Uvod u Veb i Internet tehnologije

Filip Marić

10. januar 2016

Predgovor

Ovaj tekst predstavlja skriptu za predmet „Uvod u Veb i Internet tehnologije" na Matematičkom fakultetu u Beogradu. Materijal je namenjen, pre svega studentima koji slušaju pomenuti kurs, ali svima drugima zainteresovanim za detaljnije i sistematsko upoznavanje sa softverskim tehnologijama koji se koriste u okviru Interneta i Veba.

Naravno, s obzirom da je Veb izrazito popularan, pregled svih relevantnih tehnologija nije moguće napraviti u okviru jednog udžbenika, tako da su određene odluke i izbori morale biti napravljene na samom početku. Prednost je data slobodnom softveru tehnologijama otvorenog kôda, tako da materijal uglavnom pokriva tzv. *LAMP (Linux-Apache-MySQL-PHP)* platformu.

Materijal je izlagan pristupom „odozdo-naviše" (eng. bottom-up), te samim tim, možda nije najpogodniji za prvo upoznavanje sa materijalom. Sa druge strane, smatra se da veliki broj studenata i potencijalnih čitalaca ovog materijala već poseduje određena elementarna znanja o računarskim (pa i o Veb) tehnologijama. Kreiranje Veb stranica, kao proces donekle jednostavniji od programiranja, često je primamljiv i za amatere u računarstvu tako da je moguće naići na veliki broj „instant" rešenja koji materiju prikazuju parcijalno i nesistematski. Za razliku od ovakvih receptaških materijala (eng. „cookbook"), u ovom materijalu se sve tehnologije izlažu, postupno, prilično detaljno i pridržavajući u što većoj meri relevantnih standarda, specifikacija i zvaničnih preporuka. Autor je mišljenja da određeni delovi materijala mogu da služe i kao referentni priručnik tokom realnog rada. Naravno, zbog obima materijala, vršena su određena uprošćavanja tako da su neki delovi izostavljeni i moguće ih je naći pre svega u zvaničnim specifikacijama određenih tehnologija.

Tokom izrade Veb stranica, u praksi se koriste i mnogi WYSIWYG alati (npr. Macromedia Dreamveawer, Microsoft Front Page). Iako ovakvi alati mogu značajno da ubrzaju rad, oni ne mogu da odmene detaljno poznavanje Veb tehnologija. Iz ovog razloga, odlučeno je da se oni ne pominju u okviru materijala, već da se čitaocima prepusti samostalno upoznavanje sa njima, tek nakon što se izloženi materijal savlada.

Pošto se tehnologije, naročito u oblasti informacionih tehnologija, izrazito brzo menjaju, opasnost za svaki materijal koji prikazuje ove tehnologije je da veoma brzo zastareva. Zbog toga izložene podatke treba uzeti sa rezervom i imati na umu trenutak nastajanja ovog materijala — 2010. godina.

Veliki problem prilikom pisanja materijala je postojanje različitih terminologija koje se koriste u ovoj oblasti računarstva. Svesno, odabran je pristup koji izbegava korišćenje anglicizama i većina termina je korišćena u prevedenom obliku (iako je autor svestan da u praksi anglicizmi, definitivno, dominiraju). Tako je npr. za engleski termin *web browser*, umesto rogobatnog anglicizma bra-

uzer (zašto ne brauser ili brovzer?) korišćen termin *pregledač Veba*. Osnovna motivacija za ovaj pristup je pokušaj uklapanja terminologije u postojeći korpus jezika (pošto se, na primer, lekovi kupuju u apoteci, knjige iznajmljuju u biblioteci, zašto se podaci ne bi skladištili u datoteci nego u „fajlu"?). Engleski termini su uvek navedeni prilikom prvog pojavljivanja termina, dok je u dodatak uključen i terminološki rečnik.

Na ovom mestu ću se rado zahvaliti svim studentima i ostalim čitaocima koji mi budu ukazali na greške i propuste u materijalu (kojih, svakako, ima).

U Beogradu, Filip Marić
Februar, 2011.

Sadržaj

Glava 1

Uvod u računarske mreže

U današnje vreme nezamislivo je korišćenje računara koji nisu na neki način povezani sa drugim računarima. Izgradnja računarskih mreža i naročito nastanak i razvoj Interneta i njegovih servisa poput Veba dovele su do izrazito značajnog proširenja kruga korisnika računara i promene uloge računara u odnosu na ranije. Može se slobodno reći da je pojava savremenih računarskih mreža dovela do revolucije slične onoj koju je pojava parne mašine izazvala u XVIII veku. Sve je veći broj različitih uređaja koji postaju umreženi i vremenom se pojavljuje sve više vrsta usluga koje nam nudi mrežno okruženje.

1.1 Uloga računarskih mreža i način rada u mreži

Možemo reći da su osnovne uloge računarskih mreža:

Komunikacija - korišćenjem računara ljudi danas komuniciraju putem elektronske pošte, instant poruka, ćaskanja (eng. chat), video konferencija, itd.

Deljene informacija i podataka - u mrežnom okruženju, moguće je pristupati informacijama koje se nalaze na drugim računarima u okviru mreže. Moguć je prenos podataka, najčešće u obliku preuzimanja različitih vrsta datoteka. Prenos informacija je moguće vršiti i okviru lokalnih mreža, obično u okviru jedne kompanije, kao i u okviru globalne svetske mreže. Internet i Veb se danas smatraju glavnim izvorima informacija.

Deljene softvera - Korisnici povezani u mrežu mogu da koriste mnoge usluge koje im pruža softver koji radi na računarima u okviru mreže. Na primer, putem Veba je moguće kupovati, rezervisati karte i slično. Takođe, na primer, distribuiran i paralelizovan sofver se može izvršavati na više povezanih računara čime se mogu ubrzati zahtevna izračunavanja.

Deljene hardverski resursa - u mrežnom okruženju, moguće je zajedničko korišćenje hardvera (npr. štampača, skenera) od strane više korisnika. Često se pokazuje da je kompaniji jeftinije da instalira jedan kvalitetniji uređaj, koji kroz računarsku mrežu može biti simultano korišćen od strane više korisnika, nego da svakog korisnika oprema zasebnim uređajima.

Računarski resursi u mreži mogu da budu raspoređeni na različite načine tako da obezbeđuju različite načine izvršavanja poslova. Navedimo neke od najčešćih:

Centralizovana obrada - svi poslovi se izvršavaju na jednom centralnom računaru, dok se ostali čvorovi koriste samo kao terminali za unos podataka i prikaz rezultata. Ovaj način rada je bio svojstven za rane računarske mreže i vremena velikih centralnih računara.

Klijent-server okruženje - jedan računar ima ulogu servera na kome se nalaze podaci i aplikativni softver, koji se stavljaju na raspolaganje klijentima na njihov zahtev. Serveri obično (mada ne i neophodno) bivaju moćniji računari i na njima se obavljaju poslovi koji zahtevaju više resursa. Podela na klijentski i serverski je primerenija za softver nego za računare (tako npr. isti računar može da istovremeno pokreće i Veb server i klijent elektronske pošte čime ima ulogu servera u jednoj, a klijenta u drugoj komunikaciji).

Mreža ravnopravnih računara (eng. peer-to-peer — P2P) - računari direktno komuniciraju jedan sa drugim, dele podatke i operemu. Mreže ravnopravnih računara se koriste sve više i više za masovnu razmenu velikih količina podataka (npr. Napster, Bittorent).

1.2 Komponente računarskih mreža

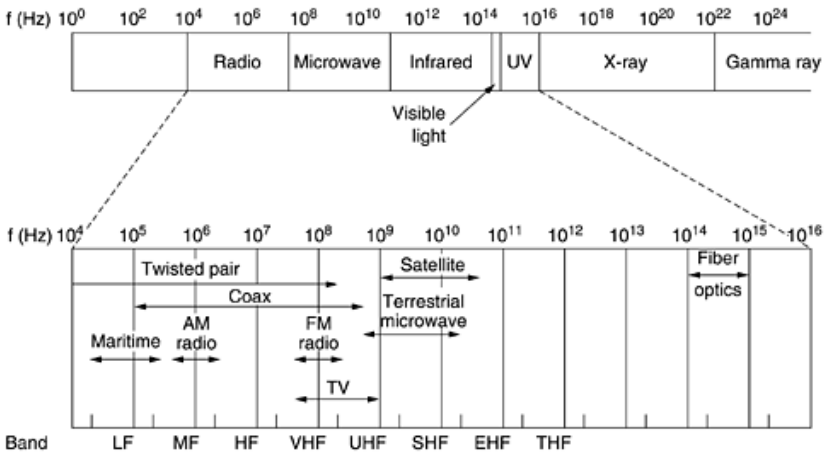
Računarska mreža je sistem koji se sastoji od skupa *hardverskih uređaja* međusobno povezanih *komunikacionom opremom* i snabedeven odgovarajućim *kontrolnim softverom* kojim se ostvaruje kontrola funkcionisanja sistema tako da je omogućen prenos podataka između povezanih uređaja.

1.2.1 Mrežni hardver

Tradicionalno se podrazumeva da se u okviru računarske mreže povezuju računari. Takođe, u okviru mreža čest je slučaj povezivanja pomoćnih uređaja kao što su, na primer, štampači i skeneri kako bi se omogućilo njihovo deljeno korišćenje od strane više računara. Međutim, u novije vreme, pojava tehnološke konvergencije dovodi da se oštra granica između klasičnih računara i ostalih digitalnih uređaja specijalizovane namene briše i sve je češće slučaj da se u okviru računarske mreže mogu povezati i PDA (eng. personal digital asistant) uređaji, mobilni telefoni, foto-aparati i slično. Realno je očekivati da će u skorije vreme sastavni deo računarskih mreža biti i uređaji poput, na primer, automobila ili frižidera čime će biti omogućeno daljinsko upravljanje ovakvim uređajima[1].

Da bi uređaj mogao biti umrežen neophodno je da sadrži specijalizovan deo hardvera namenjen umrežavanju koji se smatra delom komunikacione opreme. Obično je to *mrežna kartica (mrežni adapter) - NIC (eng. network interface card)* koja omogućava uređaju fizički pristup mreži. Svaku mrežnu karticu karakteriše jedinstvena *fizička (MAC) adresa* kojom se uređaj jedinstveno identifikuje prilikom komunikacije. Neke mrežne kartice obezbeđuju pristup žičanim, a

[1] Osnovni problem u ovom slučaju je bezbednost, odnosno strah od preuzimanja kontrole od strane nedobronamernih lica



Slika 1.1: Podela elektromagnetnog spektra

neke bežičnim komunikacionim kanalima. Osim mrežnih kartica, za umrežavanje se koriste *modemi* (telefonski, kablovski), kao i neki drugi slični uređaji.

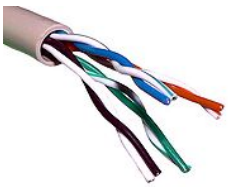
1.2.2 Komunikacioni kanali

Da bi se uređaji unutar mreže povezali među sobom koriste se komunikacioni kanali kao što su kablovi ili bežični prenosni sistemi.

Osnovna mera kvaliteta komunikacionog kanala jeste brzina prenosa koja se meri u broju bita koji se mogu preneti u jednoj sekundi (bit/s). Brzina prenosa se dakle meri brojem elementarnih informacija koje mogu da proteku u sekundi. Uzimajući u obzir aktuelne tehnologije prenosa na računarskim mrežama, češće se koristi jedinica Megabit (milion bita) u sekundi — Mbps, ili Gigabit (milijarda bita) u sekundi — Gbps. Brzina prenosa je fizička karakteristika komunikacionog kanala i zavisi od frekvencijskog opaega (eng. bandwidth) koji se može propustiti kroz kanal bez gubitka signala. Na slici 1.1 prikazan je raspon frekvencija koje se koriste pri različitim prenosnim tehnologijama. S obzirom da je brzina prenosa podataka srazmerna frekvencijskom opsegu, sa slike se jasno vidi zbog čega optički kablovi daju najbolje performanse.

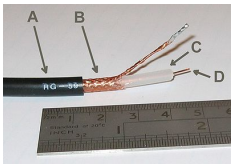
Žičane komunikacije

Parice (eng. twisted-pair wire) - Ovo je najkorišćeniji način komunikacije. Uređaji se povezuju korišćenjem uvijenih uparenih izolovanih bakarnih žica, veoma slično povezivanju običnih telefona(u telefonijskom žargonu ove se žice obično nazivaju *parice*). Žice se uparuju i uvijaju kako bi se smanjile smetnje u komunikaciji. Razlikuju se obično kablovi kategorije 3 koji se koriste u telefoniji i kablovi kategorije 5 koji se koriste u povezivanje računara.




Brzina prenosa kroz ovakav medijum obično varira od 2Mbps do 100Mbps.

Koaksijalni kablovi - Ovakvi kablovi se obično koriste za televizijske kablovske sisteme, a koriste se i u LAN mrežama u kompanijama itd. Kablovi se sastoje od centralne bakarne ili aluminijumske žice obmotane savitljivim izolatorskim slojem, oko kojega je opet obmotan provodni sloj tankih žica, sve obmotano spoljašnjom izolacijom.



Koaksijalni kablovi omogućuju brzinu prenosa do 200Mbps (čak i do 500Mbps), uz manju osetljivost na elektromagnetne smetnje.

Optički kablovi - Optički kablovi se prave od velikog broja (stotina, hiljada) veoma tankih staklenih vlakana umotanih u zaštitni sloj. Podaci se prenose svetlosnim talasima koje emituje mali laserski uređaj. Na ovakve kablove ne utiču smetnje prouzrokovane elektromagnetnim zračenjima. Nedostatak je što su skupi i komplikovani za instalaciju, pa se uglavnom koriste za osovinski deo mreže (mrežnu kičmu), na koji se onda koaksijalnim kablovima ili upredenim žicama povezuju pojedinačni uređaji.


Brzina prenosa je velika, i može da ide i do nekoliko triliona bita u sekundi. Najčešće se koriste za mreže koje imaju brzinu od 10Gbps.

Bežične tehnologije. Bežični prenosni sistemi ne koriste kablove za prenos podataka. To je posebno praktično u slučaju prenosivih računara, mobilnih uređaja ili relativno udaljenih lokacija za koje bi uspostavljanje kablovske mreže bilo nedopustivo skupo.

Umesto kablova koriste se radio talasi, mikro talasi, infracrveni zraci. Podaci se prenose moduliranjem amplitude, frekvencije ili faze talasa. Tehnologije koje se danas najčešće koriste su:

Bluetooth – Bežična tehnologija koja se koristi za komunikaciju na veoma malim razdaljinama (do deset ili do sto metara u zavisnosti od klase uređaja). Brzine prenosa idu do 3Mbps. Koristi radio talase i može da prođe i kroz čvrste prepreke. Koristi se uglavnom za komunikaciju računara sa periferijskim uređajima kao i u mobilnoj telefoniji.

Bežični LAN - Wireless LAN (WLAN, WiFi) je tehnologija koja koristi radio talase za bežičnu komunikaciju više uređaja na ograničenom rastojanju (nekoliko desetina ili stotina metara). U zavisnosti od standarda, brzina prenosa ide od 10Mbps do 50Mbps (u najnovije vreme i do 600Mbps). Najraširenij standard za bežičnu LAN komunikaciju je IEEE 802.11.

Ćelijski sistemi - Način prenosa podataka veoma sličan onom koji se koristi u mobilnoj telefoniji. Za komunikaciju se koriste radio talasi i sistemi antena koje pokrivaju određenu geografsku oblast, pri čemu se signal od odredišta do cilja prenosi preko niza antena.

Zemaljski mikrotalasi - Koriste antensku mrežu na Zemlji, pri čemu se za komunikaciju koriste mikrotalasi niske frekvencije koji zahtevaju da antene budu optički vidljive tako da se one obično smeštaju na visoke tačke (vrhove brda, tornjeve, nebodere). Antene mogu da budu udaljene i do pedesetak kilometara.

Komunikacioni sateliti – Koriste mikrotalase za komunikaciju tako što se prenos između dve tačke koje nemaju optičku vidljivost ostvaruje poprečnom komunikacijom preko komunikacionih satelita koji se obično nalaze u orbiti na visini od 36 hiljada kilometara[2]. Na ovaj način se pored računarske komunikacije obično prenose televizijski i telefonski signal. Brzina komunikacije je relativno mala (npr. 100Mps) u poređenju sa optičkim kablovima, ali ipak ima nekoliko scenarija u kojima je korišćenje satelitske komunikacije pogodnije.

1.2.3 Mrežni softver

Sama mreža ne može ničemu da posluži bez određene inteligencije koja će joj omogućiti da funkcioniše. Ulogu te inteligencije ima mrežni softver. Kako bi se savladala kompleksnost računarskih mreža, mrežni softver se organizuje hijerarhijski. Npr. programer pregledača Veba ne treba da misli o tome da li će Veb stranice primati preko bežične mreže ili preko Ethernet mreže. On treba da se koncentriše samo na aspekte značajne za njegovu konkretnu aplikaciju, a da sve niže detalje mrežne komunikacije prepusti nižem sloju mrežnog softvera (prisutnom u okviru operativnog sistema, ili čak samog mrežnog hardvera).

[2]Na ovoj visini, tačno iznad Ekvatora, sateliti orbitiraju istom brzinom kao i zemlja i izgleda da se ne pomeraju na nebu.

Najgrublje posmatrano, mrežni softver može da se podeli na dva nivoa.

Mrežni softver koji omogućuje korišćenje različitih mrežnih uređaja, npr. mrežnih kartica ili modema, jeste *mrežni softver niskog nivoa*. Ova vrsta softvera nalazi se obično u jezgru operativnog sistema računara, uglavnom u obliku upravljača perifernim uređajima, tzv. drajvera (eng. driver). On upravlja računarskim hardverom i komunikacionom opremom. Korisnik računara nikada ne koristi ovaj softver direktno, u opštem slučaju on nije ni svestan da taj softver postoji. Osnovni zadatak ovog softvera je da pruži usluge mrežnim aplikacijama (tj. njihovim programerima) koje korisnici koriste. Ove aplikacije čine *mrežni softver visokog nivoa* i pružaju različite usluge i servise korisnicima na mreži, kao što je slanje i prijem elektronske pošte, pregledanje Veba i sl.

Podela na samo dva nivoa predstavlja prilično uprošćeno gledište. S obzirom na značaj ovog koncepta, o slojevitosti mreža će biti više reči u poglavlju 1.5.

1.3 Raspon mreža

Jedan od kriterijuma za klasifikovanje mreža je i njihova fizička veličina, tj. geografski raspon koji mreža pokriva. Ova klasifikacija je izrazito bitna zbog činjenice da raspon mreže direktno određuje tehnologije komunikacije pogodne za korišćenje u okviru te mreže (npr. bežična Bluetooth komunikacija je pogodna za male lične mreže, dok je kod velikih mreža koje povezuju cele države pogodno koristiti satelitsku komunikaciju ili optičke veze).

Personal area network (PAN) - mreže koje su namenjene za jednog čoveka. Na primer, bežična mreža kojom su spojeni računar, miš i štampač je PAN. Ovakve mreže obično pokrivaju raspon od nekoliko metara i koristi bilo žičanu bilo bežičnu komunikaciju.

Local area network (LAN) - mreža koja povezuje uređaje na relativno malim udaljenostima, najčešće nekoliko kancelarija u okviru jedne poslovne zgrade. Ovakve mreže se tradicionalno vezuju na žičanu komunikaciju kroz mrežne kablove, iako nove tehnologije daju mogućnost korišćenja postojećih kućnih instalacija (koaksijalnih kablova, telefonskih linija i električnih linija) za komunikaciju kao i korišćenja bežične komunikacije.

Campus area network(CAN) - Ove mreže povezuju više lokalnih mreža u okviru ograničenog geografskog prostora (npr. u okviru jednog univerziteta, kompanije, vojne baze, itd.). Na primer, više mreža zasebnih fakulteta (departmana) u okviru jedne lokacije univerziteta (kampusa) se povezuje u jedinstvenu celinu. Tehnologija koja se koristi za povezivanje je obično ista kao i u slučaju LAN. U novije vreme, između odvojenih zgrada se obično uspostavlja bežična komunikacija.

Metropolitan area network (MAN) - Ove mreže povezuju veće geografske prostore (najčešće nivoa grada ili jako velikog kampusa). MAN obično povezuje više lokalnih mreža (LAN) korišćenjem veoma brze kičme komunikacije (eng. backbone), najčešće izgrađene od optičkih veza.

Wide area network (WAN) - ove mreže povezuju izrazito velike geografske prostore, često šire od granica jednog grada, oblasti i često i države. U

današnje vreme, WAN mreže su obično u sastavu Interneta. WAN infrastrukturu obično održavaju komercijalne kompanije (obično telefonske i telekomunikacione) i iznajmljuju usluge korišćenja. Za povezivanje u okviru kičme se koriste brze veze, najčešće optičke i satelitske.

1.4 Topologija mreža

Topologija mreže predstavlja način na koji su povezane među sobom različite komponente mreže, i način na koji interaguju. Radi jednostavnosti, u daljem tekstu neće biti pravljena razlika između fizičke i logičke topologije. Različite topologije razlikuju se prema osnovnoj ceni (koliko se ulaže u specifičan oblik povezivanja čvorova u mrežu), ceni komunikacije (koliko je vreme potrebno za prenos poruke od jednog do drugog čvora pri tom specifičnom obliku povezivanja), i pouzdanosti tj. mogućnosti prenosa podataka u slučaju otkaza nekog čvora ili veze.

Najšire posmatrano, postoje dva ključna načina povezivanja: broadcast i point-to-point.

Zajednički komunikacioni kanal (broadcast). Ove mreže se sastoje od zajedničkog komunikacionog kanala preko kojega komuniciraju sve mašine povezane u mrežu. Mašine šalju kratke poruke (pakete) na mrežu postavljajući ih na komunikacioni kanal, pri čemu svaka poruka sadrži i identifikaciju željenog primaoca. Poruku svi primaju, pri čemu je primaoc jedini prihvata, dok je svi ostali odbacuju. Ovaj način povezivanja se obično koristi za komunikaciju u okviru manjih, lokalnih mreža.

Isti fizički medijum se može koristiti za simultanu komunikaciju više čvorova bez međusobnog ometanja. Pristup uređaja kanalu se može određivati *statički*, kada svaki uređaj ima unapred određena pravila kako i u kom delu kanala sme da vrši komunikaciju ili *dinamički* kada se mogućnost pristupa uređaja kanalu određuje na osnovu trenutnog stanja i dostupnosti kanala. Neki od osnovnih načina deljenja zajedničkog kanala su:

Deljenje vremena (eng. time division multiplexing — TDM) - Jedan od načina statičke alokacije kanala je tzv. *deljenje komunikacionog kanala korišćenjem deljenja vremena* u kom slučaju svaki uređaj komunicira u tačno određenom vremenskom trenutku pri čemu se uređaji naizmenično smenjuju.

Deljenje frekvencije (eng. frequency division multiplexing — FDM) - Drugi od načina statičke alokacije kanala je tzv. *deljenje frekvencije* u kom slučaju svaki uređaj komunicira u okviru određenog frekvencijskog opsega.

Deljenje talasne dužine (eng. wave division multiplexing — WDM) - Ovo je specijalni naziv za deljenje frekvencije u slučaju optičke komunikacije.

Deljenje kodiranjem (eng. code division multiple access — CDMA) - Jedan od novijih načina statičkog deljenja kanala u okviru kojega se koristi teorija kodiranja kako bi se iz primljenog paketa informacija izdvojile informacije relevantne za određeni čvor.

CSMA/CD — Što se tiče dinamičkog deljenja kanala, najkorišćenija je tehnika *CSMA/CD (eng. carrier sense multiple access with collision detection)* koja se koristi u okviru Ethernet LAN mreža. U ovom slučaju se poštuje protokol da svaki uređaj posmatra da li kanalom već teče neka komunikacija pre nego što počne da šalje podatke. Ukoliko se primeti da neko istovremeno pokušava da pošalje podatak prekida svoje slanje, čeka određeno vreme i pokušava ponovo.

Direktne čvor-čvor veze (point-to-point). Ove mreže se sastoje od mnogo direktnih veza između individualnih parova mašina. Kako bi informacija stigla od jednog do drugog čvora, obično je potrebno da prođe kroz niz posrednih čvorova. Ovaj način komunikacije se obično koristi u okviru velikih mreža. Obično je moguće da informacije putuju različitim putanjama, tako da je izbor pogodne putanje obično veoma značajan za efikasnost komunikacije. U zavisnosti od načina određivanja putanje i načina slanja informacije razlikuju se sledeći tipovi komunikacije.

Kanalno komutiranje (eng. circuit switching) - Pre započinjanja komunikacije ostvaruje se trajna fiksirana putanja (kanal) između čvorova i sva informacija se prosleđuje kroz uspostavljenu putanju (na ovaj način funkcionišu mreže fiksne telefonije). Kanal je rezervisan sve dok se eksplicitno ne raskine, te je ovaj način komunikacije prilično skup. Kroz brze direktne veze između unutrašnjih čvorova obično se simultano odvija prenos podataka vezanih za komunikaciju između različitih parova perifernih čvorova. Ovo se obično postiže korišćenjem FDM ili TDM.

Komutiranje poruka (eng. message switching) - Svaka poruka koja se šalje putuje zasebnom putanjom.

Paketno komutiranje (eng. packet switching) - Poruke se pre slanja dele na zasebne manje pakete, i svaki paket putuje svojom zasebnom putanjom da bi se na odredištu paketi ponovo sklopili u jedinstvenu poruku. Prednost ovog načina slanja je u tome što delovi poruke (paketi) praktično paralelno putuju kroz mrežu i time mogu brže da stignu do odredišta.

Razlikuju se četiri glavna tipa topologije mreže: zvezda, magistrala, prsten, i opšta grafoidna topologija (potpuna ili delimična povezanost). Definišimo ove tipove i razmotrimo njihove prednosti i nedostatke.

Magistrala - Mreža sa topologijom magistrale povezuje svoje komponente jednim istim kablom i informacija se istovremeno raznosi svim primaocima. Preuzima se samo na odredišnim mestima. Za ovaj tip mreže tipično je korišćenje koaksijalnog kabla. Saobraćaj se odvija u oba smera, pa pri većem opterećenju može da dođe do sudaranja poslatih paketa ili zagušenja kanala.

Zvezda - U zvezdastoj mreži, svi učesnici su povezani u jednu istu centralnu tačku (čvor-računar ili drugi uređaj) a informacija putuje od pošiljaoca (emitera) prema primaocu isključivo preko te centralne tačke. Cena uspostavljanja mreže je niska kao i cena komunikacije, ali je zagušenje u centralnom čvoru često. Zato se obično na nivou centralnog čvora postavlja komutator (svič, eng. switch).

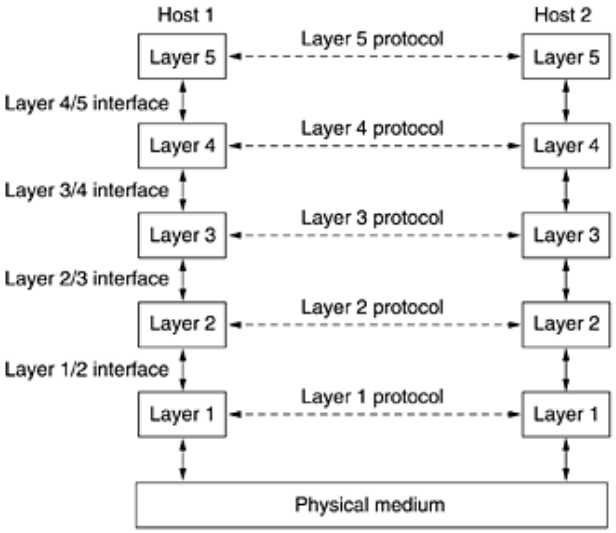
Prsten - I mreža sa topologijom prstena ima sve svoje komponente na istom kablu, ali taj kabl nema krajeve. Šta više, informacija se kreće samo u jednom, strogo određenom pravcu. Ukoliko neki od čvorova mreže sa topologijom prstena otkaže, to neće uticati na funkcionisanje ostatka mreže. Međutim, otkaz na komunikacionom kanalu rezultuje potpunim prekidom mrežnog saobraćaja.

Potpuna povezanost - U mreži sa topologijom potpune povezanosti svaki čvor poseduje posebnu vezu sa svakim od preostalih čvorova. Koristi se samo kod sasvim malih mreža i to iz razloga pouzdanosti jer redundansa smanjuje osetljivost na padove u mreži. Varijante topologije potpune povezanosti su topologije delimične povezanosti u kojima neke od od veza između čvorova izostaju, iz bilo kojih razloga.

Navedene definicije se odnose na male mreže. Jedna velika mreža sastoji se od velikog broja međusobno povezanih malih mreža, od kojih svaka ima sopstvenu topologiju. Velika mreža će, dakle, imati različite komponente sa različitim topologijama, ali će takođe imati i jednu opštu (generalnu) topologiju koja će biti ili zvezda, ili magistrala, ili prsten.

1.5 Slojevitost mreža

Današnje mreže su izrazito kompleksni entiteti. Kako bi se savladala kompleksnost mreža, mreže i mrežni softver se moraju kreirati hijerarhijski, uz postojanje velikog broja zasebnih, precizno definisanih, nivoa tj. slojeva. Broj slojeva se razlikuje od mreže do mreže. Na svakom sloju, sprovodi se odgovarajući protokol komunikacije. *Protokol* je dogovor dve strane o načinu komunikacije. Narušavanje protokola čini komunikaciju nemogućom.

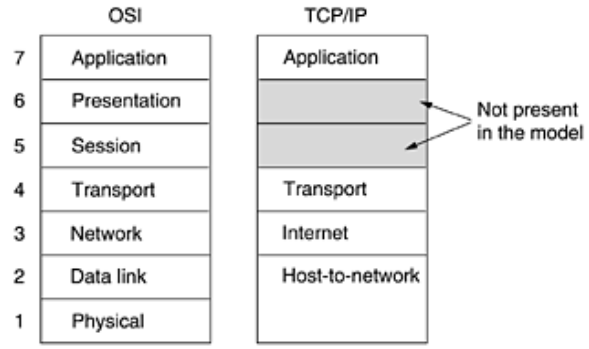


Istorijski, mreže se obično posmatraju u okviru dva referentna modela:

Open Systems Interconnection — OSI - model sa 7 slojeva, standardizovan od strane ISO.

TCP/IP - model sa 4 sloja, prisutan u okviru Interneta.

OSI model je razvijen pod okriljem međunarodne organizacije za standardizaciju (ISO) i, iako se konkretni protokoli koje definiše više ne koriste, predstavlja značajan apstraktni opis mreža. Sa druge strane, TCP/IP model je takav da se njegova apstraktna svojstva ne koriste značajno, dok se konkretni protokoli intenzivno koriste i predstavljaju osnovu Interneta. Slojevi u okviru referentnih modela i njihov međusobni odnos, grafički su prikazani na slici 1.2.



Slika 1.2: Referentni modeli

U nastavku će biti opisane uloge najznačajnijih slojeva u okviru ovih referentnih modela.

Fizički sloj (eng. physical layer). Najniži, fizički sloj obezbeđuje postojanje komunikacionog kanala i mogućnost slanja i primanja pojedinačnih bitova kroz komunikacioni kanal. Na ovom sloju se obično ne vrši nikakva kontrola grešaka.

Sloj veze podataka (eng. data link layer). Sloj veze podataka, višim slojevima obezbeđuje postojanje pouzdanog kanala komunikacije u kome se greške automatski detektuju i ispravljaju (error control), pri čemu se takođe automatski vodi računa o brzini slanja podataka kako se ne bi desilo da brzi uređaji zagušuju sporije (flow control). Ukoliko se koristi zajednički kanal komunikacije (broadcast mreže), na ovom sloju se vrši kontrola pristupa uređaja komunikacionom kanalu (Medium Access Control).

Mrežni sloj (eng. network layer) Mrežni sloj se bavi povezivanjem više računara u mrežu. Osnovni zadatak u okviru ovog sloja je rutiranje (eng. routing), tj. određivanja putanja paketa koji putuju kroz mrežu kako bi se odredio efikasan način da stignu na svoje odredište. Kako bi se odredila putanja neophodno je uvođenje sistema adresiranja. Ukoliko se povezuju heterogene mreže (sa različitim shemama adresiranja), na ovom sloju se vrši prevođenje adresa (na primer, na nižim slojevima se obično koriste fizičke MAC adrese, a na višim

IP adrese). Svaki čvor u mreži uključen u komunikaciju mora da implementira mrežni protokol, da razume odredišnu adresu i da na osnovu ovoga odluči kome će da primljenu poruku prosledi.

Najpoznatiji protokol ovog sloja je koji se koristi u okviru Interneta je *Internet Protocol (IP)*.

Transportni sloj (eng. transport layer) Transportni sloj ima zadatak da prihvata podatke sa viših slojeva, deli ih na manje jedinice (pakete), šalje te pakete na odredište korišćenjem nižih slojeva i protokola mreže. Obično se na ovom sloju razlikuju dve vrste protokola: *protokoli sa uspostavljanjem konekcije (eng. connection oriented)* i *protokoli bez uspostavljanja konekcije (eng. connectionless)*. Protokoli koji zahtevaju uspostavljanje konekcije garantuju da će poslati podaci zaista i stići na odredište (u istom redosledu u kojem su i poslati). Protokoli bez uspostavljanja konekcije ne daju ovakve garancije, ali je prenos podataka obično brži. Za razliku od protokola mrežnog sloja koji moraju da budu implementirani u svakom čvoru lanca komunikacije, protokoli transportnog sloja moraju biti implementirani jedino na krajnjim čvorovima komunikacije, tj. u host računarima. Ruteri (uređaji koji posredno učestvuju u komunikaciji prenošenjem paketa) obično nisu svesni detalja transportnih protokola. Transporni protokoli se, dakle, mogu smatrati protokolima kojim dva host računara komuniciraju. S obzirom da na istom host računaru obično postoji više različitih programa koji imaju potrebu za komunikacijom (svaki korišćenjem zasebnog aplikacionog protokola, ali zajedničkim korišćenjem transportnog protokola), zadatak transportnih protokola je i da vrše tzv. *multipleksovanje (eng. multiplexing)*. Ovo se obično ostvaruje kroz koncept *portova (eng. port)* koji predstavljaju brojeve na osnovu kojih se određuje kom programu pokrenutom na host računaru pripada paket primljen na transportnom sloju.

Najkorišćeniji protokoli ovog sloja (koji se koriste u okviru Interneta) su *Transfer Control Protocol (TCP)* i *User Datagram Protocol (UDP)*.

Aplikacioni sloj (eng. application layer) Aplikacioni sloj definiše protokole koje direktno koriste korisničke aplikacije u okviru svoje komunikacije. Ovi protokoli su prilagođeni specifičnim zahtevima aplikacija. Aplikacioni protokoli se smatraju protokolima kojima dva programa tj. dve aplikacije komuniciraju.

Najkorišćeniji protokoli ovoga sloja u okviru Interneta su *HyperText Transfer Protocol (HTTP)* koji se koristi za prenos Veb stranica, *SMTP, POP3, IMAP* koji se koriste u za prenos elektronske pošte, *File Transfer Protocol (FTP)* koji se koristi za prenos datoteka, itd.

Glava 2

Internet, usluge i protokoli

Internet je najveća i najznačajnija mreža današnjice. Ona povezuje veliki broj različitih mreža i računare širom cele planete. S obzirom na to da Internet veoma kompleksan, teško je definisati ga jednom rečenicom. Dve grupe opisa Interneta se mogu sresti u literaturi: *strukturni opisi* i *funkcionalni opisi*.

