.childNodes.item(1).text;
alert(data);
</script>
</head>

//ili
<XML SRC="http://localhost/xmlFile.xml"></XML>
```

MS XMLHTTP

XMLHTTP objekat kreiran je sa namerom da se obezbedi pristup sa klijentske strane serveru pomoću XML-a kroz HTTP protokol. Na taj način obezneđena je tehnologija koja omogućava mikro izmene, tj. prilikom

izmene podataka u XML-u, nije potrebno da se cela struktura XML-a šalje na server da bi se registorvala izmena.

Dovoljno je poslati samo podatak ili manji skup podataka koji je izmenjen. Pri tome izmenjeni elementi ostaju kod klijenta bez potrebe da se ponovo zahtevaju novi podaci sa servera, time se smanjuje saobraćaj na serveru.

```
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");

xmlhttp.Open("POST", "https:\\www.24x7.com.mk\service\get.asp", false);

xmlhttp.Send(objXml);

objXml = xmlhttp.responseXML;

alert (objXml.documentElement.text);
```

a ASP strana na serveru https:\\www.24x7.com.mk\service\get.asp

```
<%@ language=javascript %>
<%
Response.Expires = -1000;
// Ucitava XML
var doc = Server.CreateObject("Msxml2.DOMDocument");
doc.load(Request);
var result = Server.CreateObject("Msxml2.DOMDocument");
// Procesiranje i vracanje obradjenog XML dokumenta
Response.ContentType = "text/xml";
result.save(Response);
%>
```

9. TRANSFORMACIJA

XSLT - eXtensible Stylesheet Language: Transforamtions je jezik za transformaciju XML, on transformiše strukturu XML dokumenta. XSLT sadrži komande kao kod tradicionalnih programskih jezika kao što su promenljive, fukncije, iteracije i provera uslova.

XPath je jezik za pristup strukturi XML dokumenta, tj. predstavlja izraze koji mogu da sadrže putanje ka elementima u XML dokumentu, pozive funkcija, refrence na promenljive, poređenja, matematičke operacije itd. Koristi se zajedno sa XSLT da se odabrao jedan ili skup elementa iz XML koji se želi transformisati.

XSLT omogućava:

- Razdvajanja sadržaja od prezentacije
  Razdvajanje vršimo kao bi mogli da iste informacije prikažemo na različitim uređajima, u različitim bojama, različitim oblicima itd.. Najčešće se upotrebljava za transformaciju XML u HTML.
- Razmenu podataka između aplikacija
  XSLT jednostvno trasformiše i XML u XML, kako bi uskladili protokole i integrisali različita rešenja. XML možemo da transformišemo u bilo šta.

10. MICROSOFT OFFICE INFOPATH 2003[6]

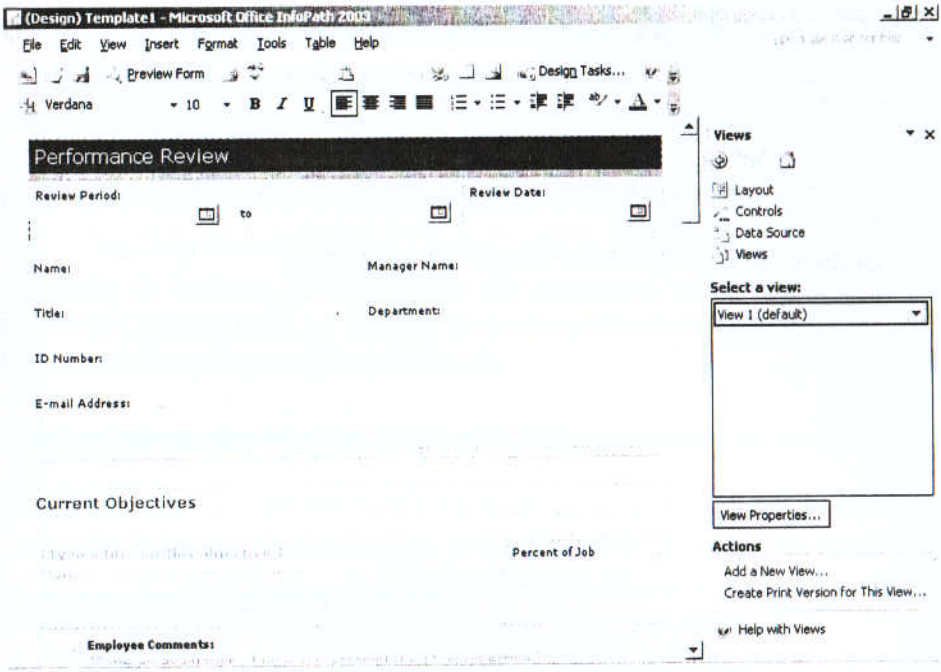
Ova aplikacija je napravljena na bazi XML zapisa, koristi forme koje su veoma pogodne za korišćenje i modifikaciju. Može se koristiti za pravljenje obrazaca, odnosno poluobrazaca, tj. obrazaca koje korisnik lako može da modifikuje pri upotrebi.

Kreiranje formi je veoma jednostavno u poređenju sa pisanjem tagova u nekom xml editoru, jer je sve napravljeno na principu korišćenja gotovih kontrola. Takođe veliko olakšanje u samom prikazu napravljne forme, a na kraju i gotovog dokumenta ogleda se u tome što ova aplikacija sama u pozadini pravi xsl fajl, koji za korisnika više ne predstavlja posao.

Na sledećem primeru (Slika 1) se vidi karakterističan prozor Microsoft Office InfoPath 2003. Globalno posmatrano pravljenje forme je podeljeno u četiri pomoćna dela: Layout, Controls, Data Source i View.

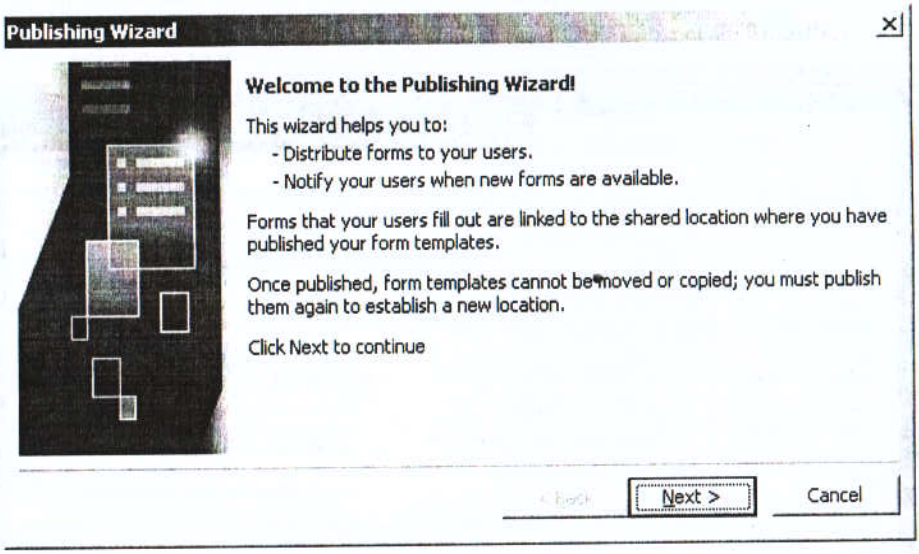
---

[6] *Deo seminarskog rada Jovanović Višnje, broj indeksa 49/02-M, urađenog pod mentorstvom Prof.dr Dragice Radosav, iz predmeta Informatika, na posle-diplomskim studijama na Tehničkom fakultetu u Zrenjaninu*

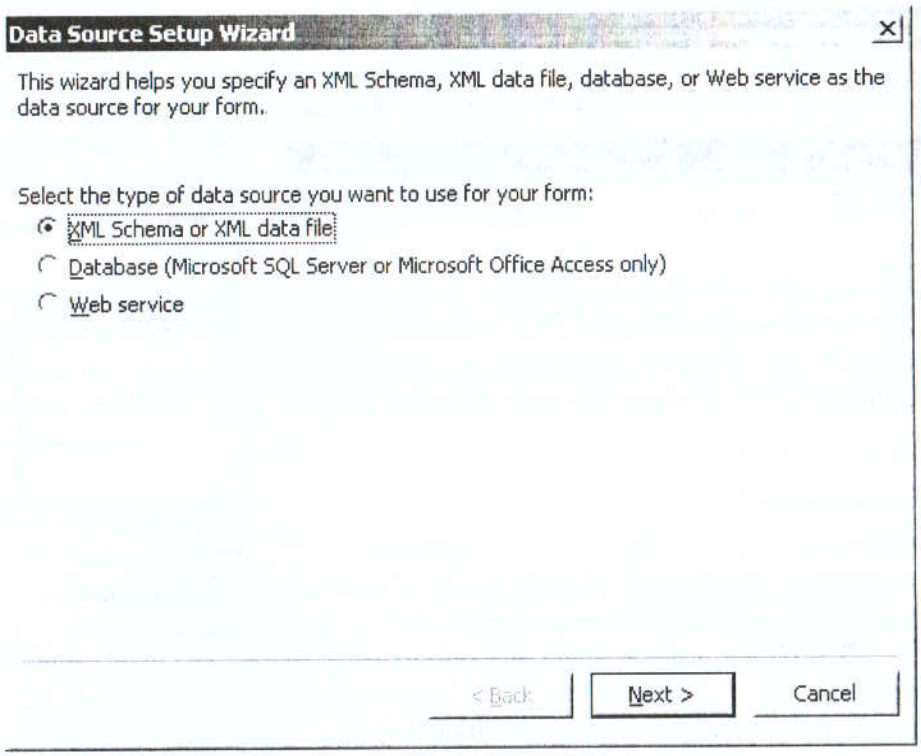


Slika 1.

Način na koji se pravi generisan xml zapis je pomoću čarobnjaka, što je interesantno posmatrati jer se radi u dva koraka (Slika 2 i Slika 3).

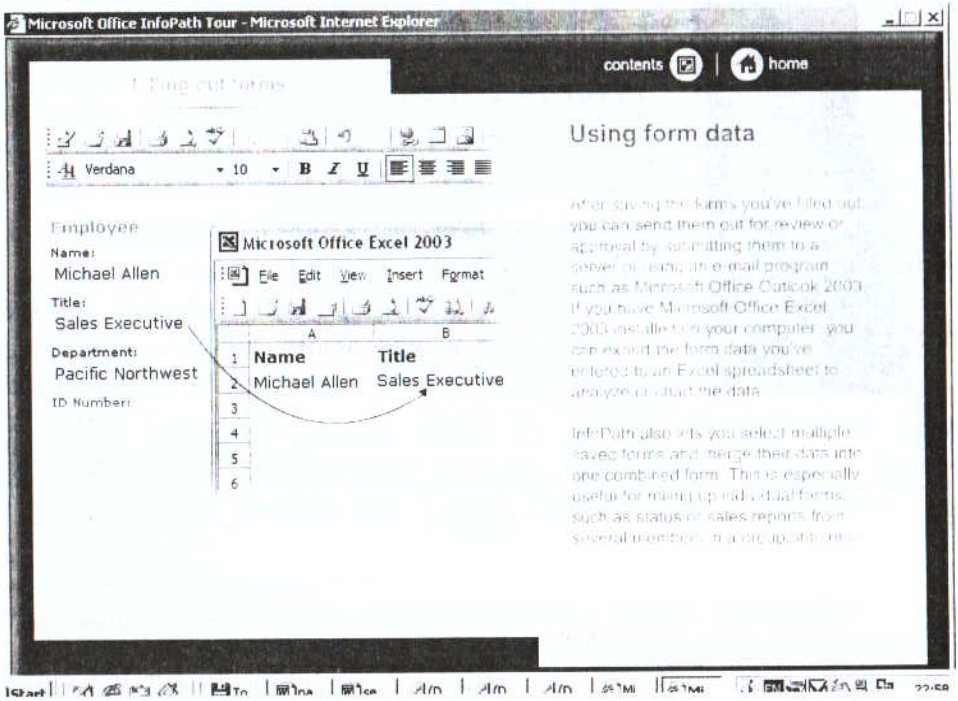


Slika 2



Slika 3

Povezivanje podataka iz drugih Microsotf Office aplikacija je takođe veoma jednostavno – na primer iz Microsoft Excel-a (Slika 4).

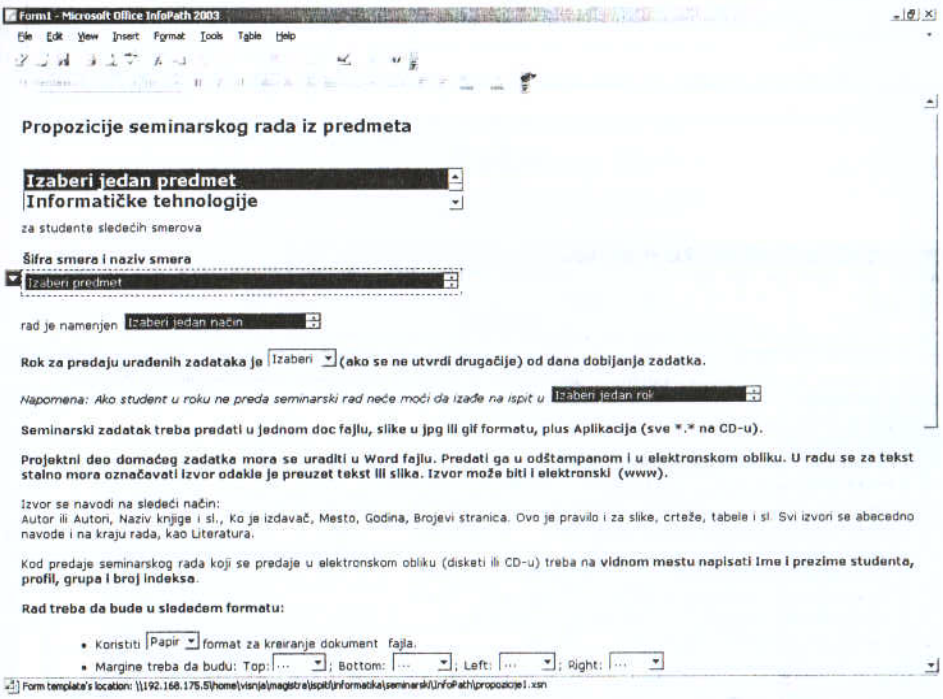


Slika 4

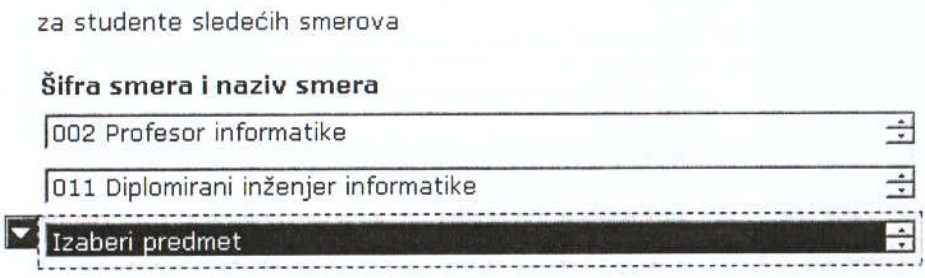
Dizajniranje formi za Propozicije seminarskih radova

Forma koja je napravljena služi za lako pravljenje, ustvari odabiranje propozija za seminarske radove iz konkretnih predmeta. Ono što je za xml tehnologiju poseban kuriozitet je neograničena mogućnost pozivanja, tj. ponavljanja prikaza dela ili celine određene komponente (tabele, forme ...).