Sa strukturnog stanovišta, Internet se definiše preko hardverskih, komunikacionih i softverskih komponenti koje ga sačinjavaju. Sa ovog stanovišta, Internet je WAN mreža koja povezuje mnoštvo manjih privatnih ili javnih mreža. Internet omogućava računarima i drugim uređajima povezanim na ove mreže da međusobno komuniciraju. Komunikacioni kanali su izgrađeni od veoma različitih fizičkih komunikacionih tehnologija (raznih vrsta kablova, bežičnih veza, satelitskih veza). Krajnji računari se nazivaju i *host računari (eng. hosts)*. Između host računara postoje obično samo posredne veze preko uređaja koji se nazivaju *ruteri (eng. router)*. Struktura Interneta je hijerarhijska: host računari su povezani u mrežu njihovih lokalnih *Internet dobavljača (eng. Internet Service Provider – ISP)*, uređaji lokalnih dobavljača su povezani u regionalne mreže, regionalne mreže su povezane u nacionalne i internacionalne mreže, itd. I host računari i ruteri poštuju *IP protokol* komunikacije koji, između ostalog, svakom od njih dodeljuje jedinstvenu logičku adresu koja se naziva *IP adresa*. IP protokol definiše mogućnost slanja paketa informacija između hostova i rutera. Paketi informacija od hosta do hosta putuju preko niza rutera, pri čemu se putanja automatski određuje i hostovi nemaju kontrolu nad putanjom paketa (koristi se paketno komutiranje). Softver koji je instaliran na host računarima korisnicima pruža različite usluge.

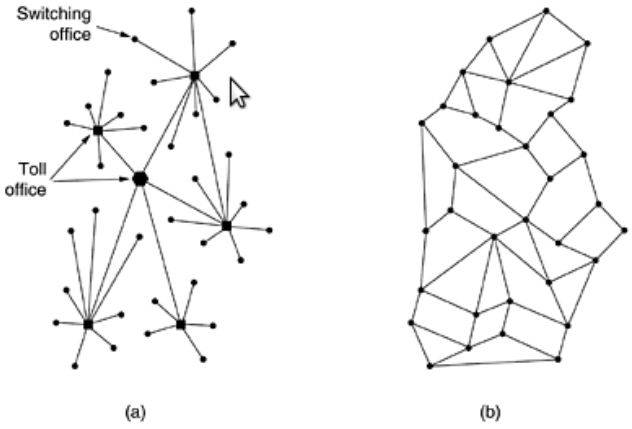
Sa funkcionalnog stanovišta, Internet se definiše preko usluga koje nudi svojim korisnicima. Sa tog stanovišta, Internet je mrežna infrastruktura koja omogućava rad *distribuiranim aplikacijama* koje korisnici koriste. Ove aplikacije uključuju *Veb (eng. World Wide Web)* koji omogućava korisnicima pregled hipertekstualnih dokumenata, *elektronsku poštu (eng. e-mail)*, *prenos datoteka (ftp, scp)* između računara, upravljanje računarima na daljinu preko *prijavljivanja na udaljene računare (telnet, ssh)*, slanje *instant poruka (im)*, itd. Vremenom se gradi sve veći i veći broj novih aplikacija. Ove aplikacije međusobno komuniciraju preko svojih specifičnih aplikacionih protokola (npr. *HTTP*, *SMTP*, *POP3*, ...). Svi aplikacioni protokoli komuniciraju korišćenjem dva transportna protokola: *TCP* koji je protokol sa uspostavljanjem konekcije koji garantuje da će podaci koji se šalju biti dostavljeni ispravno, u potpunosti i u redosledu u

kome su poslati, i *UDP* koji je protokol bez uspostavljanja konekcije i koji ne daje nikakve garancije o dostavljanju.

2.1 Istorijat Interneta

2.1.1 ARPANET

Priča o internetu počinje kasnih 1950-tih godina. Na vrhuncu hladnog rata, američko ministarstvo odbrane (*eng. Department of Defence - DoD*) je želelo da uspostavi mrežu komunikacije projektovanu tako da može da preživi nuklearni rat. U to vreme vojne komunikacije su koristile javnu telefonsku mrežu, što se smatralo veoma ranjivim. Slika 2.1(a) prikazuje hijerarhijski način organizacije telefonske mreže — jasno je da ukoliko dođe do kvara u malom broju čvorova, većina komunikacije biva presečeno. Oko 1960. godine DoD angažuje RAND



Slika 2.1: Organizacija mreže

korporaciju čiji radnik, Pol Baran (eng. Paul Baran) predlaže rešenje prikazano na slici 2.1(b). Podaci od čvora do čvora putuju bilo kojom od dostupnih putanja. Pošto su u tom slučaju neke putanje predugačke i analogni signal nije mogao da se šalje tako daleko, predloženo je da se koristi digitalno paketno komutiranje (packet-switching). U Pentagonu je ovaj koncept prihvaćen, međutim, nakon konsultacija sa AT&T, vodećom telefonskom kompanijom u SAD, koncept biva odbačen — konzervativni ljudi u AT&T nisu želeli da ih neki novajlija uči kako treba praviti komunikacioni sistem!

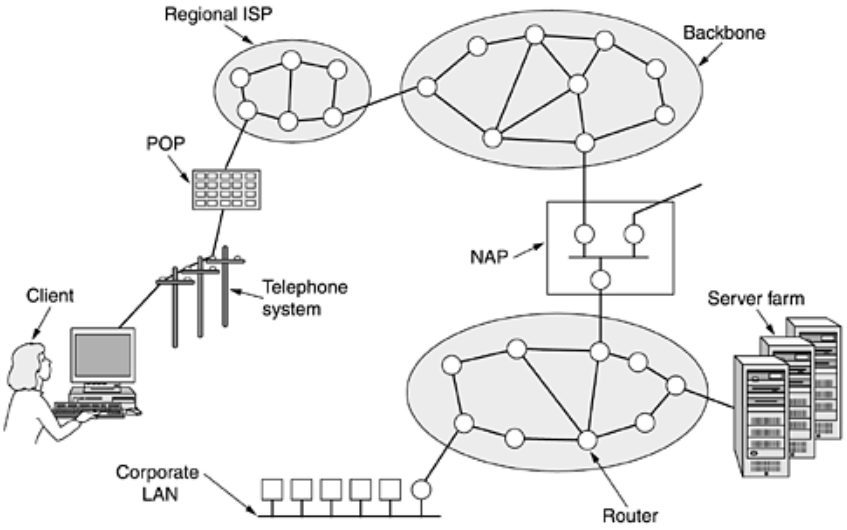
U oktobru 1957., kao odgovor na rusko lansiranje satelita Sputnjik, predsednik SAD Ajsenhauer (eng. Eisenhower) osniva *ARPA (eng. Advanced Research Project Agency)* — agenciju čiji je zadatak bio da subvencioniše istraživanja pri univerzitetima i kompanijama čije se ideje čine obećavajuće. 1967. godine, direktor ARPA Lari Roberts (eng. Larry Roberts), odlučuje da jedan od zadataka ARPA treba da bude i ulaganje u komunikacije. Nailazi se na ranije odbačen Baranov rad, čiji je minijaturni prototip već bio implementiran u Velikoj Britaniji. Nakon uvida u ova pozitivna iskustva, Roberts odlučuje da sagradi mrežu koja će biti poznata pod imenom *ARPANET*. Svaki čvor mreže se sastojao od

mini računara (hosta) na koji je nadograđen uređaj pod imenom *IMP (eng. Interface Message Processor)*. Kako bi se povećala pouzdanost, svaki IMP je bio povezan bar sa još dva udaljena IMP-a. Udaljeni IMP-ovi su među sobno bili povezani žičanim komunikacionim linijama brzine 56Kbps — najbržim koji su u to vreme mogli da se zamisle. Poruke koje su slane između hostova su se delile na pakete fiksirane dužine i svaki paket je mogao da putuje alternativnim putanjama. Svaki paket je morao u potpunosti da bude primljen u jedan IMP pre nego što se prosledi sledećem. Dakle, ARPANET je bila prva *store-and-forward packet-switching mreža*. Tender za izgradnju dobila je američka kompanija *BBN*. U pisanju softvera učestovao je i određen broj postdiplomaca uglavnom iznenađenih činjenicom da se ovako ozbiljan projekat oslanja na njihovu pamet i entuzijazam, bez učešća velikih eksperata i kompanija. Mreža je prvi put javno prikazana u decembru 1969. godine sa četiri povezana čvora: UCLA (University of California at Los Angeles), UCSB (University of California at Santa Barbara), SRI (Stanford Research Institute) i UU (University of Utah). Mreža je izrazito brzo rasla i do kraja 1972. godine bilo je povezano četrdesetak velikih čvorova u SAD. Kako bi se pomoglo rastu ARPANET-a, ARPA je takođe finansirala i istraživanja na polju satelitskih komunikacija i pokretnih radio mreža. Uskoro se uvidelo da je za dalji rast mreže uz mogućnost korišćenja različitih komunikacionih tehnologija potrebno ustanoviti i kvalitetne komunikacione protokole. 1974. godine dizajniran je *TCP/IP* model i protokol. Kako bi se ubrzalo širenje ovog protkola, ARPA daje zadatak kompaniji BBN i univerzitetu Berkley da ugrade softversku podršku ovih protkola u Berkley Unix operativni sistem, što je i urađeno kroz uvođenje programskog interfejsa za mrežno programiranje (tzv. *sockets*) i izgradnju niza aplikacija za rad u mrežnom okruženju.

Tokom 1980-tih veliki broj dodatnih mreža, naročito LAN, je povezan na ARPANET. Povećanjem dimenzije pronalaženje odgovarajućeg hosta postaje problematično i uvodi se DNS (Domain Name System).

2.1.2 NSFNET

Kasnih 1970-tih, američka nacionalna naučna fondacija *U.S. National Science Foundation (NSF)* uviđa ogroman pozitivan uticaj ARPANET-a na razvoj nauke kroz omogućavanje udaljenim istraživačima da dele podatke i učestvuju u zajedničkim istraživanjima. Ipak, kako bi neki univerzitet mogao da koristi ARPANET, neophodno je bilo da ima ugovor sa DoD što mnogi univerziteti nisu imali. NSF odlučuje da se izgradi naslednik ARPANET mreže koja bi omogućila slobodan pristup svim univerzitetskim istraživačkim grupama. Projekat je započeo izgradnjom kičme mreže (eng. backbone) koja je povezivala šest velikih računarskih centara u SAD. Superračunarima su priključeni komunikacioni uređaji koji su nazivani *fuzzball* (poput IMP u slučaju ARPANET). Hardverska tehnologija je bila identična tehnologiji korišćenoj za ARPANET, međutim, softver se razlikovao — mreža je odmah bila zasnovana na TCP/IP protokolu. Pored kičme, NSF je izgradio i dvadesetak regionalnih mreža koje su povezane na kičmu čime je zvanično izgrađena mreža poznata kao *NSFNET*. Ova mreža je priključena na ARPANET povezivanjem fuzball i IMP na univerzitetu CMU (Carnegie-Mellon University). NSFNET je bio veliki uspeh i komunikaciona tehnologija u kičmi mreže je kroz nekoliko faza proširivana i unapređivana do brzina od 1.5Mbps početkom 1990-tih.



Slika 2.2: Arhitektura Interneta

Vremenom se shvatilo da Vlada SAD nema mogućnost samostalnog finansiranja održavanja i proširivanja NSFNET mreže. Odlučeno je da se mreža preda komercijalnim kompanijama koje bi, uz ostvarivanje sopstvenog profita, izvršile značajne investicije u razvoj. Ovo se pokazuje kao dobar potez i 1990-tih godina, uključivanjem komercijalnih kompanija, brzina komunikacije u okviru NSFNET kimčme, povećana je sa 1.5Mbps na 45Mbps.

S obzirom da različite kompanije počinju da grade zasebne kičmene komunikacione kanale, kako bi bila moguća komunikacija različitim kanalima svi oni bivaju povezani u okviru čvorova pod imenom *NAP (Network Access Point)*. Ovo znači da umesto postojanja jedinstvene kičme mreže, paket koji putuje između dva NAP može da bira bilo koju od raspoloživih kičmenih infrastruktura.

2.1.3 Internet

Paralelno sa razvojem ARPANET-a i NSFNET-a, i na ostalim kontinentima nastaju mreže pravljene po uzoru njih (npr. u Evropi su izgrađene EuropaNET i EBONE). Sve ove postepeno bivaju povezane u jedinstvenu svetsku mrežu. Sredinom 1980-tih godina ljudi počinju da ovu kolekciju različitih spojenih mreža posmatraju kao *međumrežu (eng. internet)*, a kasnije i kao jedinstveni svetski entitet - *Internet*[1]. Danas se može se smatrati da je uređaj priključen na Internet ukoliko koristi softver koji komunicira TCP/IP protokolima, ima IP adresu i može da šalje IP pakete ostalim mašinama na Internetu.

Arhitektura današnjeg Interneta prikazana je na slici 2.2. Klijent se povezuje, nekom od pristupnih tehnologija, u ovom slučaju modemskim pristupom sa računarom njegovog dobavljača Interneta (*eng. Internet Service Provider - ISP*). ISP održava regionalnu mrežu svojih rutera i povezan je na neku od kičmi

[1]Veliko početno slovo u imenu Internet označava da je u pitanju jedinstven entitet tako da se Internet smatra vlastitim imenom.

Interneta. Različite kičme su povezane u okviru NAP - stanice rutera koji pripadaju različitim kičmama, a u okviru NAP su povezani brzom LAN vezom.

2.1.4 Tehnologije pristupa Internetu

Tehnologije pristupa Internetu (eng. access networks) su deo Internet infrastrukture između host računara i prvog rutera. Ovaj deo komunikacije se ponekad naziva *lokalna petlja (eng. local loop)* ili *poslednja milja (eng. last mile)*. Iako predstavlja jako mali procenat geografske razdaljine koji podaci prelaze, često predstavlja usko grlo u komunikaciji. Naime, komunikacija u ovom delu se obično vrši korišćenjem zastarele postojeće infrastrukture fiksne telefonije i vrši se na analogan način. Ipak, promene na tom polju i napredak tehnologije polako počinju da bivaju vidljive, čak i u nerazvijenim zemljama.

Modemski pristup - Jedan od načina uključivanja kućnih računara u Internet mrežu je korišćenje već postojeće infrastrukture *fiksne telefonije (eng. plain old telephone system, POTS)*. Kako bi se uspostavila veza potrebno je nazvati telefonski broj tako da ova grupa povezivanja spada u grupu *pozivnog povezivanja (eng. dial up)*. Fiksna telefonija podrazumeva postojanje parica koje povezuju udaljene tačke prenošenjem analognog signala. Računar se priključuje na telefonsku infrastrukturu preko uređaja koji se naziva *modem* koji ima zadatak da vrši analogno/digitalnu konverziju. Na drugom kraju veze, u okviru dobavljača interneta, nalazi se sličan modem koji je povezan na ruter uključen u Internet mrežu. Fizičke karakteristike komunikacije kroz telefonsku mrežu ograničavaju brzinu komunikacije na nekoliko desetina hiljada bitova u sekundi (standardna brzina modema je 56Kbps). Naime, s obzirom da je fiksna telefonija prvobitno bila namenjena pre svega prenosu glasa, a smatra se da je za dobro razumevanje glasa dovoljno preneti frekvencije između 300Hz i 3400Hz, na kraju lokalne petlje instalirani su filtri koji uklanja sve frekvencije van ovog pojasa kako bi se poboljšao kvalitet prenosa glasa. Ovim je sav prenos podatak ograničen na ovaj uski frekvencijski pojas što značajno ograničava brzinu prenosa podataka.

DSL - *Digitalna pretplatna linija (eng. Digital subscriber line)* je tehnologija za istovremeni prenos glasovnog signala i digitalnih podataka velikim brzinama preko parica fiksne telefonske mreže. Ovo znači da korisnici istovremeno mogu i da telefoniraju i da prenose podatke, što ranije nije bilo moguće. DSL ostvaruje stalnu vezu i nema potrebe za okretanjem broja prilikom uspostavljanja veze (nije dial up). U nastavku će ukratko biti objašnjen princip funkcionisanja DSL. Filtriranje frekvencija van standardnih frekvencija ljudskog govora na kraju telefonskih linija ograničava mogućnost prenosa podataka telefonskim linijama. Kako bi se se povećao frekvencijski opseg, filtri se modifikuju i odsecanje frekvencija se ne vrši. Ovim, frekvencijski opseg veze postaje zavisan samo od dužine kabla (na dugačkim paricama dolazi do slabljenja visokofrekvencijskih signala). Jedno od ograničenja DSL tehnologije je nemogućnost instalacije na mestima koje su fizički previše udaljeni od telefonske centrale (DSL pristojne brzine se obično može ugraditi na rastojanjima do 4km). U slučaju kratkih veza, prošireni frekvencijski raspon obično biva preko 1MHz. Ovaj raspon se zatim deli

na pojaseve širine 4Khz i svaki pojas se nezavisno koristi za komunikaciju. Dakle, u pitanju je *multipleksovanje deljenjem frekvencija (eng. frequency division multiplexing, FDM)*. Obično se jedan pojas alocira za prenos glasovnog signala, dva pojasa za kontrolu prenosa podataka, dok se svi ostali pojasevi (njih oko 250) alociraju za prenos podataka. S obzirom na to da se obično više vrši preuzimanje podataka nego slanje, obično se više pojaseva odvaja za dolazni saobraćaj (download) nego odlazni (upload). Ovaj pristup se naziva *Asimetričnim digitalnim pretplatnim linijama (Asymmetric DSL, tj. ADSL)*. Brzina prenosa podataka, obično je do 16Mbps u dolaznom i 1Mbps u odlaznom saobraćaju. Na korisnikovom kraju linije, instalira se *razdelnik (eng. splitter)* koji prvi pojas (frekvencije do 4Khz) usmerava ka telefonskom uređaju, a ostale pojaseve ka računaru (ili ulazu u lokalnu računarsku mrežu). Između računara i razdelnika nalazi se tzv. ADSL modem koji je relativno kompleksan uređaj jer ima zadatak da vrši deljenje i objedinjavanje podataka koji se šalju na veliki broj nezavisnih komunikacionih kanala. Sličan uređaj (koji se naziva DSLAM), instalira se na drugom kraju žice (u okviru telefonske centrale). On prihvata podatke od velikog broja korisnika, objedinjuje informacije, i šalje ih ka ISP.

ISDN - Slično DSL tehnologiji, *Integrated Services Digital Network (ISDN)* uvodi direktne digitalne veze zasnovane na žicama javne telefonije kojima se istovremeno prenosi glasovni signal i digitalni podaci (na zasebnim kanalima) što korisnicima omogućava da istovremeno razgovaraju telefonom i koriste mrežu. Za razliku od DSL, ISDN zahteva uspostavljanje veze pozivom broja tako da spada u grupu dial up pristupa. Brzina prenosa podataka je obično 128Kbps. ISDN je danas u velikoj meri potisnut od strane DSL tehnologije jer zahteva kompleksnije promene u postojećoj telefonskoj infrastrukturi, a ne donosi značajno povećanje brzine prenosa podataka u odnosu na dial up.

HFC - *Optičko-kablovske mreže (eng. Hybrid fibre-coaxial)*[2] su mreže koje se zasnivaju na kombinovanom prenosu podataka kroz optička vlakna i koaksijalne kablove koje se koriste za istovremeni prenos televizijskog signala, radio signala, i digitalnih podataka. Ruter u centrali ISP se povezuje optičkim kablovima sa čvorovima, koji su dalje povezani sa korisnicima korišćenjem koaksijalnih kablova (obično već postojećih kablova kablovske televizije). Signal iz koaksijalnih kablova se zatim razdeljuje na radio i TV signal i na digitalne podatke. Veza sa računarom se ostvaruje preko tzv. kablovskog modema. Na jedan čvor, obično se povezuje oko 500 korisnika. Signal u kablovima se obično prostire radio talasima frekvencije između 5MHz i 1GHz (obično se početni pojas širine nekoliko desetina MHz koristi za odlazni saobraćaj, a ostatak frekvencijskog pojasa se koristi za dolazni saobraćaj. Slično kao kod DSL, korišćenjem FDM, frekvencijski opseg se deli na pojaseve koji se alociraju za prenos različitih vrsta signala i podataka. Jako je važno napomenuti da svi korisnici povezani na lokalni čvor dele komunikacioni kanal i svi dolazni paketi bivaju dostavljeni istovremeno svim kablovskim modemima priključenim na isti čvor. Zbog ovoga, brzina prenosa može da varira u zavisnosti od aktivnosti ko-

[2] U svakodnevnoj upotrebi je obično naziv *kablovski Internet*

risnika priključenih na lokalni čvor. Brzina dolaznog saobraćaja može da ide i do 60Mbps, a odlaznog 2Mbps (pod pretpostavkom da lokalni čvor nije opterećen).

Mreže mobilne telefonije - Razvoj mobilne telefonije karakteriše se generacijama. U prvoj generaciji vršen je analogni prenos glasa, u drugoj digitalni prenos glasa, dok se u okviru treće generacije vrši digitalni prenos glasa i podataka. Tehnologije pristupa internetu koje koriste postojeće mreže mobilne telefonije u novije vreme postaju sve naprednije i sve šire korišćene. Tehnologije koje se danas, u okviru treće generacije, najviše koriste su *High Speed Packet Access (HSPA)* koje omogućavaju brzine prenosa i do 14Mbps u dolaznom i 6Mbps u odlaznom saobraćaju. HSPA je unapređenje *Wideband Code Division Multiple Access (W-CDMA)* tehnologije. Starija *General Packet Radio Service (GPRS)* tehnologija je omogućavala brzine od samo 56 do 114Kbps i korišćena je za prenos podataka u okviru druge generacije (tzv. 2.5G).

2.2 Internet servisi

Broj različitih servisa koje nudi Internet vremenom raste. Osnovni servisi prisutni još iz doba ARPANET-a su elektronska pošta, diskusione grupe, upravljanje računarima na daljinu i prenos datoteka.

Elektronska pošta (eng. e-mail). Elektronska pošta predstavlja jedan od najstarijih servisa Interneta. U današnje vreme, elektronska pošta ima tendenciju da skoro u potpunosti zameni klasičnu poštu. Godišnje se razmeni više milijardi poruka. Elektronska pošta funkcioniše tako što svaki korisnik poseduje svoje „poštansko sanduče" (eng. mailbox) na nekom serveru. Sanduče jedinstveno identifikuje elektronska adresa koja obavezno sadrži znak `@` (izgovara se kao *et* ili *majmunče*) koji razdvaja ime korisnika, od domena servera elektronske pošte. Na primer, `filip@matf.bg.ac.rs`. Sandučići se nalaze na serverima na Internetu i obično ih obezbeđuju kompanije, univerziteti i dobavljači Interneta, ali takođe postoje i javni, besplatni serveri elektronske pošte.

Poruke koje se šalju su u tekstualnom formatu (bilo u obliku čistog teksta, bilo u obliku hiperteksta označenog jezikom HTML), ali mogu da obuhvate i priloge u proizvoljnom formatu (koji se iz istorijskih razloga takođe kodira i šalje u obliku teksta). Uz svaku poruku, poželjno je navođenje teme poruke (eng. subject) i, naravno, elektronske adrese primaoca.

Slanje i primanje pošte korisnik obično obavlja preko klijenta instaliranog na svom računaru. Najpoznatiji klijenti za elektronsku poštu danas su Microsoft Office Outlook, Microsoft Outlook Express, Apple Mail, Mozilla Thunderbird, Lotus Notes, Eudora, mapine, elm, ... Sve više na značaju dobijaju i klijenti za mobilne uređaje u kojima prednjači iPhone/iPod Touch. Značajan obim elektronske pošte se odvija preko javnih servisa elektronske pošte vezanih za Veb koji ne zahtevaju korišćenje posebnog klijenta elektronske pošte, već se rad sa elektronskom poštom obavlja korišćenjem Veb aplikacija. Korišćenje ovih servisa obezbeđuju velike kompanije, obično besplatno. Najznačajniji servisi ovog tipa su Yahoo! Mail, Microsoft Hotmail, Google Gmail, itd.

Protokoli koji se koriste u okviru elektronske pošte su SMTP, POP3 i IMAP. Svi oni koriste TCP i to na portovima 25, 110 i 143.

Diskusione grupe (eng. usenet). Diskusione grupe predstavljaju distribuirani Internet sistem za diskusije koji datira još od 1980. godine. Korisnici mogu da čitaju i šalju javne poruke. Poruke se smeštaju na specijalizovane servere (eng. news server). Diskusije su podeljene u grupe (eng. newsgroups) po određenim temama, i grupe se imenuju hijerarhijski. Tako, na primer, `sci.math` označava grupu za diskusije na temu matematičke nauke, dok je `alt.binaries.boneless` grupa sa najvećim saobraćajem (preko 4 milijarde poslatih poruka) koja se pre svega koristi za razmenu piratizovanog sadržaja.

Pristup diskusionim grupama se vrši korišćenjem specijalizovanog softvera (eng. newsreader). Obično su klijenti elektronske pošte istovremeno i klijenti za korišćenje diskusionih grupa.

Iako u današnje vreme Veb forumi predstavljaju alternativni način diskusija, diskusione grupe se i dalje koriste u značajnoj meri.

Diskusione grupe koristi NNTP protokol koji koristi TCP na portu 119.

Prijavljivanje na udaljene računare (eng. remote login) Prijavljivanje i korišćenje udaljenih računara je jedan od najstarijih servisa Interneta. Ovaj servis omogućava korisnicima (tj. klijentima) da se korišćenjem Interneta prijave na udaljeni računar (server) i da nakon uspešnog prijavljivanja rade na računaru kao da je u pitanju lokalni računar. Korisnik na ovaj način dobija terminal kojim upravlja udaljenim računarom izdajući komande (najčešće u okviru nekog komandnog interfejsa). Udaljeni računar prima komande i izvršava ih korišćenjem svojih resursa, a rezultate šalje nazad klijentu koji ih korisniku prikazuje u okviru terminala.

Prijavljivanje na udaljeni računar se obično vrši preko Telnet protokola i SSH protokola koji koriste TCP na portu 23, odnosno 22.

Za ovaj servis, klijenti najčešće se koriste aplikacije kao što su `telnet` (komandna aplikacija koja implementira Telenet protokol), `PuTTY` (aplikacija koja implementira i Telnet i SSH protokol), `OpenSSH`, `SSH Secure Shell Client` (aplikacije koje implementiraju SSH protokol) i slično. Telnet aplikacije ne vrše enkripciju podataka prilikom slanja tako da imaju problem sa stanovišta bezbednosti i sve manje se koriste. Sa druge strane, telnet klijenti se mogu koristiti i nezavisno od Telnet protokola kao veoma tanak sloj iznad TCP konekcije i koriste se za debagovanje protokola aplikacionog sloja.

Prenos datoteka (eng. file transfer) Prenos datoteka predstavlja jedan od klasičnih servisa Interneta i datira još od ranih 1970-tih. Prenos datoteka se vrši između klijentskog računara i serverskog računara u oba smera (mogu se preuzimati i postavljati datoteke na server). Ovaj servis danas obično koristi za postavljanje datoteka na Veb servere kao i za preuzimanje velikih binarnih datoteka (za manje datoteke, obično se koristi HTT protokolP). Serveri koji čuvaju kolekcije datoteka obično se identifikuju adresom koja počinje sa `ftp` (slično kao što se Veb serveri identifikuju adresom koja počinje sa `www`).

Za prenos datoteka koristi se FTP protokol koji koristi TCP na portu 20 i 21, zatim SCP i SFTP protokoli bazirani na SSH koji nude enkripciju pri prenosu datoteka, itd.

Za prenos datoteka, na klijentskim računarima se obično koriste programi poput `ftp` (komandni program koji direktno implementira FTP protokol), `scp` (komandni program koji kopira datoteke uz korišćenje enkripcije), zatim Veb pregledači koji omogućavaju preuzimanje datoteka sa FTP servera, klijenti poput GnuFTP, Windows Commander i slično.

Ćaskanje (eng. chat) Ćaskanje korisnicima Interneta omogućava uspostavljanje kontakata i „priču" na razne teme kucanjem uživo (eng. on-line). Korisnici pristupaju sobama za ćaskanje (eng. chat room) i time mogu da se uključe u grupnu ili privatnu komunikaciju. Ćaskanje je u današnje vreme zasnovano ili na specifičnim protokolima (npr. IRC) i aplikacijama (npr. Xchat, mIRC) ili se koriste Veb zasnovane sobe za ćaskanje.

Instant poruke (eng. instant messaging) Instant poruke takođe mogu da se podvedu pod ćaskanje. Osnovna razlika je da se instant poruke uglavnom razmenjuju „oči-u-oči" između poznanika, tj. daju direktnu privatnu komunikaciju između dva učesnika, dok ćaskanje u užem smislu obično podrazumeva grupnu komunkaciju u sobi za ćaskanje.

Preteča instant poruka je UNIX program `talk` koji je omogućavao komunikaciju korisnika ulogovanih na isti server. Najpoznatiji servisi koji nude razmenu instant poruka danas su AOL Instant Messenger (AIM), Microsoft MSN, Google Talk, Skype, ICQ, ... Klijentske aplikacije neophodne za slanje instant poruka su specijalizovane aplikacije koje odgovaraju navedenim servisima (npr. Microsoft MSN Messenger). Postoje i aplikacije koje daju mogućnost korišćenja različitih IM sistema (npr. Pidgin). Instant poruke se danas mogu razmenjivati i preko Veba (npr. GMail chat, Facebook chat).

Veb (eng. World Wide Web — WWW) World Wide Web, tj. Veb je Internet servis nastao tek ranih 1990-tih godina, međutim veoma brzo je stekao ogromnu popularnost i postao je najznačajniji Internet servis današnjice. To je sistem međusobno povezanih dokumenata poznatih kao *Veb stranice* koje mogu da sadrže tekst, slike, video snimke i druge multimedijalne materijale. Veb stranice su povezane korišćenjem *veza (linkova)*, tj. predstavljaju *hipertekst*. Korisnici aktivirajući veze (obično jednostavnim klikom mišem) prelaze sa jedne stranice na drugu.

Stranice se čuvaju na specijalizovanim Veb serverima i na zahtev klijenata se prenose klijenske računare gde ih specijalizovani programi prikazuju. Ovi programi nazivaju se *pregledači Veba (eng. Web browsers)*. Najpoznatiji pregledači danas su Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera, itd.

Dostava Veb sadržaja je zasnovana na HTTP protokolu koji koristi TCP na portu 80.

Peer-to-peer (P2P) servisi Popularizacija P2P servisa desila se 1999. kada je servis pod imenom Napster iskorišćen za razmenu velike količine muzičkih MP3 datoteka između velikog broja korisnika širom sveta. S obzirom na kršenje autorskih prava Napster je već 2001. zabranjen, ali je nastao veliki broj P2P protokola i aplikacija. Za razliku od većine Internet servisa koji funkcionišu po klijent-server modelu komunikacije, P2P servisi se zansnivaju na direktnoj

razmeni podataka između različitih klijenata, pri čemu serveri samo služe za koordinaciju komunikacije, bez direktnog kontakta sa samim podacima koji se razmenjuju. P2P servisi se obično koriste za razmenu velikih datoteka (obično video i audio sadržaja). Počevši od 2009. P2P aplikacije čine najveći deo Internet saobraćaja. Najkorišćeniji P2P servisi i protokoli danas su Bittorent, DC++, Gnutella, G2, E-mule, KaZaA (FastTrack). Postoji veliki broj aplikacija koje korisnicima omogućuju korišćenje ovih protokola.

Socijalne mreže Iako su sastavni deo Veba u poslednje vreme socijalne mreže doživljavaju izrazitu ekspanziju i imaju sve veći i veći društveni značaj. Najkorišćenije socijalne mreže današnjice su Facebook, Tweeter i MySpace.

2.3 Internet protokoli

2.3.1 Protokol mrežnog sloja - IP

Internet protokol (eng. Internet Protocol — IP) je protokol koji se koristi za komunikaciju u okviru mrežnog sloja Interneta. Dve osnovne verzije ovog protokola su IPv4 i IPv6. S obzirom da je verzija IPv4 i dalje dominantna (iako se to vremenom menja), u nastavku će ona biti detaljnije opisana.

Osnovni zadatak ovog protokola je da pokuša da dopremi (tj. rutira) paket od izvorišta do odredišta u okviru mreže sa paketnim komutiranjem, isključivo na osnovu navedene adrese (bez analiziranja samog sadržaja i prirode paketa), bez obzira da li su izvorište i odredište u okviru iste mreže ili između njih postoji jedna ili više drugih mreža. Protokol ne daje nikakve garancije da će paketi zaista i biti dopremljeni, ne daje garancije o ispravnosti dopremljenih paketa, ne garantuje da će paketi biti dopremljeni u istom redosledu u kojem su poslati i slično. Garancije ovog tipa obezbeđuju se na višim slojevima komunikacije.

IP adrese. IP protokol uvodi sistem adresa poznatih kao *IP adrese*. U okviru IPv4, adrese su 32-bitni neoznačeni brojevi, koji se obično predstavljaju kao 4 dekadno zapisana broja između 0 i 255. Postoji ukupno $2^{32} \approx 4.3$ milijarde različitih adresa, što se u današnje vreme pokazuje kao nedovoljno. IPv6 će doneti 128-bitne adrese, što bi trebalo da reši ovaj problem. Za dodelu IP adresa, zadužena je *The Internet Assigned Numbers Authority (IANA)*, kao i pomoćni regionalni registri *Regional Internet Registries (RIRs)*.

Svaki uređaj priključen na Internet ima jedinstvenu IP adresu koja ga identifikuje. Neki uređaji imaju uvek istu IP adresu (tzv. *statički dodeljenu*), dok se nekim uređajima dodeljuje različita adresa prilikom svakog povezivanja na mrežu (tzv. *dinamički dodeljenu*). Tako, na primer, studentski server Matematičkog fakulteta u Beogradu ima statički dodeljenu adresu `147.91.64.2` ili binarno zapisano `10010011 1011011 01000000 00000010`. Dinamičke IP adrese se dodeljuju korišćenjem specijalizovanog *protokola za dinamičku konfiguraciju (eng. Dynamic Host Configuration Protocol — DHCP)*. Specijalizovani server (tzv. DHCP server) je zadužen za skup IP adresa koje određuje administrator mreže i na zahtev uređaja koji se priključuje na mrežu dodeljuje mu neku u tom trenutku slobodnu adresu. Server se može konfigurisati tako da dodeljuje bilo koju slobodnu IP adresu, uvek istu adresu koja se određuje na osnovu MAC adrese uređaja koji zahteva IP adresu i slično.

Prvi deo IP adrese određuje mrežu, dok drugi određuje računar u okviru mreže. Ruter do kojega dođe paket, određuje da li je paket potrebno poslati na neki lokalni čvor (koji se nalazi u istoj mreži kao i ruter) ili na neki spoljašnji čvor. U slučaju da paket treba proslediti na neki spoljašnji čvor, ruter gleda samo deo adrese koji određuje mrežu (u ovom slučaju identifikacija pojedinačnog računara nije relevantna) i korišćenjem svojih tabela i algoritama rutiranja određuje na koji od njemu susednih čvorova treba proslediti paket. Ranije, IP adrese su deljene na klase (A, B, C, D, E) i svaka klasa je definisala broj bita za prvi i broj bita za drugi deo deo IP adreses. Tako su, adrese klase A bile dodeljivane jako velikim mrežama (8+24 bita — 128 mreža sa mogućih preko 16.7 korisnika), adrese klase B dodeljivane srednjim mrežama (16+16 bita — preko 16 hiljada mreža sa mogućih 65536 korisnika), a adrese klase C dodeljivane malim mrežama (24+8 bita — preko dva miliona mreža sa mogućih 256 korisnika). Vremenom se pokazalo da ovakva organizacija nije skalabilna (obično su mreže kompanija imale potrebu za više od 256 uređaja, tako su uzimale adrese klase B, čime je veliki broj adresa ostajao nedodeljen, jer je uređaja bilo ipak mnogo manje od 65 hiljada). U novije vreme se koristi pristup označen kao *eng. Classless Inter-Domain Routing (CIDR)*. U ovom slučaju, bitovi adrese mogu biti na proizvoljan način podeljeni između adrese mreže i adrese računara u mreži. Uz IP adrese, šalje se i podatak o broju bita koje određuju mrežu (tzv. subnet mask). Notacija koja se obično koristi je `a.b.c.d/n` (npr. 194.24.16.0/20).

Još jedan od načina da se prevaziđe nedostatak IP adresa je uvođenje privatnih mreža i preslikavanja mrežnih adresa (eng. network address translation — NAT). Naime, u nekim slučajevima nije neophodno da svaki računar ima globalno jedinstvenu IP adresu. Na primer, dovoljno je da ruter (u okviru kućne ili kompanijske mreže) ima globalno jedinstvenu IP adresu, dok računari priključeni na njega mogu da koriste (lokalno jedinstvene) privatne adrese. Za privatne adrese koristi se 16.7 miliona adresa oblika `10.x.x.x`, ili milion adresa oblika `172.16.x.x` ili 65536 adresa oblika `192.168.x.x`. U slučaju da ruter detektuje odredišnu adresu ovog oblika, jasno je da je paket namenjen za lokalnu komunikaciju i šalje se jedinstvenom uređaju sa navedenom lokalnom adresom. U slučaju da je odredišna adresa javna, ruter adresu pošiljaoca zamenjuje svojom adresom (globalno jedinstvenom) i paket prosleđuje na odredište. U slučaju dolaznog paketa, nije odmah jasno na koju privatnu adresu je potrebno poslati paket koji je pristigao. Kako bi se ovo razrešilo lokalna adresa se pakuje i postaje sastavni deo paketa koji se šalje, a ruter, pre prosleđivanja paketa vrši njegovo raspakivanje i određivanje lokalne adrese. Sve ovo, narušava osnovne principe i koncepte IP protokola i zato se NAT smatra prelaznim rešenjem problema nestašice IP adresa, dok IPv6 ne zaživi.

Sistem Imena Domena (DNS). IP adrese su pogodne za korišćenje od strane računara, ali nisu pogodne za ljudsku upotrebu. Kako bi se ljudima olakšalo pamćenje imena računara, uveden je *sistem imena domena (eng. domain name system — DNS)*. DNS se smatra „telefonskim imenikom" Interneta koje imenima domena dodeljuje razne informacije (najčešće IP adrese). Tako npr. već pomenuti studentski server Matematičkog fakulteta u Beogradu ima domen `alas.matf.bg.ac.rs`. Domeni su hijerarhijski organizovani i čitaju se s desna na levo. Tako, npr. domen `rs` označava Republiku Srbiju, `ac.rs` označava akademsku mrežu u Srbiji, `bg.ac.rs` njen čvor u Beogradu, `matf.bg.ac.rs`

Matematički fakultet, dok `alas.matf.bg.ac.rs` označava konkretan studentski server koji imenovan u čast velikog matematičara Mihaila Petrovića Alasa. Domeni najvišeg nivoa mogu biti bilo nacionalni (kao u navedenom primeru), bilo generički (npr. `.com`, `.org`, `.net`). Domeni se koriste u okviru jedinstvenih lokatora resursa na Vebu (URL), u okviru adresa elektronske pošte, itd.

Prilikom preslikavanja domena u adrese, koriste se usluge distribuirane DNS baze podataka. Specijalizovani DNS serveri čuvaju delove ove baze. Ovi serveri su hijerarhijski organizovani i ova hijerarhija uglavnom prati hijerarhiju domena.

2.3.2 Protokoli transportnog sloja - TCP, UDP

TCP

TCP (Transmission Control Protocol) je protokol transportnog sloja u okviru Interneta koji pre komunikacije vrši uspostavljanje pouzdane konekcije između dva hosta. Kanal komunikacije je dvosmeran (eng. full duplex). Konekcija se uspostavlja tako što klijent i server razmene tri poruke (eng. three way handshake) u kojima klijent traži uspostavljanje konekcije, server potvrđuje da prihvata konekciju i konačno klijent potvrđuje da je konekcija uspostavljena. Prava komunikacija može da započne tek nakon što je konenkcija uspostavljena, što može da traje neko vreme.

TCP garantuje *pouzdanost prenosa podataka (eng. reliable transfer)* čime se garantuje da će paketi koji su poslati biti primljeni (i to u istom redosledu u kojem su poslati). S obzirom da niži mrežni slojevi ne garantuju dostavu paketa, TCP protokol mora da se stara o tome da paketi koji zalutaju automatski budu ponovno poslati, kao i da na prihvatnoj strani automatski permutuje primljene pakete tako da odgovaraju redosledu slanja. Kako bi ovo moglo da bude realizovano, uvodi se potvrda prijema paketa (eng. acknowledgment), tj. nakon prijema jednog ili više paketa, vrši se slanje poruke pošaljiocu koja govori da su ti paketi zaista primljeni. Pošaljioc, na osnovu ovoga, može da odluči da ponovno pošalje paket koji je ranije već bio poslat, u slučaju da u određenom vremenskom periodu ne dobije potvrdu prijema.

TCP takođe uvodi *kontrolu i korekciju grešaka*. Ovo je dodatna slaba provera (vrši se samo kontrola parnosti), jer se pretpostavlja da se jača provera (obično CRC) vrši na nižim slojevima. Ipak, u praksi se pokazuje da ova provera ima smisla i uspeva da uoči i ispravi veliki broj grešaka koje promaknu ostalim kontrolama.

TCP uvodi i *brzinu protoka (eng. flow control)* kojom se kontroliše brzina slanja kako se ne bi desilo da brzi uređaji šalju pakete brzinom većom od one kojom spori uređaji mogu da ih prime (npr. brz računar koji šalje podatke na spor mobilni telefon). Važna odlika TCP protokola je da vrši *kontrolu zagušenja (eng. congestion control)*. Pojava zagušenja se javlja kada više čvorova pokušava da pošalje podatke kroz mrežu koja je već na granicama svoje propusne moći. U takvim situacijama, dešava se da brzina komunikacije u celoj mreži opada za nekoliko redova veličina. Naime, broj izgubljenih paketa se višestruko povećava jer unutrašnji čvorovi mreže (ruteri) ne mogu da prihvate nove pakete jer su im prihvatni baferi prepuni. TCP pokušava da detektuje ovakve situacije i da u tim slučajevima uspori sa slanjem paketa dok se mreža ne rastereti. Jedna od tehnika koje se koriste u cilju smanjenja zagušenja je da se pri početku komunikacije paketi šalju sporije (eng. slow-start), a da se brzina slanja postepeno

povećava kada se utvrdi da paketi zaista i stižu na odredište. Ovo je jedan od razloga zbog čega TCP spada u grupu sporijih protokola i ne koristi se za aplikacije kod kojih je brzina prenosa presudna.

UDP

UDP (User Datagram protocol) je protokol transportnog sloja u okviru Interneta koji ne vrši uspostavljanje konekcije pre započinjanja komunikacije. Prilikom korišćenja UDP protokola ne vrši se potvrda prijema poslatih paketa, tako da se komunikacija može smatrati nepouzdanom. Osnovni razlozi korišćenja UDP protokola su, pre svega, njegova brzina i zbog toga se uglavnom koristi od strane aplikacija koje imaju potrebu za komunikacijom u *realnom vremenu (eng. real time)*, kao što su npr. audio-video prenosi, internet telefonija, igrice i sliňco. Takođe, UDP se koristi za aplikacione protokole koji daju elementarne mrežne usluge i vrše kontrolu mreže (npr. DHCP, DNS, SNMP).

Soketi (eng. sockets) - TCP/IP programski interfejs

Većina savremenih operativnih sistema i programskih jezika daje direktnu podršku za korišćenje Internet (TCP/IP familije) protokola. Podrška za korišćenje ovih protokola u okviru programa se realizuje kroz koncept soketa (eng. socket). Socket je apstrakcija kojom se programeru predstavlja kanal komunikacije (zasnovan bilo na TCP bilo na UDP protokolu). Programer piše podatke u soket ili čita podatke iz soketa, obično na sličan način kao da je u pitanju obična datoteka, dok se operativni sistem brine o svim aspektima stvarne mrežne komunikacije.

Ilustrujmo ovo jednostavnim primerom aplikacije u programskom jeziku Java koja koristi TCP sokete za mrežnu komunikaciju, prikazanim na slici 2.3. Klijent prima rečenicu od korisnika i šalje je serveru vršeći upis u odgovarajući soket. Nakon prijema odgovora od servera (u ovom slučaju izmenjene rečenice), klijent ga prikazuje korisniku. Server čeka konekcije od klijenata korišćenjem za to specijalizovanog soketa (`ServerSocket`). Kada klijent uspostavi komunikaciju otvara se običan soket za komunikaciju, prihvata se rečenica od klijenta, menjaju se sva mala slova u velika i izmenjena rečenica se šalje nazad klijentu, nakon čega se nastavlja iščekivanje nove konekcije.

2.3.3 Protokoli aplikacionog sloja

HTTP

Hypertext Transfer Protocol (HTTP), je protokol aplikacionog sloja koji predstavlja osnovu Veba. HTTP je implementiran u okviru dve vrste programa: klijentskim programima, najčešće pregledačima Veba i serverskim programima, najčešće Veb serverima. Ovi programi međusobno komuniciraju razmenom HTTP poruka. HTTP definiše strukturu ovih poruka i način na koji klijenti i serveri razmenjuju ove poruke. Pre nego što objasnimo detalje protokola, objasnimo neke od osnovnih pojmova Veba. Veb je distribuirana aplikacija zasnovana na *Veb stranicama*. Veb stranice se sastoje od objekata — hipertekstualnih datoteka opisanih na jeziku HTML, slika u raznim formatima (npr. JPG, PNG, GIF), Java apleta i slično. Svaki pojedinačni objekat ima jedinstvenu adresu u obliku tzv. *URI (Uniform Resource Identifier)*.

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String userSentence, modifiedSentence;
        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer =
          new BufferedReader(new InputStreamReader(
                                clientSocket.getInputStream()));
        userSentence = inFromUser.readLine();
        outToServer.writeBytes(userSentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}
```

```
import java.io.*;
import java.net.*;
class TCPServer {
  public static void main(String argv[]) throws Exception {
      String clientSentence, modifiedSentence;
      ServerSocket welcomeSocket = new ServerSocket(6789);
      while(true) {
           Socket connectionSocket = welcomeSocket.accept();
           BufferedReader inFromClient =
             new BufferedReader(new InputStreamReader(
                                  connectionSocket.getInputStream()));
           DataOutputStream outToClient =
             new DataOutputStream(connectionSocket.getOutputStream());
           clientSentence = inFromClient.readLine();
           modifiedSentence = clientSentence.toUpperCase() + '\n';
           outToClient.writeBytes(modifiedSentence);
        }
    }
}
```

Slika 2.3: Java klijent i server koji koriste TCP sokete za komunikaciju.

HTTP protokol dolazi u dve verzije. Rana verzija, HTTP 1.0 korišćena je u samom početku razvoja Veba i krajem 1990-tih, zamenjena je novijom verzijom HTTP 1.1 koja je i danas aktuelna i koja zadržava kompatibilnost sa prvom verzijom. Obe verzije koriste TCP za komunikaciju nižeg nivoa. Razlika je, na primer u tome, što se u okviru starije verzije TCP konekcija automatski zatvara nakon prijema HTTP odgovora, dok se u okviru novije verzije ista konekcija koristi za prenos više objekata, što doprinosi brzini zbog sporog uspostavljanja TCP konekcija.

HTTP protokol funkcioniše tako što klijent uspostavlja TCP konekciju (obično na portu 80) sa serverskim računarom, i zatim šalje *HTTP zahteve (eng. HTTP request)* za određenim Veb objektima serverskom računaru. Ukoliko traženi objekti postoje na serveru, server kroz uspostavljenu TCP konekciju objekte šalje u obliku *HTTP odgovora (eng. HTTP request)*. Važno je naglasiti da nakon slanja odgovora, server ne održava tj. ne koristi apsolutno nikakve informacije o klijentu, odnosno da je HTTP *protokol bez stanja (eng. stateless protocol)*.

I HTTP zahtevi i odgovori se navode u precizno specifikovanom obliku.

Navedimo primer HTTP zahteva.

```
GET /~filip/uvit/ HTTP/1.1
Host: www.matf.bg.ac.rs
User-Agent: Mozilla/5.0 Firefox/3.5.8
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Ovaj zahtev prati opšti format HTTP zahteva:

```
metod putanja verzija
polje: vrednost
...
polje: vrednost

sadrzaj
```

HTTP zahtev se šalje nakon što je uspostavljena TCP konekcija sa nekih host računarom. U prvoj liniji, navodi s metoda, putanja (na serveru) do objekta koji se zahteva i verzija HTTP protokola.

Postoji nekoliko različitih metoda, pri čemu se najčešće koriste `GET`, `POST` i donekle `HEAD`.

`HEAD` metod se uglavnom koristi isključivo za debagovanje jer se njime od servera zahteva da dostavi samo zaglavlje HTTP odgovora, bez stvarne dostave Veb objekata.

Ukoliko je zahtev takav da služi samo da zatraži čitanje i dostavljanje nekih podataka sa servera, preporučuje se korišćenje GET metoda. U ovom slučaju dodatne informacije koje se prenose sa klijenta na server se kodiraju u okviru URI. Primer takvog URI je npr.

`http://www.matf.bg.ac.rs/~filip/index.php?content=courses`

U ovom slučaju, dodatni parametar je `content` čija je vrednost `courses`.

Ukoliko je zahtev takav da služi da se od servera zahteva da izvrši neki sporedni efekat (npr. da upiše nešto u bazu podataka), preporučuje se korišćenje

POST metoda. U ovom slučaju dodatne informacije koje se prenose sa klijenta sa server se ne vide u okviru URI, već se navode kao sadržaj HTTP zahteva. Klijentski programi se obično pišu tako da upozore korisnika ukoliko šalju POST zahtev koji je već poslat.

HTTP zahtev sadrži i niz polja i njihovih vrednosti kojima klijent serveru saopštava neke relevantne informacije. Navedimo neke od najčešće korišćenih:

Host: - ovo je obavezno polje u HTTP/1.1 i sadrži ime hosta na koji se zahtev šalje.

User-Agent: - ovim se identifikuje klijentski softver koji šalje zahteev

Accept: - ovim klijent navodi vrstu sadržaja (MIME) koju priželjkuje. `q` vrednosti određuju prioritet.

Accept-Language: - ovim klijent navodi jezik koji priželjkuje.

Accept-Charset: - ovim klijent navodi kodnu stranu koju priželjkuje.

Connection: - ovim se navodi da li se želi perzistentna (`keep-alive`) ili jednokratna (`close`) TCP konekcija.

If-modified-since: - ovim klijent serveru naglašava da mu objekat pošalje samo ako je bio modifikovan od datuma navedenog u ovom polju (ukoliko objekat nije modifikovan, on se ne šalje ponovo već klijent prikazuje verziju koja je prethodno bila dostavljena i sačuvana je u kešu).

Nakon prijema HTTP zahteva, server šalje HTTP odgovor.

```
HTTP/1.1 200 OK
Date: Sun, 07 Mar 2010 14:53:05 GMT
Server: Apache/2.2.9 (Unix) mod_ssl/2.2.9 OpenSSL/0.9.8h PHP/5.2.6
X-Powered-By: PHP/5.2.6
Content-Length: 2905
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
```

Ovaj odgovor prati opšti format HTTP odgovora:

```
verzija kod status
polje: vrednost
...
polje: vrednost

sadrzaj
```

Kod i status su najčešće nešto od sledećeg:

200 OK - Zahtev je uspešan i informacija se vraća u okviru odgovora

301 Moved Permanently - Zahtevani objekat je premešten na lokaciju koja je navedena u polju `Location:`. Klijentski program može automatski da pošalje novi zahtev na dobijenu lokaciju.

304 Not Modified - Zahtevani objekat nije promenjen od datuma navedenog u zahtevu i nema ga potrebe ponovo slati.

400 Bad Request - Server nije uspeo da razume zahtev.

404 Not Found - Zahtevani objekat nije nađen na serveru.

500 Internal Server Error - Došlo je do neke interne greške u radu serverskog programa.

505 HTTP Version Not Supported - Server ne podržava verziju HTTP protokola.

Primetimo da kodovi koji počinju sa 2 govore o tome da je sve proteklo kako treba, kodovi koji počinju sa 3 obaveštavaju korisnika o nekoj redirekciji, kodovi koji počinju sa 4 govore o nekoj grešci u zahtevu (grešci koju je napravio klijent), a kodovi koji počinju sa 5 govore o nekoj greči na strani servera.

Neka od najčešće navedenih polja u HTTP odgovorima su:

Date: - tačno vreme kada je odgovor poslat

Server: - identifikacija veb server programa koji je poslao odgovor

Content-Type: - vrsta sadržaja (MIME) poslana u okviru odgovora

Content-Length: - dužina sadržaja u bajtovima

Last-Modified: - poslednje vreme kada je sadržaj modifikovan na serveru

SMTP, POP3 i IMAP

Pre nego što budu opisani pojedinačni protokoli, objasnimo ukratko osnovne principe funkcionisanja elektronske pošte. Kako bi se dostavila elektronska poruka sa računara pošaljioca na računar primaoca, potrebno je da u komunikaciju budu uključeni i server elektronske pošte pošaljioca, kao i server elektronske pošte primaoca. Pošaljilac sa svog računara dostavlja poruku svom serveru od kojega se zahteva da poruku dostavi serveru primaoca i smesti je u poštansko sanduče primaoca. U ovom delu komunikacije koristi se protokol za slanje pošte SMTP. Jednom kada je izdao nalog svom serveru pošaljilac je završio svoj posao i njegov server nastavlja da brine o dostavljanju poruke tj. vrši komunikaciju sa serverom primaoca i pokušava da dostavi poruku sve dok ili ne uspe ili dok ne ustanovi da dostavljanje poruke nije moguće (dobije informaciju da postoji greška u adresi ili prođe određeno vreme, a ne uspe da poruku dostavi, i slično). U slučaju da dostavljanje poruke nije uspelo, server obično obaveštava pošaljioca da dostavljanje nije uspelo. Kada se poruka uspešno dostavi na server primaoca, ona se smešta u njegovo poštansko sanduče gde je smeštena sve dok primaoc ne proveri svoju poštu i ne poželi da pročita dobijenu poruku. U tom trenutku potrebno je dostaviti poruku sa servera primaoca do njegovog ličnog računara za šta se koriste protokoli za primanje pošte kao što su POP i IMAP.

Simple Mail Transfer Protocol (SMTP) je standardni protokol za slanje pošte. Navedimo jedan primer SMTP sesije između klijenta koji šalje poštu i servera koji je prima, kako bi je dalje prosledio:

```
Server: 220 smtp.example.com ESMTP Postfix
Client: HELO relay.example.org
Server: 250 Hello relay.example.org, I am glad to meet you
Client: MAIL FROM:<bob@example.org>
Server: 250 Ok
Client: RCPT TO:<alice@example.com>
Server: 250 Ok
Client: RCPT TO:<theboss@example.com>
Server: 250 Ok
Client: DATA
Server: 354 End data with <CR><LF>.<CR><LF>
Client: From: "Bob Example" <bob@example.org>
        To: Alice Example <alice@example.com>
        Cc: theboss@example.com
        Date: Tue, 15 Jan 2008 16:02:43 -0500
        Subject: Test message

        Hello Alice.
        This is a test message with 5 header fields and
        5 lines in the message body.
        Your friend,
        Bob
        .
Server: 250 Ok: queued as 12345
Client: QUIT
Server: 221 Bye
```

Post Office Protocol (POP) je jednostavni protokol za preuzimanje poruka sa servera, pri čemu se prilikom preuzimanja poruke obično brišu sa servera. Preuzete poruke se snimaju na klijentski računar i sva njihova dalja obrada (pretraga, organizovanje, sortiranje) se vrši na klijentskom računaru koji nakon preuzimanja poruka više ne mora da ima pristup Internetu. POP protkol koristi TCP konekciju na portu 110. Navedimo jedan primer POP3 sesije između klijenta i servera:

```
Server:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
Client:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
Server:    +OK mrose's maildrop has 2 messages (320 octets)
Client:    STAT
Server:    +OK 2 320
Client:    LIST
Server:    +OK 2 messages (320 octets)
           1 120
           2 200
           .
Client:    RETR 1
Server:    +OK 120 octets
           <the POP3 server sends message 1>
           .
Client:    DELE 1
Server:    +OK message 1 deleted
Client:    RETR 2
Server:    +OK 200 octets
           <the POP3 server sends message 2>
Client:    QUIT
Server:    +OK dewey POP3 server signing off (maildrop empty)
```

Osnovne komande koje klijentski softver izdaje su

`APOP` - ovim se vrši autorizacija klijenta navođenjem njegovog korisničkog imena

i kriptovane lozinke.

`STAT` - statistika o stanju poštanskog sandučeta

`LIST` - lista poruka

`RETR` - primanje poruke sa navedenim rednim brojem

`DELE` - brisanje poruke sa navedenim rednim brojem

`QUIT` - prekidanje sesije

Internet Message Access Protocol (IMAP) je znatno napredniji protokol za primanje pošte. On je prevashodno namenjen korisnicima koji su mobilni tj. koji svojoj pošti pristupaju sa različitih računara. Kako bi ovakvi korisnici imali mogućnost pristupa svim svojim porukama, nije poželjno brisati ih sa servera (iz poštanskog sandučeta) prilikom preuzimanja. Klijenti za elektronsku poštu na lokalnim računarima obično omogućavaju korisnicima sortiranje poruka, organizovanje u fascikle, pretragu i slično. IMAP protokol je projektovan tako da se ovakva funkcionalnost obezbedi tako što se korisnicima omogući da ove funkcije izvode direktno u svom poštanskom sandučetu na serveru. Mane ovog pristupa su što se zahteva da korisnici imaju pristup Internetu sve vreme dok rade sa svojom elektronskom poštom. Određeni broj Veb aplikacija za rad sa elektronskom poštom je zasnovan na IMAP protokolu.

FTP

File Transfer Protocol (FTP) je protokol za prenos datoteka između računara. Protokol datira još od 1970-tih i doba ranog Interneta, ali se i danas koristi. U okviru tipične FTP sesije, korisnik sedi za jednim host računarom i želi da prenosi datoteke na ili sa drugog host računara. FTP koristi TCP kao protokol za komunikaciju nižeg nivoa. FTP protokol ostvaruje dve TCP konekcije za prenos datoteka. Jedna konekcija (obično na portu 21) se koristi za prenos kontrolnih informacija, a druga (obično na portu 20) za prenos samih podataka. Za svaku datoteku, otvara se nova konekcija za prenos podataka, koja se automatski zatvara kada se završi prenos datoteka. Za to vreme kontrolna konekcija sve vreme ostaje otvorena. Za razliku od HTTP protokola, tokom FTP sesije server mora da čuva određene podatke o korisniku (na primer, tekući direktorijum) tj. FTP je protokol koji *čuva stanje (eng. statefull protocol)*. Kontrole koje se izdaju serveru putem kontrolne konekcije se zapisuju u čitljivom ASCII obliku. Navedimo neke:

`USER` **username** - koristi se za slanje identifikacije korisnika serveru

`PASS` **password** - koristi se za slanje lozinke korisnika serveru

`LIST` - koristi se kako bi se serveru poslala poruka da pošalje listu datoteka u tekućem direktorijumu. Ova lista se šalje preko konekcije za podatke

`RETR` **filename** - koristi se kako bi se sa servera (iz tekućeg direktorijuma) prenela datoteka sa datim imenom na klijent (u tekući direktorijum)

`STOR` **filename** - koristi se kako bi se sa klijenta (iz tekućeg direktorijuma) prenela datoteka sa datim imenom na server (u tekući direktorijum)

Server obično na zahteve klijenta otvara konekciju za prenos podataka, a istovremeno preko kontrolne konekcije šalje statusne poruke ili poruke o greškama kao što su npr.

```
331 Username OK, password required
25  Data connection already open; transfer starting
425 Can't open data connection
452 Error writing file
```

SSH

Ovaj protkol omogućava enkrpiciju podataka prilikom njihovog slanja i ključan je protokol za bezbednost podataka.

Glava 3

Jezici za obeležavanje

U današnjem dobu računara, izdvajaju se dva paradigmatična pristupa za kreiranje tekstualnih dokumenata — (i) WYSIWYG (eng. What You See Is What You Get) pristup i (ii) korišćenje jezika za obeležavanje. U nastavku će ukratko biti opisana oba pristupa. Naglasak će biti stavljen na eksplicitno obeležavanje teksta korišćenjem jezika za obeležavanja zbog mnogobrojnih prednosti koje ovaj pristup donosi.

WYSIWYG pristup. Alati zasnovani na WYSIWYG pristupu zahtevaju od korisnika da tekst uredi u obliku koji je spreman za konačno prikazivanje na ciljnom medijumu (npr. štampanje na papiru). Tekst se uređuje oslanjajući se direktno na njegovu grafičku prezentaciju, najčešće korišćenjem miša i sličnih elemenata grafičkog korisničkog okruženja. Tipični primeri ovakvih alata su alati za kancelarijsko poslovanje (npr. Microsoft Office, OpenOffice.org).

Ekplicitno obeležavanje teksta. Tehnika obeležavanja teksta seže još iz perioda pre postojanja računara, kada su ljudi pre štampanja teksta, ručno, na rukopisu označavali na koji način bi pojedini delovi trebalo da budu odštampani (npr. kojom vrstom i veličinom slova). Slična tehnika se koristi i danas, u eri računara, i tekst se dodatno obeležava informacijama koje ga opisuju. Tehnika eksplicitnog obeležavanja strukture dokumenata olakša njihovu automatsku obradu. Obeleženi dokumenti postaju uskladištene informacije koje je moguće automatski obrađivati korišćenjem raznovrsnim računarskih aplikacijama, ali i prikazivati u obliku pogodnom za čitanje od strane čoveka.

Ovaj pristup na svom značaju dobija kada se izvrši jasno i eksplicitno razdvajanje obelažavanja *logičke strukture* i obeležavanja *vizuelne prezentacije* dokumenta. Logička struktura dokumenta podrazumava njegovu organizaciju na manje jedinice (npr. poglavlja, sekcije, pasuse), kao i označavanje njegovih istaknutih delova (npr. primeri, citati, mdefinicije i teoreme). Vizuelna prezentacija određuje izgled dokumenta u trenutku prikazivanja ili štampanja. Na primer, ona određuje vrstu i veličinu slova kojima se određeni delovi teksta predstavljaju, prored koji se koristi, raspored delova dokumenta na papiru ili ekranu, boju delova dokumenta i slično. Razdvajanje logičke strukture dokumenata od njihove grafičke prezentacije daje mogućnost da se uz minimalan trud istim podacima pridruže sasvim različiti vizuelni prikazi.

Prilikom eksplicitnog obeležavanja teksta, koriste se jezici za obeležavanje teksta (eng. *markup languages*). To su veštački jezici u kojima se korišćenjem posebnih *oznaka* opisuje logička struktura teksta ili njegov grafički izgled. Najpoznatiji jezici za obeležavanje su HTML, TEXtj. LATEX, PostScript, RTF, itd.

Svaki od ovih jezika odlikuje se konkretnom sintaksom označavanja i koristi se za označavanje jednog tipa dokumenta (npr. HTML se koristi za označavanje hipertekstualnih dokumenata). Zbog toga se ovi jezici nazivaju i *objektni jezici*. U praksi se često javlja potreba za označavanjem velikog broja različitih tipova dokumenata (npr. označavanje pisama, tehničkih izveštaja, zbirki pesama, itd). Jasno je da svaki pojedinačni tip dokumenata zahteva svoj način označavanja i skup oznaka pogodnih za njegovo označavanje. Ovo dalje omogućava izradu specifičnih softverskih alata pogodnih za određenu vrstu obrade specifičnih tipova dokumenata. Kako bi se na precizan i uniforman način omogućilo definisanje konkretnih objektnih jezika za označavanje različitih tipova dokumenata, razvijeni su i *meta* jezici. Najpoznatiji meta jezici za obeležavanje su SGML i XML u čijem okviru su definisani objektni jezici HTML, XHTML, MathML, SVG itd.

3.1 SGML

Standardni opšti jezik za obeležavanje (*Standard Generalized Markup Language*) je meta jezik za obeležavnje standardizovan od strane međunarodne organizacije za standarde (pod oznakom „ISO 8879:1986 SGML"). Jezik je razvijen za potrebe kreiranja mašinski čitljivih dokumenata u velikim projektima industrije, državne uprave, vojske itd. Osnovna motivacije prilikom standardizovanja ovog jezika je bila da se obezbedi trajnost dokumentima i njihova nezavisnost od aplikacija kojima su kreirani. Informacije skladištene u okviru SGML dokumenta postaju nezavisne od platforme tj. od softvera i hardvera. Pretečom jezika SGML smatra se jezik GML (Generalized Markup Language) nastao u kompaniji IBM 1960-tih. Jedna od značajnijih primena jezika SGML je bila izrada drugog, elektronskog, izdanja „Oksfordskog rečnika engleskog jezika (OED)". Fragment ovog rečnika je prikazan na Slici 3.1.

Slika 3.1: Fragment oksfordskog rečnika obeležen SMGL elementima.

Slobodno se može reći da je najznačajnija primena jezika SGML došla kroz objektni jezik HTML čije su prve verzije definisane upravo u okviru jezika SGML. Jezik HTML služi za obeležavanje hipertekstualnih dokumenata i postao je standardni jezik za obeležavanje dokumenata na vebu. Svaki jezik za obeležavanje koji je definisan u SGML-u naziva se i *SGML aplikacija*. Tako i za jezik HTML kažemo da je SGML aplikacija[1].

3.1.1 Uvodni primeri dokumenata

U ovom poglavlju će biti prikazano nekoliko primera SGML dokumenata i na njima će biti opisano nekoliko osnovnih pojmova jezika SGML. Detaljniji opis ovih pojmova će biti dat u poglavljima koja slede.

SGML se koristi da bi se obeležila struktura dokumenata određenog tipa. Tako, na primer, zbirka pesama sadrži nekoliko pesama, pri čemu se svaka pesma sastoji od nekoliko strofa, a svaka strofa od nekoliko stihova. SGML uvodi oznake kojima se obeležavaju elementi dokumenta.

```
<!DOCTYPE zbirka SYSTEM "zbirka-pesama.dtd">
<zbirka>
    <pesma autor="Čika Jova Zmaj">
        <strofa>
            <naslov>Žaba čita novine</naslov>
            <stih>Sedi žaba sama
            <stih>na listu lokvanja.
            <stih>Od žarkoga sunca
            <stih>štitom se zaklanja.
        </strofa>
    </pesma>
</zbirka>
```

Navedimo i primer jednog jednostavnog HTML dokumenta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Moj prvi HTML dokument</title>
  </head>
  <body>
    <p>Zdravo svete! <img src="zdravo.gif" alt="zdravo">
    <p>Copyright (&copy;) Milena
  </body>
</html>
```

U oba primera, sadržaj dokumenta je obeležen oznakama koje određuju njegovu strukturu. Dokumenti se sastoje od međusobno ugnježđenih *elemenata*. Za obeležavanje elemenata se koriste *etikete (tagovi)* oblika `<ime-elementa>` i `</ime-elementa>` (na primer `<strofa>` i `</strofa>` ili `<body>` i `</body>`). Elementi sadrže tekst, druge elementi ili kombinaciju i jednog i drugog. Elementi mogu biti dodatno okarakterisani *atributima* oblika `ime-atributa="vrednost-atributa"` (na primer `naslov= "Žaba čita novine"`). U okviru teksta mogu

[1] Kasnije će biti precizirano da najnovije verzije jezika HTML predstavljaju XML aplikacije.

se pojaviti i *karakterski entiteti* oblika `&ime-entiteta;` (na primer `&copy;`) koji označavaju određene karaktere.

Sadržaj i značenje elemenata nije propisano meta jezikom već svaki objektni jezik definisan u okviru SGML-a definiše sopstveni skup etiketa koje koristi za obeležavanje i definiše njihovo značenje kao i moguće međusobne odnose. Svakom dokumentu, pridružen je njegov tip. Tip dokumenta određuje sintaksu dokumenta tj. određuje koji elementi, atributi i entiteti se mogu javiti u okviru dokumenta i kakav je njihov međusobni odnos. Posebni programi koje nazivamo *SGML parserima* ili *SGML validatorima* mogu da ispitaju da li je dokument u skladu sa svojim tipom tj. da li zadovoljava sva sintaksna pravila propisana odgovarajućim tipom. Pripadnost određenom tipu dokumenta, izražava se deklaracijom `<!DOCTYPE>` koja se navodi na početku samog dokumenta. U okviru ove deklaracije, pored informacija o imenu tipa dokumenta, organizaciji koja ga je kreirala i slično, nalazi se obično uputnica na *definiciju tipa dokumenta (eng. Document type definition — DTD).*

U prvom primeru tip dokumenta je definisan datotekom `zbirka-pesama.dtd`, dok je u drugom primeru tip dokumenenta definisan datotekom `http://www.w3.org/TR/html4/strict.dtd`. Oznaka `PUBLIC` u drugom primeru ukazuje na to da je tip dokumenta javan i dostupan. Ove datoteke definišu elemente od kojih se grade konkretni dokumenti. Tako, tip dokumenta zbirke pesama uvodi elemente `zbirka`, `pesma`, `strofa` i `stih` i zahteva da se zbirka sastoji od nekoliko pesama, da se svaka pesma sastoji od nekoliko strofa, a da se svaka strofa sastoji od nekoliko stihova. Takođe, u okviru ove definicije tipa dokumenta, specifikovano je da pesma ima atribut `autor` kao i šta sve može biti vrednost ovog atributa. Ova specifikacija, data je na formalnom i precizno definisanom jeziku koji će biti opisan u zasebnom poglavlju:

```
<!ELEMENT zbirka - -  (pesma+)>
<!ELEMENT pesma  - -  (naslov?, strofa+)>
<!ATTLIST pesma autor CDATA #REQUIRED>
<!ELEMENT naslov - O  (#PCDATA)>
<!ELEMENT strofa - O  (stih+)>
<!ELEMENT stih   O O  (#PCDATA)>
```

Dakle, korišćenje SGML-a podrazumeva kreiranje sopstvenih ili korišćenje javnih tipova dokumenata i obeležavanje dokumenata u skladu sa njihovim željenim tipom. Proces kreiranja novih tipova dokumenata podrazumeva izradu:

SGML deklaracije - formalnog opisa leksike samih dokumenata koja prevashodno određuje koji karakteri se koriste prilikom kreiranja dokumenata.

Definicije tipa dokumenta - formalnog opisa sintakse samih dokumenata koja određuje od kojih elemenata, etiketa, atributa i entiteta se dokument sastoji i kakav je njihov međusobni odnos.

Semantičke specifikacije - neformalnog opisa semantike elemenata, etiketa i atributa koji se koriste u okviru dokumenata. Ovakva specifikacija može u sebi da sadrži i neka dodatna ograničenja koja se ne mogu izraziti u okviru formalne definicije tipa dokumenta.

3.1.2 Osnovni konstrukti

Elementi i etikete. Osnovna gradivna jedinica SGML dokumenata su *elementi*. Elementi su obično označeni *etiketama (eng. tag)*. Razlikuju se *otvarajuće etikete (eng. opening tag)* koje označavaju početak elementa i koje su oblika <ime-elementa> i *zatvarajuće etikete (eng. closing tag)* koje označavaju kraj elementa i koje su oblika </ime-elementa>. Napomenimo da elementi nisu isto što i etikete. Element sačinjava početna etiketa, završna etiketa i sav sadržaj (tekst i drugi elementi) koji se nalaze između njih. Ime elementa se navodi i početnoj etiketi i u završnoj etiketi. Imena elemenata dozvoljeno je pisati i malim i velikim slovima i ne pravi se razlika između velikih i malih slova.

Na primer, element ul jezika (tipa dokumenta) HTML, služi da označi neku listu nabrojanih stavki, i u primeru koji sledi njegov sadržaj čine tri elementa li, čiji su sadržaji niske `Lista 1`, `Lista 2` i `Lista 3`:

```
<ul>
<LI>Lista 1</LI>
<li>Lista 2</li>
<li>Lista 3</li>
</ul>
```

Kod nekih SGML elemenata moguće je izostaviti završne etikete, dok je kod nekih čak moguće izostaviti i početne etikete. Na primer, u jeziku HTML, elementi p služe da označe pasuse. Pasusi ne zahtevaju navođenje završne etikete . Početak novog pasusa implicitno označava kraj prethodnog, slično kao i oznaka kraja okružujućeg elementa </body>.