Na ovoj formi (Slika 5) se npr. više puta može pozivati smer studija (za slučaj da izabrani predmet sluša više smerova), na taj način što se levim ili desnim klikom miša na strelicu levo od combo box-a insertuje ceo combo box iznad ili ispod postojećeg (Slika 6).



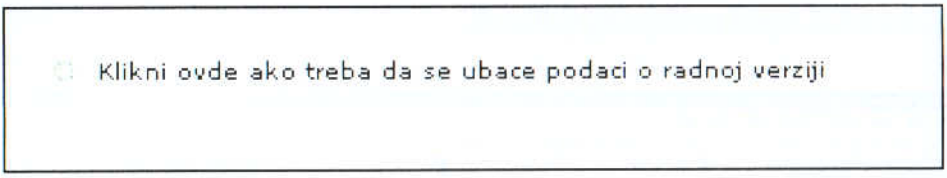
Slika 5



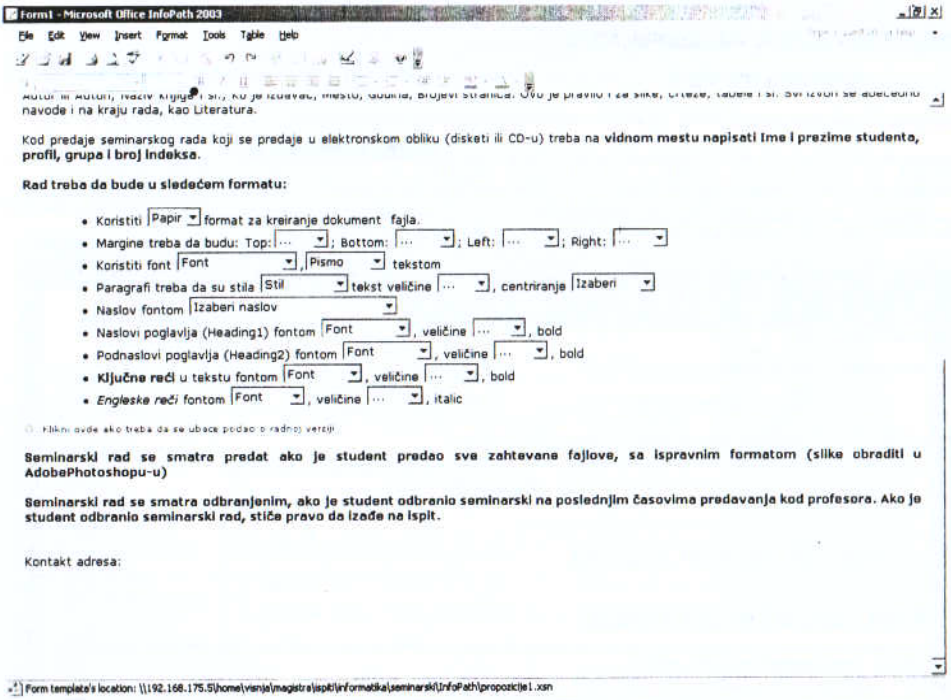
Slika 6

Za slučaj da je potrebna informacija o radnoj verziji, ona se može pozvati jednim klikom miša (inače je skrivena - Slika 7, Slika 8 i Slika 9). Na taj način se brzim odabirom lako modifikuju bitni sadržaji.

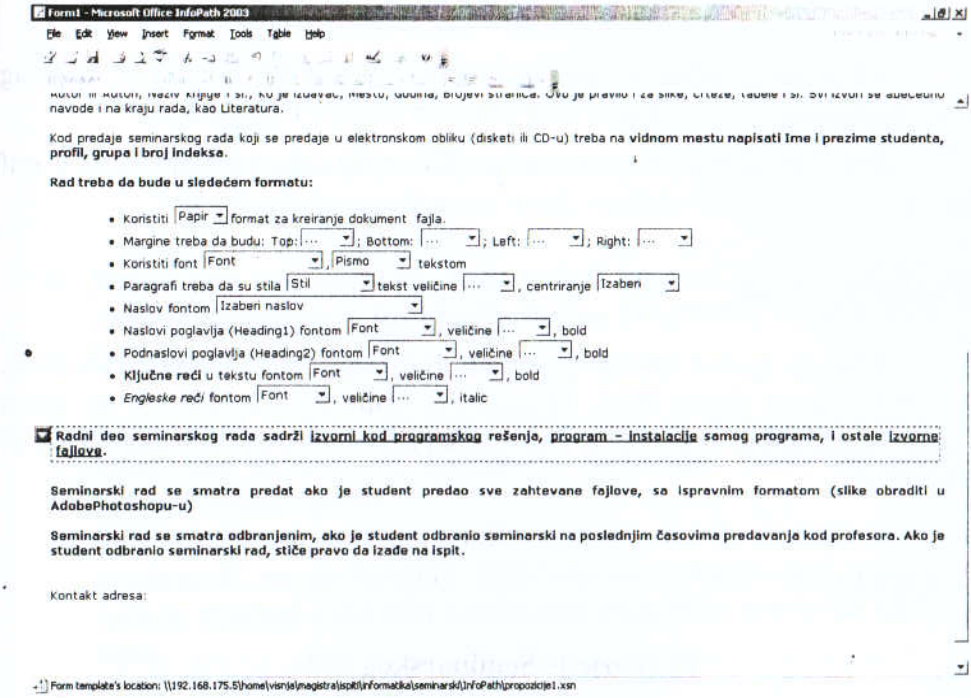
Print Preview prikaz dobijenog dokumenta za Propozicije seminarskog rada iz predmeta Obrazovni računarski softver, dat je na - Slika 10.



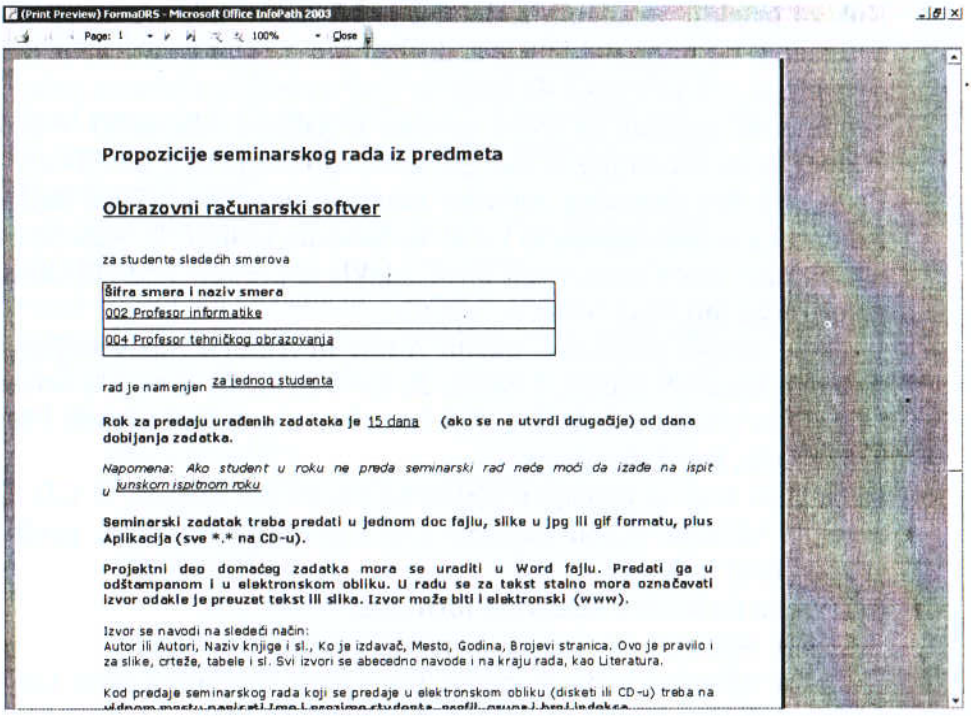
Slika 7.



Slika 8



Slika 9



Slika 10.

Na kraju je dobijeni xml fajl Propozicije za ORS otvoren u Architag Xray XML Editor –u i dobijeni kod se može videti u Prilogu 2.

Dobijeni dokument propozicija (Slika 10), se može prebaciti u pdf format.

11. ZAKLJUČAK

XML je moćno oružije koje vam može omogućiti jednostavan život. Svi veliki igrači poput IBM, Microsoft-a prihvatili su XML i na scenu stupaju mnogi proizvodi, usluge i ko zna šta još. Budite i vi deo toga, XML je sadašnjost.

12. PRILOG 1

Propozicije Seminarskog rada

1. **Svaki student dobija svoj domaći seminarski zadatak.**
2. **Rok za predaju urađenih zadataka je 15. maja (ako se ne utvrdi drugačije) od dana dobijanja zadatka.** *Ako student u roku ne preda seminarski rad neće moći da izađe na ispit u junskom ispitnom roku.*
3. **Seminarski zadatak se treba predati u jednom Microsoft Word (*.doc) fajlu,slike u jpg ili kao gif, plus Aplikacija (sve na CD-u)**
4. **Projektni deo domaćeg zadatka mora se uraditi u Word fajlu. Predati ga u odštampanom i u elektronskom obliku. U radu se za tekst stalno mora označavati izvor odakle je preuzet tekst ili slika. Izvor može biti i elektronski (www).**
5. **Izvor se navodi na sledeći način: Autor ili Autori, Naziv knjige i sl.,Ko je izdavač, Mesto, Godina, Brojevi stranica. Ovo je pravilo i za slike, crteže, tabele i sl. Svi izvori se abecedno navode i na kraju rada, kao Literatura.**
6. Seminarski koji se predaje u elektronskom obliku (disketi ili CD-u) treba na **vidnom mestu napisati Ime i prezime studenta, profil, grupa i broj indeksa.**
7. **Rad treba da bude u sledećem formatu:**
   - Koristiti A4 format za kreiranje dokument fajla.
   - Margine treba da budu: Top:1,8cm; Bottom: 2,6cm; Left: 2,6cm; Right: 1,8cm, videti sliku
   - Koristiti font Arial latiničnim tekstom
   - Normal tekst veličine 11, centriran po levoj i desnoj margini (Justify)
   - Naslov fontom (Title) Arial,veličine 22, bold

- Naslovi poglavlja (Heading1) fontom Arial, veličine 14, bold
- Podnaslovi poglavlja (Heading2) fontom Arial, veličine 11, bold
- **Ključne reći** u tekstu fontom Arial, veličine 11, bold
- *Engleske reći* fontom Arial, veličine 11, Italic

8. **Radni deo seminarskog rada sadrži izvorni kod programskog rešenja, program – instalacije samog programa, i ostale izvorne fajlove.**

9. **Seminarski rad se smatra predat ako je student predao sve zahtevane fajlove, sa ispravnim formatom (slike obraditi u AdobePhotoshopu-u)**
10. **Seminarski rad se smatra odbranjenim, ako je student odbranio seminarski na poslednjim časovima predavanja kod profesora. Ako je student odbranio seminarski rad, stiče pravo da izađe na ispit.**

Kontakt adresa: radosav@tf.zr.ac.yu

13. PRILOG 2

```
<?xml version="1.0" encoding="UTF-8"?><?mso-infoPathSolution
solutionVersion="1.0.0.77" productVersion="11.0.5531"
PIVersion="1.0.0.0"
href="file:///\\192.168.175.5\home\visnja\magistra\ispiti\informatika\seminar
ski\InfoPath\propozicije1.xsn" ?><?mso-application
progid="InfoPath.Document"?><my:myFields
xmlns:my="http://schemas.microsoft.com/office/infopath/2003/myXSD/2004
-03-05T12:37:58" xmlns:xhtml="http://www.w3.org/1999/xhtml"
xml:lang="en-us">
        <my:group1>
                <my:group2></my:group2>
        </my:group1>
        <my:group4>
                <my:group5></my:group5>
        </my:group4>
        <my:group6>
                <my:group7></my:group7>
        </my:group6>
        <my:group9>
                <my:group10>
                        <my:field10></my:field10>
```

<my:group13>
<my:group14></my:group14>
</my:group13>
</my:group10>
</my:group9>
<my:group11>
<my:group12>
<my:field11></my:field11>
<my:field12></my:field12>
<my:group23>
<my:field24></my:field24>
</my:group23>
<my:field25>002 Profesor informatike</my:field25>
</my:group12><my:group12>

<my:field11></my:field11>

<my:field12></my:field12>

<my:group23>

<my:field24></my:field24>

</my:group23>

<my:field25>004 Profesor tehniÄ□kog obrazovanja</my:field25>
</my:group12>
</my:group11>
<my:field13>za jednog studenta</my:field13>
<my:field14 xsi:nil="true"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></my:field14>
<my:field15>15 dana</my:field15>
<my:group15>
<my:group16></my:group16>
</my:group15>
<my:group18></my:group18>
<my:group19></my:group19>
<my:group20>
<my:field16></my:field16>
</my:group20>
<my:field17>Obrazovni raÄ□unarski softver</my:field17>
<my:field18></my:field18>
<my:field19></my:field19>
<my:group21>

```
            <my:field20></my:field20>
        </my:group21>
        <my:group22>
            <my:field21></my:field21>
        </my:group22>
        <my:field22></my:field22>
        <my:field23>junskom ispitnom roku</my:field23>
        <my:group24></my:group24><my:group25>
            <my:field26></my:field26>
        </my:group25>
        <my:group26>
            <my:field27></my:field27>
        </my:group26>
        <my:field28>A4</my:field28>
        <my:field29>1,8 cm</my:field29>
        <my:field30>2,6 cm</my:field30>
        <my:field31>2,6 cm</my:field31>
        <my:field32>1,8 cm</my:field32>
        <my:field33>Arial</my:field33>
        <my:field34>latiniÄ□nim</my:field34>
        <my:field35>Normal</my:field35>
        <my:field36>11</my:field36>
        <my:field37>Justify</my:field37>
        <my:field38>Title, Arial, vel. 22, bold</my:field38>
        <my:field39>Arial</my:field39>
        <my:field40>14</my:field40>
        <my:field41>Arial</my:field41>
        <my:field42>11</my:field42>
        <my:field43>11</my:field43>
        <my:field44>Arial</my:field44>
        <my:field45>11</my:field45>
        <my:field46></my:field46>
        <my:group27>
            <my:field47></my:field47>
        </my:group27>
        <my:group28>
            <my:field48></my:field48>
        </my:group28>
        <my:group29>
            <my:group30>
                <my:field49>radosav@tf.zr.ac.yu</my:field49>
            </my:group30>
        </my:group29>
</my:myFields>
```

14. PRILOG 3

Kratak pregled o XML-u[7]

Šta je XML?