```
<body>
  <p>Zdravo svete!
  <p>Copyright (&copy;) Milena
</body>
```

Sličan je slučaj i sa elementima stih navedenim u primeru zbirke pesama.

Neki SGML elementi nemaju svoj sadržaj. Npr. HTML element koji označava prelazak u novi red br. Kod praznih elemenata najčešće je zabranjeno navoditi završnu etiketu.

Svi elementi koji se mogu koristiti u okviru jednog dokumenta se navode u okviru DTD. Za svaki element se navodi da li je obavezno korišćenje etiketa i precizno se opisuje njegov dozvoljeni i nedozvoljeni sadržaj.

Atributi. Atributi sadrže dodatne informacije o SGML elementima. Atributi imaju svoj naziv i vrednost. Naziv atributa je razdvojeno od vrednosti znakom jednakosti. Vrednost atributa mora biti navedena u okviru dvostrukih (`""`) ili jednostrukih navodnika (`''`). U okviru dvostrukih navodnika moguće je korišćenje jednostrukih i obratno. Ponekad navodnici, kod vrednosti atributa, mogu biti izostavljeni. Atributi elementa se navode u okviru njegove početne etikete. Na primer, atribut href elementa a jezika HTML određuje odredište hiperveze:

```
<a href="http://www.google.com">Link na google</a>
```

Imena atributa su nezavisna od veličine slova, dok vrednosti nekada zavise, a nekada ne zavise od veličine slova.

Entiteti. SGML daje mogućnost imenovanja delova sadržaja na portabilan način. Koncept *eniteta* u SGML uvodi izvesnu vrstu makro zamena. Zamena entiteta se vrši kada se dokumenti analiziraju odgovarajućim parserom. Na primer, moguće je deklarisati entitet pod imenom `uvit` koji se zamenjuje tekstom `Uvod u Veb i Internet tehnologije`, i zatim se u okviru ovog dokumenta na ime predmeta pozivati korišćenjem *reference na entitet*. Postoji nekoliko vrsta entiteta i referenci na entitete.

Prva vrsta entiteta su *obični entiteti (eng. regular entities)*. Reference na njih počinju sa znakom `&` i završavaju se sa `;` i moguće ih je navoditi u okviru teksta dokumenta (ne u okviru DTD). Tako, ako se negde u okviru dokumenta javi sadržaj:

```
Nastava iz predmeta "&uvit;" se odvija utorkom.
```

ovim je u stvari kodiran tekst:

```
Nastava iz predmeta "Uvod u Veb i Internet tehnologije"
se odvija utorkom.
```

Druga vrsta entiteta su *parametarski entiteti (eng. parameter entities)*. Reference na njih počinju znakom `%` i završavaju se sa `;` i moguće ih je navoditi samo u okviru DTD dokumenta (ne u okviru objektnih dokumenata).

Treća vrsta entiteta su *karakterski entiteti (eng. character entities)*. Njima se uvode imena koja označavaju određene karaktere. Koriste se da bi se naveli karakteri koji imaju specijalno značenje, zatim neki retko korišćeni karakteri, karakteri koji nisu podržani tekućim kodiranjem karaktera ili karakteri koje je nemoguće uneti u okviru softvera za kreiranje dokumenata. Na primer, u jeziku HTML `"<"` označava karakter `<`, dok `"&quot;"` označava karakter `"`.

Pored referenci na karakterske entitete, za predstavljanje karaktera u dokumentima je moguće koristiti i *numeričke karakterske reference*. One navode kao brojevi zapisani (bilo dekadno, bilo heksadekadno) između `&#` i `;`. Obično ove vrednosti odgovaraju ISO 10646, tj. UNICODE kodovima. Na primer, `"&#1114;"` i `&#x45A` označavaju ćirilično malo slovo `њ`. Heksadekadni kodovi počinju sa malim ili velikim slovom `x`.

Komentari. U okviru SGML dokumenata moguće je navoditi i komentare, i to na sledeći način:

```
<!-- Ovo je jedan komentar -->
<!-- Ovo je komentar,
   koji ne staje u jedan red -->
```

Označene sekcije. *Označene sekcije (eng. marked sections)* se koriste da bi se označili delovi dokumenta koji zahtevaju posebnu vrstu procesiranja. Oblika su

```
<![ ključna reč [ ... označena sekcija ... ]]>
```

Najčešće korišćene ključne reči su `CDATA` koja označava doslovan sadržaj koji se ne parsira, `IGNORE` koja označava da se sekcija ignoriše tokom parsiranja, `INCLUDE` koja označava da se sekcija uključuje tokom parsiranja, `TEMP` koja označava da je sekcija privremeni deo dokumenta, itd.

Instrukcije procesiranja. *Instrukcije procesiranja (eng. processing instructions)* su lokalne instrukcije aplikaciji koja obrađuje dokument i napisane su na način specifičan za aplikaciju. Navode se između `<?` i `?>`. Npr.

```
<p>Sada je <?php echo date("h:i:s"); ?> </p>
```

Instrukcija `<?php echo date("h:i:s"); ?>` okviru HTML dokumenata govori PHP interpetatoru koji obrađuje dokument da je u pitanju deo PHP koda koji je onda potrebno interpretirati.

3.1.3 Definicije tipa dokumenta (DTD)

Svaki element i atribut u okviru neke SGML aplikacije se definiše u okviru definicije tipa dokumenta (DTD).

Deklaracije entiteta. Entiteti se deklarišu korišćenjem `<!ENTITY` za kojim ide ime entiteta, vrednost entiteta pod navodnicima i završni `>`.

Na primer:

```
<!ENTITY uvit "Uvod u Veb i Internet tehnologije">
```

U slučaju parametarskih entiteta, koristi se oznaka `%`.

Na primer:

```
<!ENTITY % fontstyle "TT | I | B | BIG | SMALL">
```

Ovako deklarisan entitet se može dalje koristiti u okviru DTD, uključujući i u okviru deklaracija drugih entiteta.

```
<!ENTITY % inline
  "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">
```

Deklaracije elemenata. Većina DTD se sastoji od deklaracija elemenata i njihovih atributa. Deklaracija elementa počinje sa `<!ELEMENT`, završava se sa `>`, a između se navodi:

1. Ime elementa.
2. Pravila minimalizacije koja određuju da li je neka od etiketa opciona[2]. Dve crtice `-`nakon imena označavaju da su obe etikete obavezne, crtica `-` za kojom sledi `O` označava da se završna etiketa može izostaviti, dok dva slova `O` označavaju da se obe etikete mogu izostaviti.
3. Sadržaj elementa. Dozvoljeni sadržaj elementa se naziva model sadržaja (eng. content model). Za definiciju modela sadržaja koriste se:

   `EMPTY` - označava elemente koji nemaju sadržaj, tj. prazne elemente.

   `ANY` - označava da element može imati proizvoljan sadržaj koji se sastoji od teksta i drugih elemenata.

[2]Pravila minimalizacije u nekim slučajevima mogu biti isključena i ne navode se u okviru DTD. Npr. jezik XML zabranjuje minimalizaciju elemenata i u okviru XML DTD pravila minimalizacije ne navode.

`CDATA` **(character data)** - označava tekst koji se neće analizirati pomoću SGML parsera. Sadržaj se tumači doslovno kako je napisan tj. reference na enititete se ne zamenjuju entitetima dok etikete koje se u njemu nalaze ne označavaju elemente.

`RCDATA` **(replacable character data)** - slično kao CDATA, osim što se reference zamenjuju (etikete i dalje ne označavaju elemente).

složene modele sadržaja - koriste se u slučaju kada element može da sadrži druge unježđene elemente. Model grupe su predstavljene izrazima u zagradama. Atomi u ovim izrazima mogu biti:

imena elemenata - označavaju unježđene elemente.

`#PCDATA` **(parsed character data)** - označava tekst koji će se analizirati pomoću parsera. Reference na enititete se u okviru ovog teksta zamenjuju entitetima i etikete koje se u njemu nalaze označavaju elemente.

Ovi atomi se dalje mogu kombinovati sledećim veznicima[3]:

`A?` - atom `A` se može, ali ne mora pojaviti.

`A+` - atom `A` se mora pojaviti jedan ili više puta

`A*` - atom `A` se mora pojaviti nula ili više puta

`A | B` - ili atom `A` ili atom `B` se mora pojaviti, ali ne oba.

`A, B` - oba atoma `A` i `B` se moraju pojaviti u tom redosledu.

`A & B` - oba atoma `A` i `B` se moraju pojaviti u bilo kom redosledu.

Moguće je dodatna pravila uključivanja i isključivanja sadržaja.

`+(S)` - sadržaj `S` se može pojaviti.

`-(S)` - sadržaj `S` se ne sme pojaviti.

Naravno, definicije elemenata mogu da sadrže reference parametarskih entiteta.

Razmotrimo nekoliko primera:

```
<!ELEMENT zbirka - -  (pesma+)>
```

Element `zbirka` u sebi sadrži jedan ili više elemenata `pesma`, pri čemu se obe etikete moraju navoditi.

```
<!ELEMENT pesma - - (naslov?, strofa+)>
```

Element pesma može, a ne mora, da sadrži element `naslov` za kojim sledi jedan ili više elemenata `strofa`. Obe etikete se opet moraju navesti.

```
<!ELEMENT stih O O (#PCDATA)>
```

Sadržaj elementa `stih` je proizvoljan tekst koji može da uključi i reference entiteta, ali ne sme da uključi druge elemente.

```
<!ELEMENT A - - (%inline;)* -(A)>
```

U ovom primeru korišćeno je dodatno pravilo isključivanje sadržaja. element `A` sadrži nula ili više elemenata obuhvaćenih parametarskim entitetom `%inline;`, ali ne sme da sadrži drugi element `A`.

[3]Naglasimo da su ovi veznici dosta bliski formalizmu *regularnih izraza (eng. regular expressions)*

Deklaracije atributa. Deklaracija atributa u okviru DTD počinje sa `<!ATTLIST`. Nakon ovoga, navodi se element za koji se deklariše atributa, nakon toga lista deklaracija pojedinačnih atributa i na kraju se navodi simbol `>`. Svaka deklaracija pojedinačnih atributa je trojka koja definiše:

1. Ime atributa
2. Tip vrednosti atributa, ili eksplicitno naveden skup dopustivih vrednosti. Najčešće korišćeni tipovi su:

   CDATA (character data) - kao i u slučaju elemenata, označava tekst koji se neće analizirati pomoću SGML parsera

   NAME - označava imena.

   ID - Označava jedinstvene identifikatore tj. imena koja moraju biti jedinstvena u celom dokumentu.

   NUMBER - Označava brojevne vrednosti.
3. Naznaku da li je vrednost atributa: implicitna (ključna reč `#IMPLIED`) - podrazumevanu vrednost određuje softver koji vrši obradu dokumenta; fiksirana (ključna reč `#FIXED`) - podrazumeva da atrubut može da ima samo jedinu moguću vrednost koja je u nastavku navedena; ili zahtevana (ključna reč `#REQUIRED`). Na ovom mestu moguće je i eksplicitno specifikovati podrazumevanu vrednost atributa.

Naravno, definicije atributa mogu da sadrže reference parametarskih entiteta.

Navedimo nekoliko primera:

```
<!ATTLIST pesma
   autor CDATA #REQUIRED
>
```

Ovim je za element `pesma` deklarisan atribut `autor` čija je vrednost neki tekst, pri čemu je navođenje atributa obavezno.

```
<!ATTLIST td
   rowspan NUMBER 1
   colspan NUMBER 1
>
```

Ovim se za element `td` uvode atributi `rowspan` i `colspan` čije su vrednosti brojevi, dok je podrazumevana vrednost za oba atributa 1.

```
<!ATTLIST html
   version CDATA #FIXED "%HTML.Version"
>
```

Ovim se označava da vrednost atributa `version` elementa `html` može da bude isključivo vrednost određena parametarskim entitetom `HTML.Version` (koji definiše tekuću verziju).

Uključivanje DTD. DTD može biti naveden ili kroz unutrašnju ili kroz spoljašnju deklaraciju.

Unutrašnja deklaracija podrazumeva da se DTD deklaracije nalaze u zaglavlju datoteke u kojoj je smešten dokument. Na primer:

```
<!-- test.sgml -->
<!DOCTYPE test
  <!ELEMENT test - - PCDATA>
>
<test>Zdravo</test>
```

Spoljašnja deklaracija podrazumeva da se DTD deklaracije nalaze u spoljašnjoj datoteci, bilo na lokalnom sistemu ili javno na Vebu. U tom slučaju se u okviru `<!DOCTYPE>` navodi ime datoteke koja sadrži DTD. Na primer:

```
<!-- test.sgml -->
<!DOCTYPE test SYSTEM "test.dtd">
<test>Zdravo</test>
```

```
<!-- test.dtd -->
<!ELEMENT test - - PCDATA>
```

3.2 XML

SGML je zamišljen kao izrazito opšti jezik koji omućava kodiranje veoma raznorodnih dokumenata. Formalizam DTD omogućava korisnicima da na jezgrovit način iskažu širok spektar sintaksnih pravila za određeni tip dokumenata. Kako bi ovo bilo moguće postići, SGML je dizajniran kao veoma kompleksan jezik sa mnoštvom različitih sintaksnih konstrukcija. Kako bi se autorima dokumenata pomoglo, dopušten je veliki broj višeznačnosti i proizvoljnosti (npr. mogućnost izostavljanja etiketa u okviru elemenata). Sa druge strane, sve ovo otežava rad sa SGML dokumentima i čini izradu alata koji obrađuju SGML dokumente veoma komplikovanim.

eXstensible Markup Language (skr. XML) je meta jezik za obeležavanje koji je nastao sredinom 1990-tih kao rezultat potrebe za postojanjem jezika za obeležavanje veoma sličnog jeziku SGML, a koji bi bio jednostavniji za parsiranje (rasčlanjavanje) i obradu. Ovaj jezik je iz SGML-a izbacio proizvoljnosti tako da se pisanjem dokumenata u ovom jeziku moraju poštovati mnogo striktnija pravila. Svaki ispravan XML dokument je ujedno i ispravan SGML dokument[4]. Osnovni ciljevi koji su vodili dizajn jezika XML su:

- XML će biti korišćen na Internetu.
- XML će podržavati veliki broj aplikacija.
- XML će biti kompatibilan sa jezikom SGML.
- Biće jednostavno pisati programe koji procesiraju XML dokumente.
- XML neće sadržati opcione i proizvoljne delove.
- XML dokumenti moraju biti čitljivi i jasni za ljude.
- XML će biti brzo razvijen.
- Dizajn jezika XML će biti formalan i koncizan.

[4]Ovde se ne uzima u obzir validnost u odnosu na neki DTD već samo dobra formiranost dokumenta.

- XML dokumente će biti jednostavno kreirati.
- Jezgrovitost u XML deklaracijama nije naročito značajna.

Vremenom se pojavio se jako veliki broj XML aplikacija. Navedimo samo neke:

XHTML - zapis hipertekstualnih dokumenata,

MathML - zapis matematičkog sadržaja,

SVG - zapis vektorskih crteža,

SMIL - zapis multimedijalnog sadržaja,

WML - zapis sadržaja pogodnog za pregledanje na mobilnim uređajima,

SOAP - korišćenje veb servisa

Otvoreni formati dokumenata (koji su ISO standardizovani, a vezuju se za kancelarijski paket OpenOffice.org), kao i najnoviji Microsoft Office formati su takođe zasnovani na jeziku XML. S obzirom na oštriju sintaksu XML dokumenata, XML je vremenom postao jezik za zapisivanje raznovrsnih struktuiranih i polustruktuiranih informacija i došlo je do razvoja specijalnih baza podataka koje su zasnovane na skladištenju informacija u XML dokumentima i korišćenju specijalizovanih jezika (npr. XPath, XQuery) za pretraživanje.

3.2.1 Ispravnost dokumenata - dobro formirani i validni dokumenti.

Navedimo neka opšta sintaksna pravila koje uvodi XML, naglašavajući pri tom razlike između jezika SGML i XML:

- Svi elementi u jeziku XML moraju sadržati i početnu i završnu etiketu, što nije slučaj u jeziku SGML.
- U jeziku XML, ukoliko elementi nemaju sadržaj, umesto početne i završne etikete moguće je koristiti specijalnu vrstu etiketa kojima se obeležavaju prazni elementi. Na primer, umesto `<br></br>`, moguće je pisati `<br />`. U jeziku SGML prazni elementi često ne smeju imati završnu etiketu.
- XML razlikuje velika i mala slova i imena XML elemenata se obično pišu se malim slovom, dok SGML nije osetljiv na veličinu slova.
- Isto kao u jeziku SGML, i u jeziku XML etikete moraju biti zatvarane u obratnom redosledu od otvaranja. Svi elementi moraju biti dobro ugnježđeni i nije dozvoljeno njihovo preplitanje. Na primer, nije dopušteno pisati:

```
<b><i>Tekst</b></i>
```

- Svaki XML dokument mora imati sadržaj napisan u okviru jednog elementa elementa koji se nalazi na najvišem nivou i koji se naziva *koreni element (eng. root element)*. Na primer, u okviru svakog HTML dokumenta, ceo sadržaj se nalazi u okviru elementa `html`:

```
<html>
  Sadzaj....
</html>
```

SGML dokumente nije neophodno pisati u okviru jednog elementa koji se nalazi na najvišem nivou.

- U jeziku XML, nije dozvoljeno izostavljanje vrednosti atributa, kao ni izostavljanje navodnika prilikom navođenja vrednosti atributa, što je dozvoljeno u jeziku SGML.
- U okviru XML dokumenta (isto kao i u okviru SGML dokumenta) karakteri `<` i `&` imaju specijalno značenje i nije ih dozvoljeno koristiti osim za označavanje etiketa i referenci entiteta. Umesto njih potrebno je koristiti reference entiteta `<` i `&`. Karakteri `>`, `"` i `'` nisu zabranjeni, ali je umesto njih preporučljivo koristiti `>`, `&quot;` i `&apos;`. Ovo su jedini predefinisani eniteti jezika XML.
- Na početku XML dokumenta poželjno je navesti *XML deklaraciju* — instrukciju procesiranja oblika

  ```
  <?xml version="1.0" encoding="utf-8" ?>
  ```

  koja govori o verziji XML-a i korišćenom kodiranju karaktera. Ukoliko je kodiranje različito od `utf-8` ili `utf-16`, neophodno je navesti xml deklaraciju koja specifikuje kodiranje.

S obzirom na kompleksnost i striktnost opšte sintakse koju svaki XML dokument mora da zadovolji, dokumenti koji poštuju ove sintaksne zahteve smatraju se ispravnim (u slabom smislu te reči) i za njih se kaže da su *dobro formirani (eng. well formed)*.

Na primer, dokument narednog sadržaja se može smatrati ispravnim HTML 4.01 dokumentom jer poštuje sva pravila sintakse jezika SGML (u kome je jezik HTML 4.01 definisan).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <title>Moj prvi HTML dokument</title>
  </HEAD>
  <BODY>
    <p>Zdravo svete! <img src=zdravo.gif alt=zdravo>
    <p>Copyright (&copy;) Milena
  </body>
</html>
```

Ipak, dokument prethodnog sadržaja se ne može smatrati dobro formiranim XML dokumentom jer narušava mnoga opšta pravila sintakse jeizka XML. Na primer, nedostaju završne etikete elemenata `<p>` i `<img>` i mešana su velika i mala slova.

Naredni sadržaj predstavlja dobro formirani dokument koji odgovara prethodnom, a koji je opisan u jeziku XHTML (definisanom u okviru jezika XML):

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Moj prvi HTML dokument</title>
  </head>
  <body>
    <p>Zdravo svete! <img src="zdravo.gif" alt="zdravo" /> </p>
    <p>Copyright (&copy;) Milena</p>
  </body>
</html>
```

Pored ovih opštih pravila, isto kao i u jeziku SGML, svakom pojedinačnom tipu dokumenta je pridružena njegova specifična sintaksa kojom su propisani elementi, atributi i entiteti od kojih se dokument sastoji. Jezik XML zadržava koncept *definicije tipa dokumenta (DTD)* za specifikovanje tipova dokumenta, ali uvodi i alternativni način da se ovo uradi korišćenjem *XML sheme (eng. XML schema)*. Dokumenti koji su dobro formirani i koji dodatno poštuju sva sintaksna pravila svog tipa dokumenta smatraju se ispravnim (u jakom smislu te reči) i za njih se kaže da su *validni (eng. valid)*. Tako, na primer, iako je naredni dokument dobro formiran, on nije validan (jer element `html` jezika XHTML mora da sadrži zaglavlje koje ovde nedostaje, i takođe element `body` sadrži atribut `color` koji nije na ovom mestu dopušten).

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <body color="black">
    <p>Zdravo svete! <img src="zdravo.gif" alt="zdravo" /> </p>
    <p>Copyright (&copy;) Milena</p>
  </body>
</html>
```

XML DTD i XML Schema Iako su na prvi pogled XML DTD i SGML DTD izgledaju veoma slično, XML značajno smanjuje izražajnost prilikom pisanja DTD ukidajući neke konstrukcije i uvodeći neke restrikcije. Za detaljni opis pravila za pisanje XML DTD, čitalac se upućuje na XML standard, a u nastavku će biti samo veoma grubo skicirane razlike u odnosu na SGML:

- Veličina slova nije bitna u okviru SGML DTD, dok u XML DTD postaje bitna.
- Nije dozvoljena minimalizacija elemenata tako da DTD ne može da sadrži pravila minimalizacije.
- Mnogi skraćeni zapisi u okviru zapisa DTD nisu više podržani. Na primer, nije moguće koristiti komentare u okviru deklaracija, nije moguće koristiti istovremenu deklaraciju više elemenata i slično.
- Mnogi tipovi atributa su ukinuti.
- U slučaju deklaracije mešanog modela sadržaja dokumenta uvedena su striktnija pravila. Na primer, `#PCDATA` mora da se javi na prvom mestu u

deklaraciji i ne može biti vezano veznikom `,` koji određuje redosled teksta i elemenata.

- Dodatno uključivanje i isključivanje sadržaja (`+(S)` i `-(S)`) nije dozvoljeno.
- Zabranjeno je korišćenje veznika `&`.
- `CDATA` i `RCDATA` modeli sadržaja nisu više dozvoljeni. Umesto njih se mora koristiti `(#PCDATA)`. Ovim su tvorci dokumenata primorani da sve delove dokumenta koji ne bi trebalo da se obrađuje XML parserom označe sa označenom `CDATA` sekcijom u okviru samog dokumenta.
- ...

XML Schema (XSD) predstavlja XML-zasnovanu alternativu za DTD tj. XML Schema dokumenti su dokumenti koji su zapisani u XML formatu. Ilustracije radi navodimo samo jedan primer XML Schema dokumenta, dok se čitaoci za detaljnije informacije upućuju na literaturu.

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ime" type="xs:string"/>
      <xs:element name="prezime" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

U ovom primeru definisan je element `student`. Element sadrži, redom, elemente `ime` i `prezime` koji imaju tekstuelni sadržaj.

3.3 HTML/XHTML

HyperText Markup Language je jezik za obeležavanje *hipertekstualnih dokumenata* i predstavlja jednu od osnova Veba.

3.3.1 Istorijat

Jezik HTML je nastao 1980-tih godina u istraživačkom centru CERN. Razvio ga je fizičar Tim Berners-Li (eng. Tim Berners-Lee) u okviru sistema koji omogućava istraživačima jednostavno korišćenje i razmenu dokumenata. Ključne ideje koje Berners-Li koristi su da se dokumenti razmenjuju korišćenjem Interneta i da se dokumenti predstavljaju kao *hipertekst*, tj. tekst koji sadrži veze (uputnice, linkove) ka drugim dokumentima i koji, ukoliko se čita uz podršku specijalizovanog softvera, omogućava izuzetno jednostavnu navigaciju kroz veliki broj dokumenata i time značajno povećava dostupnost informacija. S obzirom na to da se u CERN-u za obeležavanje dokumenata već koristio SGML, novi jezik je kreiran kao SGML aplikacija (kreiran je poseban DTD, pri čemu je većina naziva elemenata preuzeto iz već korišćenih SGML DTD-ova u okviru CERN-a i ubačeno je nekoliko novih elemenata, pre svega onaj za obeležavanje veza). Godine 1989. Berners-Li sa grupom kolega predlaže postojeću infrastrukturu Internet-a kao platformu za razmenu ovih dokumenata i 1990. kreira

prototip klijentskog i serverskog sofvera i definiše prvu verziju HTTP protokola za njihovu komunikaciju. Iako je ovaj predlog odbijen u okviru CERN-a, ovo se smatra početkom Veba. Rad na HTML-u se nastavlja i 1993. u okviru IETF se objavljuje prva formalna HTML specifikacija. U to vreme se u NCSA (National Center for Supercomputing Applications u SAD) pojavljuje pregledač *Mosaic* koji mimo HTML standarda uvodi mogućnost direktnog predstavljanja i prikazivanja slika u okviru dokumenata što mu odmah donosi popularnost i veliki broj korisnika. Ubrzo se pojavljuje mogućnost i predstavljanja tabela i formi za popunjavanje i sva ova proširenja bivaju obuhvaćena standardima koji se pojavljuju. Popularnost veba i HTML-a raste i tokom 1990-tih velike softverske kompanije proizvode svoje pregledače (Microsoft Internet Explorer i Netscape Navigator) i započinju borbu za primat na Veb tržištu (ovaj period poznat je i kao period ratova pregledača — eng. browser wars). Napredak tehnologije prikaza (pre svega pojava grafičkih kontrolera i monitora u boji) i sve veća rasprostranjenost grafičkih korisničkih okruženja dovela je do toga da autori teže kreiranju dokumenata sa bogatim elementima grafičke prezentacije (npr. mnoštvom boja, slovnih likova, animiranim sadržajem) i pod tim pritiskom proizvođači pregledača proširuju ad hoc smišljenim elementima koji HTML proširuju elementima koji služe isključivo za definisanje vizuelne prezentacije i koji značajno opterećuju označavanje logičke strukture dokumenata [5]. Počinje velika tržišna utrka koja dovodi naglog i nekontrolisanog razvoja HTML tehnologija, van uticaja zvaničnih standardizacionih institucija (pre svega IETF-a). Kako bi se kanalisao dalji razvoj veba i koordinisali industrijski proizvođači softvera, 1994. Tim Berners-Li formira neprofitnu organizaciju *World Wide Web Consortium (W3C)* koja okuplja nekoliko stotina, pre svega akademskih, stručnjaka i koja preuzima kontrolu nad veb tehnologijama. Danas se W3C smatra jedinim relevantnim telom za razvoj veba i njihove preporuke se smatraju *de facto* standardima.

3.3.2 Verzije jezika

Striktno pridržavanje standarda je osnovni preduslov kreiranja kvalitetnih dokumenata. Autori bi sve vreme trebalo da imaju u vidu da će njihovi dokumenti biti tumačeni korišćenjem različitih alata u različitim okruženjima i na različitim uređajima (npr. verovatno je da će biti korišćeni različiti pregledači, moguće je da će dokumenti biti pregledani na ekranima sa različitom veličinom i rezolucijom, na mobilnim telefonima, čak je moguće da će biti prikazivani Brajevom azbukom ili zvučno za osobe sa oštećenim vidom). Dakle, kontrolu kvaliteta dokumenta ne bi trebalo vršiti samo proverom u omiljenom pregledaču, već je za svaki dokument poželjno proveriti njegovu saglasnost sa standardom putem validacije. Na žalost, nekontrolisani razvoj HTML-a tokom 1990-tih, doveo je do toga da je čak i danas neophodno voditi računa o tome kako se dokumenti prikazuju na starim korisničkim agentima koji ne poštuju standarde. Takođe, ukoliko bi se dokumenti koji nisu u skladu sa standardima, a koji su prilagođeni za prikazivanje iz korisničkih agenata koji ne poštuju standarde, prikazali na novijim korisničkim koji poštuju standarde, njihov izgled bi verovatno bio drugačiji od onoga što su autori imali na umu kada su ih kreirali[6]. Pošto je veliki broj

[5] Najpoznatiji primeri su elementi `font`, `marquee` i `blink`.

[6] Do najvećih problema dolazi zbog problema sa tumačenjem jezika CSS.

ovakvih starih dokumenata i dalje prisutan na Vebu, proizvođači su primorani da u korisničke agente ugrađuju dva moda: noviji standardni mod i mod namjen za prikaz nestandardnih dokumenata (eng. quirks mode).

Standardi. U vreme pisanja ovog teksta, dominantna su dva HTML standarda. Standard *HTML 4.01* iz predstavlja definiciju tekuće verzije jezika HTML u SGML okviru. Standard je usvojen 1999. kao W3C preporuka, a 2000. godine biva usvojen i kao ISO standard.

S obzirom na popularnost i prednosti jezika XML nad jezikom SGML (o čemu je ranije već bilo reči) 1999. godine se pojavljuje i jezik *XHTML 1.0* koji predstavlja reformulaciju jezika HTML 4.01 kao XML aplikacije. Osim opštih sintaksnih razlika koji potiču iz odnosa jezika SGML i XML, nema značajnih razlika između HTML 4.01 i XHTML 1.0 jezika. Prednosti jezika XML u odnosu SGML su već diskutovane. Veliki broj novih jezika za obeležavanje specifičnih vrsta sadržaja se definišu kao XML aplikacije (npr. MathML, SVG, SMIL, itd.). Da bi sadržaji opisani na ovim jezicima mogli biti uključeni u HTML dokumente, potrebno je da i HTML dokumenta budu opisana u okviru XML-a. Takođe, zbog striktnije sintakse XML-a, programi za unošenje teksta kao i programi za obradu se mnogo bolje snalaze sa XML dokumentima nego sa SGML dokumentima. Imajući sve ovo u vidu, izvesno je da će i sve naredne verzije jezika HTML biti formulisane kao naslednice XHTML-a tj. u XML okviru. Stoga će u nastavku ovog teksta biti razmatran isključivo jezik XHTML i razumno je očekivati da će čitaoci biti u stanju da samostalno prouče i HTML 4.01, ukoliko se za tim javi potreba. Ipak, trebalo bi naglasiti da nisu svi korisnički agenti, naročito oni stariji, u stanju da obrađuju XML.

Tipovi dokumenata. Već je pomenuto kako je sredinom 1990-tih godina pod pritiskom tržišta i industrije sofvera jezik HTML (čak i njegove standardizovane verzije) proširen nizom elemenata koji isključivo služe za definisanje grafičke prezentacije dokumenta. Međutim, od verzije HTML 4.0, odlučeno je da se pokuša sa ispravljanjem ovako loših rešenja i uvedeno je potpuno razdvajanje opisa logičke strukture i vizuelne prezentacije dokumenata. Za opis prezentacije dokumenata koriste se *stilski listovi (eng. style-sheets)*, tj. uveden je zaseban jezik CSS (koji će biti kasnije opisan). Odlučeno je da se elementi (i atributi) koji se odnose na opis prezentacije uklone iz jezika HTML. Ipak, s obzirom na veliki broj postojećih dokumenata koji koriste ove elemente odlučeno je da se ovo izbacivanje radi postepeno. Za početak, ovi elementi su proglašeni zastarelim. Uvedena su dva različita tipa dokumenta (DTD): *striktni (eng. strict)* i *prelazni (eng. transitional)*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Striktni dokumenti poštuju znatno oštrija pravila i ne sadrže sporne elemente. Prelazni tip dokumenata je formulisan tako da su pravila znatno blaža i dokumenti i dalje mogu da sadže sve sporne elemente. Na primer, telo striktnih dokumenata ne može da sadrži tekst ili sliku koja nije deo nekog pasusa, tabele ili slično, dok je kod prelaznih dokumenata ovako nešto dopušteno. Svakako, W3C preporučuje korišćenje striktnog DTD pri kreiranju svih novih dokumenata.

Pored ova dva tipa, standard uvodi i specijalni tip dokumenta (DTD) za kreiranje opisa zona stranice korišćenjem *okvira (eng. frame)*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Pošto danas postoje znatno bolji načini za podelu stranica na zone, ovaj DTD se sve ređe i ređe koristi i njegova upotreba se takođe ne preporučuje.

MIME tip HTML dokumenata. Kao što je već opisano u poglavlju o HTTP protokolu, prilikom slanja sadržaja preko veba, HTTP odgovori sadrže `Content-Type` polje koje sadrži informaciju o tipu sadržaja u obliku MIME formata. HTML 4.01 dokumenti, kao i dokumenti ranijih verzija HTML-a se predstavljaju MIME tipom `text/html`. Kod XHTML dokumenata, situacija je komplikovanija. Ovi dokumenti se mogu predstaviti kao dokumenti MIME tipa `text/html`, `application/xhtml+xml`, `application/xml` i `text/xml`. Zadnja dva tipa su pogodnija za predstavljanje „čistih" XML dokumenata i ne preporučuju se prilikom predstavljanja HTML sadržaja. Ukoliko je korisnički agent takav da može da obrađuje XML, savet je da se dokument predstavi kao `application/xhtml+xml`. Međutim, veliki broj korisničkih agenata (pre svega oni stariji) ne može da obrađuje XML i ne ume da iskoristi prednosti koje XML donosi. U tim slučajevima, savet je da se koristi MIME tip `text/html` i da se XHTML dokumenti „lažno" predstavljaju kako da su HTML. Međutim, kako bi ovo uspelo, potrebno je pridržavati određenih saveta poznatih kao *saveti za HTML kompatibilnost (eng. HTML compatibility guidelines)*. Ova pravila su takva da ne narušavaju ispravnost XML dokumenata, a doprinose mogućnosti njihovog ispravnog tumačenja od strane starijih korisničkih agenata koji razumeju samo HTML. Saveti su dati u dodatku XHTML 1.0 standarda i ovde će biti navedeno samo nekoliko primera:

- Ne navoditi XML deklaraciju i koristiti UTF-8 ili UTF-16 kodiranje karaktera.
- Za elemente sa praznim sadržajem navoditi skraćeni zapis etiketa (npr. `<br />`)).
- Za elemente sa mogućim nepraznim sadržajem ne koristiti skraćeni zapis etiketa ako se jave prazni (npr. `<p></p>`).
- ...

Jezik	MIME tip	Tretira se kao:
HTML	`text/html`	HTML
XHTML	`application/xhtml+html`	XML
	`application/xml`	
	`text/xml`	

Tabela 3.1: MIME tipovi HTML i XHTML dokumenata

3.3.3 Skup karaktera i kodiranje karaktera

Dokumenti se sastoje od *karaktera*, kao što su npr. latinično `A`, ćirilično `И`, arapsko slovo س ili kinesko 水. Jezik HTML definiše *skup karaktera (eng. character set)* koji je dozvoljeno koristiti u HTML dokumentima. Skup karaktera se odlikuje apstraktnim karakterima koji se mogu koristiti u dokumentima, pri čemu je svaki karakter karakterisan jedinstvenim imenom i jedinstvenim celobrojnim kodom (tj. pozicijom). Za skup karaktera HTML dokumenata moguće je koristiti *Univerzalni skup karaktera (eng. Universal character set — UCS)* definisan standardom *ISO 10646*. Ovaj standard obuhvata veliki broj karaktera koji se koriste u svim svetskim jezicima. Najznačajnija je tzv. osnovna višejezička ravan (eng. basic multilingual plane — BMP). Karakteri koje obuhvata skup *UNICODE* se svi nalaze i u UCS i to na istim pozicijama kao u UNICODE-u.