- **XML – eXtensible Markup Language (proširivi označavajući jezik) je jezik za označavanje strukture dokumenta, unutar njegovog sadržaja:**

  - World Wide Web Consortium (W3C) je propisao osobine fajl formata radi jednostavnijeg i jeftinijeg distribuiranja elektronskih dokumenata na World Wide Web-u
  - proširiv, ne tako ograničen kao HTML (nove oznake se mogu i moraju definisati)
  - podržava obimnije strukture, kao što su objekti ili hijerarhije
  - podržava zakonitosti i dobro oblikovane osobine
  - odvaja formu (kako izgleda) od sadržaja (šta je to)

Označavajući jezici

- **Postoje mnogi označavajući jezici**
  - MS Word koristi Rich Text Format (RTF), zaštićeni označavajući jezik
  - Word Perfect
  - HTML (HyperText Markup Language)
  - XML (međustranični word procesori koriste XML)
  - SGML – Standard Generalised Markup Language
- **XML i HTML su podskupovi od SGML-a**
  - XML obavlja 80% SMGL-ovih funkcija sa 20% njegove složenost
  - Izvestaj kod SMGL-a je 155 strana, a kod XML je 35 strana
  - Uklanja sve neobavezne osobine SMGL-a

XML koncepti

- **XML je samoopisujuć i može biti validovan**

---

[7] *Prevod Lecture 13. XML (PDF), MIT OpenCourseWare » Civil and Environmental Engineering » 1.264J Database, Internet, and Systems Integration Technologies, Fall 2002*

- XML dokumenta sadrže pravila po kome se podaci mogu uskladiti
- Pravila mogu biti ponovo korišćena u drugim dokumentima: klasama dokumenata

- **XML aplikacije**
  - Oblik razmene podataka između kompjutera
    - Koristi Web servere za protok podataka između baza podataka
  - Uobičajen format za Web, EDI, papirni dokument,...
    - XML kao opšti označavajući jezik
  - Daljinske procedure call/invocation (zvanje/pozivanje) protokola
    - Izvršavaju Web servisi ili procesi na drugim kompjuterima

XML alati

- **HTML prikazuje podatke; XML opisuje podatke**
- **XML u sebi sadrži sledeće jezike:**
  - CSS , Cascading Style Sheets (pre-XSL)
  - XSLT: extensible stylesheet language/transformation
    - Ulazni dokument - XML (bez šema) -> Vrste izlaznih dokumenata: XML, HTML ili običan tekst
    - Skriptovani jezik, slobodan unos, toleriše greške
  - XQuery
    - XML (sa šemama) -> XML
    - Strogo sintaksni jezik, pristup bazama podataka, garantuje tačnost
    - Osnovno preklapanje sa XPath-om (deo XSL-a), koordiniraju
  - XML Šema
    - Zamenjuje DTD (Document Type Definition), koji nisu XML, sa XML definicijom
  - Xlink
    - Dopušta više glavnih hiperlinkova nego HTML
  - XHTML
    - Preispituje HTML koji je važeci XML
  - **X**Forms
    - XML verzija HTML-a: definiše značenje, ne samo izgled

XML planiranje cilja

- **Čitljiv za ljude**
  - Proizveden pomoću standardnih browser-a (IE 5, Netscape 5)
- **Čiljiv za mašine**
  - Automatizovana obrada razmenjenih dokumenata putem XML raščlanjivača napravljenih u browser-ima i drugim Web alatima
- **Pristupačan dokumentima**
  - XML se koristi za priručnike, CD-ove, help i ostale tekstualne dokumente
  - Word procesori koriste XML kao standardni jezik

Struktura XML dokumenta

- **HTML**
  - <Head>
    - <Title>
  - <Body>
- **XML**
  - Prolog
    - XML deklaracija (definisanje verzije)
    - Deklaracija tipa dokumenta (definisanje tagova)
  - Telo dokumenta
    - Tagovi definišu element podataka

Tipovi XML dokumenata

- **Važeći dokumenti**
  - Slede sva pravila:
    - Npr. #REQUIRED za elemente
- **Dobro Formiran dokument**
  - Sledi XML sintaksu, ali može da bude neprihvaćena
  - Browser-i ga koriste za prijem XML dokumenata koji su već bili prihvaćeni pomoću servera
  - Ne treba da se download-uje DTD i da se ponovo validuje
- **Takođe se koriste u manjim aplikacijama, koje mogu brzo da se napišu**
  - Npr. Deljenje dokumenata između odeljenja ili radnih grupa

Primer dokumenta

- **Katalog proizvoda i dostupnost**
  - Vrednovanje proizvoda i servisa, i dostupnost
- **Načini naručivanja**
  - Datum prijema narudžbine, fakturisanje, otpremanje, itd.
- **Vrsta poreza**
  - Nadzornici, mušterije, vrsta poreza
- **Knjige i pregled podataka**
  - Opšti tehnički podaci
- **Korisnički priručnici i instrukcije**
  - Operativne instukcije za posebne proizvode i servise

Primer XML dokumenta: Email

```
<?xml version="1.0"?>
<!DOCTYPE EMAIL SYSTEM "Example2.dtd">

<EMAIL LANGUAGE="Western" ENCRYPTED="128"
        PRIORITY="HIGH">
        <TO>Xin@mit.edu</TO>
        <FROM>&SIGNATURE;@mit.edu</FROM>
        <CC>Shuang@mit.edu</CC>
        <BCC>Ming@mit.edu</BCC>
        <SUBJECT>Sample Document with External DTD
        </SUBJECT>
        <BODY>
                Hello, this is &SIGNATURE;.
                Take care, -&SIGNATURE;
        </BODY>
</EMAIL>
```

Primer: Katalog u XML-u

```
<parts>
        <part>
                <partNO>45891</partNO>
                <description>40mm Rubber Washer
                </description>
                <inStock>Yes</inStock>
                <branch>Chicago, IL</branch>
```

```
            <cost>$0.45</cost>
        </part>
        <part>
            <partNO>40892</partNO>
            <description>45mm Rubber Washer
            </description>
            <inStock>No</inStock>
            <branch<Crested Butte, CO</branch>
            <cost>$0.50</cost>
        </part>
</parts>
```

DATA MINING[1]

[1] *Seminarski rad Sonje Stanković, br.ind.: 12/02M-10, urađen pod mentorstvom Prof.dr Dragice Radosav, u okviru predmeta Informatika, na posle-diplomskim studijama na Tehničkom fakultetu u Zrenjaninu*

SADRŽAJ:

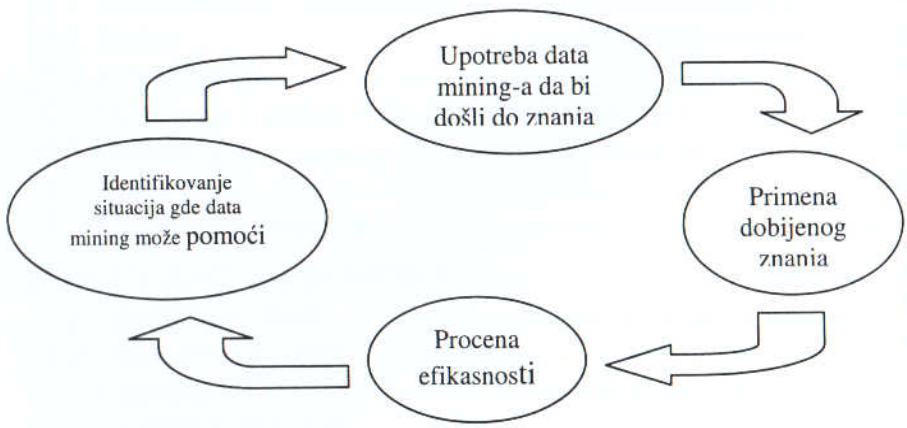
1. UVOD

Tržište svakim danom postaje sve konkurentnije. Razliku između uspeha i propadanja kompanija često čini tanka linija veće efikasnosti. Ova prednost se često postiže boljim i preciznijim informacijama koje su osnova dobrih poslovnih odluka. Problem nastaje kad treba odabrati način na koji ćemo doći do ovih odluka. Jedan od mogućih načina je: analizom veoma vrednih i često potcenjenih postojećih podataka.

Usled automatizacije prikupljanja podataka, baze podatka danas stižu i do veličine od terabajta (više od 1.000.000.000.000 bajtova podataka). U okviru ovih podataka nalaze se informacije od strateške važnosti. Samo mali deo podataka koji se prikupljaju se analizira (procenjuje se oko 5%). Ogromne količine podataka se prikupljaju da se ne bi izostavile neke bitne informacije. Međutim, količina podataka se povećava velikom brzinom, tako da se stari podaci nikad ne analiziraju.

Još jedna otežavajuća pojava za stručnjake koji treba da analiziraju sakupljene podatke je internet. Internet je danas sastavni deo našeg života i broj njegovih korisnika i njegovih primena se svakim danom povećava. Kroz bilione Web stranica obuhvaćeni su gotovo svi aspekti ljudskih delatnosti. Međutim, pretraga kroz podatke, razumevanje podataka i korišćenje informacija sačuvanih u raznim oblicima, predstavlja pravi izazov, jer su na internetu raspoloživi podaci veoma raznovrsni i mnogo su dinamičniji od podataka koji se nalaze u bazi podataka neke kompanije.

Najnovije rešenje ovog problema donosi nam data mining (DM) (negde nazvan i Knowledge Discovery in Databases (KDD) – otkrivanje znanja u bazama podataka), koji se koristi da bi se povećali rezultati i smanjili troškovi.



Slika 1. Prikaz ciklusa upotrebe data mining-a

Potencijal je ogroman. Inovativne organizacije širom sveta koriste data mining da otkriju i kontaktiraju potencijalne potrošače, da promene ponudu proizvoda u cilju povećanja prodaje, i da minimiziraju gubitke zbog grešaka ili prevara. U farmaceutskoj industriji, data mining se koristi da se analiziraju podaci za odabir hemijskih sastojaka za pravljenje novih lekova. U bioinformatici ima primenu u sortiranju gena po sličnosti njihovih osobina.

Stručnjaci razvijaju data mining tehnike svaki dan i ova tehnologija će igrati veoma bitnu ulogu u razvoju inteligentnih pretraga i predviđanja budućih podataka na osnovu postojećih.

Data mining je, u stvari, poluautomatsko otkrivanje šablona, veza, promena, zavisnosti i anomalija u velikoj količini podataka.

Data mining ima primenu i u poslovnim istraživanjima i u nauci. Ovo je neophodan proces zbog svakodnevnog rasta količine podataka i shvatanja da ovaj proces može da se iskoristi kao sredstvo za ostvarivanje tzv. poslovne inteligencije.

2. ŠTA JE DATA MINING?

Važnost sakupljanja podataka, koji mogu uticati na poslovanje ili naučne aktivnosti, da bi se stekle prednosti u odnosu na konkurenciju, je priznata u celom svetu. U svim velikim kompanijama postoje moćni sistemi, za prikupljanje podataka i njihovo skladištenje u velike baze podataka. Problem nastaje kad treba ove podatke pretvoriti u informacije koje mogu koristiti, zbog teškoće analiziranja velike količine podataka i dobijanja korisnih informacija iz istih.

Analiza koju izvršavaju ljudi, bez pomoći specijalnih alata, nema više smisla zbog količine podataka koju treba obraditi, da bi se donele poslovne odluke.

Data mining je moderan pristup analizi podataka koji ne zamenjuje tradicionalne statističke metode. On kombinuje statističke metode sa računarom da bi obradio velike količine promenljivih podataka. Neki delovi data mining-a se mogu automatizovati, ali to su samo mali delovi procesa. Razumevanje problema, odabir bitnih podataka, transformisanje podataka u adekvatan oblik i interpretacija rezultata su aktivnosti koje trenutno nisu automatizovane, i ne izgleda da će biti u skorijoj budućnosti.

Glavna ideja data mining-a je da se izvrši pretraga i analiza postojećih podataka iz kojih se izdvajaju šabloni. Na osnovu ovih šablona dolazimo do znanja i do predviđanja, tj. na osnovu postojećih podataka predviđamo buduće podatke.

Sistemi baza podataka ne podržavaju upite kao što su:

- "Ko bi kupio proizvod X",
- "Prikaži mi sve probleme koji su slični ovom",
- "Obeleži sve lažne transakcije",
- "Koju robu treba da ponudim ovom potrošaču? ",
- "Koja je verovatnoća da će potrošač odgovoriti na ovu ponudu? ",
- "Mogu li se predvideti najprofitabilniji načini kupovine/prodaje u narednom periodu? ",
- "Da li će korisnik odložiti plaćanje zajma ili će ga isplatiti na vreme? ",
- "Koja medicinska dijagnoza treba da se postavi ovom pacijentu? ",
- "Kolika će potrošnja električne energije biti u narednom periodu? ",
- "Zašto se u proizvodnji povećao broj neispravnih proizvoda? ", itd.

Ali, ovo su veoma bitni upiti, čije rezultate možemo dobiti uz pomoć Data Mining-a.

Data mining je proces koji koristi razne alate za analizu podataka da bi se otkrili šabloni i veze među podacima koji mogu biti iskorišteni da bi se došlo do informacija, a analizom tih informacija mi dolazimo do znanja. Znanje nam je potrebno da bi smo došli do mudrosti (kao što je ilustrovano na slici ispod).

Podaci ← predstavljene činjenice

Informacije ← razumevanje podataka

Znanje ← razumevanje dobijenih informacija

Mudrost ← primena prikupljenog znanja

Glavni zadaci data mining-a su opisivanje i predviđanje. Metode predviđanja koriste promenljive da bi predvidele nepoznate ili buduće vrednosti drugih promeljivih. Opisne metode pronalaze šablone koji opisuju podatke.

Prvi korak u data mining analizi je opisivanje podataka (definisanje statističke osobine – kao što su značenja i standardna odstupanja), vizualni pregled podataka (koristeći dijagrame i grafikone), i traženje mogućih smislenih veza između promenljivih (npr. vrednosti koje se često pojavljuju zajedno). Kao što će kasnije biti naglašeno, prikupljanje, pretraživanje i odabiranje pravih podataka je od kritične važnosti.