Ipak, izuzetno je važno imati na umu da skup karaktera ne određuje jedinstveno kako se karakteri kodiraju bajtovima prilikom smeštanja u datoteke. Ovo je određeno kroz specifično *kodiranje karaktera (eng. character encoding)*[7]. U nastavku će biti opisano nekoliko najčešće korišćenih kodiranja karaktera. Imena koja će biti navedena su standardna (pod nadležnošću IANA) i trebalo bi ih koristiti prilikom navođenja kodiranja karaktera. Veličina slova nije bitna prilikom navođenja ovih oznaka.

`ASCII` - *American Standard Code for Information Interchange* - jedan od starijih standarda za kodiranje koji definiše samo 127 karaktera (uglavnom slova engleske abecede, cifre, interpunksijske znake i neke specijalne karaktere). Iako je standard sedmobitan, obično se za kodiranje svakog karaktera koristi ceo jedan bajt.

Naredne kodne strane predstavljaju proširenja ASCII tabele i definišu po 255 karaktera. Prvih 127 karaktera se poklapaju sa ASCII tablicom, dok su naredni karakteri specifični, obično za neko geografsko područje. Prilikom kodiranja, naravno, svaki karakter zauzima 1 bajt. Dve familije kodnih strana dominiraju - ISO-8859 familija standardizovana od strane međunarodne organizacije za standardizaciju i windows-125x protežirana od strane kompanije Majkrosoft. Naredne kodne strane su izdvojene zbog toga što se najčešće koriste za zapis tekstova na srpskom jeziku.

`ISO_8859-1` - Latin-1. Pored ASCII simbola, definiše latinične simbole korišćene u zapadnoevropskim jezicima.

[7] Naglasimo, da ono što se u ovom tekstu (i u HTML standardima) naziva imenom *character encoding*, u nekim starijim dokumentima se javlja pod imenom *charset*.

`ISO_8859-2` - Latin-2. Pored ASCII simbola, definiše latinične simbole korišćene u centralnoevropskim jezicima (između ostalog i dijakritike š, ć, đ, č i ž koji se koriste u srpskom).

`ISO_8859-5` - Latin-5. Pored ASCII simbola, definiše ćirilične simbole.

`windows-1250` - Majkrosoftova kodna strana koja pored ASCII simbola definiše latinične simbole korišćene u centralnoevropskim jezicima (između ostalog i dijakritike š, ć, đ, č i ž koji se koriste u srpskom).

`windows-1251` - Majkrosoftova kodna strana koja pored ASCII simbola definiše i ćirilične simbole.

Naredne kodne sheme su novije i omogućavaju kodiranje šireg skupa karaktera. Za razliku od prethodnih „jednobajtnih" kodnih strana, ovde je moguće u okviru istog dokumenta mešati različite karaktere (npr. dijakritike, ćirilicu, grčko i hebrejsko pismo).

`ISO-10646-UCS-2` - Kodiranje koje omogućava prikaz osnovne višejezičke ravni tako što se svaki od ovih karaktera zapisuje sa po dva bajta.

`UTF-8` - Pametan algoritam kodiranja koji karaktere kodira tako da ASCII karaktere zapisuje sa po jednim bajtom (kompatibilno sa ASCII kodiranjem), naredne karaktere (između ostalih i dijakritike korišćene u latiničnom zapisu srpskog jezika i ćirilične simbole) sa po dva bajta, naredne sa po tri bajta itd. S obzirom da se u engleskom jeziku i u ostalim evropskim jezicima koji koriste latinicu uglavnom koriste ASCII karakteri

`UTF-16` - Pametan algoritam koji omogućava kodiranje punog ISO-10646 skupa, pri čemu se za karaktere osnovne višejezičke ravni koriste dva bajta, a za ostale karaktere više bajtova.

Za navođenje karaktera u okviru dokumenata moguće je koristiti i numeričke karakterske reference. Npr. reč lišće se može kodirati kao `li&#x161;&#x107;e`. S obzirom da ovo čini dokumente prilično nečitljivim, ovaj način kodiranja karaktera se ne preporučuje. Naglasimo još jednom, da izuzetak predstavljaju karakteri `<`, `>` i `&` koji imaju specijalno značenje u okviru HTML dokumenata, tako da se za njihovo kodiranje zahteva korišćenje karakterskih referenci `<`, `>` i `&`, respektivno.

Navođenje kodne strane za HTML dokumente. Kako bi sadržaj mogao korektno da se protumači, neophodno je da korisnički agenti imaju informaciju koje je kodiranje karaktera korišćeno. Ova informacija često biva dostavljena od strane servera na kome se dokument nalazi.

Standardan način da se ovo uradi je da se informacija o kodiranju dostavi u okviru HTTP odgovora kojim se HTML dokument dostavlja. Standardna oznaka korišćenog kodiranja se u tom slučaju navodi u okviru `Content-Type` polja. Na primer:

```
Content-Type: text/html; charset=UTF-8
```

govori da je sadržaj koji sledi kodiran korišćenjem `UTF-8` kodiranja.

S obzirom da neki serveri ne uključuju ovu informaciju u HTTP odgovore, kodiranje karaktera je moguće izvršiti i u okviru samog HTML dokumenta. Ovo se radi u okviru `http-equiv` atributa elementa `meta` koji se navodi u zaglavlju dokumenta. Npr.

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Detaljniji opis `meta` elementa biće dat u narednim poglavljima. Ukoliko se koristi XHTML, moguće je informaciju o kodiranju navesti u `xml` deklaraciji:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Još jedan od načina je da se naglasi kodiranje je da se u okviru elementa koji upućuje na neki dokument (npr. u okviru veze) naglasi koje je kodiranje korišćeno za dokument na koji element (veza) upućuje. Npr.

```
<a href="page.html" charset="utf-8">...</a>
```

Ukoliko korisnički agent ne dobije informaciju o kodiranju dokumenta koji treba da obradi, moguće je ili da sam pokuša da pogodi kodiranje ili da ponudi korisniku da odabere kodiranje (npr. „View — Character Encoding” u programu Firefox).

Ukoliko je kodiranje takvo da ne može da obuhvati neke karaktere iz univerzalnog skupa karaktera, oni se navode koristeći reference karakterskih entiteta ili numeričke karakterske reference.

3.3.4 Osnovna struktura HTML dokumenata

Započnimo pregled XHTML-a minimalnim primerom ispravnog dokumenta.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="sr" lang="sr">
  <head>
    <title>Prvi primer</title>
  </head>
  <body>
    <p>Dobrodošli na
       <a href="http://www.matf.bg.ac.rs/~filip/uvit/">UVIT</a>.
    </p>
  </body>
</html>
```

Prvi red sadrži XML deklaraciju. S obzirom da ona nije obavezna, a u skladu sa savetima za kompatibilnost sa starijim verzijama HTML-a, preporučuje se njeno izostavljanje. Nakon ovoga, naveden je DTD i nakon toga element koreni `html`. Svaki HTML dokument se sastoji iz dva dela: *zaglavlja (eng. header) i tela (eng. body)*. Tako element `html` mora da sadrži obavezno elemente `head` i `body`.

```
<!ELEMENT html (head, body)>
<!ATTLIST html
  %i18n;
  id          ID             #IMPLIED
  xmlns       %URI;          #FIXED 'http://www.w3.org/1999/xhtml'
  >
```

Od atributa, element `html` može samo da sadrži samo one koje se odnose na internacionalizaciju (`lang`, `xml:lang` i `dir`), jedinstveni identifikator `id` i oznaku imenskog prostora `xmlns`.

Generički atributi. Određeni broj atributa može da bude pridružen jako širokom spektru elemenata.

```
<!ENTITY % attrs "%coreattrs; %i18n; %events;">
```

Parametarski entitet `%attrs` opisuje tri grupe atributa (`%coreattrs`, `%i18n` i `%events`). One će u nastavku biti opisane.

```
<!ENTITY % coreattrs
 "id          ID           #IMPLIED
  class       CDATA        #IMPLIED
  style       %StyleSheet; #IMPLIED
  title       %Text;       #IMPLIED"
  >
```

`id` - dodeljuje ime elementu. Za ovo ime se zahteva da bude jedinstveno na nivou celokupnog dokumenta. U narednom primeru, dva pasusa imaju jedinstvene identifikatore:

```
<p id="moj-pasus">Ovo je moj pasus.</p>
<p id="tvoj-pasus">Ovo je tvoj pasus.</p>
```

Identifikatori elemenata se obično koriste na sledeće načine.

- Kao selektori prilikom korišćenja CSS-a. Na ovaj način je moguće dodeliti poseban grafički izgled ovom specifičnom elementu. Tako je npr. moguće postaviti crvenu boju mom pasusu:

  ```
  p#moj-pasus {color : red}
  ```

- Kao dolazna sidra u okviru veza. Na ovaj način je moguće napraviti vezu iz nekog spoljašnjeg dokumenta ka ovom specifičnom elementu. Aktiviranjem veze, obično korisnički agent automatski prikazuje deo strane koji sadrži identifikovani element.

  ```
  <a href="strana.html#tvoj-pasus">Tvoj pasus</a>
  ```

- Kao način da se iz nekog skripta pristupi ovom specifičnom elementu. Na primer, naredni JavaScript kod sakriva `moj-pasus`.

  ```
  document.getElementById("moj-pasus").style="display : none";
  ```

- Za različite specifične vrste procesiranja dokumenata specijalizovanim softverom.

`class` - dodeljuje jednu ili više klasa dokumentu. U slučaju da se kroz ovaj atribut navodi više klasa, one se razdvajaju razmacima. Više različitih elemenata u okviru istog dokumenta mogu da pripadaju istoj klasi. Klase su korisnički definisane. U narednom primeru, pasusu su dodeljene klase `greska` i `engleski`.

```
<p class="greska engleski">This is an error</p>
```

Klase se obično koriste na sledeće načine.

- Kao selektori prilikom korišćenja CSS-a. Na ovaj način je moguće dodeliti poseban grafički izgled grupi elemenata grupisanih u istu klasu. Tako je npr. moguće postaviti crvenu boju svim elementima u dokumentu koje pripadaju klasi `greska`, a plavu boju pozadine svim pasusima u klasi `engleski`:

```
 .greska   {color: red}
p.engleski {background : blue}
```

- Za različite specifične vrste procesiranja dokumenata specijalizovanim softverom.

`style` - dodeljuje informacije o grafičkom izgledu (tj. stilu) elementa. Ove informacije se navode u podrazumevanom jeziku stilskih listova, a to je obično CSS. Na primer, u sledećem pasusu se postavlja boja teksta na crvenu:

```
<p style="color: red">Neki tekst</p>
```

S obzirom na probleme koji nastaju prilikom održavanja dokumenata u kojima se informacije o grafičkom izgledu nalaze isprepletane sa samim sadržajem dokumenta, često korišćenje `style` atributa se ne preporučuje. Umesto toga, poželjno je elemente obeležiti identifikatorima ili klasama, a informacije o grafičkom izgledu postavljati u zasebnim sekcijama ili zasebnim spoljašnjim dokumentima.

`title` - za razliku od elementa `title` kojim se zadaje naslov celog dokumenta, atribut `title` daje dodatne informacije o elementu u okviru koga je postavljen. Npr. narednoj vezi je dat naslov:

```
<a href="uvit.html"
   title="Stranica predmeta Uvod u Veb i Internet tehnologije">uvit</a>
```

Korisnički agenti obično ove informacije ili ignorišu ili ih prikazuju u kao balončiće (eng. tool tip) kada se mišem dođe do elementa sa naslovom.

```
<!ENTITY % i18n
 "lang        %LanguageCode; #IMPLIED
  xml:lang    %LanguageCode; #IMPLIED
  dir         (ltr|rtl)      #IMPLIED"
  >
```

Ovi atributi se odnose na internacionalizaciju (eng. internationalization)[8].

[8]Skraćenica `i18n` potiče od toga što između slova `i` i slova `n` u reči *internationalization* postoji 18 slova.

`lang` - označava jezik na kome je napisan sadržaj elementa. Informacija o jeziku može biti korisna:

- pretraživačkim mašinama
- softveru koji čita dokumente osobama sa oštećenim vidom
- korisničkim agentima kako bi odabrali način zapisa navodnika, kvalitetne glifove (npr. ligature), kako bi mogli da vrše automatski prelom teksta, itd.
- prilikom provere ispravnosti pravopisa i gramatike (eng. spell and grammar checking).

Atribut `lang` se obično koristi na nivou celog dokumenta (u okviru elementa `html`) i u okviru elemenata koji su napisani na različitom jeziku od dominantnog jezika dokumenta.

`dir` - označava smer ispisa teksta (`ltr` – s leva na desno ili `rtl` – s desna na levo). Koristi se uglavnom kod jezika koji se pišu s desna na levo (npr. hebrejski, arapski, ...).

```
<!ENTITY % events
  onclick     %Script; #IMPLIED
  ondblclick  %Script; #IMPLIED
  onmousedown %Script; #IMPLIED
  onmouseup   %Script; #IMPLIED
  onmouseover %Script; #IMPLIED
  onmousemove %Script; #IMPLIED
  onmouseout  %Script; #IMPLIED
  onkeypress  %Script; #IMPLIED
  onkeydown   %Script; #IMPLIED
  onkeyup     %Script; #IMPLIED
  >
```

Ova grupa atributa služi da poveže događaje sa skriptovima koji se izvršavaju prilikom tih događaja. U narednom primeru, ukoliko korisnik mišem klikne na označeni pasus, biće prikazana poruka `Kliknuli ste na pasus`. Funkcija `alert` je deo DOM koji se obično koristi iz jezika JavaScript.

```
<p onclick="alert('Kliknuli ste na pasus')">Ovo je jedan pasus</p>
```

Zaglavlje dokumenta Zaglavlje HTML dokumenta predstavljeno je elementom `head`. U okviru zaglavlja, navode se informacije o dokumentu, kao što su naslov, ključne reči, ime autora itd. Ove informacije mogu biti korisne pretraživačkim mašinama i drugom softveru koji se bavi obradom i arhiviranjem dokumenata. Korisnički agenti obično ne prikazuju direktno sadržaj zaglavlja, ali koriste ovaj sadržaj posredno i informacije navedene u zaglavlju ponekad diktiraju način prikazivanja tela dokumenta.

Sadržaj zaglavlja je tačno jednom naveden element `title`, opciono naveden element `base`, proizvoljno kombinovani sa elementima `script`, `style`, `meta`, `link` i `object`.

```
<!ENTITY % head.misc "(script|style|meta|link|object)*">
<!ELEMENT head (%head.misc;,
     ((title, %head.misc;, (base, %head.misc;)?) |
      (base, %head.misc;, (title, %head.misc;)))>
```

Napravimo na trenutak digresiju kako bismo uporedili datu definiciju sa definicijom elementa `head` u SGML zasnovanom HTML 4.01 standardu:

```
<!ENTITY % head.misc "script|style|meta|link|object">
<!ENTITY % head.content "title & base?">
<!ELEMENT head O O (%head.content;) +(%head.misc;) >
```

Primetimo kako bogat SGML jezik omogućava znatno jezgrovitije i razumljivije navođenja sadržaja elemenata, međutim, ovakav opis je teži za obradu od strane parsera u odnosu na eksplicitno navedene varijante u okviru prikazane XHTML definicije.

Uz atribute za internacionalizaciju i atribut `id`, zaglavlje može da sadrži još atribut `profile`[9].

```
<!ATTLIST head
  %i18n;
  id          ID             #IMPLIED
  profile     %URI;          #IMPLIED
  >
```

U nastavku će detaljno biti opisani elementi koji se navode u okviru zaglavlja.

`title` - Element `title` se koristi za navođenje naslova dokumenta. Naslov je obavezan deo zaglavlja. Autori bi trebalo da koriste element `title` kako bi čitaoci na osnovu njega mogli da identifikuju potencijalni sadržaj dokumenta. Pošto korisnici obično ovaj naslov vide van konteksta (npr. naslov im se prikazuje kao rezultat pretrage korišćenjem neke pretraživačke mašine), preporučuje se da naslov dokumenta bude širok i dovoljno informativan. Npr. umesto kratkog i neinformativnog naslova `Uvod`, bolje je koristiti informativniji naslov `Uvod u jezik HTML`. Element naslov može da ima atribute vezane za internacionalizaciju i identifikator.

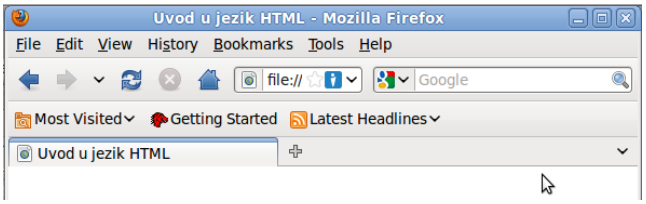
```
<!ELEMENT title (#PCDATA)>
<!ATTLIST title
  %i18n;
  id          ID             #IMPLIED
  >
```

Primer navođenja naslova može biti:

```
<head>
  <title lang="sr">Uvod u jezik HTML</title>
</head>
```

Veb pregledači obično naslov dokumenta prikazuju u naslovu okvira prozora ili u naslovu tab-polja u kome se dokument prikazuje. Npr.

[9] Profili određuju vrstu meta podataka koje bi trebalo da budu navedeni u okviru zaglavlja. Za detalje, čitalac se upućuje na konsultovanje standarda.



Dok element `title` definiše naslov celog dokumenta, naslovi pojedinačnih elemenata (slika, veza i slično) se mogu postavljati korišćenjem atributa `title`.

`base` - Ovim elementom, tj. njegovim atributom `href` se navodi bazni URI koji se koristi prilikom razrešavanja svih relativnih veza, o čemu će biti više reči u poglavlju o vezama.

```
<!ELEMENT base EMPTY>
<!ATTLIST base
  href        %URI;          #REQUIRED
  id          ID             #IMPLIED
  >
```

`meta` - Metainformacije su informacije o samom dokumentu (i ne smatraju se delom samog dokumenta). Ove informacije mogu da uključe informacije u autoru, datumu i načinu kreiranja dokumenta i slično. U okviru samog dokumenta, metainformacije se navode korišćenjem `meta` elementa.

```
<!ELEMENT meta EMPTY>
<!ATTLIST meta
  %i18n;
  id          ID             #IMPLIED
  http-equiv  CDATA          #IMPLIED
  name        CDATA          #IMPLIED
  content     CDATA          #REQUIRED
  scheme      CDATA          #IMPLIED
  >
```

Metainformacije se zadaju u obliku vrednosti nekih svojstava (eng. property-value pairs). Tako, svojstvo `author` nekog dokumenta može da ima vrednost `Filip Marić`. Razlikuju se dve vrste svojstava:

1. svojstva definisana u okviru HTTP protokola (HTTP ekvivalenti),
2. svojstva opšteg tipa nezavisna od HTTP mehanizma.

U zavisnosti od vrste svojstva, za njihovo postavljanje koristi se ili atribut `http-equiv` ili atribut `name`. U oba slučaja vrednost svojstva se navodi u okviru atributa `content`. HTTP ekvivalente navedene u okviru samog dokumenta neki Veb serveri koriste, ekstrahuju ih i šalju u okviru HTTP odgovora. U nekim slučajevima, korisnički agenti vrše ekstrakciju HTTP ekvivalenata iz samih dokumenata i ponašaju se na isti način kao da su ova svojstva bila poslata u okviru HTTP odgovora. Svojstva opšteg tipa koriste pretraživačke mašine, sistemi za arhiviranje dokumenata i slično. Važno je, međutim, napomenuti da se u današnje vreme veliki broj ovih

svojstava zanemaruje, zbog zloupotrebe ovog mehanizma i lažnog prijavljivanja sadržaja stranica od strane pornografskih sajtova.

Razmotrimo nekoliko primera:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Language" content="sr" />
<meta name="language" content="sr" />
<meta name="author" content="Filip Marić" />
<meta name="keywords" lang="en" content="html, metainformation" />
<meta name="keywords" lang="sr" content="html, metainformacije" />
<meta name="description" content="Dokument sa metainformacijama" />
```

U primeru je kroz postavljanje tipa dokumenta i kodnu stranu, korisnički agent informisan da je u pitanju HTML sadržaj kodiran u UTF-8. Zatim je, na nekoliko načina zapisano da je sadržaj na srpskom jeziku. Naveden je autor dokumenta, navedene su ključne reči na dva različita jezika i dat je kratak opis.

Atribut `scheme` daje dodatne informacije o tome kako je potrebno tumačiti vrednost nekog svojstva. Tako se u sledećem primeru navodi da je identifikator u stvari izdavački broj knjige (ISBN).

```
<meta scheme="ISBN" name="identifier" content="978-0201485677" />
```

`script` - Klijentski skript (eng. client-side script) je program koji je priložen uz HTML dokument ili se nalazi u okviru njega. Ovaj program se izvršava na klijentskoj mašini, najčešće u okviru Veb pregledača u trenutku kada se dokument učita ili u nekom naknadnom trenutku kao odgovor na akciju korisnika (aktiviranje veze, pritisak na dugme, itd.) Podrška jezika HTML za skriptove je opšta i ne zavisi od specifičnog jezika koji se koristi za pisanje skriptova. Zbog izuzetnog značaja koji klijentski skriptovi imaju, ovoj temi će biti posvećena posebna glava (glava 4)ove knjige.

```
<!ELEMENT script (#PCDATA)>
<!ATTLIST script
  id          ID             #IMPLIED
  charset     %Charset;      #IMPLIED
  src         %URI;          #IMPLIED
  type        %ContentType;  #REQUIRED
  defer       (defer)        #IMPLIED
  >
```

Element `script` služi da se u dokument umetne skript. Ovaj element može da se navede proizvoljan broj puta u zaglavlju, ali i u telu dokumenta. Sadržaj elementa je tekst skripta na nekom skript programskom jeziku. Atributima se daju neke detaljnije informacije o samom skriptu. Opišimo najznačajnije. Atribut `src` služi da se ukaže na spoljašnju datoteku koja sadrži kod skripta, u kom slučaju telo `script` elementa biva prazno. Atribut `type` služi da se navede jezik u kome je skript napisan. Jezik se navodi u obliku tipa sadržaja (najčešće `text/javascript`). Na primer, skript iz datoteke `script.js` se uključuje korišćenjem:

```
<script type="text/javascript" src="script.js"></script>
```

`style` - Element `style`, koji se navodi u zaglavlju dokumenta, služi da uključi informacije u grafičkom izgledu (tj. o vizuelnoj prezentaciji) dokumenta.

```
<!ELEMENT style (#PCDATA)> <!ATTLIST style
  %i18n;
  id          ID             #IMPLIED
  title       %Text;         #IMPLIED
  type        %ContentType;  #REQUIRED
  media       %MediaDesc;    #IMPLIED
  >
```

Sadržaj ovog elementa najčešće predstavlja opis na jeziku CSS. Atribut `type` služi da se navede jezik u kome je dat stilski opis. Jezik se navodi u obliku tipa sadržaja (najčešće je to `text/css`). Atribut `media` služi za opis medijuma prikazivanja za koji je ovaj stilski opis prilagođen. Podrazumevana vrednost ovog atributa je `screen` i odnosi se na prikaz na ekranu. U narednom primeru, kroz jezik CSS daje se stilski opis kojim se boja teksta svih pasusa postavlja na crveno prilikom prikaza na ekranu i štampanja.

```
<style type="text/css" media="screen, print">
  p {color : red}
</style>
```

`link` - HTML dokumenti su retko zasebni i obično su deo većih kolekcija dokumenata (npr. svako poglavlje neke elektronske knjige se nalazi u okviru zasebne HTML stranice). Element `link` se koristi da se u zaglavlju pojedinačnog dokumenta opiše njegov odnos sa drugim dokumentima i time na neki način cela kolekcija dokumenata poveže u jedinstvenu celinu. Ove informacije koriste uglavnom pretraživačke mašine, ali ih mogu koristiti i korisnički agenti (npr. neki pregledači, poput Opere i Mozile prikazuju poseban meni koji sadrži veze ka navedenim dokumentima, neki pregledači automatski učitavaju naredni dokument kolekcije očekujući da će je korisnik uskoro zatražiti i slično).

```
<!ELEMENT link EMPTY>
<!ATTLIST link
  %attrs;
  charset     %Charset;      #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  type        %ContentType;  #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  media       %MediaDesc;    #IMPLIED
  target      %FrameTarget;  #IMPLIED
  >
```

Atribut `href` se koristi da bi se navela adresa (URI) dokumenta sa kojim se uspostavlja neki odnos. Atribut `rel` se koristi kako bi se navela vrsta odnosa. Navedimo neke od najčešće korišćenih vrednosti ovog atributa:

`start`, `contents`, `previous`, `next`, `index`, `end` - ove relacije se koriste kako bi se ukazalo na početnu stranu neke kolekcije dokumenata,

sadržaj neke kolekcije dokumenata, prethodnu stranu, narednu stranu, indeksnu stranu, poslednju stranu, itd.

`stylesheet` - koristi se za ukazivanje na zaseban dokument koji sadrži stilski opis (najčešće CSS) koji opisuje grafičku prezentaciju koju treba primeniti na tekući dokument.

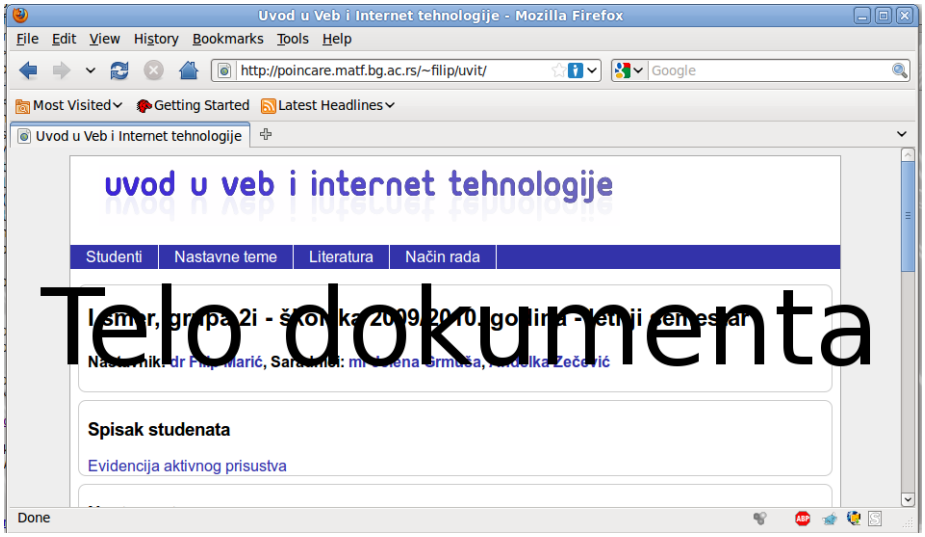
`alternate` - koristi se za ukazivanje na alternativne verzije tekućeg dokumenta (npr. verzije na drugim jezicima, verzije za štampu, itd.)

Na primer:

```
<link rel="next" type="text/html"
      title="Strana 2" href="page-2.html" />

<link rel="alternate" type="text/html"
      hreflang="en" title="Verzija na engleskom"
      href="page1-en.html" />

<link rel="alternate" type="application/pdf"
      title="Verzija za štampu"
      href="page1.pdf" />
```

`object` - Iako `object` element može da bude naveden u okviru zaglavlja, on se mnogo češće upotrebljava u okviru tela dokumenta te će njegova upotreba biti naknadno objašnjena.

Telo dokumenta Telo dokumenta predstavlja sadržaj dokumenta koji različiti korisnički agenti mogu predstaviti na različite načine. Pregledači Veba prikazuju telo u centralnom prozoru ekrana, ispisujući tekst, iscrtavajući slike, prikazujući veze i slično. Korisnički agenti za osobe sa oštećenim vidom izgovaraju korisnicima sadržaj tela dokumenta.



Telo dokumenta je predstavljeno elementom `body`.

```
<!ELEMENT body (%Block;)+>
<!ATTLIST body
  %attrs;
  onload          %Script;   #IMPLIED
  onunload        %Script;   #IMPLIED
  >
```

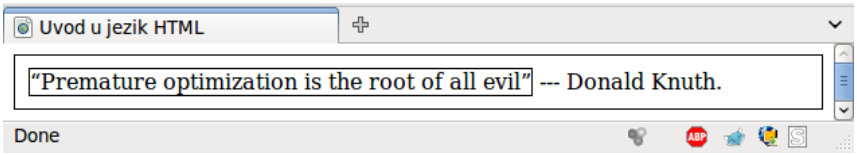
Uz osnovne generičke atribute, element body može još biti okarakterisan isključivo[10] atributima onload i onunload koji sadrže skript kôd koji se poziva prilikom učitavanja ili zatvaranja dokumenta.

Blok i linijski elementi. U okvir tela dokumenta, moguće je postaviti veliki broj različitih HTML elemenata. O svakom specifičnom elementu biće reči u nastavku. Na ovom mestu navedimo jedino narednu, izuzetno značajnu klasifikaciju elemenata koji sačinjavaju telo. Elemente možemo podeliti na *elemente nivoa bloka (eng. block level elements)* i na *linijske elemente (eng. inline elements)*. Razlika između ove dve vrste elemenata se primećuje u nekoliko aspekata:

- Blok elemenati mogu da sadrže druge blok elemente, tekst i linijske elemente, dok linijski elementi mogu da sadrže samo tekst i linijske elemente. Blok elementi, dakle označavaju veće strukture.
- Blok elementi se prilikom prikazivanja formatiraju drugačije od linijskih. Blok elementi obično automatski započinju nov red i prostiru se od početka do kraja širine prozora. Linijski elementi ne započinju nove redove (već su umetnuti u tekući red) i ne šire se do kraja prozora. Ilustrujmo ovo i primerom.

  ```
  <p><q>Premature optimization is the root of all evil</q> ---
     Donald Knuth.</p>
  ```

  Element p kojim se predstavlja pasus je blok element, a element q kojim se predstavlja citat je linijski. Zbog toga se pasus prostire celom širinom prozora, dok se citat prostire samo onoliko koliko je neophodno.

- Blok elementi i linijski elementi se razlikuju i po tome kako se odnose prema jezicima koji se pišu s desna ulevo.

Navedimo sada DTD fragmente[11] koji definišu blok i linijske elemente:

[10] Napomenimo da se mnoštvo atributa koje opisuju grafičku prezentaciju tela (npr. background, text, link, alink, vlink) smatra zastarelim i izbačeno je iz striktne verzije standarda.

[11] Fragment ovde prikazan je preuzet iz HTML 4.01. DTD fragment jezika XHTML 1.0 je donekle komplikovaniji jer definiše veći broj pomoćnih parametarskih entiteta kojima se kodiraju specijalni uslovi koji se u SGML-u jednostavno izražavaju dodatnim pravilima uključivanja i isključivanja sadržaja.

```
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ENTITY % phrase "em | strong | dfn | code | samp | kbd | var | cite |
                   abbr | acronym" >
<!ENTITY % special "a | img | object | br | script | map | q | sub | sup |
                    span | bdo">
<!ENTITY % formctrl "input | select | textarea | label | button">
<!ENTITY % Inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">

<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % list "ul | ol">
<!ENTITY % preformatted "pre">
<!ENTITY % Block
     "p | %heading; | %list; | %preformatted; | dl | div | noscript |
      blockquote | form | hr | table | fieldset | address">

<!ENTITY % Flow "%Block; | %Inline;">
```

O svim ovim elementima biće više reči u nastavku.

Naglasimo sada da se telo HTML dokumenta sastoji od elemenata nivoa bloka — direktno navođenje teksta i/ili linijski elemenata u okviru tela dokumenta nije ispravno u striktnoj verziji jezika HTML.

Elementi za grupisanje. Jezik HTML definiše dva elementa kojima se ne pridaje nikakva istaknuta semantika već isključivo služe za grupisanje drugih elemenata i predstavljaju generički mehanizam za nametanje strukture dokumentima. Ovo su element `span` koji je linijski element i element `div` koji je element nivoa bloka. Ovi elementi mogu da imaju isključivo generičke atribute.

```
<!ELEMENT span %Inline;>
<!ATTLIST span
  %attrs;
  >
<!ELEMENT div %Flow;>
<!ATTLIST div
  %attrs;
  >
```

Korišćenje elemenata za grupisanje obično ide uz korišćenje atributa `id` i `class` i korišćenje CSS-a za opis grafičke prezentacije. Element `div`, uz mogućnost njegovog pozicioniranja kroz CSS, postaje veoma korišćen u savremenom Veb dizajnu za određivanje i opisivanje strukture Veb stranice i sve više zamenjuje tabele koje su se nekada koristile za to.

Pretpostavimo, na primer, da u okviru dokumenta želimo da prikažemo podatke o poslovnim i privatnim kontaktima. Pošto HTML nema posebne elemente kojima se predstavljaju objekti kao što su „kontakt”, „telefon”, „elektronska adresa” i slično, za predstavljanje se mogu koristiti generički elementi:

```
<div id="kontakt-1" class="kontakt">
<table class="podaci">
<tr><th>Prezime:</th> <td>Marić</td></tr>
<tr><th>Ime:</th>     <td>Filip</td></tr>
<tr><th>e-mail:</th>  <td><span class="e-mail">filip@math.rs</span></td></tr>
</table>
</div>
```

Kasnije, je korišćenjem CSS moguće postaviti način prezentacije svih kontakata, svih elektronskih adresa i slično. Npr.

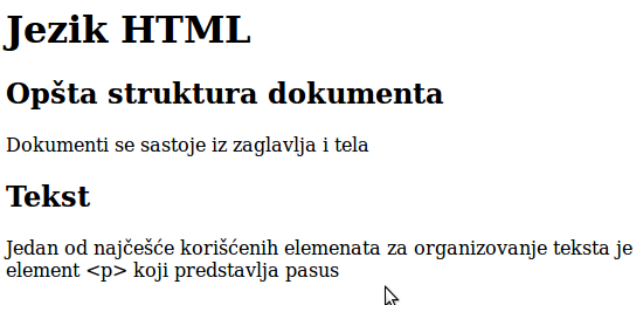
```
.kontakt { border: 1px solid black }
.e-mail  { font-style: italic }
```

Naglasimo još, da je u slučaju da postoji odgovarajući element sa specifičnom semantikom, uvek poželjnije koristiti njega nego generički element za grupisanje. Npr. HTML definiše element `address` za predstavljanje adresa i njega je uvek poželjnije koristiti nego npr. `<div class="address">`.

Naslovi. Naslovi obično ukratko opisuju temu poglavlja koje sledi. Korisnički agenti mogu iskoristiti informacije date kroz naslove, na primer, za automatsko generisanje sadržaja dokumenta. Postoji šest nivoa naslova pri čemu element `h1` označava naslove najvišeg nivoa, dok `h6` označava naslove najnižeg nivoa. Naslovi su blok elementi i obično se prikazuju centrirano, većim simbolima od ostatka teksta. Obično veličina teksta opada sa snižavanjem nivoa naslova.

Naredni tekst sadrži tri naslova i dva pasusa.

```
<h1>Jezik HTML</h1>
<h2>Opšta struktura dokumenta</h2>
<p>Dokumenti se sastoje iz zaglavlja i tela</p>
<h2>Tekst</h2>
<p>Jedan od najčešće korišćenih elemenata za organizovanje teksta
   je element &lt;p&gt; koji predstavlja pasus</p>
```



3.3.5 Tekst

Osnovni gradivni element svih dokumenata je tekst. Iako tekst ne može da bude direktno deo tela HTML dokumenta (podsetimo se, telo mora da se sastoji od niza elemenata nivoa bloka), on može biti sadržaj većine kako linijskih, tako i blok elemenata.

Beline, formatiranje teksta i preformatirani tekst. HTML dopušta korišćenje belina (eng. white space characters) prilikom kreiranja dokumenata. Pod belinama se standardno podrazumevaju razmaci, tabulatori, i prelasci u novi red. Međutim, uobičajeno ponašanje korisničkih agenata je da u većini slučajeva ignoriše beline prilikom prikazivanja teksta. Naime, beline se koriste kako bi se odredile granice „reči", a reči se raspoređuju i prikazuju korišćenjem za to specijalizovanih algoritama. Ovo znači da raspored teksta u izvornom kodu HTML dokumenta neće direktno uticati na raspored prilikom prikazivanja.