Ali sam opis podataka ne može biti dobar plan za akciju. Mora se izgraditi model za predviđanje, zasnovan na šablonima koji su određeni na osnovu poznatih rezultata, zatim testirati model na rezultatima, van originalnog uzorka. Dobar model ne sme se pomešati sa stvarnošću (zna se da mapa puta nije perfektan prikaz puta), ali može biti koristan vodič u razumevanju posla. Poslednji korak je empirijsko verifikovanje modela. Na primer, iz baze podataka potrošača koji su već odgovorili na određenu ponudu, može se napraviti model za predviđanje koji su potencijalni

kandidati koji bi odgovorili na istu ponudu. Možete li se osloniti na ovo predviđanje? Kontaktirajte određen broj kandidata sa nove liste i pogledajte koje će te rezultate dobiti.

3. ZAŠTO DATA MINING?

Data mining može biti veoma vredan alat vašoj kompaniji, ali samo ako znate da otkrijete znanje sakriveno u podacima. Data mining omogućava izdvajanje dijamanata znanja iz podataka koji su uskladišteni i predviđanje ishoda budućih situacija. Pomoćiće u optimizaciji poslovnih odluka, povećaće vrednost svake mušterije i komunikacije i povećaće zadovoljstvo mušterije vašim uslugama.

Podaci koji zahtevaju analizu se razlikuju u zavisnosti od kompanija. Primeri uključuju:

- Prošlost prodaje mušterijama i njihovom kontaktiranju,
- Podatke koji su uskladišteni o pozivima mušterija,
- Demografske podatke o mušterijama,
- Podatke o dijagnozama i prepisanim lekovima pacijentima,
- Podatke koji se nalaze na vašem Web sajtu...

U svim ovim slučajevima data mining vam može otkriti znanje skriveno u podacima i pretvoriti ovo znanje u prednost nad konkurencijom. Danas sve više kompanija priznaje vrednost ove nove pojave nazvane data mining, čiji je cilj da se optimiziraju operacije i poveća profit.

4. DATA MINING: ŠTA MOŽE DA URADI?

Data mining je alat, ne magični štapić. On ne eliminiše potrebu da znate o vašem poslu, da razumete vaše podatke ili analitičke metode. Data mining pomaže poslovnim analitičarima da pronađu šablone i veze među podacima – on ne govori vrednost šablona za organizacije. I što je najbitnije, šabloni koje data mining otkrije, moraju biti potvrđeni u stvarnom svetu.

Na prvi pogled, data mining se ne razlikuje od tradicionalnih metoda pretraživanja podataka, ali se on razlikuje pre svega, po svom cilju. Cilj data mininga nije da pronađe činjenice (gde se tradicionalne metode zaustavljaju), već da generiše hipoteze; da nas usmeri na pitanja koja su pred nama, a ne da nam generiše odgovore.

Treba imati na umu da predviđajuće veze između podataka koje se pronalaze pomoću data mining-a ne prouzrokuju ni jednu akciju ili ponašanje. Na primer: otkrijete da su muškarci sa mesečnim prihodom između 15.000 i 20.000 dinara koji kupuju određene novine, mogući kupci proizvoda koji prodajete. Ovaj šablon možete iskoristiti, recimo da marketinški ciljate na ovu populaciju potrošača, treba imati na umu da ovi faktori ne znače da će oni nužno kupiti vaš proizvod.

Da bi ste osigurali rezultate koji će biti od koristi od vitalne važnosti je da shvatate podatke sa kojima radite. Kvalitet izlaznih podataka će često biti veoma osetljiv na izuzetke (podaci koji imaju veoma različite vrednosti od tipičnih vrednosti u vašoj bazi podataka), nebitne stavke, ili stavke koje su bitne zajedno (kao što su godište, datum rođenja), način kodiranja vaših podataka, podaci koje ostavljate i podaci koje isključujete. Algoritam varira u svojoj osetljivosti na takve podatke, ali nije mudro oslanjati se na alat data mining-a da donosi sve bitne odluke sam.

Data mining alat neće automatski otkriti sva rešenja bez navođenja čoveka. Umesto stavljati za cilj: "Pomozi mi da poboljšam odgovor na moje ponude e-mail-om", koristite data mining da otkrijete osobine ljudi koji:

- odgovaraju na vaše e-mail-ove i
- kupuju vaše proizvode.

Šabloni koje otkrije data mining alat za ova dva cilja mogu biti veoma različiti.

Data mining zahteva od vas da razumete rad sa alatom koji odaberete i algoritam na osnovu kojeg je taj alat baziran. Odluke koje napravite u postavljanju alata za data minig i optimizacije koju koristite će uticati na preciznost i brzinu vašeg modela.

Data mining ne zamenjuje vešte poslovne analitičare ili menadžere, ali im daje moćan nov alat da poboljšaju svoj učinak.

Često se podaci sa kojima radimo već nalaze u nekom skladištu podataka. Baza podataka može biti logički podskup skladišta podataka. Skladište podataka nije uslov za data mining. Postavljanje velikog skladišta podataka koje usklađuje podatke iz više izvora, rešava problem integriteta podataka, ali, sa druge strane, učitavanje upita u takvo skladište podataka može biti ogroman posao, koji dugo traje i veoma je skup. Iz svega ovoga sledi bitan zaključak: za uspešan proces data mining-a uopšte nisu potrebna skladišta podataka – sve što vam je potrebno, to su podaci. Podaci se mogu učitavati iz jedne ili više baza podataka jednostavno izdvajajući te podatke kao read only podatke (podatke koji se mogu samo čitati).

5. STANDARDI DATA MINING-a

Za data mining ne postoje propisani standardi. Velike informatičke kompanije se "bore" da njihov način rada bude proglašen standardom, što bi uslovilo i prodaju njihovih data mining alata, kurseva za buduće *data miner*-e... Među "takmičarima" se javljaju zaista poznata imena, kao što su:

- IBM – predlaže PMML (Predictive Modelling Markup Language), kao i SQL/MM,
- Oracle – predlaže JSR-73 (Java Data Mining Standards),
- SAS i Microsoft – predlažu XML za analizu...

Međutim, još uvek ništa nije odlučeno, tako da svaki data miner radi na svoj način, metodama koje njemu najviše odgovaraju. Naravno, kad se usvoji standard, ništa neće sprečiti *data miner*-e da rade metodom koja njima najviše odgovara.

6. DATA MINING, MAŠINSKO UČENJE I STATISTIKA

Data mining preuzima prednost u poljima veštačke inteligencije i statistike. Obe ove discipline rade na problemu prepoznavanja šablona i klasifikacije i veoma su doprinele razumevanju i primenjivanju neuralnih mreža i stabla pretraživanja.

Mašinsko učenje se bavi problemom kako bi mašine (računari) mogle same da stiču znanje. Data mining se bavi omogućavanjem ljudima da uče iz postojećih podataka.

Data mining je nadogradnja statističkih metoda koja je posledica velikih promena u statističkoj zajednici. Razvoj većine statističkih metoda je, do nedavno, baziran na teorijama i analitičkim metodama koje su radile dosta dobro kad je trebalo manji skup podataka da se analizira. Povećana moć kompjutera i njihova sve manja cena, pridonela je potrebi da se analizira sve više i više podataka, skupovi od miliona redova, tehnikama koje se baziraju na "nasilnom" pretraživanju mogućih rešenja. Nove tehnike uključuju algoritme kao što su neuralne mreže i stabla pretraživanja i nove prilaze starijim algoritmima kao što je diskriminantna analiza. Ove tehnike mogu skoro svaku funkciju ili interakciju da obave same. Tradicionalne statističke tehnike se baziraju na kreatoru, da im odredi funkcije i interakcije.

Bitno je znati da je data mining primena ovih i drugih statističkih metoda (baziranih na veštačkoj inteligenciji) da se ustaljeni problemi u poslu reše na način koji je pristupačan i radnicima koji se razumeju u posao i

problem, kao i uvežbanim statističarima. Data mining je alat za povećanje produktivnosti ljudi koji se trude da izgrade modele za predviđanje.

7. DATA MINING U ZAVISNOSTI OD HARDVERA /SOFTVERA

Bitna stvar koja omogućuje data mining je veliko sniženje u ceni hardvera i povećanje njegovih karakteristika. Samim tim, računari su sve pristupačniji ljudima. Veliki broj ljudi ima pristup internetu, ostavlja svoje podatke preko raznih upitnika, a ti podaci se sakupljaju u bazama podataka i čekaju obradu.

Tokom godina, drastično se smanjuje cena skladištenja podataka, kao i kompjuterska obrada podataka. Kompjuterska obrada podataka ima manju cenu, što je ta obrada automatizovanija. Automatizaciju obrade podataka omogućuje upravo data mining.

8. USPEŠNI DATA MINING

Da bi se data mining uspešno izveo, mora se voditi računa o dve ključne stvari:

- Precizna formulacija problema koji treba rešiti – precizno određivanje obično daje najbolje rezultate
- Korišćenje pravih podataka – nakon odabira podataka koji su na raspolaganju, možda ih je potrebno transformisati i kombinovati, da bi dobili što bolje rezultate

Što se graditelj modela može više "igrati" sa podacima, građenjem modela, procenjivanjem rezultata, bolji će biti konačni model.

Idealno bi bilo kad bi alat za pretraživanje podataka (grafički/vizuelni, upit/OLAP) bio dobro integrisan sa analizom ili algoritmom na osnovu kog se gradi model.

Data mining je proces koji je sam po sebi specifičan za svaki problem koji treba da reši. Zbog toga je teško reći pravila ili postupak samog data mining-a, ali u nastavku su predloženi koraci koji svaki data mining proces treba da sadrži.

Konkretno, data mining se može posmatrati kao izdvajanje skupa podataka iz jedne (ili više) baze podataka i primena data mining algoritma da proizvede predviđajući model ili skup. Na raspolaganju su razne strategije,

što zavisi od oblika podataka, raspodele podataka, raspoloživih resursa, zahtevane preciznosti...

9. KORACI DATA MINING-A

Svako ko se bavi data mining-om razvija svoje metode i korake koje usavršava praksom. Postoji mnogo preporučenih koraka kako "uspešno odraditi" data mining. Neki autori predlažu sledeće korake:

- Odabir podataka (skupa sa kojim će se raditi)
- Čišćenje podataka (uklanjanje netačnih i nepotpunih podataka)
- Transformacija podataka (smanjenje dimenzija)
- Generisanje šablona (algoritmi: klasifikacija, podela...)
- Procena test rezultata
- Vizualizacija (transformisanje rezultata u oblik koji će ljudi razumeti)

Međutim, iako, svaki *data miner* ima svoj način rada, suština je ista.

U nastavku je objašnjen proces izdvajanja skrivenog znanja iz postojećih podataka na još jedan način (drugačije su koraci nazvani i posao je drugačije raspoređen).

Može se desiti da se svi koraci odrade (naizgled) korektno, a opet da rezultati ne zadovoljavaju, što dovodi do vraćanja na neki od prethodnih koraka, a nekad i do vraćanja na prvi korak.

9.1. Identifikovanje cilja

Pre nego što se počne, treba raščistiti sa tim šta želimo postići sa analizom. Treba unapred znati koji je poslovni cilj data mining-a. Potrebno je ustanoviti da li je cilj merljiv. Neki mogući ciljevi su:

- Pronaći veze između prodaje nekih proizvoda ili usluga
- Identifikovanje šablona kupovine tokom nekog perioda vremena
- Identifikovanje potencijalnih vrsta potrošača
- Pronaći veze između prodaje nekih proizvoda u zavisnosti od godišnjeg doba...

9.2. Odabir podataka

Kad smo definisali cilj, sledeći korak je odabrati podatke pomoću kojih ćemo stići do ovog cilja. Ovi podaci mogu biti podskup podataka nekog skladišta podataka. To mogu biti, zaista, različiti podaci (o proizvodima, proizvođačima, potrošačima, prodajnim mestima...). Ovo su neke ključne stvari kod odabira podataka:

- Da li su podaci adekvatni da opišu pojavu koju data mining analiza pokušava da modeluje?
- Mogu li se nadograditi interni podaci o potrošačima sa eksternim demografskim podacima, ili podacima o načinu života?
- Ako se podaci čitaju iz više baza podataka, može li se pronaći zajedničko polje, na osnovu kog bi se ti podaci povezali?
- Koliko su bitni podaci za postizanje cilja?

9.3. Priprema podataka

Kad se podaci odaberu, mora se odrediti koje atribute ćemo konvertovati u oblike koji će biti korisni. Treba uzeti u obzir unose kreatora i korisnika podataka.

- Uspostaviti strategije za ponašanje kad se naiđe na nepotpune podatke, netačne podatke, krajnje vrednosti
- Identifikovanje redundantnih promenljivih u skupu podataka i odlučivanje koja će se polja isključiti iz obrade
- Odlučivanje o transformacijama, ako je potrebno
- Vizualno pregledanje skupa podataka, da bi se stekla slika o bazi
- Određivanje frekvencije raspodele podataka

Neke od ovih odluka mogu se odložiti dok se ne odabere alat za data mining. Na primer, ako vam treba neuralna mreža, možda ćete morati transformisati neka od polja.

9.4. Pregled podataka

Potrebno je pregledati strukturu podataka da bi se odredili odgovarajući alati.

- Koja je priroda i struktura baze podataka?
- Kakva je raspodela u skupu podataka?
- U kakvom su stanju podaci?

Treba izbalansirati strukturu podataka sa potrebama korisnika da bi razumeli rezultate. Neuralne mreže, na primer, ne objašnjavaju njihove rezultate.

9.5. Odabir alata

Odabir alata za data mining se odvija u zavisnosti od: cilja kojem težimo i strukture podataka. Oba bi trebalo da vode do istog alata. Kad odabiramo potencijalni alat treba uzeti u obzir ova pitanja:

- Da li su podaci kategorisani?
- Na kojim platformama alati rade?
- Da li alati podržavaju ODBC?
- Koje oblike podataka alati mogu da koriste?

Malo je verovatno da postoji jedan alat koji će da pruži pravi odgovor na vaš projekat. Neki alati integrišu nekoliko tehnologija u celinu sačinjenu od statističko-analitičkih programa, neuralnih mreža i simboličkih klasifikatora.

9.6. Oblikovanje rešenja

Zajedno sa podacima, cilj pretrage i odabir alata određuju oblik rešenja. Ključna pitanja o kojima treba voditi računa su:

- Koji je optimalni oblik rešenja (drva odlučivanja, pravila, C kod, SQL...)?
- Koji su mogući oblici rešenja dostupni?
- Kakav oblik je potreban krajnjim korisnicima (graf, izveštaj, kod)?

9.7. Oblikovanje modela

Sada počinje sam proces data mining-a. Obično je prvi korak korišćenje broja, odabranog slučajnim izborom, da bi se podelili podaci na skupove za proveru, izradu i procenu modela. Generisanje pravila klasifikacije, drva odlučivanja, podela podataka na podgrupe, ocenjivanje, kodiranje, merenje i procena podataka (grešaka) se odvija u ovoj etapi data mining-a. Treba voditi računa o pitanjima:

- Da li su greške na prihvatljivom nivou?
- Mogu li se greške smanjiti?
- Koji neodgovarajući atributi postoje u podacima? Mogu li se odstraniti?
- Da li su potrebni dodatni podaci?
- Da li je potrebna neka druga metodologija?
- Hoće li biti potrebno koristiti i testirati novi skup podataka?

9.8. Procena rezultata

Rezultate treba prikazati i o analizi tih rezultata prodiskutovati sa poslovnim klijentom ili sa ekspertom u tom domenu poslovanja. Treba se uveriti da su rezultati ispravni i prilagođeni cilju kojem smo težili. Ovo je veoma bitno, jer rezultati mogu izgledati logički ispravni, a u praksi da nemaju nikakvu primenu, jer u stvari nemaju smisla. Ovde treba obratiti pažnju na:

- Da li rezultati imaju smisla?
- Da li se treba vratiti na neki prethodni korak da bi popravili rezultate?
- Može li se koristiti neki drugi data mining alat u cilju poboljšanja rezultata?

9.9. Dostavljanje rešenja

Treba dostaviti konačni izveštaj poslovnoj jedinici ili klijentu. Izveštaj treba da sadrži podatke o celokupnom procesu data mining-a, uključujući pripremu podataka, korištene alate, probne rezultate, izvorni kod, pravila. Neka pitanja koja se ovde javljaju su:

- Da li bi dodatni podaci poboljšali analizu?
- Koje unutrašnje informacije su otkrivene i kako se mogu upotrebiti?
- Da li rezultati zadovoljavaju postavljeni cilj?

9.10. Primena rešenja

Podelite rezultate sa svim zainteresovanim krajnjim korisnicima u odgovarajućoj poslovnoj jedinici. Rešenja mogu uticati na poslovne procese preduzeća. Neka data mining rešenja mogu imati oblik:

- SQL sintakse namenjenoj krajnjim korisnicima
- C koda integrisanog u sistem proizvodnje
- Pravila integrisanih u sistem odlučivanja

Iako data mining alati automatizuju analizu baze podataka, oni mogu voditi do pogrešnih rezultata i zaključaka ako niste pažljivi. Treba imati na umu da je data mining poslovni proces sa određenim ciljem – da pronađe skriveno znanje iz postojećih podataka u bazi.

U nastavku su detaljnije objašnjeni koraci data mining-a, uključujući i etape koje nisu nabrojane u ovih deset, ali su opisane da bi budući data miner-i imali u vidu da su na raspolaganju.

10. ZADACI KOJE REŠAVA DATA MINING

Data mining je proces koji omogućava rešavanje i automatizaciju raznih zadataka u cilju olakšavanja posla stručnjacima (analitičarima, statističarima...) u cilju otkrivanja budućih vrednosti. Neki od zadataka data mining-a su bolje objašnjeni u nastavku:

10.1. Razumevanje i vizualizacija

Pre nego što se izgradi dobar model za predviđanje, moraju se razumeti podaci. Počećemo sakupljanjem različitih numeričkih podataka (uključujući opisne statističke podatke, npr.: prosek, standardne devijacije...) i kategorisanjem podataka.

Podaci mogu biti svrstani u nizove (numeričke vrednosti) ili kategorički svrstani u diskretne klase. Kategorički podaci mogu biti, dalje, definisani kao ordinalni (sa smislenim redosledom) ili nominalni (bez redosleda).

Grafički i vizuelni alati su od vitalne pomoći pri pripremi podataka i analizi istih. Grafički se podaci uglavnom predstavljaju histogramima ili kvadratima u kojima se prikazuju vrednosti. Mogućnost naknadnog dodavanja promenljivih, uveliko povećava korisnost nekih vrsta grafova.

Vizualizacija je korisna zato što omogućava ljudima da vide šumu i da posmatraju drveće ponaosob. Šabloni, veze, specijalne vrednosti i vrednosti koje nisu tu se lakše uoče ako se vizuelno predstave, nego kad su u obliku brojeva i tekstova.

Problem kod vizualizacije je u tome što se podaci mogu predstaviti samo u dve dimenzije na ekranu računara, ili na papiru, a u stvarnosti oni imaju više promenljivih, a samim tim i više dimenzija. Na primer: želimo da prikažemo vezu između zloupotrebe kreditne kartice sa godinama, polom, bračnim stanjem, godinama staža... ovi podaci se veoma teško mogu prikazati u samo dve dimenzije, a da to bude jasno vidljivo korisniku alata. Razvijaju se alati koji mogu prikazati podatke u više dimenzija (bez obzira na ograničenost ekrana računara), ali korisnici ovih alata moraju da uvežbaju oči za uočavanje podataka koji im se predstavljaju.

10.2. Podela

Podatke u bazi je potrebno podeliti na različite grupe. Cilj je da se pronađu one grupe koje se razlikuju od stalih i čiji članovi su međusobno slični. Za razliku od klasifikacije, ne znamo kolike će grupe biti kad počnemo, ni koji će atributi biti sortirani. Ovaj proces je potrebno da nadgleda neko ko se razume u posao kompanije.

Često je potrebno promeniti neke parametre isključivanjem promenljivih koje se javljaju u više slučajeva, jer se često proverom utvrdi da su takve promenljive nebitne. Nakon podele baze na razumne segmente, ovi odeljci mogu biti iskorišćeni za klasifikaciju novih podataka. Ne treba mešati podelu baze sa segmentacijom.

Segmantacija se odnosi na uopštene probleme identifikovanja grupa koja imaju zajedničke karakteristike. Kod podele baze podataka, podaci se grupišu u grupe koje prethodno nisu definisane.

Slika 2. Primeri podele podataka u grupe

10.3. Klasifikacija

Problemi klasifikacije teže da identifikuju karakteristike, koje određuju grupe, kojima svaki od slučajeva pripada. Ovaj šablon se može iskoristiti za razumevanje postojećih podataka i da se predvidi kako će se nove instance ponašati.

Data mining klasifikuje podatke (slučajeve) i induktivno pronalazi predviđajući šablon.

10.4. Otkrivanje i analiza veza među podacima

Analiza veza je opisni prilaz pretraživanju podataka koji može pomoći pri identifikovanju odnosa među vrednostima u bazi podataka. Dva najčešća prilaza pri analizi veza su:

- otkrivanje sličnosti i
- otkrivanje ponavljanja

Pri otkrivanju sličnosti pronalazimo pravila na osnovu kojih se podaci pojavljuju zajedno (npr. kao u kupovini), dok je otkrivanje ponavljanja dosta slično, sa tim da se ponavljaju veze, koje se odlikuju sličnošću tokom vremena.

Da bi smo otkrili smislena pravila moramo posmatrati frekvenciju ponavljanja podataka i njihovih kombinacija. Moramo posmatrati koliko često se ponavlja podatak A (prethodnik), a koliko često podatak B (posledica)? U stvari, zanima nas koliko se često pojavljuje B u odnosu na A? Ovo se može preformulisati u pitanje: "Kad ljudi kupuju čekić, koliko često kupuju eksere?".

Izraz za ovaj uslov predviđanja nazvaćemo sigurnost. Sigurnost je proračunata kao:

$$\frac{\text{Frekvencija A i B}}{\text{Frekvencija A}}$$

Vezu zavisnosti između dva događaja označićemo oznakom: **Þ**.

Da bi ilustrovali ovaj koncept, odredićemo hipotetičku bazu podataka sa podacima:

Ukupna prodaja u gvožđari: 1000
Ukupno objekta "čekić" : 50
Ukupno objekta "ekser" : 80
Ukupno objekta "čekić" i "ekser" : 15

Sad možemo računati:

Podrazumeva kupovinu "ćekić i ekser" = 1.5% (15/1000)
Verovatnoća "čekić Þ ekser" = 30% (15/50)
Verovatnoća "ekser Þ čekić" = 19% (15/80)

Na osnovu ovoga može se videti verovatnoća da će kupac čekića kupiti eksere (30%) koja je veća od verovatnoće da će kupac eksera kupiti čekić (19%). Podrazumevanje ove veze čekić-ekser je dovoljno velika (1.5%) da bi predočila smisleno pravilo.

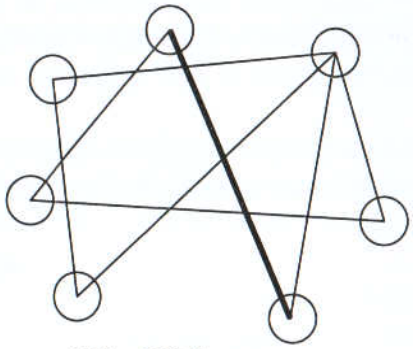
Algoritmi povezivanja pronalaze ova pravila sortirajući podatke, dok beleže pojavljivanje pojava, tako da mogu izračunati njihovu verovatnoću pojavljivanja. Efikasnost sa kojom izračunavaju ove verovatnoće čini razliku među algoritmima. Ovo je veoma bitno, jer kombinatorna eksplozija može prouzrokovati ogroman broj pravila. Neki algoritmi mogu kreirati bazu podataka koju čine pravila, faktori pojavljivanja i podrška za upite (npr.: "Prikaži sve veze sladoleda koje se ponavljaju, imaju verovatnoću preko 80% i podrazumevaju se 2% ili više").

Još jedna zajednička osobina generatora pravila za povezivanje je da ih mogu hijerarhijski predstaviti. U našem primeru, gledali smo sve eksere i čekiće, ne individualne pojave. Važno je odabrati odgovarajući nivo agregacije, inače će biti mala verovatnoća za pronalazak veza koje nas zanimaju. Hijerarhija podataka nam omogućava da kontrolišemo nivo agregacije i da eksperimentišemo sa različitim nivoima. Moramo da zapamtimo da pravila za vezivanje ili ponavljanje nisu zaista pravila, već opisi veza u određenoj bazi podataka. Nije moguće testiranjem modela na

drugim podacima, povećati snagu ovih pravila. U stvari, krećemo sa pretpostavkom da će prošle pojave da se ponavljaju i u budućnosti.

Često je teško odlučiti šta raditi sa pravilima veza koje smo otkrili. U planiranju prodavnice, na primer, staviti povezane objekte fizički jedne do drugih, može smanjiti prihode (potrošači manje kupuju, jer više ne stavljaju u korpu robu koju nisu planirali kupiti, dok idu kroz radnju tražeći robu koju žele kupiti). Analiza i eksperiment su često potrebni da bi iskoristili pravila povezivanja do kojih smo došli.

Grafički metodi mogu biti veoma korisni u viđenju strukture povezivanja. Na primer, gledanjem u bazu podataka osiguranja da bi videli potencijalne prevare, možemo otkriti da određeni doktor i advokat rade zajedno u neuobičajeno mnogo slučajeva.



Slika 3.Dijagram veza

10.5.Modelovanje

Modelovanje ima za zadatak pronalaženje pravila (načina) za opis zavisnosti koje se javljaju među promenljivama.

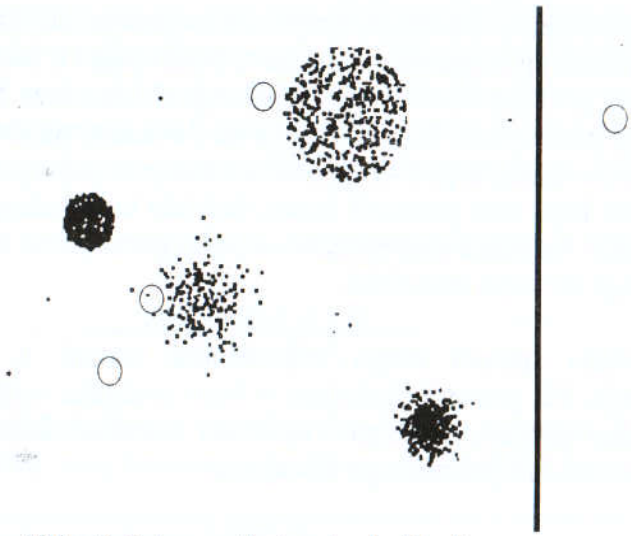
10.6. Predviđanje

Predviđanje ima za zadatak uočavanje šablona i korišćenje razvijenog modela da se predvide buduće vrednosti ciljne promenljive.

10.7. Otkrivanje devijacija

Otkrivanje devijacija ima za zadatak otkrivanje najznačajnijih promena, u ključnim veličinama podataka, od prethodnih ili očekivanih vrednosti.

Ovo je u stvari proces posmatranja skupa podataka i njihovih "normalnih" vrednosti. Zatim je potrebno otkriti podatke koji značajno odstupaju od "normalnih" vrednosti.



Slika 4. Primer otkrivanja devijacija

11. PREDVIĐAJUĆI DATA MINING

Data mining je alat za predviđanje na osnovu sadašnjeg (poznatog). Ako bi samo analizirao podatke, bez davanja informacija kako bi oni (podaci) mogli uticati na buduća stanja, onda se ovaj proces ne bi razlikovao od tradicionalnih statističkih metoda.

11.1. Hijerarhija izbora

Cilj data mining-a je da proizvede novo znanje na osnovu kojeg korisnik može delovati. Ovo može raditi gradeći model realnog sveta na osnovu prikupljenih podataka iz različitih izvora koji mogu uključivati novčane transakcije preduzeća, potrošačku istoriju i demografske informacije. Ovi podaci se mogu obraditi uključujući eskterne baze podataka kao što su podaci iz banke ili vremenski podaci. Rezultat građenja modela je opis šablona i veza među podacima koji se mogu koristiti za predviđanja.

Da bi izbegli konfuziju različitih aspekata data mining-a, pomaže hijerarhijski poređati izbore i odluke koje će nam biti potrebne, pre početka:

- poslovne ciljeve
- vrsta predviđanja
- vrsta modela
- algoritam
- proizvod

Na najvišem nivou su poslovni ciljevi: šta je svrha pretraživanja ovih podataka? Na primer, traženje šablona u podacima koji će pomoći da se zadrže dobre mušterije. Može se izgraditi model za predviđanje profitabilnosti mušterija i drugi model koji će otkriti koje mušterije će najverovatnije to prestati da budu.

Znanje o organizacijskim potrebama i ciljevima će vas voditi u formulisanju ciljeva modela.

Sledeći korak je odlučivanje koja metoda predviđanja je najprikladnija:

- klasifikacija – predviđanje kojoj kategoriji ili klasi slučaj pripada
- regresija – predviđanje koju brojevnu vrednost će imati promenljiva (ako je to promenljiva koja vremenom varira, onda se ova metoda naziva predviđanje u «vremenskim serijama»)

Sad može da se odabere tip modela. Neuralna mreža može biti pogodna da se izvede regresija (možda) i drvo odlučivanja za klasifikaciju. Postoje i tradicionalni statistički modeli, među kojima se može birati kao što su: logistička regresija, analiza diskriminanti ili uopšte linearni modeli.