Na primer, sadržaj nekog elementa (npr. pasusa) može biti tekst:

```
Ovo    je

   sve    jedan red.
```

Međutim, korisnički agenti bi ovaj tekst prikazali kao:

Ovo je sve jedan red.

pretvarajući sve višestruke beline i nove redove u obične razmake.

U nekim slučajevima, ipak, poželjno je da se očuva i korisnicima prikaže tačan raspored teksta i belina (eng. indentation) iz izvornog HTML koda (na primer, želi se prikazati neki programski kôd). Kako bi se ovo postiglo, HTML uvodi poseban element `pre`.

```
<!ELEMENT pre %pre.content;>
<!ATTLIST pre
  %attrs;
  >
```

Element `pre` je element nivoa bloka, a njegov sadržaj može sačinjavati tekst i linijski elementi, osim elemenata `img`, `object`, `big`, `small`, `sub` i `sup`. XHTML DTD mora eksplicitno da definiše listu svih ovih elemenata u okviru entiteta `%pre.content` (što ovde ne navodimo), dok HTML 4.01 DTD koristi dodatno pravilo isključivanja, kako bi elegantno ovo izrazio:

```
<!ENTITY % pre.exclusion "img|object|big|small|sub|sup">
<!ELEMENT pre - - (%inline;)* -(%pre.exclusion;) -->
<!ATTLIST pre
  %attrs;
  >
```

Ukoliko bi tekst iz prethodno primera bio sadržan u okviru elementa `pre`, pregledači bi ga prikazivali korišćenjem nekog neproporcionalnog slovnog lika fiksirane širine (eng. monospaced font) i rasporedili na isti način kao što je u izvornom HTML dokumentu.

```
<pre>Ovo    je

   sve    jedan red.</pre>
```

```
Ovo    je

   sve    jedan red.
```

Pasusi i linije. Prilikom kreiranja dokumenata autori obično svoje misli grupišu u *pasuse (eng. paragraph)*. Jezik HTML definiše element `p` kojim se predstavljaju pasusi.

```
<!ELEMENT p %Inline;>
<!ATTLIST p
  %attrs;
  >
```

Pasusi su elementi nivoa bloka koji mogu da sadrže isključivo tekst i linijske elemente. Pasusi ne mogu da imaju druge atribute osim generičkih. Napomenimo da su ranije verzije definisale atribute poput npr. `align` za način poravnavanja teksta, ali se ovo smatra zastarelim i preporučuje se upotreba CSS-a.

Raspored reči i linija u okviru pasusa obično određuju korisnički agenti. Ipak, ukoliko korisnik na nekom mestu želi da eksplicitno označi prelazak u novu liniju, ovo može da uradi korišćenjem elementa br.

```
<!ELEMENT br EMPTY>
<!ATTLIST br
  %coreattrs;
  >
```

Element br je praznog sadržaja, tako da se u okviru XHTML obično navodi kao <br /> i od atributa može da sadrži samo generičke atribute id, class, title i style.

Naglasimo još da je moguće i eksplicitno naglasiti da se na nekom mestu zabranjuje prelom linije. U tom slučaju je potrebno umesto običnog razmaka upotrebiti entitet   (eng. no breaking space). Korisnički agenti ponekad vrše hifenaciju (rastavljanje reči na kraju linije). Korišćenjem entiteta moguće je sugerisati mesto u reči na kojem je poželjno izvršiti prelom. Ukoliko se reč zaista prelomi na tom mestu se prikazuje kao crtica na kraju reda, međutim, ukoliko se reč ne prelomi, se ignoriše i umesto njega se ne prikazuje ništa.

Oznake strukture delova deksta. Naredni elementi koji se u HTML standardima nazivaju *elementi fraza (eng. phrase elements)* služe da naglase vrstu i prirodu određenih delova i fraza u tekstu.

```
<!ENTITY % phrase "em | strong | dfn | code | samp | kbd | var | cite |
                   abbr | acronym">
<!ELEMENT (%phrase;) %Inline;>
<!ATTLIST (%phrase;)
  %attrs;
  >
```

Svi ovi elementi su linijski elementi i samim tim mogu da sadrže isključivo tekst i druge linijske elemente. Osim generičkih atributa, ne mogu da sadrže druge atribute. Navedimo sada ukratko i njihovu semantiku i neke primere upotrebe.

em - označava istaknut tekst.

strong - označava naročito istaknut tekst.

dfn - označava definicije nekog termina (koji je sadržan u okviru samog elementa).

code - označava fragmente programskog koda.

samp - označava primere izlaza nekog programa ili skripta.

kbd - označava tekst koji korisnik treba da unese.

var - označava promenljive ili parametre programa.

cite - sadrži citat ili referencu na neki izvor.

abbr - označava da je u pitanju neka skraćenica (eng. abbreviation).

acronym - označava da je u pitanju akronim — posebna vrsta skraćenice koja je sačinjena od početnih slova ili slogova termina.

Vizuelna prezentacija ovih elemenata zavisi od korisničkih agenata i može biti izmenjena korišćenjem stilskih listova (CSS). Npr. element em, korisnički agenti obično prikazuju iskošenim slovima (eng. italics), dok element strong obično prikazuju debelim slovima (eng. bold).

```
<p>Više detalja o <abbr lang="en" title="World Wide Web">WWW</abbr> se može naći
u <cite>[ISO-0000]</cite>. Pogledajte <em>obavezno</em> poglavlje 9.2.</p>
```

Neki jezici (npr. francuski) zahtevaju korišćenje izdignutog ili spuštenog teksta (eng. superscripts and subscripts). Takođe, ovaj način prikaza se koristi za zapis indeksa i eksponenata u okviru naučnih formula. HTML definiše element sup za zapis izdignutog teksta i sub za zapis spuštenog teksta.

```
<!ELEMENT (sub|sup) %Inline;>
<!ATTLIST sub %attrs;>
```

Oba elementa su linijska i osim generičkih ne dopuštaju druge atribute.
Navedimo i primere:

```
H<sub>2</sub>O, E = mc<sup>2</sup>,
<span lang="fr">M<sup>lle</sup> Dupont</span>
```

$$H_2O,\ E = mc^2,\ M^{lle}\ Dupont$$

Često je u okviru dokumenata potrebno citirati određeni tekst. U tu svrhu, HTML uvodi elemente blockquote i q.

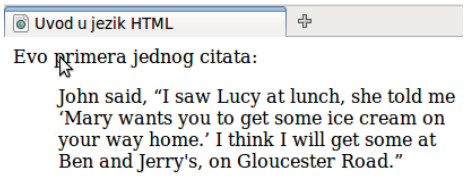
```
<!ELEMENT blockquote %Block;>
<!ATTLIST blockquote
  %attrs;
  cite        %URI;          #IMPLIED
  >

<!ELEMENT q %Inline;>
<!ATTLIST q
  %attrs;
  cite        %URI;          #IMPLIED
  >
```

Element blockquote služi za citiranje većih delova teksta, spada u blok elemente i njegov sadržaj moraju da sačinjavaju blok elementi. Korisnički agenti ovaj element obično prikazuju kao uvučeni blok.

Element q služi za navođenje kraćih citata, spada u linijske elemente i samim tim može da sadrži samo linijske elemente. Prilikom prikazivanja, korisnički agenti obično oko sadržaja elementa q automatski stavljaju navodnike, pri čemu se vrsta navodnika određuje na osnovu jezika.

Navedimo primer upotrebe ovih elemenata.

```
<blockquote cite="http://www.w3.org/TR/REC-html40/struct/text.html">
<p>John said, <q lang="en-us">I saw Lucy at lunch, she told me
<q lang="en-us">Mary wants you to get some ice cream on your way home.</q>
I think I will get some at Ben and Jerry's, on Gloucester Road.</q></p>
</blockquote>
```



Oznake stila slova. Za razliku od ranije navedenih elemenata fraza koji služe da označe logičku strukturu teksta i ulogu neke fraze u okviru dokumenta, naredni elementi služe da se direktno odredi grafička prezentacija (kaže se i fizički stil) nekih delova teksta. Iako ovi elementi nisu još zvanično dobili status zastarelih elemenata (eng. deprecated), njihova intenzivna upotreba se ne preporučuje.

```
<!ENTITY % fontstyle "tt | i | b | big | small">
<!ELEMENT (%fontstyle;) %Inline;>
<!ATTLIST (%fontstyle;)
  %attrs;
  >
```

Svi ovi elementi su linijski elementi i samim tim mogu da sadrže isključivo tekst i druge linijske elemente. Osim generičkih atributa, ne mogu da sadrže druge atribute. Navedimo sada ukratko i njihovu semantiku.

`tt` - označava da treba koristiti neproporcionalna slova — `teletype font`.

`i` - označava da treba koristiti iskošena slova — *italic font*.

`b` - označava da treba koristiti podebljana slova — **bold font**.

`big` - označava da treba koristiti povećana slova — big font.

`small` - označava da treba koristiti smanjena slova — small font.

3.3.6 Liste

Liste se u dokumentima koriste kada treba da se izrazi neko nabrajanje stavki. Jezik HTML podržava tri tipa lista:

- nenumerisane liste (eng. unordered lists), predstavljene elementom `ul`,
- numerisane liste (eng. ordered lists), predstavljene elementom `ol`,
- Definicione liste (eng. definition lists), predstavljene elementom `dl`.

```
<!ENTITY % lists "ul | ol | dl">
```

Numerisane i nenumerisane liste sadrže niz stavki označenih elementom `li`.

```
<!ELEMENT ul (li)+>
<!ATTLIST ul
  %attrs;
  >
<!ELEMENT ol (li)+>
<!ATTLIST ol
  %attrs;
  >
<!ELEMENT li %Flow;>
<!ATTLIST li
  %attrs;
  >
```

Liste su elementi nivoa bloka. Stavke u listama mogu da sadrže bilo tekst, blok elemente, bilo linijske elemente. Iako su starije verzije HTML-a definisale određeni broj atributa kojima se podešavao izgled lista, oni su uklonjeni, tako da u savremenom HTML-u elementi koji predstavljaju liste i njihove stavke mogu da sadrže samo generičke atribute.

Definicione liste sadrže niz termina koji se definišu. Ono što se definiše (lat. definiendum) se označava elementom `dt`, a ono čime se definiše (lat. definiens) se označava elementom `dd`. Ni ovi elementi nemaju atribute osim generičkih.

```
<!ELEMENT dl (dt|dd)+> <!ATTLIST dl
  %attrs;
  >
<!ELEMENT dt %Inline;>
<!ATTLIST dt
  %attrs;
  >
<!ELEMENT dd %Flow;>
<!ATTLIST dd
  %attrs;
  >
```

Moguće je kombinovati različite vrste lista, pri čemu liste mogu biti i ugnježđene. Navedimo primer jedan primer.

```
<dl>
<dt><strong>Sastojci:</strong></dt>
<dd>
<ul>
<li>100 g. brašna</li>
<li>10 g. šećera</li>
<li>1 čaša vode</li>
<li>2 jaja</li>
<li>so, šećer</li>
</ul>
</dd>
<dt><strong>Postupak: </strong></dt>
<dd>
<ol>
<li>Izmešaj suve sastojke.</li>
<li>Dodaj vlažne sastojke.</li>
<li>Mešaj 10 minuta.</li>
<li>Peci sat vremena na temperaturi od 300 stepeni.</li>
</ol>
</dd>
</dl>
```

Sastojci:

- 100 g. brašna
- 10 g. šećera
- 1 čaša vode
- 2 jaja
- so, šećer

Postupak:

1. Izmešaj suve sastojke.
2. Dodaj vlažne sastojke.
3. Mešaj 10 minuta.
4. Peci sat vremena na temperaturi od 300 stepeni.

3.3.7 Veze

Ono što jezik HTML razdvaja od ostalih jezika za kreiranje struktuiranih dokumenata i pripremu dokumenata za štampu je mogućnost kreiranja veza, tj. kreiranja hipertekstualnih dokumenata. *Veze (eng. links)* povezuju dva resursa na Vebu. Krajevi veza se ponekad nazivaju *sidra (eng. anchors)* pri čemu se razlikuju polazna i dolazna sidra. Veza kreće od polaznog sidra (obično istaknutu frazu u okviru HTML dokumenta) i ukazuje ka dolaznom sidru (koje može biti drugi HTML dokument, slika, video isečak, itd).

Sidra se kreiraju elementom `a`.

Najčešći oblik korišćenja elementa `a` je uz navođenje samo atributa `href` čime se omogućava kreiranje veze ka drugom resursu na Vebu. Na primer:

```
<p>Primer veze: <a href="http://www.matf.bg.ac.rs">Matematički fakultet</a></p>
```

Ovim se kreira veza ka dokumentu na adresi `http://www.matf.bg.ac.rs` pri čemu se u polaznom dokumentu kreira aktivno sidro koje prikazuje tekst *Matematički fakultet*.

Primer veze: Matematički fakultet

Ukoliko korisnik aktivira sidro (najčešće kliktanjem mišem), pregledač učitava stranu sa adrese `http://www.matf.bg.ac.rs`.

Često se uz samu vezu navodi i atribut `title` koji imenuje objekat na koji se ukazuje i koji pregledači obično prikazuju u okviru balončića (eng. tool-tip) koji iskače kada se mišem pređe preko sidra.

Naredni DTD fragment definiše element `a` i njegove atribute.

```
<!ELEMENT a %a.content;>
<!ATTLIST A
  %attrs;
  %focus;
  charset     %Charset;      #IMPLIED
  type        %ContentType;  #IMPLIED
  name        CDATA          #IMPLIED
  href        %URI;          #IMPLIED
  hreflang    %LanguageCode; #IMPLIED
  rel         %LinkTypes;    #IMPLIED
  rev         %LinkTypes;    #IMPLIED
  shape       %Shape;        rect
  coords      %Coords;       #IMPLIED
  >
<!ENTITY % focus
 "accesskey   %Character;    #IMPLIED
  tabindex    %Number;       #IMPLIED
  onfocus     %Script;       #IMPLIED
  onblur      %Script;       #IMPLIED"
  >
```

Sadržaj elementa `a` čine linijski elementi (osim drugih elementa `a`). Prikazani XHTML DTD fragment zahteva ekplicitno navođenje ovakvih elemenata u okviru entiteta `%a.content`, dok SGML zasnovan HTML 4.01 DTD omogućava da se ovo na jezgrovitiji način izrazi.

```
<!ELEMENT A - - (%inline;)* -(A) -->
```

Dakle, pored generičkih atributa, element `a` može biti okarakterisan i narednim atributima:

`charset` - kodna stranu korišćenu za kodiranje resursa na koji veza ukazuje.

`type` - MIME tip dokumenta na koji veza ukazuje (npr. "text/html", "image/png")

`name` - ime tekućeg sidra koje se navodi kako bi ovo sidro moglo da se koristi kao dolazno sidro (videti pasus o identifikatorima fragmenata). Ovo ime mora biti jedinstveno (i ne sme se čak poklapati sa jedinstvenim identifikatorima elemenata navedenim kroz generički `id` atribut).

`href` - lokacija resursa na Vebu (tj. dolazno sidro) na koji veza ukazuje.

`hreflang` - koristi se isključivo u kombinaciji sa `href` i opisuje jezik resursa na koji ukazuje atribut `href`.

`rel` - odnos između trenutnog i povezanog dokumenta. Odnosi su isti kao u slučaju elementa `link` (str. 62). Npr. ukoliko se iz nekog poglavlja elektronske knjige upućuje na njen sadržaj u okviru veze se može navesti `rel="contents"`.

`rev` - odnos između povezanog i trenutnog dokumenta. Ovaj odnos je recipročan odnosu koji se navodi atributom `rev`. Npr. ukoliko se iz sadržaja knjige upućuje na neko njeno poglavlje, u okviru veze se može navesti atribut `rev="contents"`.

`shape`, `coords` - Ovi atributi se koriste za kreiranje mape slike i biće objašnjeni u odgovarajućem poglavlju.

Takođe, moguće je korišćenje atributa koji kontrolišu fokus ovog elementa.

`accesskey` - taster karaktera koji kada se pritisne u kombinaciji sa nekim od specijalnih tastera na tastaturi (najčešće `alt`) uzrokuje da element dobije fokus. Tako npr.

```
<a href="http://www.google.com" accesskey="g">Google</a>
```

omogućava korisnicima da ovu vezu aktiviraju pritiskom na `Alt+g`. Ovo je, naravno, veoma zavisno od korisničkog agenta kao i od operativnog sistema koji klijent koristi.

`tabindex` - redni broj u listi elemenata koja kontroliše redosled fokusiranja elemenata korišćenjem tastera `tab`.

`onfocus` - akcija (skript) koji se izvršava u trenutku kada element dobije fokus.

`onblur` - akcija (skript) koji se izvršava u trenutku kada element izgubi fokus.

Adresiranje. U okviru `href` atributa navodi se adresa resursa na koji veza upućuje. Adresa se zadaje bilo *apsolutno* (u obliku URI) bilo *relativno*. Na primer, sidro

```
<a href="http://www.matf.bg.ac.rs">Matematički fakultet</a>
```

koristi apsolutno adresiranje, dok sidro

```
<a href="images/logo.png">Logo</a>
```

koristi relativno adresiranje. Ukoliko adresa ne predstavlja ispravno kreiran kompletan URI, smatra se da je adresiranje relativno. Tako, sidro

```
<a href="www.matf.bg.ac.rs">Matematički fakultet</a>
```

verovatno sadrži grešku.

Na relativno navedene adrese primenjuje se postupak razrešavanja adresa. Prilikom razrešavanja koristi se tzv. bazna adresa. Bazna adresa se može navesti korišćenjem elementa base u zaglavlju dokumenta. Na primer,

```
<base href="http://www.matf.bg.ac.rs/~filip/index.html" />
```

Relativne adrese se razrešavaju korišćenjem oznake protokola, imena servera i direktorijuma na serveru iz bazne adrese. Tako, ukoliko se u okviru dokumenta čiji je `base` element prikazan u prethodnom primeru nađe sidro:

```
<a href="images/logo.png">Logo</a>
```

će ukazivati na adresu:

`http://www.matf.bg.ac.rs/~filip/images/logo.png`.

Slično, sidro:

```
<a href="../index.html">Glavna strana fakulteta</a>
```

će ukazivati na adresu:

`http://www.matf.bg.ac.rs/index.html`.

Ukoliko se element `base` izostavi, što je veoma često slučaj, za baznu adresu se uzima adresa sa koje je preuzet tekući dokument i relativne adrese se razrešavaju u odnosu na ovu adresu.

Opšta je preporuka da se u okviru stranica istog dokumenta (iste Veb prezentacije) koristi relativno adresiranje, a da se apsolutno adresiranje koristi isključivo prema spoljašnjim dokumentima, čiji adrese su opšte poznate i nepromenljive.

Fragmenti i adresiranje fragmenata. U nekim slučajevima nije dovoljno uspostaviti vezu samo sa HTML dokumentom već je potrebno klijenta uputiti na tačno određeni deo (fragment) u okviru stranice. Da bi to bilo moguće potrebno je identifikovati željeni fragment što se čini bilo navođenjem identifikatora uz neki element, bilo eksplicitnim kreiranjem dolaznog sidra. Na primer, fragment koji započinje nekim naslovom, može biti označen zadavanjem identifikatora tom naslovu:

```
<h1 id="spisak">Spisak studenata</h1>
```

Dolazna sidra se kreiraju navođenjem elementa `a`, pri čemu se u ovom slučaju atribut `href` izostavlja i umesto njega se navodi atribut `name`. Na primer.

```
<h1><a name="spisak">Spisak studenata</a></h1>
```

Kao što je rečeno, vrednost atributa `name` mora biti jedinstvena i ne sme se poklapati čak ni sa jednim drugim identifikatorom u okviru dokumenta. Preporučeno je da sadržaj dolaznog sidra nikada ne bude prazan jer se neki korisnički agenti ponašaju neispravno u takvim slučajevima.

U oba slučaja, na označen fragment, moguće je uputiti navođenjem tzv. *identifikatora fragmenta* `#spisak` na kraju URI adrese stranice koja sadrži spisak. Npr.

```
Spisak studenata se može videti
<a href="http://www.matf.bg.ac.rs/~filip/studenti.html#spisak"
   title="Spisak studenata">ovde</a>.
```

Ukoliko se povezivanje vrši u okviru jedne HTML stranice, čest je slučaj da se navede samo identifikator fragmenta, dok se ostali elementi adrese preuzimaju iz (najčešće izostavljene) bazne adrese.

```
<h1 id="sadrzaj">Sadrzaj</h1>
...
<p>Pogledaj <a href="#sadrzaj">sadrzaj</a> ovog dokumenta.</p>
```

Moguće je kreirati veze *unapred* tj. veze ka fragmentima koji su definisani u dokumentu tek nakon navođenja same veze.

Kod odlučivanja da li koristiti `id` ili `name` prilikom definisanja fragmenata treba imati u vidu i sledeće:

- Atribut `id` se pored identifikovanja fragmenta može upotrebiti i u druge svrhe (npr. za zadavanje stilskog opisa),
- Neki stariji korisnički agenti ne podržavaju identifikovanje fragmenata korišćenjem `id`.
- `name` može da sadrži bogatiji skup imena (npr. moguće je navođenje entiteta u okviru imena).

3.3.8 Objekti, slike, apleti

Jezik HTML dopušta umetanje različitih multimedijalnih sadržaja (multimedijalnih objekata) u dokumente. Tako, moguće je umetati slike, zvučne i video isečke, aplete, itd. Preporučeni način umetanja multimedijalnih objekata je korišćenje elementa `object`. Npr.

```
<object type="image/png" data="portrait.png">Portret autora dokumenta</object>
```

Ranije verzije jezika HTML nudile su posebne elemente (`img`, `applet` i `iframe`) za umetanje slika, apleta i drugih HTML dokumenata. Element `applet` se u novijim verzijama HTML-a smatra zastarelim i nije preporučljivo upotrebljavati ga. Elementi `img` i `iframe` su i dalje podržani, ali se ipak preporučuje opšteg elementa `object`. Osnovni problem uvođenja ovakvih specifičnih elemenata u jezik HTML je činjenica da je različitih vrsta multimedijalnih sadržaja sve više i više i ovi elementi ne rešavaju problem podrške za raznorazne sadržaje koji će se u budućnosti koristiti.

Generičko uključivanje sadržaja: `object` element. Većina korisničkih agenata ima ugrađene mehanizme za prikaz uobičajenih tipova multimedijalnih sadržaja (npr. tekst, JPEG, GIF, PNG slike). Međutim, postoje i tipovi sadržaja koje korisnički agenti ne umeju da samostalno prikažu već za njihov prikaz koriste dodatne spoljašnje aplikacije. Prilikom uključivanja sadržaja korišćenjem generičkog elementa `object` moguće je navesti tri vrste informacija:

- Podatke koje je potrebno prikazati (pri čemu se pretpostavlja da korisnički agent te podatke ume da prikaže). Na primer ukoliko se želi uključivanje neke slike potrbno je navesti lokaciju datoteke koja predstavlja sliku.
- Implementaciju programa koji treba da prikaže neki multimedijalni sadržaj u okviru dokumenta. Na primer, ukoliko se želi uključivanje Java apleta koji prikazuje tačno vreme, potrebno je navesti lokaciju bajt koda apleta.
- Dodatne vrednosti tj. parametre koji su neophodni tokom prikaza multimedijalnog sadržaja. Na primer, neki apleti zahtevaju da im se postave određene inicijalne vrednosti.

Najčešće nije potrebno navoditi sve ove informacije, već se neke podrazumevaju. Na primer, ukoliko se navedu podaci o JPEG slici, nije potrebno posebno navoditi lokaciju programa koji prikazuje sliku već se podrazumeva da korisnički agent ume samostalno da prikaže sliku.

Umesto davanja potpunog opisa elementa `object`, navešćemo samo neke najčešće načine njegove upotrebe.

data - lokacija na kojoj se nalaze podaci koji se uključuju

type - MIME tip podataka koji su navedeni atributom `data`. Ukoliko korisnički agenti nemaju mogućnost prikaza ovog tipa podataka, obično se podaci ni ne preuzimaju. Ukoliko se tip naveden ovde razlikuje od tipa prijavljenog u okviru zaglavlja HTTP odgovora u kojem su podaci stigli, prednost ima HTTP zaglavlje.

classid - lokacija implementacije multimedijalnog sadržaja. Ovaj atribut može da se koristi bilo sa bilo umesto atributa `data` kako bi se naveo softver koji treba da prikaže multimedijalni objekat.

codebase - bazni adresa (URI) koja se koristi za razrešavanje relativnih adresa navedenih u okviru `classid` ili `data` atributa.

codetype - MIME tip implementacije navedene putem `classid`. Slično kao u slučaju `type` atributa, sprečava preuzimanje nepodržanih objekata.

standby - tekst koji se prikazuje dok se multimedijalni sadržaj ne učita.

Sadržaj elementa `object` može biti proizvoljna kolekcija blok i linijskih elemenata i može dodatno sadržati poseban element `param` koji služi za navođenje dodatnih parametara. Sadržaj elementa `object` se prikazuje samo u slučaju da nije moguće prikazati sam objekat, ali se on u svakom slučaju analizira kako bi se izdvojili `param` elementi. Ovo se može koristiti kako bi se, u slučaju da korisnički agent ne ume da prikaže određenu vrstu sadržaja, ponudile alternativne varijante.

```
<!-- Prvo se pokušava prikaz Python apleta -->
<object title="The Earth as seen from space"
        classid="http://www.observer.mars/TheEarth.py">
  <!-- Inače, pokušava se MPEG video -->
  <object data="TheEarth.mpeg" type="application/mpeg">
    <!-- Inače, pokušava se GIF slika -->
    <object data="TheEarth.gif" type="image/gif">
      <!-- Inače, prikazuje se tekst -->
      The <strong>Earth</strong> as seen from space.
    </object>
  </object>
</object>
```

Element `param` služi da se zadaju dodatni parametri objektu zadatom putem `classid`. Obično se parametri zadaju putem parova nekih svojstava i njihovih vrednosti. Za zadavanje ovih parova se koriste atributi `name` i `value`. Značenje svakog od svojstava je specifično za objekat koji ih koristi i ne postoje nikakva standardna svojstva. Navedimo jedan primer upotrebe elementa `param`.

```
<object classid="java:Clock.class"
        codetype="application/java"
        width="100"
        height="100" title="A clock java applet!"
        standby="Do you know what time it is?">
        <param name="type" value="analog" />
        <param name="bgcolor" value="white" />
</object>
```

Ovim se u HTML dokument umeće Java aplet čiji je bajt kod predstavljen datotekom `Clock.class`. Apletu su zadata dva parametra: `type` sa vrednošću `analog` (čime se zadaje da se sat iscrtava kao analogni sat) i `bgcolor` sa vrednošću `white` (čime se boja pozadine postavlja na belu).

Uključivanje slika korišćenjem `img` elementa. Za uključivanje slika u dokumente moguće je koristiti i specifični `img` element.

```
<!ELEMENT img EMPTY>
<!ATTLIST img
  %attrs;
  src         %URI;          #REQUIRED
  alt         %Text;         #REQUIRED
  longdesc    %URI;          #IMPLIED
  height      %Length;       #IMPLIED
  width       %Length;       #IMPLIED
  usemap      %URI;          #IMPLIED
  ismap       (ismap)        #IMPLIED
  >
```

Sadržaj elementa `img` mora biti prazan. Obavezni atributi prilikom korišćenja ovog elementa su:

`src` - adresa slike. Adresa se zadaje bilo apsolutno (u obliku URI) bilo relativno (na isti način kao u slučaju veza, što je to opisano na strani 74.

`alt` - alternativni tekst. Ovaj atribut je značajan u slučaju da korisnički agent iz nekog razloga nije u stanju da prikaže sliku. U tom slučaju, korisniku se prikazuje ovde naveden tekst. Sličnu ulogu ima i generički `title` atribut tako da je poželjno i njega navoditi.

Na primer:

```
<p>
<img src="earth.jpg" alt="Planeta zemlja" title="Planeta zemlja"/>
Čuvajmo našu planetu!
</p>
```

Čuvajmo našu planetu!

Osim navedenih, element `img` može da sadrži i sledeće atribute:

`longdesc` - duži opis slike. Ovaj opis pregledači obično ne prikazuju eksplicitno.

`width`, `height` - širina i visina slike (najčešće u pikselima ili procentima elementa koji sadrži sliku. Ukoliko su vrednosti ovako navedene različite od prirodne dimenzije slike korisnički agenti proširuju ili sakupljaju sliku. U slučaju širenja slike obično se gubi na njenom kvalitetu. U slučaju da se želi prikaz slike u dimenzijama manjim od njenih prirodnih dimenzija, umesto da se to radi navođenjem ovih atributa, preporučljivo je smanjiti sliku i izmeniti njene prirodne dimenzije (korišćenjem nekog od alata za obradu slika) kako bi se uštedelo prilikom prenosa slike kroz mrežu. Dakle, poželjno je da vrednosti ovih atributa odgovaraju vrednostima prirodne dimenzije slike. Iako se u tom slučaju naizgled ne dobija ništa (prikaz ostaje identičan kao i u slučaju da ovi atributi nisu navedeni), navođenje ovih atributa ipak ima smisla jer se time korisničkom agentu daje do znanja kolika je slika i pre nego što se sama slika učita, tako da je moguće pravilno rasporediti ostatak sadržaja.

`ismap`, `usemap` - odnose se na mape slika i biće objašnjeni u odgovarajućem delu teksta.

3.3.9 Tabele

HTML tabele omogućavaju grupisanje podataka u vrste i kolone ćelija.

Tabelama je moguće pridružiti naslov (korišćenjem elementa `caption`) kojim se ukratko opisuje svrha tabele. Takođe, moguće je zadati i duži opis tabele (korišćenjem atributa `summary`).

Vrste tabela se mogu grupisati u zaglavlje (eng. header) tabele, podnožje (eng. footer) tabele i sekcije u telu (eng. body) tabele (korišćenjem `thead`, `tfoot` i `tbody` elemenata). Korisnički agenti koriste ovakvo grupisanje vrsta tabele na različite načine. Na primer, neki pregledači omogućavaju skrolovanje tela tabele nezavisno od zaglavlja i podnožja. Takođe, prilikom štampanja dugačkih tabela, često se zaglavlje i podnožje ponavljaju na svakoj strani koja sadrži podatke iz tabele.

Takođe, moguće je grupisati i kolone tabele (korišćenjem `colgroup` i `col` elemenata). Svim kolonama u okviru grupe je moguće istovremeno menjati osobine.

Ćelije tabele su ili naslovne ćelije (označene elementom `th`) ili ćelije koje sadrže obične podatke (označene elementom `td`). Ćelije mogu da se prostiru i kroz više vrsta i/ili kolona. Svakoj ćeliji sa podacima je moguće pridružiti informaciju o tome koja naslovna ćelija odgovara tim podacima čime se omogućuje da specifični korisnički agenti (npr. audio agenti za slabovide, pregledači za male ekrane) na odgovarajući način prikažu podatke.

Tabele su jako često korišćene za raspoređivanje sadržaja na stranici (eng. layout). U novije vreme preporučuje se raspoređivanje sadržaja korišćenjem stilskih listova (CSS).

Pre detaljne specifikacije svakog od elemenata za opis tabela, navedimo primere.

```
<table border="1"
       summary="Ova tabela daje određene podatke o voćnim mušicama:
                prosečnu visinu i težinu i procenat jedinki sa crvenim
                očima (za muške i za ženske jedinke).">
    <caption><em>Podaci o voćnim mušicama</em></caption>
    <tr>
       <th rowspan="2"> </th>
       <th colspan="2">Prosečna</th>
       <th rowspan="2">Crvene<br/>oči</th>
    </tr>
    <tr>
       <th>visina</th>
       <th>težina</th>
    </tr>
    <tr>
       <th>Muški</th>
       <td>1.9</td>
       <td>0.003</td>
       <td>40%</td>
    </tr>
    <tr>
       <th>Ženske</th>
       <td>1.7</td>
       <td>0.002</td>
       <td>43%</td>
    </tr>
</table>
```

Podaci o voćnim mušicama

	Prosečna		Crvene oči
	visina	težina	
Muški	1.9	0.003	40%
Ženske	1.7	0.002	43%

Opišimo sada detaljno elemente koji se koriste za opis tabela.

Opis cele tabela je predstavljen elementom `table`.

```
<!ELEMENT table
     (caption?, (col*|colgroup*), thead?, tfoot?, (tbody+|tr+))>

<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
<!ENTITY % TRules "(none | groups | rows | cols | all)">
<!ATTLIST table
  %attrs;
  summary     %Text;         #IMPLIED
  width       %Length;       #IMPLIED
  border      %Pixels;       #IMPLIED
  frame       %TFrame;       #IMPLIED
  rules       %TRules;       #IMPLIED
  cellspacing %Length;       #IMPLIED
  cellpadding %Length;       #IMPLIED
  >
```

Sadržaj elementa `table` čine redom opcioni element `caption` koji predstavlja opis tabele, opcioni niz elemenata `col` ili opcioni niz elemenata `colgroup` kojim se opisuje grupisanje kolona, opcioni elementi `thead` i `tfoot` kojima se opisuju zaglavlje i podnožje tabele neprazan niz `tbody` elemenata ili neprazan niz `tr`

elemenata kojima se opsuje telo tabele, odnosno vrste koje sačinjavaju telo tabele.

Osim generičkih, element `table` može imati i sledeće atribute:

`summary` - duži opis tabele.

Moguće je koristiti i atribute za opis vizuelne prezentacije tabela. Alternativni, preporučeni način da se opiše vizuelna prezentacija tabele je korišćenje stilskih listova (CSS).

`width` - širina tabele.

`cellpadding` - margine sadržaja ćelije, tj. prostor (u pikselima) između okvira ćelije i njenog sadržaja.

`cellspacing` - razmak između susednih ćelija.

Pored ovih, narednim atributima je moguće opisati okvir tabele:

`border` - širina okvira u pikselima.

`frame` - određuje se koje stranice spoljašnjeg pravougaonog okvira koji uokviruje tabelu će biti prikazane. Dozvoljene su vrednosti:

- `void`: ne prikazuje se ni jedna stranica,
- `above`: prikazuje se samo gornja stranica,
- `below`: prikazuje se samo donja stranica,
- `lhs`: prikazuje se samo leva stranica,
- `rhs`: prikazuje se samo desna stranica,
- `hsides`: prikazuje se samo horizontalne stranice (gornja i donja),
- `vsides`: prikazuje se samo vertikalne stranice (leva i desna),
- `box`: prikazuju se sve četiri stranice,
- `border`: prikazuju se sve četiri stranice.

`rules` - određuje koje stranice unutrašnje mreže linija između ćelija tabele će biti prikazane. Dozvoljene su vrednosti:

- `none`: linije mreže neće biti prikazane,
- `groups`: linije će biti prikazane samo između grupa, kolona (opisanih kroz `colgroup` i `col`) i grupa vrsta (opisanih kroz `thead`, `tfoot` i `tbody`),
- `rows`: linije će biti prikazane samo između vrsta tabele,
- `cols`: linije će biti prikazane samo između kolona tabele,
- `all`: linije će biti prikazane između svih vrsta i kolona.

Naslov tabele. Naslov tabele se zadaje elementom `caption`, čiji sadržaj čine isključivo linijski elementi, a koji ne može da ima drugih atributa osim generičkih.

```
<!ELEMENT caption %Inline;>
<!ATTLIST caption
  %attrs;
  >
```

Najviše jedan element `caption` se može navesti isključivo neposredno nakon etikete `<table>`.

Vrste i ćelije tabele. Vrste tabele se opisuju elementom `tr`. Sadržaj ovog elementa je neprazan niz ćelija tabele (opisanih kroz elemente `th` ili `td`). Element `tr` može imati samo generičke atribute ili atribute za poravnavanje sadržaja ćelija koje sadrži.

```
<!ELEMENT tr       (th|td)+>

<!ENTITY % cellhalign
  "align      (left|center|right|justify|char) #IMPLIED
   char       %Character;    #IMPLIED
   charoff    %Length;       #IMPLIED"
  >

<!ENTITY % cellvalign
  "valign     (top|middle|bottom|baseline) #IMPLIED"
  >

<!ATTLIST tr
  %attrs;
  %cellhalign;
  %cellvalign;
  >
```

Osim generičkih atributa, moguće je navesti atribute za poravnavanje sadržaja ćelija. Poravnavanje je moguće definisati bilo na nivou vrste, bilo na nivou pojedinačnih ćelija, bilo na nivou grupe vrsta i/ili kolona.

`align` - opisuje horizontalno poravnavanje sadržaja ćelije tabele. Dopuštene su vrednosti:

- `left`: sadržaj je poravnat na levu marginu,
- `center`: sadržaj je centriran,
- `right`: sadržaj je poravnat na desnu marginu,
- `justify`: sadržaj je poravnat i na levu i na desnu marginu,
- `char`: tekst je poravnat u odnosu na određeni karakter koji se navodi kroz atribut `char`. Najčešća upotreba ovog poravnavanja je u slučaju tabela koje sadrže decimalne brojeve, koji se onda poravnavaju na decimalnu tačku.

`valign` - opisuje vertikalno poravnavanje sadrăja ćelija tabele. Dopuštene su vrednosti:

- `top`: sadržaj je poravnat na gornju marginu,
- `middle`: sadrăj je vertikalno centriran u ćelijama,
- `bottom`: sadržaj je poravnat na donju marginu,
- `baseline`: sadržaj svih ćelija u redu sa ovim atributom je tako poravnat da prva linija teksta bude na istoj visini.

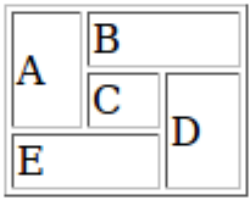
Ćelije tabele mogu biti naslovne ćelije i tada se opisuju elementom `th` ili obične ćelije sa podacima i tada se opisuju elementom `td`.

```
<!ELEMENT th       %Flow;>
<!ELEMENT td       %Flow;>

<!ATTLIST (th|td)
  %attrs;
  abbr        %Text;          #IMPLIED
  axis        CDATA           #IMPLIED
  headers     IDREFS          #IMPLIED
  scope       %Scope;         #IMPLIED
  rowspan     %Number;        "1"
  colspan     %Number;        "1"
  %cellhalign;
  %cellvalign;
  >
```

Sadržaj ćelija može biti bilo kakav niz blok ili linijskih elemenata. Osim generičkih atributa i već opisanih atributa za poravnavanje sadržaja, ćelije tabele mogu da budu okarakterisane i sledećim atributima.

`rowspan`, `colspan` - ovim atributima se definiše da ćelija treba da se prostire kroz više vrsta i/ili kolona. Na primer:

```
<table width="100" border="1">
<tr> <td rowspan="2">A</td>  <td colspan="2">B</td> </tr>
<tr> <td>C</td> <td rowspan="2">D</td> </tr>
<tr> <td colspan="2">E</td> </tr>
</table>
```



Pored ovih moguće je navesti i naredne, ređe korišćene atribute.

`headers`, `scope` - logički povezuju ćelije sa podacima i njima odgovarajuće naslovne ćelije. Ovakve veze se obično koriste u slučaju specijalizovanih korisničkih agenata. Na primer, audio agenti prilikom čitanja sadrža ćelije obično čitaju i odgovarajuću naslovnu ćeliju. Atribut `headers` se navodi u okviru običnih ćelija i sadrži listu jedinstvenih identifikatora (`id`) njoj odgovarajućih naslovnih ćelija. Atribut `scope` se navodi u okviru naslovnih ćelija i opisuje na koje ćelije se ta naslovna ćelija odnosi. Dopuštene vrednosti su:

- `row`: naslovna ćelija za ostatak vrste,
- `col`: naslovna ćelija za ostatak kolone,
- `rowgroup`: naslovna ćelija za ostatak grupe vrsta,
- `colgroup`: naslovna ćelija za ostatak grupe kolona,

`abbr` - skraćeni sadržaj ćelije koji se u nekim slučajevima prikazuje umesto pravog sadržaja.

`axis` - dodatna kategorizacija ćelija.

Grupisanje vrsta i kolona tabele. Vrste tabele se grupišu u zonu zaglavlja, zonu podnožja i zone tela. Za ovo se koriste elementi `thead`, `tfoot` i `tbody`.

```
<!ELEMENT thead    (tr)+>
<!ELEMENT tfoot    (tr)+>
<!ELEMENT tbody    (tr)+>

<!ATTLIST (thead|tbody|tfoot)
  %attrs;
  %cellhalign;
  %cellvalign;
  >
```

Sadržaj sva tri navedena elementa je neprazan niz vrsta tabele (opisanih kroz `tr` elemente). Ovi elementi mogu biti karakterisani isključivo generičkim atributima ili atributima za opis poravnavanja sadržaja ćelija i u tom slučaju se poravnavanje odnosi na sve ćelije u okviru grupe vrsta.

Kolone tabele je moguće grupisati i njihova svojstva opisati kombinacijom elemenata `colgroup` i `col`.

```
<!ELEMENT colgroup (col)*>
<!ELEMENT col      EMPTY>

<!ATTLIST (colgroup|col)
  %attrs;
  span        %Number;       "1"
  width       %MultiLength;  #IMPLIED
  %cellhalign;
  %cellvalign;
  >
```

Element `colgroup` može da sadrži nula ili više elemenata `col`, dok je sadržaj elementa `col` prazan. Osim generičkih atributa, ovi elementi mogu biti okarakterisani atributima koji određuju poravnavanje sadržaja sadržanih ćelija (cele grupe, ukoliko se navedu u okviru `colgroup` ili pojedinačne kolone, ukoliko se navedu u okviru `col`), kao i atributima:

`width` - određuje širinu sadržanih ćelija (cele grupe, ukoliko se navedu u okviru `colgroup` ili pojedinačne kolone, ukoliko se navedu u okviru `col`).

`span` - određuje broj ćelija u grupi, odnosno broj ćelija na koje se odgovarajuća podešavanja poravnavanja i širine odnosi. Ukoliko `colgroup` sadrži pojedinačne `col` elemente, ovaj atribut se zanemaruje.

Na primer,

```
<colgroup span="40" width="20"></colgroup>
```

definiše grupu od 40 kolona pri čemu je svaka kolona široka 20 piksela.

```
<colgroup valign="top">
  <col width="50" align="right" />
  <col span="2" width="100" align="center" />
  <col width="50" align="left" />
</colgroup>
```

definiše grupu od 4 kolone. Kod sve četiri kolone sadrăj je vertikalno poravnat na gornju marginu. Prva kolona je široka 50 piksela i sadržaj je desno poravnat, naredne dve kolone su široke 100 piksela i sadržaj je centriran, dok je četvrta kolona široka 50 piksela i sadržaj je levo poravnat.

3.3.10 Formulari

3.4 CSS - stilski listovi

Najrasprostranjeniji način pridruživanja vizuelne prezentacije HTML i XML dokumentima je korišćenje *stilskih listova (eng. stylesheets)* i jezika *CSS (eng. Cascading Style Sheets)*. CSS je nastao kao rezultat evolucije različitih jezika za opis vizuelne prezentacije (tj. stila) stranica i prva verzija je zvanično objavljena kao W3C preporuka u decembru 1996. godine. CSS je opisan u nekoliko različitih specifikacija (tzv. nivoa), odnosno njihovih verzija, pri čemu svaki naredni nivo predstavlja nadogradnju i dodaje nove mogućnosti prethodnom nivou. U trenutku pisanja ovog teksta, aktuelna W3C preporuka je CSS 2 (tj. 2.1), dok je CSS 3 još u fazi razvoja (još od 2005. godine).

3.4.1 Sintaksa CSS opisa

CSS opisi (sadržani u tzv. stilskim listovima) imaju veoma jednostavnu sintaksu. Prikažimo je kroz primer:

```
/* Prvo pravilo */
h1, div#prvi a:visited {color:blue; font-size: 12px}

/* Drugo pravilo */
p.vazno {
  color: black;
  /* Ovim se postavlja poravnanje teksta */
  text-align: center;
}
```

Navedeni primer sadrži dva CSS pravila. Svako pravilo se sastoji od (niza) selektora (`h1, div#prvi a:visited` u prvom i `p.vazno` u drugom opisu), za kojim se u vitičastim zagradama navodi (niz) CSS deklaracija koje se sastoje od svojstava i njihovih vrednosti. Deklaracije su međusobno razdvojene simbolom `;`, pri čemu nije pogrešno staviti ovaj simbol i na kraj poslednje deklaracije. U okviru CSS opisa moguće je koristiti komentare koji se navode između simbola `/*` i `*/`. Beline se zanemaruju, pa je prelom teksta u okviru CSS opisa je slobodan (npr. prvo pravilo je navedeno u jednom redu, dok je drugo prelomljeno na više redova).

3.4.2 CSS selektori

Svako CSS pravilo počinje navođenjem jednog ili više selektora za čim sledi CSS deklaracija. Selektori određuju elemente u okviru dokumenta na koje se deklaracija odnosi. Postoji nekoliko različitih vrsta selektora.

Ime elementa. Najjednostavni selektor je samo tip (ime) elementa. Ovaj selektor određuje da se CSS deklaracija odnosi na sve elemente navedenog tipa u okviru dokumenta. Na primer, pravilo

```
p { color : blue }
```

govori da se deklaracija odnosi na sve pasuse (elemente `p`) u okviru dokumenta.

Identifikator. Selektori oblika `#id` služe da označe samo onaj element čiji je jedinstveni identifikator naveden. Na primer

```
p#pasus13 {color : blue}
```

ili

```
#pasus7 {color : blue}
```

govore da se deklaracija odnosi na pasus (element) sa identifikatorom (atributom `id`) `pasus7`.

Klase. Selektori oblika `.class` služe da se označe samo oni elementi koji pripadaju navednoj klasi. Na primer

```
p.crveni {color : red}
```

označava da se deklaracija odnosi na sve pasuse (elemente `p`) koji pripadaju klasi `crveni` (tj. imaju naveden atribut `class` sa vrednošću `crveni`).

```
.crveni {color : red}
```

U ovom primeru se deklaracija odnosi na sve elemente (ne samo pasuse) koji pripadaju klasi `crveni`.

Naglasimo još da je navođenje selektora klase specifično za HTML.

Pseudoklase i pseudoelementi. U većini slučajeva, pridruživanje stila pojedinačnim elementima zadovoljava potrebe korisnika. Međutim, u nekim slučajevima, javlja se potreba za finijom kontrolom. Na primer, element `a` označava veze u HTML dokumentima, međutim, često se želi različit prikaz za veze koje su već posećene od onih koje nisu posećene. Kako bi se pružila podrška za ovakva finija podešavanja, CSS uvodi pseudoklase i pseudoelemente. Neki od njih su i sledeći.

`:link` - označava veze koje još nisu posećene.

`:visited` - označava veze koje su posećene.

`:hover` - označava elemente iznad koji se trenutno nalazi pokazivač (najčešće miš), ali koji još nisu aktivirani (nije se „kliknulo" mišem).

`:active` - označava elemente koji su upravo aktivirani („kliknuto" je mišem na njih ili je pritisnut taster „Enter").

`:focus` - označava elemente koji su trenutno u fokusu tastature.

`:lang(L)` - označava elemente koji su na jeziku `L`.

`:first-line` - odnosi se na prvu liniju (najčešće pasusa).

`:first-letter` - odnosi se na prvo slovo (najčešće pasusa).

Tako se, na primer,

```
a:visited {color : blue}
```

delaracija odnosi na sve veze koje su već posećene iz pregledača Veba.

Selektori atributa. U nekim slučajevima je potrebno da se promeni stil samo elementima kojima je naveden neki atribut ili kojima neki atribut ima tačno određenu vrednost. Ovo se postiže selektorima oblika `selector[attr]` ili `selector[attr="value"]`. Tako se, na primer,

```
input[type="text"] {color : blue}
```

deklaracija odnosi na sve elemente `input` čiji je atribut `type` postavljen na vrednost `text`.

Ugnježđeni elementi. U nekim slučajevima, poželjno je da se promeni stil elemenata, ali samo pod uslovom da su sadržani u okviru nekog drugog, šireg elementa. To se poštiže tako što se navedu dva selektora jedan iza drugog, razdvojeni razmakom: `selector1 selector2`, čime se podrazumeva da se deklaracija odnosi na sve elemente odabrane `selector2` koji su sadržani u okviru nekog elementa odabranog selekotorom `selector1`. Tako se, na primer,

```
table.rezultati span {color : blue}
```

deklaracija odnosi na sve elemente `span` koji se nalaze u okviru tabela (elementa `table`) koje pripadaju klasi `rezultati`.

Naglasimo, da dubina ugnježđavanja može biti i veća od 2.

U nekim slučajevima se ne želi odabir svih naslednika nekog elementa, već samo njegove dece (tj. neposrednih potomaka). Tada se koristi selektorima oblika `selector1 > selector2`.

U nekim slučajevima se želi odabir elemenata samo ako neposredno dolaze posle nekog elemente (tj. ako su susedi u stablu). U tom slučaju se navodi selektor oblika `selector1 + selector2`. Tako se, na primer,

```
h1 + p { text-indent: 0 }
```

deklaracija odnosi na sve pasuse (elemente `p`) koji dolaze neposredno nakon glavnog naslova (elementa `h1`).

Svi elementi. Ukoliko se kao selektor navede `*`, deklaracija se odnosi na sve elemente u okviru dokumenta.

3.4.3 Neka CSS svojstva i njihove vrednosti

U ovom poglavlju biće navedena neka najčešće korišćena CSS svojstva i njihove najčešće korišćene vrednosti. Potpun pregled svojstava i vrednosti moguće je naći u zvaničnoj CSS specifikaciji.

Slovni likovi (Font). Naredna svojstva definišu izgled teksta tj. slovnih likova (eng. fonts) koji se koriste prilikom ispisa teksta. Sva ova svojstva se mogu primeniti na bilo koji element i automatski se nasleđuju na sve sadržane elemente.

`font-family` - Ovo svojstvo određuje slovni lik ili familiju slovnih likova koji će se koristiti prilikom ispisa teksta. Moguće je navesti precizan naziv fonta (npr. `"Times New Roman"`, `Arial`, `"Courier New"`), ime familije (npr. `Times`) ili ime vrste (npr. `serif`, `sans-serif`, `cursive`, `fantasy`, `monospace`). Moguće je navesti i više opisa u opadajućem prioritetu. Tako npr.

```
p { font-family: "New Century Schoolbook", Times, serif }
```

označava da se koristi `New Century Schoolbook` ukoliko postoji na sistemu. Ako ne postoji, koristi se bilo koji `Times` slovni lik, a ako ne postoji, koristi se bilo koji serifni[12] lik.

`font-style` - Ovo svojstvo određuje *iskošenost* teksta. Moguće vrednosti su `normal`, `italic` i `oblique`.

`font-variant` - Ovo svojstvo daje mogućnost korišćenja „MALIH-VELIKIH" slova. Moguće vrednosti su `normal` i `small-caps`.

`font-weight` - Ovo svojstvo određuje „**debljinu**" slova. Najčešće vrednosti su `normal`, `bold`, i `lighter`.

`font-size` - Ovo svojstvo određuje veličinu slova. Najčešće se vrednosti zadaju u jedinici `pt`. Tako, na primer,

```
p {font-size: 12pt}
```

određuje da će veličina teksta u pasusima biti 12. Pored ovoga, moguće je navoditi i relativne mere (u procentima). Tako, na primer,

```
p {font-size: 120%}
```

određuje da će veličina teksta biti 120% predvidjene. Moguće je i navoditi slovne oznake veličine, poput npr. `xx-small`, `medium`, `x-large` i slično.

`font` - Ovo svojstvo objedinjuje ostala svojstva navedena u ovoj sekciji i omogućava da se istovremeno postavi više različitih karakteristika slovnog lika. Na primer

```
p { font: italic bold 12pt/14pt Times, serif }
```

Boje i pozadina. Boje se mogu zadati na nekoliko načina. Najkorišćeniji su sledeći.

ime - boja se zadaje preko imena boje (npr. `red`, `yellow`, `black`, ...). Puna lista boja je prikazana na slici 3.2. Na primer:

```
p { color : red }
```

#**rrggbb** - boja se zadaje zadavanjem tri dvocifrena heksadekadna broja koji određuju redom crvenu, zelenu i plavu komponentu boje. Na primer:

```
p { color : #ff0000 }
```

rgb(r, g, b) - boja se zadaje zadavanje tri dekadna broja (između 0 i 255) koji određuju redom crvenu, zelenu i plavu komponentu boje. Na primer:

```
p { color : rgb(255, 0, 0) }
```

`color` - Ovo svojstvo određuje boju tekstualnog sadržaja elementa. Svojstvo se nasleđuje.

[12] Serifi su „postolja" na kojima „stoje" određena slova (npr. i, f, r) u nekim likovima.

Slika 3.2: Imena boja

`background-color` - Ovo svojstvo određuje boju pozadine elementa. Svojstvo se nasleđuje. Osim specifikacije boje, vrednost ovog svojstva može da bude i `transparent` čime se element pravi providnim.

`background-image` - Ovo svojstvo određuje sliku koja će biti prikazana kao pozadina elementa. Svojstvo se ne nasleđuje. Moguće je dodatno podesiti neke parametre prikaza slika (npr. poziciju, način ponavljanja). Moguće vrednosti su `none` kada se ne koristi slika ili URI slike u obliku `url(...)`. Preporučuje se da se prilikom korišćenja ovog svojstva postavi i boja pozadine (korišćenjem svojstva `background-color`). Boja se prikazuje u slučaju da slika nije dostupna, i prikazuje se na onim delovima elementa gde slika nije postavljena ili je providna.

`background-repeat` - U slučaju kada je elementu postavljena pozadinska slika (korišćenjem svojstva `background-image`), ovim svojstvom se podešava da li će slika biti prikazana samo jednom ili će biti ponavljana dok ne ispuni celu širinu i/ili visinu elementa. Moguće vrednosti su:

repeat - slika se ponavlja i horizontalno i vertikalno.

repeat-x - slika se ponavlja samo horizontalno.

repeat-y - slika se ponavlja samo vertikalno.

no-repeat - slika se ne ponavlja (prikazuje se samo jedna kopija)

`background-attachment` - U slučaju kada je (blok) elementu postavljena pozadinska slika (korišćenjem svojstva `background-image`), ovim svojstvom se određuje da li će se prilikom pomeranja sadržaja stranice (tzv. skrolovanja) slika pomerati zajedno sa sadržajem ili će ostati fiksna u odnosu na prozor i posmatrača. U prvom slučaju navodi se vrednost `scroll`, a u drugom `fixed`.

`background-position` - U slučaju kada je (blok) elementu postavljena pozadinska slika (korišćenjem svojstva `background-image`), ovim svojstvom se određuje njegova početna pozicija. Ukoliko se navedu dve vrednosti, prva određuje horizontalnu, a druga vertikalnu poziciju. Ukoliko se navede samo jedna vrednost, za drugu se podrazumeva `center`. Vrednosti mogu biti navedene na sledeći način:

procenat - Vrednost od x% poravnava tačku koja se nalazi na x% širine (dužine) slike sa tačkom koja se nalazi na x% širine (dužine) elementa. Tako npr. `0% 0%` poravnava gornja leva temena, dok `100% 100%` poravnava donja desna.

dužina - gornje levo teme slike se postavlja na tačku pomerenu za navedene vrednosti u odnosu na gornje levo teme elementa.

top, bottom - ekvivalentno `0%`, odnosno `100%` za vertikalnu poziciju.

left, right - ekvivalentno `0%`, odnosno `100%` za horizontalnu poziciju.

center - ekvivalentno `50%` za horizontalnu poziciju (ako nije navedena) ili za vertikalnu poziciju (ako je horizontalna pozicija navedena).

`background` - Ovo svojstvo omogućava da se istovremeno navede više aspekata pozadine.

Na primer, umesto:

```
body {
    background-color : #aaaaaa;
    background-image: url('background.png')
    background-repeat: repeat-x;
    background-attachment: scroll;
    background-position: right top;
}
```

moguće je koristiti:

```
body { background: #aaaaaa url('background.png') repeat-x scroll right top }
```

Tekst.

`text-indent` - Ovo svojstvo definiše uvlačenje prve linije teksta u okviru elementa. Ovo svojstvo se nasleđuje. Vrednost se navodi bilo u obliku dužine ili u obliku procenta.

`text-align` - Ovo svojstvo definiše poravnavanje teksta u okviru elementa. Svojstvo se nasleđuje. Moguće su vrednosti `left` (poravnavanje na levu stranu), `right` (poravnavanje na desnu stranu), `center` (centriran tekst) ili `justify` (tekst poravnat na obe strane).

`text-decoration` - Ovim svojstvom se mogu zadati različite dekoracije teksta. Moguće vrednosti su: `none` (tekst nema dekoracija), `underline` (tekst je podvučen), `overline` (tekst je nadvučen), `line-trough` (tekst je precrtan) i `blink` (tekst „trepće").

`letter-spacing` - razmak između slova (vrednost je dimenzija u pikselima ili nekoj drugoj jedinici),

`word-spacing` - razmak između reči (vrednost je zimenzija u pikselima ili nekoj drugoj jedinici),

`text-transform` - Ovim svojstvom se zadaju različite automatske transformacije teksta. Vrednost `uppercase` prouzrokuje da se ceo tekst prikaže velikim slovima (bez obzira na to kako je otkucan), `lowercase` prouzrokuje da se ceo tekst prikaže malim slovima, a `capitalize` da se prva slova reči prikazuju kao velika.

`vertical-align` - vertikalno poravnavanje (najčešće vrednosti su `top`, `middle` i `bottom`).

Liste.

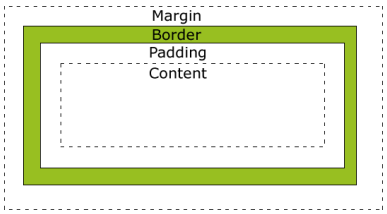
`list-style-type` - vrednost „crtice" u nenumerisanoj listi ili broja u numerisanoj listi (najčešće vrednosti su `disc`, `circle`, `square`, `lower-alpha`, `lower-roman`, `upper-alpha`, `upper-roman` itd.).

`list-style-image` - umesto „crtice" može se prikazati i neka slika koja se ovim svojstvom zadaje.

`list-style-position` - određuje da li će se crtice i brojevi biti prikazani unutar liste (tj. unutar njenog okvira) ili van nje (vrednosti ovog svojstva su `inside` i `outside`).

`list-style` - objedinjuje gore nabrojana svojstva.

„Kutije" (Box model). Svi HTML elementi mogu da se smatraju pravougaonim površinama — *kutijicama, eng. boxes.* Svaka kutijica ima svoj *sadržaj (eng. content)* i može da ima svoj *okvir (eng. border).* Okvir je razdvojen od sadržaja tzv. *punjenjem (eng. padding)*, a od okolnih elemenata *marginom (eng. margin).* Slika 3.3 prikazuje ovo.



Slika 3.3: Elementi u obliku kutijica

Korišćenjem CSS-a, moguće je prilagoditi sve ove parametre.

`width`, `height`, `box-sizing` - Ova svojstva služe za postavljanje širine i visine sadržaja elementa. Punjenje, okvir i margine podrazumevano nisu uračunate[13], ali se to može promeniti svojstvom `box-sizing`. Svojstvo `box-sizing` govori o tome da li se u širinu i visinu elementa uključuju i okvir, punjenje i margine. Podrazumevana vrednost je `content-box` i tada se računa samo širina i visina samo sadržaja (bez punjenja, okvira i margina). Sa druge strane, ako se navede vrednost `border-box` računaju se širina i visina sadržaja, punjenja i okvira (bez margina). Ovo može ponekad biti zgodno jer dizajner često zna koliko prostora na strani neki element treba da zauzme (od jedne do naspramne ivice okvira) i ako koristi `border-box` preračunavanje, tada ne mora da ručno od te vrednosti oduzima vrednosti debljine okvira i punjenje da bi postavio širinu elemenata (što bi morao da radi ako se koristi `content-box`).

[13]Napomenimo da se ovo često razlikovalo u starijim verzijma pregledača, što je stvaralo mnoge probleme.

`min-width`, `max-width`, `min-height`, `max-height` - Ovim svojstvima je moguće podesiti minimalnu ili maksimalnu dužinu tj. visinu elemenata.

`margin` - Ovim svojstvom podešava se margina. Kada ze kao vrednost navede jedna dimenzija (npr. `margin: 10px`) podrazumeva se da se ona odnosi na sve četiri margine (levu, desnu, gornju i donju). Ako se navedu dve dimenzije (npr. `margin: 10px 20px`) prva od njih se odnosi na levu i desnu, a druga na gornju i donju marginu. Ako se navedu četiri dimenzije one se odnose redom na levu, gornju, desnu i donju marginu (navode se, dakle, u pravcu kazaljke na satu). Vrednost `auto` prouzrokuje da se margine automatski podjednako rasporede što se jako često koristi u centriranju elemenata. Moguće je zasebno podešavati svaku od četiri margine i to atributima `margin-top`, `margin-right`, `margin-bottom`i `margin-left`.

`padding` - Ovim svojstvo podešava se unutrašnja margina (tj. punjenje) i ono se koristi analogno svojstvu `margin`. Pojedinačne dimenzije mogu se podešavati svojstvima `padding-top`, `padding-right`, `padding-bottom`i `padding-left`.

`border-width` - Ovim svojstvo podešva se debljina okvira (npr. `border-width: 3px`). Pojedinačni okviri (levi, gornji, desni i donji) se mogu podešavati svojstvima `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`.

`border-style` - Ovim svojstvom podešava se tip linije okvira (vrednosti su `solid`, `dashed`, `dotted` itd.). Opet se mogu podeăsvati i pojedinačni okviri i to svojstvima `border-top-style`, `border-right-style`, `border-bottom-style` i `border-left-style`.

`border-color` - Ovim svojstvima se podešava boja okvira. Opet se mogu podeăsvati i pojedinačni okviri i to svojstvima `border-top-color` i `border-right-color`, `border-bottom-color` i `border-left-color`.

`border` - Ovo svojstvo kombinuje sva tri aspekta okvira (debljinu, vrstu linije i boju). Na primer, `border: 1px solid black` kreira tanki, crni okvir, iscrtan punom linijom. Pojedinačni okviri se mogu podešavati svojstvima `border-top`, `border-right`, `border-bottom` i `border-left`.

Tabele.

`border-collapse` - ovo veoma često korišćeno svojstvo uz vrednost `collapse` prouzrokuje da se susedne ćelije „prilepe" tako da između njih nema prostora i tako da se između susednih ćelija postavlja jedinstveni okvir (u suprotnom svaka ćelija ima svoj posebni okvir).

`border`, `background-color`, `text-align`, `vertical-align`, `padding` - ova svojstva su veoma važna za stilizovanja tabela, a koriste se na potpuno isti način kao i kod svih drugih elemenata. Naglasimo da je ove atributa moguće posebno postavljati na nivou tabele (`table`)), na nivou pojedinačnih redova (`tr`) i pojedinačnih ćelija (`td`).

Selektor `tr:hover` može se koristiti za efekat promene boje reda tabele kada je pokazivač miša iznad njega. CSS3 donosi još jedan često korišćen zanimljiv

efekat: selektor `tr:nth-child(even)` se može koristiti za promene svih parnih vrsta ćelije (tako se dobijaju tzv. zebra table u kojima se vrste naizmenično boje u dve boje što doprinosi čitljivosti).

3.4.4 CSS pozicioniranje

Pozicioniranje elemenata vrši se postavljanjem njihovog svojstva `position`. Moguće su četiri vrednosti: `static`, `relative`, `absolute` i `fixed`.

`static` - Statičko pozicioniranje je podrazumevano (elementi kojima vrednost atributa `position` nije postavljena imaju statičko pozicioniranje). To podrazumeva da se oni uklapaju u takozvani *normalni tok* (engl. normal flow). Elementi nivoa bloka se postavljaju jedan ispod drugog, dok se linijski elementi slažu jedan uz drugi.

`relative` - Relativno pozicioniranje omogućava da se elementi pomere u odnosu na svoju statičku poziciju, tj. da se izmeste iz normalnog toka, pri čemu to nikako ne utiče na pozicioniranje ostalih elemenata. Koordinate se zadaju svojstvima `top`, `left`, `bottom` i `right`. Na primer,

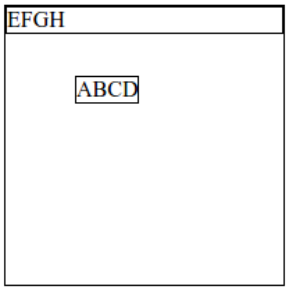
```
p {
  position: relative;
  top: -10px;
  left: 20px;
}
```

prouzrokuje da se svi pasusi pomere 10 piksela gore i 20 piksela desno u odnosu na njihovu normalnu poziciju. Promenom pozicioniranja elementi mogu i preklopiti, a koji će biti iznad, a koji ispod određuje se svojstvom `z-index` – veće vrednosti određuju elemente koji će biti iznad tj. koji će se videti. Naglasimo i da se relativno pozicioniranje često koristi u kombinaciji sa apsolutnim i da se `position: relative` bez promene koordinata često koristi kao deo apsolutnog pozicioniranja koje sledeće opisujemo.

`absolute` - Apsolutno pozicioniranje dovodi do toga da se element pozicionira u okviru nekog šireg elementa i to tako što se zadaju njegove koordinate u odnosu na taj širi element. Po definiciji, to je najbliži element koji ga sadrži a pozicioniran je nekako tj. ima vrednost svojstva `position` različitu od podrazumevane vrednosti `static`. Ako takav element ne postoji, apsolutno pozicioniranje se vrši u odnosu na element `body`. Da bi se naglasilo da će se neki element koristiti kao element u kome će se sadržaj pozicionirati apsolutno, njemu se obično samo vrednost svojstva `position` postavi na `relative`. Apsolutno pozicionirani element se izbacuje iz normalnog toka i ostali elementi se pozicioniraju kao da on ne postoji. Koordinate apsolutno pozicioniranih elemenata se opet zadaju svojstvima `top`, `left`, `bottom` i `right`. Razmotrimo naredni primer:

```
<div id='container'>
  <div id='content'>ABCD</div>
  <div>EFGH</div>
</div>
```

```
div {
  border: 1px solid black;
}

#container {
  position: relative;
  width: 500px;
  height: 500px;
  top: 50; left: 70;
}

#content {
  position: absolute;
  top: 50px; left: 50px;
}
```

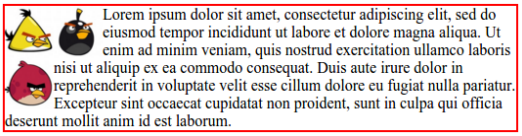


Element `#content` se pozicionira tako da mu je gornji levi ugao 50 piksela ispod i 70 piksela levo od gornjeg levog ugla elementa `container`. Element u kome piše EFGH nalazi se na vrhu elementa `#container` jer je element `#content` izbačen iz normalnog toka.

`fixed` - Fiksno pozicioniranje je donekle slično apsolutnom, ali se element pozicionira u odnosu na ekran pregledača i ne pomera se prilikom skrolovanja stranice. Najčešće se koristi za fiksiranje naslova ili potpisa stranice. Pošto se i fiksno pozicionirani elementi ibacuju iz normalnog toka, ako se želi obezbediti da ne preklapaju druge elemente, potrebno je tim drugim elementima postaviti odgovarajuće margine.

Neki elementi mogu da budu „lebdeći", tj. da se podese tako da se pomeraju u levi ili desni kraj okružujućeg elementa, a da se ostali sadržaj „obmotava" oko njih (slično kao što se tekst obmotava oko slika u novinama). Ovo se postiže svojstvom `float`, čije su vrednosti `left` i `right`. Kada dva susedna elementa imaju istu vrednost svojstva `float`, oni se pozicioniraju jedan do drugoga (na primer, ako je vrednost `left` prvi element će se pomeriti nalevo, do početka okružujućeg elementa, a drugi će odlebdeti nalevo, ali samo do kraja prvog lebdećeg elementa). Ako se želi da se lebdeći element složi ispod, a ne pored drugog lebdećeg elementa, onda je potrebno navesti mu svojstvo `clear`. Na primer:

```
<div>
   <img src='1.jpg' style='float: left;' />
   <img src='2.jpg' style='float: left;' />
   <img src='3.jpg' style='float: left; clear: left;' />
   Lorem ipsum dolor sit amet, ...
</div>
```

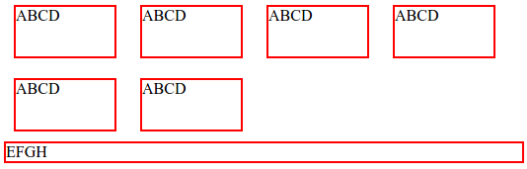


Donekle u vezi sa pozicioniranjem je i način prikaza elemenata tj. svojstvo cssdisplay. Podrazumevane vrednosti su `inline` za linijske elemente i `block` za blok elemente. Podrazumevani način prikaza se može promeniti (na primer, elementi `li` se podrazumevano prikazuju kao blok elementi, ali pošto se često koriste u menijima prikaz im se menja na `inline` da bi se prikazivali jedan pored drugog). Vrednost `none` uzrokuje da se element sakrije tj. da se „izbaci" iz stranice i ne prikazuje ni na koji način (ostali elementi se pozicioniraju kao i da ga nema, za razliku od `visibility: hidden` koje sakriva element, ali ne utiče na elemente oko njega). Na kraju, vrednost `inline-block` kombinuje linijski i blok način prikaza tj. elementi se slažu jedan pored drugoga (kao u slučaju linijskog prikaza), ali mogu da imaju fiksiranu širinu i dužinu (što je slučaj sa blok elementima). Ovo se obično koristi da se napravi matrica kutijica koje se redom slažu, lepo poravnate. Na primer,

```
div {
  border: 2px solid red;
}
.floating-box {
  display: inline-block;
  width: 100px; height: 50px;
  margin: 10px;
}
```

```
<div class='floating-box'>ABCD</div>
<div class='floating-box'>ABCD</div>
<div class='floating-box'>ABCD</div>
<div class='floating-box'>ABCD</div>
<div class='floating-box'>ABCD</div>
<div class='floating-box'>ABCD</div>
<div>EFGH</div>
```

se prikazuje kao na sledećoj slici.



3.4.5 Umetanje CSS opisa u HTML dokumente

Autori HTML dokumenata mogu umetnuti CSS opise u dokumente na nekoliko različitih načina.

Opisi na nivou elementa (atribut `style`). Jedan od načina da se pojedinačnom elementu promeni stil je da mu se navede atribut `style` čiji je sadržaj niz CSS svojstava i vrednosti (selektori se ne navode) koje određuju izgled tog pojedinačnog elementa. Na primer,

```
<p style="color:red; margin-left:10px">Ovo je pasus</p>
```

S obzirom da se na ovaj način gube prednosti koje donosi razdvajanje opisa strukture i vizuelne prezentacije jer opisi prezentacije bivaju razasuti po celom dokumentu i isprepletani sa opisom strukture, korišćenje ovog načina se preporučuje samo u izuzetnim slučajevima.

Opisi na nivou dokumenta (element `style`). Naredno mesto na kome se može navesti CSS opis je element `style` u okviru zaglavlja dokumenta (elementa `head`). Ukoliko sadrži opis na jeziku CSS element `style` bi trebalo da ima atribut `type` sa vrednošću `text/css`. CSS opis se navodi kao sadržaj elementa `style`. Na primer

```
<head>
   <style type="text/css">p {color : blue}</style>
   ...
</head>
```

Opis naveden u zaglavlju dokumenta se odnosi isključivo na taj dokument. Mehanizam CSS selektora se koristi kako bi se odredilo na koje tačno elemente dokumenta se pojedinačna navedena CSS svojstva i njihove vrednosti navode.