Mnogi algoritmi su na raspolaganju za građenje modela. Za izgradnju neuralne mreže i drva odlučivanja na raspolaganju su različiti algoritmi, o kojima će biti reči kasnije.

Pri odabiranju proizvoda data mining-a, treba imati na umu da postoje različite implementacije određenog algoritma čak i kad se identifikuje istim imenom. Ove razlike u implementaciji mogu uticati na karakteristike izvršavanja, kao što su korišćenje memorije i skladištenje podataka, kao i na performanse kao što su brzina i preciznost.

Mnoge poslovne ciljeve je najbolje postići građenjem višestrukih vrsta modela koristeći različite algoritme. Često je nemoguće odrediti koja vrsta modela je najbolja, sve dok se ne isproba nekoliko prilaza.

11.2. Neka terminologija

U modelima predviđanja, vrednosti i klase koje predviđamo se nazivaju *odgovor*, *zavisna vrednost* ili *ciljna vrednost*. Vrednosti pomoću kojih se vrši predviđanje nazivaju se *nezavisne promenljive* ili *predviđajuće vrednosti*.

Predviđajući modeli se izgrađuju koristeći podatke čije vrednosti promenljive za odgovor su unapred poznate. Ova vrsta građenja modela se nekad naziva *nadgledajuće učenje*, jer vrednosti koje izračunavamo ili procenjujemo se upoređuju sa poznatim rezultatima. Suprotno, deskriptivne tehnike, kao što jeređanje podataka, o kojem je bilo reči, nekad nazivamo *nenadgledajuće učenje*, jer u ovim slučajevima ne postoje poznati rezultati koji bi vodili algoritam.

11.3. Regresija

Regresijom postojeće vrednosti ispitujemo da bi predvideli buduće vrednosti. U najprostijem slučaju, regresija koristi standardne statističke metode, kao što je linearna regresija. Nažalost, mnogi problemi nisu jednostavne linearne projekcije prethodnih vrednosti. Zbog toga se koriste kompleksnije tehnike (logistička regresija, drva odlučivanja, neuralne mreže...). Postoje modeli koji se koristiti i za regresiju i za klasifikaciju. Na primer CART (Classification And Regression Trees – drva za klasifikaciju i regresiju) algoritam drva odlučivanja može se koristiti za izgradnju i drva klasifikacije i drva regresije. Neuralne mreže, takođe, mogu kreirati obe vrste modela.

11.4. Vremenski intervali

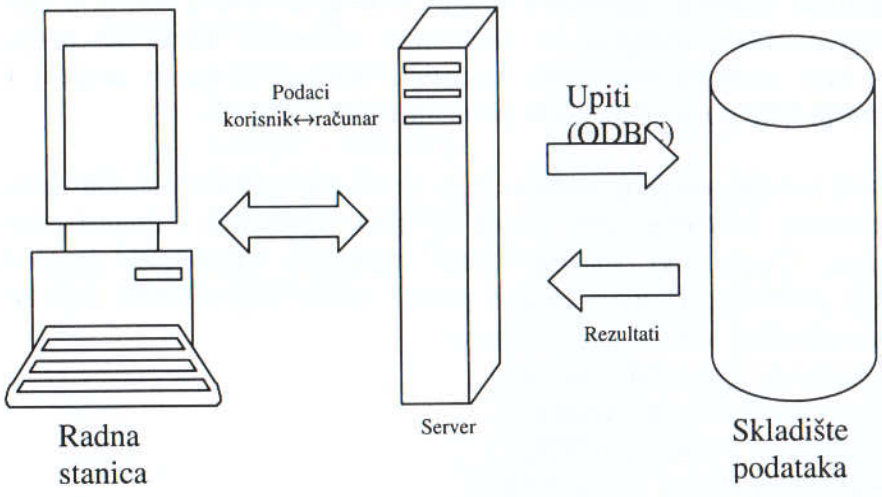
Predviđanje u vremenskim intervalima predviđa buduće nepoznate vrednosti na osnovu postojećih podataka u vremenskim intervalima. Kao i regresija, koristi poznate rezultate da dođe do predviđanja. Modeli moraju uzeti u obzir različite osobine vremena, posebno hijerarhiju u periodima (uključujući različine vrste difinicija kao što je sedmodnevna radna nedelja, mesec, godina...), uticaj godišnjih doba, kalednarski efekti kao što su praznici i posebno se mora uzeti u obzir koliko se daleko u prošlost podaci smatraju bitnim.

12. DATA MINING PROCES

Sam data mining proces se može posmatrati kao celina od 6 delova:

- Razumevanje posla (25%)
- Razumevanje podataka (20%)
- Priprema podataka (25%)
- Modelovanje (10%)
- Procena (20%)
- Puštanje u rad

Vizuelno, data mining proces se može predstaviti kao na Slici 5:



Slika 5.

13. RAZLIČITE DATA MINING TEHNOLOGIJE

Većina proizvoda koristi varijacije algoritama koji su objavljeni u nekim časopisima, sa specifičnim implementacijama koje su usmerene na dostizanje cilja individualnog proizvođača.

Zato je najvažnije imati na umu da ni jedan model ili algoritam ne može i ne bi trebalo da se koristi ekskluzivno. Za svaki problem, priroda podataka će uticati na izbor modela i algoritma. Ne postoji najbolji model ili algoritam. Takođe, biće svakom projektantu potrebno više alata i tehnologija da bi došao do najboljeg modela (za rešavanje određenog problema).

U nastavku će biti napomenuti neki od modela i algoritama koji se koriste u praksi.

13.1. Analitički sistemi

Svi ovi sistemi su, naravno, specifični za određeni domen upotrebe. Kao jedan od najrazvijenijih sistema ove vrste, spomenuće se sistem analize finansijskog tržišta zasnovan na metodi tehničke analize. Tehnička analiza predstavlja skup nekoliko desetina različitih tehnika za predviđanje dinamike cena i odabiranje optimalne strukture investiranja, zasnovane na različitim empirijskim modelima ponašanja tržišta. Ove metode se kreću od veoma jednostavnih do onih koje se baziraju na matematici, kao što je teorija fraktala ili spektralna analiza.

Empirijski model je ugrađen u ovakav sistem na osnovu izlaza, a nije dobijen automatskim učenjem iz prethodnih činjenica. Ovde se javlja kriterijum koji moramo razmotriti: statistički značaj dobijenih modela i njihova interpretativnost, ne mogu se primeniti na ove sisteme.

Ali ovi sistemi, obično, pružaju druge prednosti korisnicima. Oni rade pod specifičnim uslovima. Ovi uslovi su jasni trgovcima i finansijskim analitičarima. Često, ovi sistemi imaju specijalne opcije za pregled finansijskih podataka. U ovoj oblasti postoji veliki broj sistema koji se baziraju na tehničkoj analizi, kao na primer:

- MetaStock (Equis International)
- SuperCharts (Omega Research)
- Candlestick Forecaster (IPTC)
- Wall Street Money (Market Arts)

13.2. Statistički paketi

Sistemski prilaz je od bitnog značaja za uspešan data mining i mnogi su specificirali modele koji bi vodili korisnika (posebno novog u građenju predviđajućih modela) kroz niz koraka koji bi doveo do dobrih rezultata.

Dok većina novijih verzija, poznatih statističkih paketa, koji rade na principima tradicionalnih statističih metoda, su ugrađeni u neke delove data mining-a, njihove glavne analitičke metode ostaju klasične (regresija, uzročna analiza...). Ovi sistemi ne mogu odrediti oblik zavisnosti skrivenih u podacima i zahtevaju da korisnik dođe do hipoteze koju će testirati sistem.

Veliki nedostatak ovih sistema je da oni ne dozvoljavaju da analizu podataka izvrši korisnik koji nije upoznat sa statističkim metodama. Još jedan nedostatak statističkih paketa je da tokom istraživanja podataka, korisnik mora da u više navrata izvrši niz nekih elementarnih operacija. Alati za automatizaciju procesa, ili nisu prisutni (ne postoje), ili zahtevaju programiranje na nekom internacionalnom jeziku.

Ovi nedostaci čine statističke pakete veoma neefikasne u rešavanju kompleksnih problema koji se javljaju u realnom svetu. Primeri ovih sistema:

- SAS (SAS Institute) – **SEMMA**
  - *Sample* – uzorak
  - *Explore* – istraži
  - *Modify* – promeni
  - *Model* – modeluj
  - *Assess* – proceni
- SPSS (SPSS) – **5A**
  - *Assess* – proceni
  - *Access* – pristupi
  - *Analyze* – analiziraj
  - *Act* – delaj (čini)
  - *Automate* – automatizuj
- Statgraphics (Statistical Graphics)
- Statistica (Statsoft)

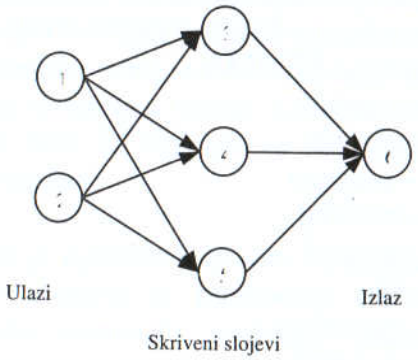
13.3. Neuralne mreže

Ovaj metod je dosta popularan, posebno u oblasti finansija. Neuralne mreže su razvijene 1940-ih na osnovu biološkog nervnog sistema u pokušaju oponašanja procesa.

Neuralne mreže su od posebnog značaja, jer nude efikasno modeliranje velikih i kompleksnih problema u kojima može biti stotine promenljivih prethodnika koje međusobno mogu imati mnoge interakcije.

Neuralna mreža započinje sa ulaznim slojem, gde svaki čvor odgovara promenljivoj pretka. Svaki ulazni čvor je povezan sa svakim čvorom u skrivenom sloju. Svaka od ovih veza među čvorovima ima određenu težinu. Čvorovi u skrivenom sloju mogu biti povezani sa čvorovima u drugom skrivenom sloju ili sa izlaznim slojem. Izlazni sloj se sastoji od jedne ili više izlaznih promenljivih.

Svaki čvor može biti posmatran kao predviđajuća promenljiva ili kao kombinacija predviđajućih promenljivih.



Slika 6. Primer jednostavne neuralne mreže sa jednim skrivenim slojem (na šemi nedostaju težine veza među čvorovima)

Krajnji rezultat neuralnih mreža je matematički model procesa. On se uglavnom koristi za numeričke podatke.

Postoje nesuglasice u vezi efikasnosti neuralnih mreža. Ove nesuglasice potiču većinom iz činjenice da razvoj modela neuralnih mreža je delom umetnost, a delom nauka, što znači da rezultati uglavnom zavise od pojedinca koji je izgradio model. Tj. oblik modela (topologija mreže) i rezultati se razlikuju od jednog do drugog istraživača za iste ulazne podatke. Ali uprkos nesuglasicama u vezi njihove efikasnosti, neuralne mreže se koriste (sa različitim uspehom) u mnogim oblastima.

Postoje mnogi sistemi kod kojih se data mining vrši preko metoda neuralnih mreža, a neki od njih su:

- PolyAnalyst (Megaputer Intelligence)
- BrainMaker (CSS)
- NeuroShell (Ward Systems Group)
- OWL (Hyperlogic)

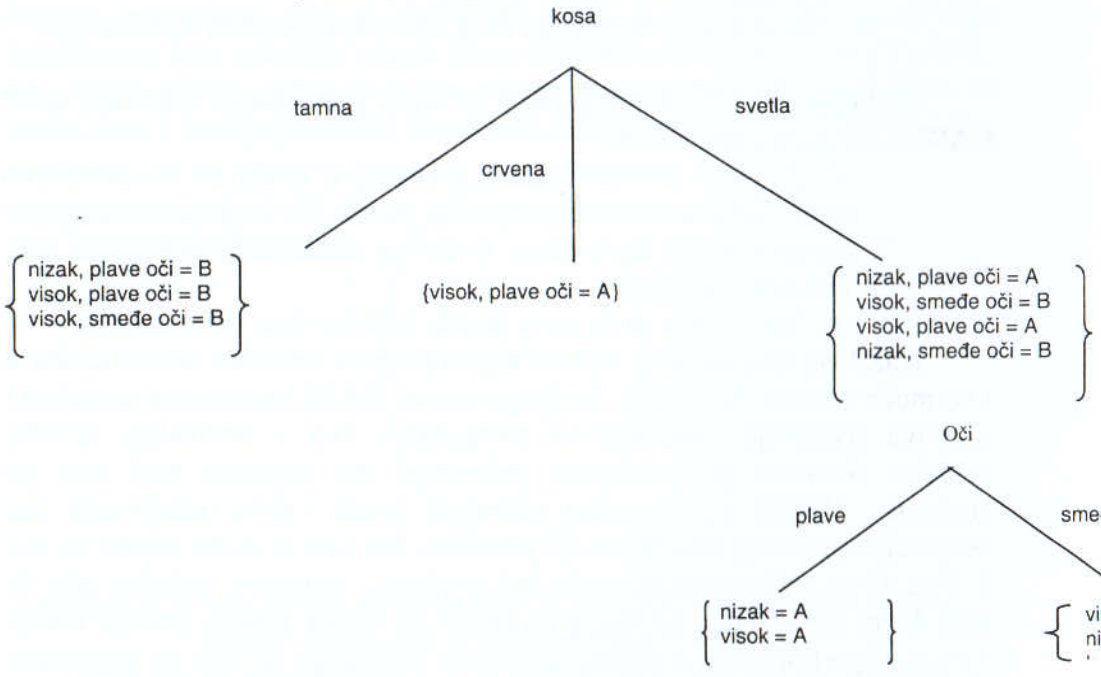
13.4. Drva odlučivanja

Metoda drva odlučivanja je tehnika koja se bazira na podeli podataka i njihovoj raspodeli u vidu drveta. Veoma je efikasna pri klasifikaciji podataka.

Drva odlučivanja čine način predstavljanja niza pravila koje vode do hijerarhijske strukture. Ovu metodu čine pravila klasifikacije tipa "*IF – THEN*" (ako – onda).

Početni čvor se naziva koren. Zavisno od algoritma, svaki čvor može imati dve ili više grana. Ako svaki čvor generiše po dve izlazne grane, takvo drvo nazivano: binarno drvo. Svaka grana vodi ili do drugog čvora odlučivanja ili do dna drveta, koji se naziva list.

Navigacijom kroz drvo odlučivanja može se odrediti vrednost ili klasa svakom podatku koji dovodi do odluke kojom granom dalje krenuti, počevši od korena i pomerajući se ka daljim čvorovima sve dok se ne dođe do lista stabla. Svaki čvor sadrži podatke koji mu govore kojom granom će se dalje kretati. Drvo odlučivanja može biti pretvoreno u niz pravila.



Slika 7. Drvo odlučivanja

Negativno kod drva odlučivanja bi bilo to što se odluke donose "usput" na osnovu ograničenih informacija. Ovaj prilaz može ostaviti bitna pravila neotkrivena, zbog odluka koje se donose rano u procesu podele podataka koje mogu dovesti do toga da se dobra pravila ne otkriju kasnije. Takođe, što je drvo "razgranatije", podaci su podeljeni na sve manje skupove. Iz malih skupova podataka se teško mogu dobiti dobra rešenja.

Mora se naglasiti da, uprkos nedostacima, zbog svoje jednostavnosti i jasnosti (kod grafičkog prikaza) ova metoda se veoma mnogo koristi. Neki od sistema koji omogućavaju data mining metodom drva odlučivanja su:

- PolyAnalyst (Megaputer Intelligence)
- C5.0 (Rule Quest)
- Clementine (Integral Solutions)
- SIPINA (University of Lyon)
- IDIS (Information Discovery)

13.5. Multivariantni prilagođavajući regresioni splajnovi (MARS)

Sredinom 1980-ih godina jedan od kreatora CART-a, Jerome H. Friedman, razvio je metod MARS.

Glavne mane koje je hteo da eliminiše (u odnosu na CART) su:

- Diskontinualna predviđanja
- Zavisnost podela od prethodnih podela
- Smanjena interpretacija zbog interacija, posebno onih na vrhu.

Osnovna ideja MARS algoritma je dosta jednostavna. Ukratko, mane CART-a su uklonjene pomoću:

- Zamene diskontinualnog grananja u čvoru sa kontinualnom tranzicijom modelovanom parom pravih. Na kraju procesa izgrade modela, prave na svakom čvoru su zamenjene funkcijama koje nazivamo splajnovi.
- Ne zahteva se da nova podela bude zavisna od prethodne.

Nažalost, ovo znači da MARS algoritam gubi strukturu drveta CART-a i ne može dovesti do pravila. Sa druge strane, MARS automatski pronalazi i izlistava najbitnije predviđajuće promenljive kao i interakcije između predaka. Rezultat je automatski nelinearni alat regresije koji radi po koracima. MARS, kao i većina neuralnih mreža i drva odlučivanja ima tendenciju suvišnog klasifikovanja podataka. Na ovo se može uticati na dva načina. Prvo, ručna provera može biti izvršena i algoritam podešen tako da daje dobra predviđanja na skupu na kojem se testira. Drugo, postoje mnogi parametri podešavanja u samom algoritmu koji mogu dovesti do unutrašnje provere.

13.6. Smanjenje pravila

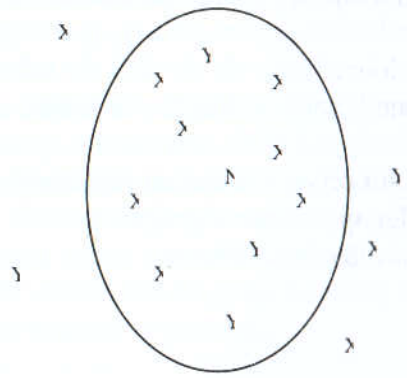
Smanjenje pravila je metoda derivacije skupa pravila u cilju klasifikacije podataka. Iako drva odlučivanja mogu dovesti do niza pravila, metoda smanjenja pravila generiše niz nezavisnih pravila koja retko formiraju drvo. Smanjenje pravila ne "sili" podelu na svakom nivou. Zbog toga je u mogućnosti da pronađe nove i nekad bolje šablone za klasifikaciju.

Za razliku od drveta, pravila generisana ovom metodom retko pokrivaju sve moguće situacije. Takođe, pravila mogu nekad da se sukobe u njihovim predviđanjima, u kojem slučaju je neophodno odlučiti koje pravilo slediti. Da bi se ovo rešilo, često se pravilima dodeljuju vrednosti sigurnosti i u slučaju konflikta koristi se ono pravilo koje ima ovu veću vrednost.

13.7. K-najbliži sused ili Zaključivanje na osnovu prošlosti (Memory Based Reasoning (MBR))

Glavna ideja ove metode je veoma jednostavna. Predviđanje budućih situacija ili donošenje ispravne odluke, ovi sistemi realizuju pronalaženjem najbliže prošlosti trenutne situacije i donose najbolje rešenje koje je bilo ispravno u prošlosti (za najsličniji problem).

Pokušavajući rešiti nove probleme, ljudi često posmatraju situacije u sličnim problemima koje su prethodno rešili. K-najbliži sused je tehnika klasifikacije koja odlučuje u koju klasu smestiti novi slučaj ispitivajući neki broj "K" u većini sličnih slučajeva suseda. Prebrojava se broj slučajeva za svaku klasu i dodeljuje se novi slučaj istoj klasi kojoj većina suseda pripada.



Slika 8. N je novi slučaj. Pripada klasi X, zato što se sedam X-ova nalazi unutar elipse, koji brojčano nadjačavaju tri Y.

Prvo što se mora uraditi u K – najbliži sused metodi je pronalaženje mere za udaljenost među atributima u podacima i onda njeno izračunavanje. Ovo je lako za numeričke podatke, dok se druge vrste promenljivih moraju

specijalno obrađivati. Kad se jednom odredi razdaljina među slučajevima, odabira se niz već klasifikovanih slučajeva koja će se koristiti kao osnova za klasifikovanje novih. Odlučuje se koliko je velika okolina u okviru koje se vrši poređenje i kako računati susede (npr. može se dati veća vrednost bližem susedu nego daljem).

Ova metoda opterećuje računar, jer se vreme izračunavanja povećava za faktorijel ukupnog broja tačaka (slučajeva). Ovaj model se veoma lako shvata kad postoji mali broj početnih promenljivih.

Nedostatak ovog modela bi bio u tome što ovi modeli donose odluke na osnovu celokupnih podataka koji postoje. Zato je nemoguće odrediti koji specifični faktori su uticali na predviđanje.

Neki od sistema koji omogućavaju data mining ovom metodom su:

- PolyAnalyst (Megaputer Intelligence)
- KATE tools (Acknosoft)
- Pattern Recognition Workbench (Unica)

13.8. Nelinearne regresione metode

Ove metode se baziraju na traženju zavisnosti ciljne promenljive u odnosu na ostale promenljive u obliku funkcije nekog određenog oblika. U nelinearnoj regresionoj metodi, zavisnost među promenljivama koje se posmatraju, se predstavlja u obliku polinoma. Ove metode pružaju rešenja koja su većeg statističkog značaja, u odnosu na rešenja koje pružaju neuralne mreže. Dobijena polinomska funkcija je bolja za analizu i prikaz činjenica (koje su u stvarnosti još uvek suviše kompleksne za to).

Ovaj metod ima dobre šanse da će doći do tačnog rešenja u oblastima finansijskog tržišta ili medicinskog dijagnosticiranja.

Primer sistema koji koriste nelinearnu regresionu metodu:

- PolyAnalyst (Megaputer Intelligence)
- NeuroShell (Ward Systems Group)

13.9. Diskriminantna analiza

Diskriminantna analiza je najstarija matematička tehnika klasifikacije. Prvi put je objavljenja 1936. godine od strane R. A. Fisher-a u cilju klasifikacije botaničkih podataka u tri vrste. Obuka za korišćenje ove metode je veoma jednostavna. Ova tehnika je veoma osetljiva na šablone u podacima. Diskriminantna analiza se često koristi u određenim disciplinama kao što je medicina, socijalne nauke i neke oblasti biologije.

Diskriminantna analiza nije popularna u data mining-u iz tri glavna razloga:

- Pretpostavlja da su sve predviđajuće promenljive normalno raspoređene, što ne mora biti slučaj,
- Nepoređane, po nekom redosledu, predviđajuće promenljive ne mogu biti korištene,
- Granice koje razdvajaju klase su sve linearne forme, a podaci često ne mogu biti razdeljeni na tako jednostavan način.

Skorašnje verzije diskriminantne analize odnose se na ove probleme dozvoljavajući granicama da budu kvadratnog oblika, kao i linearnog, što značajno povećava osetljivost u nekim slučajevima.

13.10. Generalizovani modeli dodavanja (GAM)

Postoji klasa modela koji proširuju i linearnu i logističku regresiju, poznati kao generalizovani modeli dodavanja (GAM). Nazivaju se dodavajući, jer se pretpostavalja da model može biti zapisan kao suma (verovatno) nelinearnih funkcija, za svakog pretka. GAM može biti korišten i za regresiju i/ili za klasifikaciju (sa binarnim predviđanjem).

Predviđajuća promenljiva može biti bilo koja funkcija prethodnika dokle god nema prekidanja u koracima. GAM koristeći moć računara umesto teorije ili znanja će proizvesti rezultat sumirajući veze među pretcima. Umesto procenjivanja velikog broja parametara, kao što čine neuralne mreže, GAM procenjuje vrednost izlaza za svaku vrednost ulaza – jedna tačka, jedna procena. Kao i neuralne mreže, GAM generiše krivu automatski, odabirajuću vrednost kompleksnosti na osnovu podataka.

13.11. Evolucionarno programiranje

Trenutno, ovo je najmlađa i (po mnogim stručnjacima) veoma obećavajuća oblast data mining-a. Glavna ideja ove metode je da sistem automatski formuliše hipoteze o zavisnosti ciljnih vrednosti (onih koje nas zanimaju) u odnosu na druge promenljive.

Proces dobijanja internih programa (hipoteza) je organizovan kao evolucija u svetu, svih mogućih programa (ova metoda bi se mogla nazvati prethodnikom metode genetičkog algoritma). Kad sistem pronađe hipotezu koja opisuje posmatranu zavisnost razumno dobro, počinje da traži razne modifikacije programa, u cilju pronalaženja najboljeg kroz proces, da bi se povećala preciznost predviđanja. Na ovaj način sistem razvija broj genetičkih linija programa koji se takmiče među sobom u preciznosti predstavljanja tražene zavisnosti.

Kad se dođe do programa (hipoteze) sa traženom tačnošću, specijalni modul sistema prevodi otkrivenu zavisnost sa internog jezika u oblik koji čovek može lako razumeti (matematičke formule, tabele...). Ovo omogućava korisniku da, pomoću veoma bitne unutrašnje kontrole i kontrole dobijene zavisnosti utiče na proces i omogućava grafički prikaz rezultata. Kontrola statističkog značaja dobijenih rezultata se ostvaruje kroz broj efikasnih modernih statističkih metoda, kao na primer, pomoću metode slučajnog testiranja podataka.

Postoje razni alati koji omogućavaju data mining ovom metodom, a jedan od njih je *PolyAnalyst* (od proizvođača *Megaputer Intelligence).*

13.12. Genetički algoritmi

Analiza podataka nije glavna oblast primene genetičkih algoritama. Genetičke algoritme treba posmatrati kao tehniku rešavanja raznih kombinatoričkih problema i/ili problema optimizacije. Ipak, genetički algoritmi su našli svoje mesto među standardnim modernim instrumentima za data mining.

Genetički algoritmi se koriste da vode proces učenja data mining algoritama, kao i neuralne mreže. Genetički algoritam se ponaša kao metoda za izvršavanje usmerene pretrage dobrih modela u nekom domenu.

Nazivaju se genetičkim algoritmima zato što prate šablon biološke evolucije u kojoj se članovi jedne generacije (modela) takmiče za što bolje karakteristike do sledeće generacije (modela) sve dok se najbolji model ne pronađe.

Ovaj model radi na principu:

1. Svaka karakteristika rešenja koje želimo se predstavlja nizom numeričkih ili nenumeričkih parametara.
2. Svi podaci koji se razmatraju predstavljaju se nizom parametara i oni predstavljaju hromozome koji dalje evoluiraju.
3. Evolucija se odvija pomoću tri mehanizma:
   - Selekcijom – odabira se niz hromozoma koji imaju najbolje osobine za pronalaženje rešenja
   - Ukrštanjem – proizvode se novi hromozomi (sledeća generacija) od postojećih tako što im se mešaju osobine
   - Mutacijama – slučajne promene gena u nekim organizmima u populaciji.
4. Nakon određenog broja kreiranja novih generacija, dobijamo rešenje koje se više ne može poboljšati. Ovo rešenje se uzima kao najbolje.

Genetički algoritmi su interesantan prilaz optimizaciji modela, ali dodaju mnogo kompjuterske obrade. Takođe, njihova mana (što se tiče data mining-a) se ogleda u tome što samo stručnjak može odrediti kriterijume za odabir hromozoma i efikasno formulisati problem.

Neki primeri korišćenja genetičkih algoritama u data mining-u su:

- PolyAnalyst (Megaputer Intelligence)
- GeneHunter (Ward Systems Group)

14. ODABIR DATA MINING PROIZVODA

Postoji mnogo data mining proizvoda na tržištu, a svakodnevno se pojavljuju novi. Svaki od ovih proizvoda ima svoje prednosti (i mane). Može li se odrediti koji je najbolji?

Odgovor na ovo pitanje je subjektivan. On zavisi od prethodnog iskustva, znanja i veština koje *data miner* poseduje, a takođe i zavisi od vrste problema koji želimo rešiti (sa kojim podacima radimo, u kakvom su obliku podaci koji se obrađuju, u kakvom obliku želimo dobiti izlazne podatke...).

Ovo su neki od proizvoda za data mining:

- IBM – Intelligent Miner
- SAS – Enterprise Miner
- SPSS – Clementine
- Megaputer Intelligence – PolyAnalyst
- Microsoft – Analysis Server (deo aplikacije *SQL Server 2000*)

...i još proizvoda od "manjih" proizvođača

14.1. Kategorije data mining proizvoda

Podaci se ne mogu deliti na jednostavne kategorije kao što su na primer (visoko i nisko) zato što su proizvodi prebogati u funkcionalnosti da bi delili podatke u samo jednoj dimenziji.

Postoje tri glavne vrste proizvoda za data mining:

- Alati koji pomažu u analizi za OLAP (on-line analitički procesi) – pomažu OLAP korisnicima da otkriju najbitnije dimenzije i segmente na koje treba da obrate pažnju. Vodeći alati u ovoj oblasti: Business Objects, Business Miner i Cognos Scenario.
- Čisti data mining proizvodi – horizontalni alati usmereni na data mining analize koje rešavaju različite vrste problema. Vodeći alati u ovoj oblasti: IBM Intelligent Miner, Oracle Darwin, SAS Enterprise Miner, SGI MineSet, SPSS Clementine
- Analitičke aplikacije koje implementiraju specifične poslovne procese čiji je data mining sastavni deo. Odabir proizvoda za izradu analitičke aplikacije se odnosi i na horizontalne alate i na data mining komponente analitičke aplikacije.

Bez obzira koliko je opširna lista mogućnosti izbora proizvoda pomoću kojih razvijate data mining, ništa ne može zameniti iskustvo u radu sa njima. Iako je lista uslova koje treba da zadovoljavaju proizvodi bitna, presudiće konkretne mogućnosti nekog proizvoda, u zavisnosti od vaših zahteva. Korišćenje proizvoda u pilot projektu (testiranje proizvoda) je neophodno da bi se odredilo da li je baš taj proizvod ono što zahtevate vi, vaša organizacija i, naravno, problem koji rešavate.

14.2. Osnovne karakteristike data mining proizvoda

U zavisnosti od specifičnih okolnosti – arhitekture sistema, resursima osoblja, veličine baze podataka, kompleksnosti problema – neki data mining proizvodi će davati bolje učinke od drugih. Procenjivanje data mining proizvoda uključuje procenu njegovih mogućnosti u broju ključnih oblasti:

Arhitektura sistema – da li je dizajniran da radi na jednom računaru ili u mreži? Mora se naglasiti da jačina računara na kojem proizvod radi nije pouzdan pokazivač kompleksnosti problema koje proizvod može rešiti. Veoma sofisticirani proizvodi koji mogu rešiti kompleksne probleme i zahtevaju vešte korisnike mogu raditi na samo jednom računaru, a mogu raditi i u velikim mrežama računara.

Priprema podataka – priprema podataka je vremenski najzahtevniji deo data mining-a. Sve što proizvod može da vam automatizuje u ovom delu projektovanja, veoma može ubrzati razvoj modela. Neke od funkcija koje proizvod može da pruži može uključivati:

- Čišćenje podataka – kao što je traženje nestalih podataka ili idetifikovanje povredu integriteta
- Opis podataka – kao što su brojanje redova, brojanje vrednosti ili distribucija vrednosti
- Transformacije podataka – kao što je dodavanje novih kolona, izračunavanja u okviru postojećih kolona, grupisanje promenljivih u nizove...
- Odabir podataka za izgradnju modela ili za kreiranje nizova podataka za testiranje ili procenu modela
- Odabir predaka iz prostora promenljivih i identifikacija kolinearnih kolona

Pristup podacima – Neki alati za data mining zahtevaju da podaci budu izdvojeni iz baze podataka, dok drugi alati direktno pristupaju podacima u bazi. Data mining alat će imati prednost ako može direktno da pristupi bazi podataka koja je na nekom serveru koristeći SQL, jer tako može povećati učinak i iskoristiti mogućnosti servera, kao što je paralelni pristup bazi podataka. Međutim, ni jedan proizvod ne podržava različite servere za baze podataka, tako da se mora koristiti jedna vrsta povezivanja za sve baze koje ćemo koristiti (najuobičajenije je to Microsoft-ov ODBC – Open DataBase Connectivity).

Algoritmi – karakteristike algoritama data mining proizvod koristi da bi korisnik mogao da odredi da li odgovara karakteristikama problema (npr.: koliko brzo algoritam radi sa novim podacima, kako se ophodi prema promenljivima za predviđanje, koliko se brzo izvršavaju...). Još jedna bitna osobina algoritma je: koliko je osetljiv na nepravilnosti. Pravi podaci imaju nebitne kolone, redove koji se ne uklapaju u šablon, nedostajuće ili netačne vrednosti. Koliko nepravilnosti alat za izgradnju modela može da izdrži pre nego što mu tačnost opadne? Drugim rečima, koliko je algoritam osetljiv na podatke koji nedostaju, koliko tačne šablone otkriva kad se susretne sa čudnim i netačnim podacima? U nekim slučajevima jednostavnim dodavanjem još podataka mogu se smanjiti nepravilnosti, ali nekad su i podaci koje dodajemo dosta nepravilni, tako da to može dovesti do naknadnog smanjenja tačnosti rezultata. Jedan od glavnih delova pripreme podataka se odnosi baš na smanjenje nepravilnosti u podacima sa kojima se radi.

Kompatibilnost sa drugim proizvodima – postoje mnogi alati koji mogu pomoći u razumevanju podataka pre izgradnje modela i koji mogu pomoći u predstavljanju rezultata modela. Ovde se misli na tradicionalne alate upita i izgradnje izveštaja, grafičke i alate vizualizacije i OLAP alate. Data mining softver koji pruža lak način rada sa proizvodima drugih proizvođača, pruža korisniku dodatne načine da izvuče maksimum iz procesa otkrivanja znanja.

Procena i interpretacija modela – proizvodi mogu pomoći korisniku da razume rezultate pružajući neke merne vrednosti (preciznosti, značaja...) u oblicima kao što su matrice, grafikoni, dozvoljavajući korisniku da izvrši analizu rezultata.

Primena modela – rezultati modela mogu biti upisani direktno u bazu podataka ili se oni mogu dobiti izdvajanjem podataka iz baze. Kad treba primeniti model na nove podatke, kako se dobijaju, obično je neophodno model isprogramirati pomoću API ili koda generisanog pomoću data mining alata. U svakom slučaju, jedan od ključnih problema je adaptacija modela da može obraditi nove podatke, često uz neke transformacije, da bi napravila valjana predviđanja. Mnogi data mining alati ostavljaju ovaj posao kao izdvojenu celinu koju treba da odradi korisnik ili programer.

Merljivost – koliko je efektivan alat kad obrađuje velike količine podataka – i redove i kolone – i sa koliko sofisticiranim tehnikama procene? Ovi izazovi zahtevaju mogućnost korišćenja što jačeg hardvera. Koje vrste paralelne obrade podržava alat? Koliko dobro radi data mining alat kad se broj procesora poveća? Podržava li paralelni pristup podacima?

Data mining algoritmi napisani za jednoprocesorski računar neće automatski brže raditi na paralelnim mašinama. Oni moraju biti prepravljeni, da bi iskoristili paralelne procesore. Postoje dva osnovna načina da se ovo postigne. U jednom slučaju, nezavisni delovi aplikacije se dodeljuju različitim procesorima. Što je više procesora, više se delova može istovremeno obraditi. Drugi način je da podelimo model na zadatke, izvršimo te zadatke na posebnim procesorima i posle ih iskombinujemo, da bi dobili odgovor.

Korisnički interfejs – neki proizvodi nude GUI (grafički interfejs sa korisnikom), dok neki nude neku vrstu programskog jezika. Neki proizvodi nude data mining API koji se može koristiti sjedinjenjem u programske jezike, kao što su C, Visual Basic ili Power Builder. Mnoge bitne tehničke odluke se moraju doneti pri pripremi podataka, selekciji podataka i odabiru strategije za modelovanje, što nas dovodi do zaključka da čak i grafički interfejs koji dosta pojednostavljuje izradu modela, zahteva stručnjaka koji bi pronašao najefektivniji model.

15. NEUSPEŠNI DATA MINING

Naravno, nekad data mining proces može biti neuspešan. Postoje mnogi razlozi za to, a među njima su i:

- Nerazumevanje u vezi problema koji treba da se reši, između rukovodioca kompanije i eksperta (u oblasti u kojoj je problem), kao i/ili između data miner-a
- Data mining alat ne može da obradi vrstu podataka koja mu je na raspolaganju
- Podaci nisu adekvatno odabrani
- Javlja se suviše "loših" podataka (netačne vrednosti, vrednosti kojih nema...)

Neke greške ne vidimo (dok ne bude kasno). Sve se slaže, sve je tu, a opet, rezultati nemaju smisla. Zato svako ko se prihvati posla data mining-a mora biti spreman na ovakve rezultate i situacije kad će morati da se vraća na prethodne etape posla, a nekad, možda (u zavisnosti od toga gde je greška nastala) i na početak procesa.

16. RAZLOZI SVE VEĆE POPULARNOSTI DATA MINING-A

16.1. Povećanje količine podataka

Glavni razlog neophodnosti automatizovanja kompujerskih sistema, u cilju inteligentne analize podataka, je velika količina postojećih i svakodnevno pojavljivanje novih podataka, koje treba obraditi. Količina podataka koja se svakodnevno prikupi zavisi od vrste orgnanizacije koja ih prikuplja (poslovna, naučna, državna...). Po informacijama GTE istraživačkog centra, samo naučne organizacije prikupe svakodnevno približno 1 TB (terabajt) novih podataka. A zna se da naučne organizacije nisu vodeće po količini podataka koje se sakupljaju. Nemoguće je nadati se da će se standardnim ("ručnim") analizama pregledati tako velike količine podataka

16.2. Ograničenje ljudskih analiza

Dva druga problema koji se pojavljuju kad čovek analizira podatke, su neprilagođenost ljudskog mozga u pretraživanju kompleksnih višezavisnih povezanosti među podacima i nedostatak objektivnosti u takvim analizama. Čovek (ekspert) je uvek pod uticajem prethodnih iskustava (poslovanja sa drugim firmama, rad u drugom okruženju, sa drugim problemima...). Nekad prethodna iskustva pomažu u radu, a nekad ograničavaju eksperta, ali je nemoguće da on zaboravi prethodna iskustva.

16.3. Niska cena mašinskog učenja

Dodatna prednost korišćenja automatizovanog data mining sistema, je da proces mnogo manje košta, od zapošljavanja manje armije visoko obrazovanih (i plaćenih) profesionalaca (statističara). Data mining ne isključuje čoveka iz procesa rešavanja zadatka potpuno, ali značajno pojednostavljuje posao.

17. OBLASTI PRIMENE DATA MINING-A

Data mining je sve popularniji zbog velikih rezultata koje može pružiti. Ovi alati se mogu iskoristiti da se kontroliše cena, kao i da se poveća prihod.

Kome je potreban data mining? Potreban je svakome ko prikuplja veću količinu podataka. Nema razlika o kojoj se oblasti radi, gde god ima podataka, data mining postaje nužnost.

Mnoge organizacije koriste data mining da pomognu u upravljanju svih faza "potrošačkog životnog ciklusa", uključujući i pridobijanje novih mušterija, povećanje prihoda od postojećih mušterija i zadržavanje dobrih mušterija. Određivanjem karakteristika dobrog potrošača, svaka kompanija može da se usredsredi na potencijalne potrošače koji imaju slične karakteristike. Takođe, određivanjem karakteristika potrošača koji su prestali da koriste proizvode kompanije, ista se može usredsrediti na potrošače koji imaju slične karakteristike, jer je manji trošak za kompaniju da zadrži potrošača, nego da pridobije novog.

Data mining nalazi primenu u širokom spektru industrija.

U nastavku su navedene samo neke od mogućih oblasti u kojima se može primeniti data mining:

- Bankarstvo (odobrenje za hipoteku, krediti, analiza i otkrivanje prevara...)
- Finansije (analiza i predviđanje poslovnog uspeha, analiza vrednosti deonica...)
- Osiguranje (predviđanje bankrotstva, analiza rizika, predviđanje verovatnoće naplate osiguranja, otkrivanje ljudi koji isceniraju nezgodu ili sami naprave štetu, da bi naplatili osiguranje...)
- Internet Marketing (usmereno oglašavanje, usmerene ponude za kupovinu/prodaju...)
- Vlada (procena pretnji, terorističkih napada...)
- Proizvodnja (kontrola kvaliteta, organizacija održavanja, automatizovanje sistema...)

- Medicina (analiza gena, epidemiološke analize, dijagnosticiranje, retrospektivne analize lekova, analiza faktora rizika...)
- Astronomija (klasifikacija astronomskih objekata)
- Naučno istraživanje (uopšteno modelovanje svih vrsta)

18. ZAKLJUČAK

Data mining veoma obećava, kad je reč o pomoći organizacijama da otkriju šablone, skrivene u njihovim podacima, koji mogu biti iskorišteni da se predvidi ponašanje potrošača, proizvoda i procesa. Takođe, kao što je rečeno, data mining ima veoma široku primenu i ne treba ga vezati isključivo za poslovnu primenu.

Moderni kompjuterski data mining sistemi pronalaze, formulišu i testiraju hipoteze, iz podataka koje analiziraju, a na osnovu pravila po kojima sistem radi. Kad se pronađu zaključci (znanje), koje je data mining proces otkrio, to znanje se treba upotrebiti da bi se donele mudre i dobro informisane poslovne odluke.

Data mining alate treba da koristi osoba, tj. tim ljudi, koji razumeju posao, podatke i uopšte, prirodu analitičkih metoda, koji su korišteni. Realistična očekivanja mogu da se kreću od povećanja prodaje od smanjenja troškova.

Izgradnja modela je samo jedan korak u otkrivanju znanja. Veoma je bitno pravilno sakupiti i pripremiti podatke i, naravno, uporediti model sa stvarnim svetom. Najbolji model se najčešće otkrije nakon kreiranja nekoliko različitih modela, ili različitim metodama i/ili algoritmima.

Odabir pravog proizvoda za data mining znači pronaći alat sa dobrim osnovnim mogućnostima, sa interfejsom koji odgovara nivou veštine ljudi koji će ga koristiti i sa opcijama obrade podataka koje su bitne za specifični problem koji treba da se reši.

19. LITERATURA:

1. **Introduction to Data Mining and Knowledge Discovery**, third edition, Two Crows Corporation, 1999.
2. **Nuggets and Data Mining**, white paper, *Michael Gilman PhD*, Data Mining Technologies Inc. Bethpage, NY 2002.
3. **Analysis of Data Mining Algorithms**, *Karuna Pande Joshi*, 1997.
4. **Data Mining**, *Michael I. Samos*, Institute for eComerce, 2000.
5. **Data Mining**, *Chris Williams*, Institute for Adaptive and Neural Computation, Edinburgh, UK 2002.
6. **Data Mining And High Performance Computing**, *Cristian Aflori*, Computer Engineering Faculty, Technical University Iasi – Romania
7. **An Introduction to Data Mining**, CSIRO Australia, 1999
8. **Data Mining for Web Intelligence**, *Jiawei Han Kevin, Chen-Chuan Chang*, University of Illinois at Urbana-Champaign, 2002
9. **Knowledge discovery in science as opposed to business**, *Brian J. Read*, CLRC Rutherford Appleton Laboratory, UK
10. **A System for Data Mining over Local and Wide Area Clusters and Super-Clusters**, *S. Bailey, R. Grossman, H. Sivakumar, A. Turinsky*, National Center for Data Mining, University of Illinois at Chicago, 1999
11. **What is Data Mining?**, *Simona Despa*, StatNews #55, Cornell University, Office of Statistical Consulting, March 2003.
12. **Workshop on Data Mining Standards, Services and Platforms**, *Robert Grossman*, National Center for Data Mining, University of Illinois at Chicago, 2003.
13. **Mining Very Large Databases to Support Knowledge Exploration**, WhiteCross White Paper: Version 1, 2001.
14. **Introduction to Data Mining,** *Stijn Viaene,* KBC Insurance Research Chair, Boston, MA, USA, 2002
15. ***Internet izvori:***
    a. www.whitecross.com
    b. www.microsoft.com
    c. www.megaputer.com
    d. www.thearling.com
    e. www.comp.nus.edu.sg
    f. www.spss.com
    g. www.webtechniques.com