Spoljašnji opisi. Način umetanja CSS opisa u dokumente koji se najčešće koristi i koji nudi najveće pogodnosti je kreiranje posebnih datoteka koje sadrže samo CSS opis (najčešće sa ekstenzijom `.css`) i zatim pozivanje na te datoteke iz samih dokumenata. Pozivanje se najšešće vrši korišćenjem elementa `link` u zaglavlju dokumenta (elementu `head`). U ovom slučaju kao atributi elementa link moraju se navesti `rel` sa vrednošću `stylesheet`, `type` sa vrednošću `text/css` i atribut `href` u kome se navodi URI spoljašnjeg dokumenta (najčešće samo ime `.css` datoteke). Na primer

```
<link rel="stylesheet" type="text/css" href="stil.css" />
```

Jezik CSS dopušta da se direktivom `@import` u jedan stilski list kompetno uveze neki drugi stilski list, tako da je ovo još jedan od mogućih načina korišćenja spoljašnjih datoteka sa stilskim opisima.

```
<style type="text/css"> @import("stil.css") </style>
```

Ovaj način je naročito pogodan kada se isti stilski opis uključuje iz više različitih HTML dokumenata (najčešće celog Veb sajta). Ovim se postiže to da se vizuelna prezentacija celog sajta može promeniti samo izmenom jedne `.css` datoteke.

Nasleđivanje stilskih opisa. Neka svojstva se nasleđuju kroz celo stablo dokumenta, tj. u nekim slučajevima kada se deklaracija stila pridruži nekom elementu, istu deklaraciju automatski nasleđuju i svi elementi sadržani u tom

elementu. Na primer, ako se podesi boja slova tela dokumenta, svi sadržani elementi nasleđuju navedenu boju.

```
body {color : red}
```

Neka svojstva se ne na sleđuju. Na primer, ako se podesi margina telu dokumenta, to ne znači da će i svi sadržani elementi imati istu marginu.

Kaskada stilskih opisa. Stilski opis za neki element može biti istovremeno naveden i na nekoliko različitih mesta.

Stil autora dokumenta. Autori dokumenta mogu svojim dokumentima da pridruže stilske opise na tri prethodno opisana načina.

Stil korisnika. Korisnik može svojim podešavanjima da utiče na stil i postoji stilski opis na nivou korisnika.

Stil korisničkog agenta. Ukoliko autor dokumenta ni korisnik ne navedu stilski opis za neki element, onda se podrazumeva podrazumevani stil tog elementa zadat na nivou korisničkog agenta (tj. pregledača Veba).

Različiti opisi za neki element se „sabiraju" tj. kombinuju kako bi se dobio konačan stil prikaza elementa. U slučaju da dođe do konflikta, prednost se daje stilskim listovima autora, zatim stilskim listovima korisnika, dok podrazumevani stilski listovi korisničkih agenata imaju najmanji prioritet. U slučaju da do konflikta dođe u okviru stilskog lista autora, prioritet imaju deklaracije navedene na nivou elementa, zatim deklaracije navedene u na nivou dokumenta i tek na kraju deklaracije navedene u spoljašnjim listovima.

3.5 MathML

Problem kodiranja matematičkog sadržaja primerenih za računarusku i elektronsku komunikaciju stariji je i opštiji od postojanja Interneta i Veba. Međutim, s obzirom da se Veb razvio u jedan od najznačajnijih izvora informacija današnjice problem predstavljanja matematičkih sadržaja na Veb u mašinski čitljivom obliku postaje izuzetno značajan. Jezik HTML ne nudi velike mogućnosti za predstavljanje matematičkog sadržaja. Podržano je predstavljanje eksponenata i indeksa kroz elemente `sub` i `sup`, predstavljanje izraza sa osnovnim računskim operacijama, moguće je korišćenje simbola drevnih alfabeta kroz Unicode skup karaktera (npr. moguće je koristiti grčka slova) i slično. Međutim iole kompleksnije matematičke izraze nije moguće direktno kodirati kroz jezik HTML. Rešenje koje se i danas prevashodno koristi je da se matematički izrazi predstave u slikama i da se kao slike uključe u HTML dokumente. Slike se obično kreiraju specijalizovanim alatima, najčešće verzijama LaTeX-a. Problemi ovog pristupa su evidentni: ažuriranje ovakvih dokumenata je komplikovano, nije moguće indeksirati i pretraživati formule na osnovu njihovog sadržaja, slike su relativno veliki objekti i zauzimaju mrežne resurse prilikom preuzimanja i slično.

W3C konzorcijum prepoznaje ovaj problem i predlaže standardni format za predstavljanje i razmenu matematičkog sadržaja koji je, između ostalog, prilagođen za objavljivanje na Vebu. U okviru konzorcijuma oformljena je *W3C Math Working Group* koja je zadužena za razvoj ovog formata. Odlučeno je da format bude zasnovan na XML-u, nazvan je *Mathematical Markup Langulage*

(MathML). Razvoj je započeo još davne 1998. godine i tekuća verzija standarda je MathML 2.0, pri čemu se trenutno radi na izradi nove verzije MathML 3.0. Prilikom koncipiranja MathML-a navedeni su naredni osnovni ciljevi.

- MathML omogućava zapis matematičkih sadržaja za potrebe učenja matematike, naučne komunikacije na svim nivoima, itd.
- MathML omogućava kodiranje kako sintakse, tj. vizuelne prezentacije matematičkog sadržaja tako i semantike tj. značenja sadržaja.
- Moguće je (dvosmerno) konvertovanje između MathML-a i ostalih matematičkih formata, kako na nivou sintakse matematičkih izraza, tako i na nivou njihove semantike. Neki od takvih formata su i
  - formati sistema za računarsku algebru (engl. computer algebra systems),
  - jezici za pripremu za štampu matematičkih dokumenata (pre svega LaTeX),
  - specijalizovani jezici za opis matematičkog sadržaja.

  Moguće je da prebacivanjem sadržaja između formata dođe do izvesnog gubitka informacija.
- MathML može da se prikaže:
  - u uobičajenom grafičkom obliku,
  - u obliku tekstualnog prikaza (ASCII) pogodnog za prikaz na terminalima,
  - u štampanom, uključujući i Brajevu azbuku.
- MathML može da se kombinuje sa XHTML-om i pregledači Veba omogućavaju kvalitetan prikaz matematičkog sadržaja u okviru Veb stranica.
- MathML sadržaj umetnut u Veb stranice može da interaguje sa korisnikom (npr. da reaguje na akcije mišem).

Dakle, MathML je dizajniran tako da bude fleksibilan, proširiv, da je moguća interakcija sa mnoštvom matematičkog softvera i da ima mogućnost proizvodnje visokokvalitetnog predstavljanja na različitim medijima. U idealnom slučaju, MathML bi trebalo da postane osnovni format za predstavljanje celokupnog matematičkog sadržaja i trebalo bi da bude nezaobilazan deo svakog matematičkog softverskog alata. Trebalo bi da bude moguće da se određeni matematički sadržaj u pregledaču Veba ili nekom drugom matematičkom alatu označi i ako je potrebno konvertuje u MathML format, zatim da se takav MathML zapis umetne u neki drugi alat ili u Veb stranicu i slično.

Naravno, jezik za obeležavanje koji nastoji da ispuni sve ovo mora da bude komplekasan. Puna snaga MathML-a može da se vidi tek ako se izradi dovoljan broj aplikacija koje podržavaj MathML format. Na žalost, u trenutku pisanja ovog teksta može se reći da MathML nije uspeo da se dovoljno snažno pozicionira i ispuni sve predviđene ciljeve. Posle punih dvanaest godina od početka njegovog razvoja MathML je i dalje dolazeća tehnologija koja je relativno slabo podržana

u okviru vodećih matematičkih alata. Što se tiče podrške u okviru Veb pregledača, pregledači Firefox, Opera i Safari su u mogućnosti da prikažu MathML, dok Microsoft Internet Explorer tu mogućnost dobija nakon instalacije određenih dodataka (npr. MathPlayer).

Postoje dva različita oblika MathML-a:

Presentation MathML - služi isključivo za predstavljanje sintakse (vizuelne prezentacije) matematičkih izraza,

Content MathML - služi za predstavljanje predstavljanje semantike (značenja izraza).

Na sledećem primeru MathML-a će biti prikazana razlika između Content MathML-a i Presentation MathML-a. Predstavimo izraz:

$$x^2 + 4x + 4 = 0$$

Ova jednačina u Presentation MathML-u može biti opisana sa:

```
<mrow>
  <mrow>
    <msup>
      <mi>x</mi>
      <mn>2</mn>
    </msup>
    <mo>+</mo>
    <mrow>
      <mn>4</mn>
      <mo>&InvisibleTimes;</mo>
      <mi>x</mi>
    </mrow>
    <mo>+</mo>
    <mn>4</mn>
  </mrow>
  <mo>=</mo>
  <mn>0</mn>
</mrow>
```

Element `mo` služi za predstavljanje operatora, `mi` za predstavljanje identifikatora dok element `mn` služi za predstavljanje brojeva. Slično HTML elementu `sup`, MathML element `msup` opisuje eksponent. Element `mrow` služi za grupisanje drugih elemenata.

Prethodna jednačina u Content MathML-u može biti opisana sa:

```
<apply>
  <eq />
  <apply>
    <plus />
    <apply>
      <power />
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <apply>
      <times />
      <cn>4</cn>
      <ci>x</ci>
    </apply>
    <cn>4</cn>
  </apply>
  <ci>0</ci>
</apply>
```

Primetimo da ovaj opis prilično odgovara zapisu izraza u prefiksnom obliku. Element `ci` služi za predstavljanje identifikatora, a element `cn` za predstavljanje brojeva. Element `apply` služi da označi prmenu određene matematičke operacije, dok elementi `plus`, `times`, `power` i `eq` označavaju redom sabiranje, množenje, stepenovanje i jednakost.

MathML podržava elementarnu i do određenog nivoa naprednu matematičku simboliku (podržan je tzv. K12 nivo obrazovanja, tj. matematika koja se u SAD uči tokom prvih 12 godina školovanja).

Ukoliko se MathML matematički sadržaj čuva u zasebnim XML datotekama, potrebno je koristiti DTD:

```
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
  "http://www.w3.org/Math/DTD/mathml2/mathml2.dtd"
>
```

MathML se može umetnuti isključivo u XHTML dokumente pri čemu se za kombinovani tip dokumenta koristi poseban DTD:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
   "http://www.w3.org/Math/DTD/mathml2/xhtml-math11-f.dtd">
```

Dokumentima je potrebno priključiti i posebnu instrukciju procesiranja[14]:

```
<?xml-stylesheet type="text/xsl"
                 href="http://www.w3.org/Math/XSL/mathml.xsl"?>
```

Na primer:

[14] Ukoliko se želi mogućnost čitanja dokumenata iz lokala, bez Internet konekcije, potrebno je na lokalni računar presnimiti `mathml.xsl`, `pmathml.xsl`, `ctop.xsl` i `pmathmlcss.xsl` datoteke sa adrese `http://www.w3.org/Math/XSL/`.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://www.w3.org/Math/XSL/mathml.xsl"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN"
          "http://www.w3.org/Math/DTD/mathml2/xhtml-math11-f.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Presentation MathML - primer</title>
  </head>
  <body>
    <p>Ovo što sledi je jedan primer matematičke formule:
      <math xmlns="http://www.w3.org/1998/Math/MathML">
      <msqrt> <mrow> <mi>a</mi> <mo>+</mo> <mi>b</mi> </mrow> </msqrt>
      </math>
    </p>
  </body>
</html>
```

3.6 SVG

Scalable Vector Graphics (SVG) je XML format za zapis dvodimenzionalne vektorske grafike. Format je razvijen i podržan od strane W3C konzorcijuma. Većina današnjih pregledača Veba (s izuzetkom Microsoft Internet Explorer-a) imaju mogućnost direktnog prikazivanja slika u SVG formatu. Za razliku od rasterskih slika, vektorske slike su skalabilne tj. mogu se bez gubitka kvaliteta prikazivati u različitim rezolucijama i na različitim nivoima uvećanja (eng. zoom).

SVG definiše elemente za opis osnovnih geometrijskih primitiva: `rect` za pravougaonike, `circle` za krugove, `ellipse` za elipse, `line` za duži, `polygon` za izlomljene linije, `polygon` za mnogouglove i `path` za putanje proizvoljnog oblika (koje kombinuju Bezijeove krive i duži). Ovi elementi sadrže atribute koji koordinatama opisuju položaj i dimenzije ovih objekata. Detalji vizuelne prezentacije (npr. boje, debljine linija) se mogu podešavati bilo korišćenjem posebnih, za to namenjenih atributa, bilo korišćenjem CSS-a.

Na primer:

```
<rect width="300" height="100"
      style="fill: rgb(0,0,255); stroke-width: 1; stroke: rgb(0,0,0)" />
```

Ovim je opisan pravougaonik dimenzija 300×100, koji je ispunjen plavom bojom, a uokviren crnim okvirom debljine 1.

SVG slike se obično smeštaju u zasebne datoteke (sa ekstenzijom `.svg` ili `.xml`). Datoteke koriste SVG DTD:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Na primer:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
 <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red" />
</svg>
```

SVG slike se mogu uključiti u obične HTML dokumente korišćenjem `object` elementa. Na primer:

```
<p>Ovo što sledi je primer jedne slike: <br />
   <object data="krug.svg" type="image/svg+xml">
     Vaš pregledač ne podržava SVG.
   </object>
</p>
```

3.7 SMIL

Synchronized Multimedia Integration Language (SMIL) (izgovara se „smajl”) je standard predviđen za opis audiovizuelnih prezentacija i animacija. U ove svrhe, u današnje vreme se prevashodno koriste komercijalni formati i alati (pre svega Adobe Flash). Za razliku od ovoga, SMIL je otvoren, standardizovan format koji je W3C preporuka. Navedimo neke od osobina SMIL-a.

- Može se koristiti za kreiranje prezentacija sa slajdovima (Internet odgovor na Power Point).
- Prezentacije mogu da kombinuju različite tipove multimedijalnog sadržaja (tekst, video, audio, itd.)
- Moguće je istovremeno prikazivanje različitih multimedijalnih datoteka koje dolaze sa različitih Veb servera.
- Prezentacije mogu da sadrže veze sa drugim SMIL prezentacijama.
- Prezentacije mogu da sadrže kontrolnu dugmad (stop, start, next, ...)
- Moguće je definisanje redosleda prikazivanja elemenata prezentacije kao i dužine trajanja njihovog prikazivanja.
- Moguće je definisanje pozicije na kojoj će se prikazivati odgovarajući elementi prezentacije.

Microsoft Internet Explorer ima mogućnost direktnog prikazivanja SMIL prezentacija, dok drugi pregledači zahtevaju instalaciju dodatnih alata.

SMIL je XML format i datoteke koje sadrže SMIL prezentacije bi trebalo da su tipa:

```
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
          "http://www.w3.org/2001/SMIL20/SMIL20.dtd">
```

Na primer:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
          "http://www.w3.org/2001/SMIL20/SMIL20.dtd">
<smil>
  <body>
    <seq repeatCount="indefinite">
      <img src="image1.jpg" dur="3s" />
      <img src="image2.jpg" dur="3s" />
    </seq>
  </body>
</smil>
```

U prethodnom primeru opisuje se prezentacija koja beskonačno smenjuje dve slike (`image1.jpg` i `image2.jpg`) pri čemu svaku sliku prikazuje tačno 3 sekunde.

SMIL prezentacije je moguće umetati direktno u XHTML dokumente (pri čemu se tada koristi poseban kombinovani DTD). Predviđeno je, takođe, da se SMIL koristi sa SVG u cilju izrade animacija.

Glava 4

Klijentski skriptovi - DOM, Javascript, Apleti, ActiveX

Razvojem Veba, uočena je mogućnost da se Veb stranice osim kao izvor informacija koriste i kao jednostavni aplikativni programi. Za razliku od statičkih Veb stranica čiji se sadržaj jednostavno prikazuje u okviru pregledača, dinamičke Veb stranice nude mogućnost interakcije između stranice i korisnika koji je pregleda.

Postoji nekoliko različitih načina da se kreiraju stranice koje sadrže dinamički sadržaj.

Klijentski skript jezici. Jedan od pravaca razvoja bazira se na razvoju specijalizovanih programskih jezika — *klijentskih skript jezika*, koji omogućavaju da se izvorni programski kôd umetne u HTML Veb stranice, i zatim interpretira i izvršava u okviru Veb pregledača. Jezici ovog tipa su JavaScript (kasnije standardizovan kao ECMAScript), JScript, Visual Basic Script, ... Klijentski skriptovi mogu da reaguju na akcije korisnika (pokrete miša, unos sa tastature, ...) i da na osnovu toga menjaju i prilagođavaju sadržaj Veb stranice u koju su umetnuti. Kako bi ovo moglo da se realizuje, neophodno je da postoji način da se iz skript jezika pristupi elementima stranice i u ovom cilju razvijen je *objektni model dokumenta (DOM)*. DOM omogućava da skriptovi vide Veb stranicu u koju su umetnuti kao hijerarhiju objekata koji se odlikuju svojim svojstvima koje je moguće menjati i metodama koje je moguće pozivati, a sve u cilju dinamičke promene Veb stranice. Kombinovanje HTML-a, CSS-a sa JavaScript-om korišćenjem DOM-a, ponekad se naziva *dinamičkih HTML-om* (eng. Dynamic HTML — DHTML).

Aktivni objekti. Drugi pristup ubacivanja interaktivnosti u Veb stranice je ubacivanje aktivnih objekata. Aktivni objekti dolaze u obliku kompiliranog koda koji se onda izvršava u okviru pregledača, ali uz podršku eksternog softvera koji taj kôd izvršava. Najčešći predstavnici ovog pristupa su *apleti (eng. applets)* i *ActiveX kontrole (eng. ActiveX controls)*. Aktivni objekti ugrađeni u Veb stranice predstavljaju zasebne celine i ne postoji direktna interakcija između njih i Veb stranice u koju su umetnuti — interakciju je moguće postići samo dodatnim korišćenjem klijentskih skript jezika.

Java apleti se programiraju u programskom jeziku Java, kompiliraju se do nivoa tzv. bajt koda (eng. byte code). Bajt kod se obično snima u `.class` datoteke ili pakuje u `.jar` arhive i u tom obliku se dostavlja na klijent. Na klijentu se zahteva postojanje interpretatora java bajt koda (tzv. java runtime environment — JRE) i dodatka (eng. plugin) za razgledač koji omogućava da se JRE pokrene i rezultat njegovog rada prikaže u okviru pregledača, u okviru same Veb stranice koja aplet uključuje. Ova vrsta softvera razvijena je za širok dijapazon operativnih sistema i pregledača, pa se softver programiran u jeziku Java, pa samim tim i Java apleti, smatra izrazito prenosivim. Pored Jave, apleti se mogu kreirati i u nekim drugim programskim jezicima (npr. Python).

ActiveX kontrole su tesno vezane za kompaniju Microsoft, operativni sistem Windows i pregledač Internet Explorer. Ove kontrole se programiraju korišćenjem Microsoft-ovih razvojnih alata, i moguće ih je pisati na različitim programskim jezicima (C++, C#, VisualBasic). Za razliku od Java apleta koji se mogu izvršavati na velikom broju različih platformi, ActiveX kontrole nisu portabilne i stranice koje sadrže ActiveX kontrole nije moguće u prikazati ukoliko klijent koristi okružnje različito od Microsoft-ovog, što ozbiljno smanjuje upotrebljivost ove tehnologije[1]. ActiveX kontrole Veb programima nude veće mogućnosti korišćenja usluga operativnog sistema i sa jedne strane predstavljaju moćniju programersku alatku, dok sa druge strane predstavljaju veću opasnost po bezbednost.

4.1 Objektni model dokumenta - DOM

Objektni model dokumenta (eng. Document Object Model — DOM) je interfejs koji omogućava programima tj. skriptovima da dinamički pristupe i izmene sadržaj, strukturu ili stil Veb dokumenta. DOM je nezavistan od jezika iz kojega se koristi i platforme na kojoj se koristi. U okviru DOM, elementi dokumenta se predstavljaju objektima (eng. objects) koji imaju svoja svojstva (eng. properties) i metode (eng. methods). Promenom vrednosti svojstava objekta i pozivom metoda nad objektima, menja se sadržaj, struktura ili stil dokumenta. Dakle, DOM obezbeđuje uniforman način da se iz bilo kojeg skript jezika pristupi, promeni, doda ili obriše proizvoljni HTML element.

DOM je nastao još 1990-tih, u vreme ratova pregledača. Godine 1996., kompanija Netscape u okviru njihovog pregledača ugrađuje podršku za prvu verziju jezika JavaScript koja Veb stranice proširuje mogućnošću interakcije na klijentskoj strani. Istovremeno Microsoft u Internet Explorer 3.0 ugrađuje podršku za jezik JScript, zasnovan na Nestcape-ovom jeziku JavaScript. Način na koji ova dva jezika komuniciraju sa okružujućim HTML dokumentom predstavio je osnovu za definisanje DOM. DOM iz ovog perioda se danas označava kao *„DOM Level 0"* ili *„Legacy DOM"*. Ovaj model nije zvanično standardizovan, ali je delimično opisan u okviru specifikacije jezika HTML 4. Legacy DOM omogućava pristup samo određenim elementima HTML dokumenta.

Godine, 1997. sa novijim verzijama Netscape Navigator i Internet Explorer pregledača, javila se mnogo bolja podrška za dinamički HTML, što je zahte-

[1]Na primer, autor ovog teksta je promenio svoju banku jer je bazirala Veb interfejs svog informacionog sistema na ActiveX kontrolama i time nametnula potrebu da se koristi isključivo jedan operativni sistem i razgledač Veba, što autor smatra duboko neprofesionalnom i veoma lošom poslovnom praksom.

valo proširenje objektnog modela dokumenta. Dodana je mogućnost pristupa većini HTML elemenata, mogućnost kontrolisanja izgleda stranica modifikovanjem CSS svojstava od strane skriptova, bogatiji model događaja za interakciju sa korisnikom i slično. Na nesreću, u jeku ratova pregledača, obe kompanije nezavisno vrše proširenja rudimentarnog Legacy DOM i time narušavaju kolikotoliko postojeću sinhronizovanost. Ove, međusobno različite verzije DOM danas se nazivaju *„Intermediate DOM"*.

Uvidevši probleme nesinhronizovanog razvoja, pod okriljem W3C, krajem 1990-tih započinje rad na standardizaciji klijentskih skript jezika, a nakon toga i DOM. Definiše se standard ECMAScript koji biva usvojen i u okviru JavaScript i JScript jezika. Nakon ovoga, 1998. definiše se standardna verzija DOM, poznata kao *DOM Level 1*. Ovaj standard, definiše kompletan model za celokupne HTML i XML dokumente i omogućava sredstva kojima skriptovi mogu dinamički da izmene bilo koji deo dokumenta. Ovaj model se dalje proširuje novim mogućnostima i novom funkcionalnošću i 2000., definiše se *DOM Level 2*, a nakon toga, 2004. godine i *DOM Level 3*.

DOM predstavlja sliku HTML/XML dokumenta u memoriji, u obliku drveta čiji su čvorovi DOM objekti. Standardizacijom HTML-a i DOM-a, postalo je jasno da prva komponenta svakog pregledača mora biti raščlanjavač (eng. parser) koji čita tekst HTML/XML datoteke i u memoriji ga predstavlja u obliku DOM drveta. Ove komponente se nazivaju *raspoređivačkim mašinama (eng. layout engine)* i počinju da se razvijaju u okviru zasebnih projekata, nezavisno od samih pregledača. Najpoznatiji softver ovog tipa danas predstavlja *Trident/MSHTML* koji koristi Internet Explorer, ali koji se koristi i za prikaz sadržaja (npr. help) u okviru Windows operativnog sistema, *Presto* koji koristi Opera, *Webkit* koji koriste Safari, Google Chrome, Google Android, Adobe Dreamweaver, itd, *Gecko* koji koriste svi Mozilla alati (Firefox, Sea Monkey, itd.), itd.

U okviru DOM, svakom delu dokumenta pridružen je zaseban DOM objekat. Objekti imaju svoja svojstva (atribute) i metode kojima se obično pristupa korišćenjem sintakste `objekat.svojstvo` i `objekat.metod(parametri)`. Svaki DOM objekat ima svoj tip, pri čemu tip objekta određuje svojstva i metode koje ga odlikuju. Tipovi, pa samim tim i svojstva i metodi su određeni tzv. interfejsima (eng. interface). Interfejsi su organizovani u hijararhiju nasleđivanja. Na primer, svi DOM objekti su čvorovi tzv. DOM drveta i implementiraju interfejs `Node` koji nudi osnovne operacije za rad sa drvetom (npr. metod `appendChild(c)` kojim se dodaje čvor u drvo). Međutim, objekti koji odgovaraju elementima dokumenta implementiraju interfejs `Element` koji nasleđuje interfejs `Node` i pored osnovne funkcionalnosti interfejsa `Node` nudi dodatnu funkcionalnost karakterističnu samo za rad sa elementima dokumenta (npr. metodu `setAttributeNode(attr)` kojim se ovom elementu postavlja vrednost nekog atributa). U okviru standarda, interfejsi su specifikovani korišćenjem jezika Interface Definition Language (IDL). Npr:

```
interface NodeList {
  Node                            item(in unsigned long index);
  readonly attribute unsigned long  length;
};
```

Ovaj interfejs opisuje objekat koji predstavlja listu čvorova. Ovaj objekat ima celobrojno svojstvo `length` koje samo može samo da se čita, kao i metod `item` kojim se vraća čvor liste na poziciji `index`. Važno je naglasiti da pregledači tj. njihove raspoređivačke mašine ponekad značajno proširuju standardnom de-

finisane interfejse i unose širu funkcionalnost.

DOM se može posmatrati na 3 nivoa:

- Core DOM - standardni model za bilo koji struktuirani dokument
- XML DOM - standardni model XML dokumenata
- HTML DOM - standardni model HTML dokumenata

Core DOM definiše interfejse karakteristične za sve dokumente (npr. interfejs `Node` koji je zajednički za sve čvorove DOM drveta), XML DOM definiše proširenje ovih interfejsa potrebna za predstavljanje strukture XML dokumenata (npr. interfejs `CDATASection` kojim se predstavlja `CDATA` sekcija XML dokumenta), dok HTML DOM definiše interfejse potrebne za predstavljanje strukture HTML dokumenata (npr. interfejse `HTMLDocument` kojim se predstavlja HTML dokument ili npr. `HTMLImageElement` kojim se predstavljaju elementi `img` HTML dokumenta).

4.1.1 Core DOM

Započnimo predstavljanje DOM prikazom Core modela (zajedničkog i za HTML i XML dokumente).

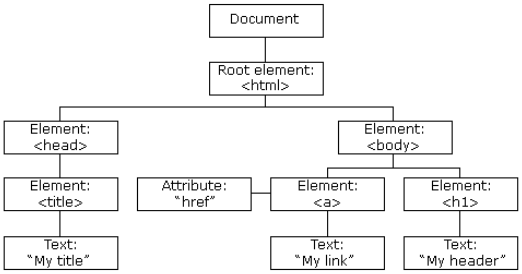
DOM drvo DOM objekti su međusobno povezani i čine strukturu drveta. Svaki objekat predstavlja jedan čvor drveta, pri čemu postoje različite vrste čvorova.

- Ceo dokument je predstavljen čvorom dokumenta.
- Svaki element je predstavljen čvorom elementa.
- Tekst u okviru elemenata je predstavljem čvorom teksta.
- Svaki atribut je predstavljen čvorom atributa.
- Komentari su predstavljeni čvorom komentara.

Prikažimo DOM drvo na jednom jednostavnom primeru.

```
<html>
  <head>
    <title>My title</title>
  </head>
  <body>
    <a href="page.html">My link</a>
    <h1>My header</h1>
  </body>
</html>
```

Primer drveta koje odgovara prethodnom HTML dokumentu dat je na narednoj slici.



Čvorovi u drvetu su u međusobnim odnosima. Jedan čvor u odnosu na drugi može da bude *roditelj (eng. parent)*, *dete (eng. child)* ili *(rođeni) brat (eng. sibling)*[2].

- U drvetu, postoji jedinstveni čvor koji se označava kao *koren (eng. root)*.
- Svaki čvor osim korena ima jedinstveno određen čvor roditelja.
- Svaki čvor može da ima proizvoljan broj dece. Lista dece svakog čvora je uređena.
- List je čvor bez dece.
- Braća su čvorovi koji imaju istog roditelja.

Na primer, u drvetu prikazanom na prethodnoj slici, čvor `<html>` je koreni čvor — on nema roditelja. Njegova deca su čvorovi `<head>` i `<body>`, koji su međusobno braća.

Interfejs `Node` Svaki čvor poseduje svojstva kojima se pristupa osnovnim podacima o čvoru.

- `nodeType` predstavlja tip čvora (npr. 1 - element, 2 - atribut, 3 - tekst, 8 - komentar, 9 - dokument).
- `nodeName` predstavlja ime čvora. Ovo svojstvo nije moguće menjati. Ime čvora elementa odgovara imenu elementa, čvora atributa imenu atributa, kod čvora teksta fiksirana je vrednost `#text`, a kod čvora dokumenta vrednost `#document`.
- `nodeValue` predstavlja vrednost čvora. Ova vrednost nije definisana za čvorove elemenata, dok kod čvora atributa ona predstavlja vrednost atributa, a kod čvora teksta predstavlja sam tekst.

Pomenimo još veoma često korišćeno, ali nestandardno svojstvo `innerHTML` koje za svaki čvor sadrži HTML kôd koji ga opisuje (uključujući i njegovu decu tj. naslednike).

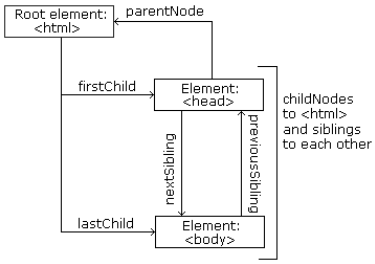
Celom dokumentu odgovara čvor kome se može pristupiti sa `document`. Korenom čvoru dokumenta može se pristupiti sa `document.documentElement`.

Svaki čvor poseduje odgovarajuća svojstva i metode za navigaciju kroz DOM drvo.

[2]Engleski termin *sibling* podrazumeva rođenu braću i sestre. S obzirom da u srpskom jeziku ne postoji odgovarajući termin za prevod, uz izvinjenje damama, odlučeno je da se koristi termin braća

- `firstChild` referiše na prvo dete čvora. U slučaju lista, vrednost ovog svojstva je NULL.
- `lastChild` referiše na poslednje dete čvora. U slučaju lista, vrednost ovog svojstva je NULL.
- `childNodes` predstavlja (potencijalno prazan) niz čvorova dece čvora.
- `parentNode` referiše na čvor roditelj. U slučaju korena vrednost ovog svojstva je NULL.
- `nextSibling` referiše na narednog brata čvora. U slučaju poslednjeg čvora u listi dece, vrednost ovog svojstva je NULL.
- `previousSibling` referiše na narednog brata čvora. U slučaju prvog čvora u listi dece, vrednost ovog svojstva je NULL.
- `attributes` predstavlja (potencijalno prazan) niz čvorova atributa čvora.
- `ownerDocument` upućuje na jedinstveni čvor dokumenta (svi čvorovi su sadržani u okviru ovog čvora).

Naredna slika ilustruje neka od ovih svojstava:



Na primer, tekstu `MyHeader` u prethodnom primeru moguće je pristupiti na sledeći način:

```
document.documentElement.firstChild.nextSibling.childNodes[1].firstChild.nodeValue
```

Osnovne metode za manipulaciju drvetom su:

- `appendChild(c)` - dodaje novo dete `c` čvoru.
- `removeChild(c)` - uklanja dete `c` iz čvora.
- `replaceChild(oc, nc)` - menja staro dete `oc` novim `nc`.
- `insertBefore(nc, rc)` - ubacuje novo dete `nc` pre postojećeg deteta `rc`.

Interfejs `Document`. Objekat koji predstavlja dokument, pored interfejsa `Node` implementira i interfejs `Document` (interfejs `Document` nasleđuje interfejs `Node`).

Dokument poseduje određeni broj metoda koji se koriste da bi se locirali određeni čvorovi bez potrebe za direktom navigacijom kroz DOM drvo dokumenta.

`getElementById` - Otkako je uveden, najčešće korišćen metod objekta dokumenta je `getElementById(id)`. Ovaj metod omogućava da se locira čvor sa datim identifikatorom (naravno, HTML standard zahteva da su identifikatori jedinstveni na nivou celog dokumenta, i u slučaju da nisu, dolazi do greške).

`getElementsByTagName` - ovaj metod vraća listu svih objekata (čvorova) koji odgovaraju datom elementu u okviru dokumenta.

Pored ovoga, interfejs `Document` nudi metode za kreiranje novih čvorova. Npr.

`createElement(name)` - kreira čvor elementa datog imena

`createTextNode(text)` - kreira tekstualni čvor datog teksta

`createAttribute(name)` - kreira čvor atributa datog imena

Interfejsi `Element` i `Attr`. Interfejs `Element` implementiraju objekti koji predstavljaju elemente dokumenta, dok interfejs `Attr` implementiraju objekti koji predstavljaju njihove atribute. Interfejs `Element` uglavnom nudi metode za baratanje sa atributima pridruženim elementu, dok interfejs `Attr` nudi svojstva `name`, `specified` i `value` za ispitivanje imena i vrednosti atributa, kao i svojstvo `ownerElement` za pristup elementu na kojeg se atribut odnosi.

4.1.2 HTML DOM

Kao što je već rečeno, HTML DOM definiše interfejse specifične za predstavljanje HTML elemenata.

Interfejs `HTMLDocument`. Za predstavljanje HTML dokumenata koristi se interfejs `HTMLDocument` koji nasleđuje osnovni interfejs `Document`. Kao što je većrečeno, celom dokumentu odgovara čvor kome se može pristupiti sa `document`. U slučaju HTML dokumenta, ovaj čvor upravo implementira interfejs `HTMLDocument`.

Osnovna svojstva koje ovaj interfejs definiše su:

`title` - naslov dokumenta

`URL` - URI dokumenta

`body` - čvor DOM drveta koji odgovara HTML `body` elementu

`images` - kolekcija čvorova koji odgovaraju elementima `img` u dokumentu — svakoj slici koja se nalazi u dokumentu se može pristupiti korišćenjem ove kolekcije.

`applets` - kolekcija čvorova koji odgovaraju elementima `applet` u dokumentu — svakom apletu koji se nalazi u dokumentu se može pristupiti korišćenjem ove kolekcije.

`links` - kolekcija čvorova koji odgovaraju elementima `link` u dokumentu.

`forms` - kolekcija čvorova koji odgovaraju elementima `form` u dokumentu — svakoj formi koji se nalazi u dokumentu se može pristupiti korišćenjem ove kolekcije.

`anchors` - kolekcija čvorova koji odgovaraju elementima `a` u dokumentu — svakom sidru koje se nalazi u dokumentu se može pristupiti korišćenjem ove kolekcije.

Kolekcije čvorova implementiraju interfejs `HTMLCollection` i njihovim članovima je moguće pristupiti bilo preko rednog broja, bilo preko identifikatora, tj. imena. Tako npr.

```
document.images[0]
```

predstavlja čvor koji odgovara prvoj slici navedenoj u dokumentu, dok

```
document.images["planeta"]
```

predstavlja čvor koji odgovara slici čiji atribut `id` ima vrednost `planeta`, ili, ako takva slika ne postoji, slici čiji atribut `name` ima vrednost `planeta`.

Pored ovoga, često se koriste i nestandardna svojstva kao što su

`lastModified` - vreme poslednje promene,

`readyState` - proverava da li je dokument učitan,

`documentMode` - označava mod koji pregledač koristi za prikazivanje dokumenta.

Metodi koje definiše interfejs `HTMLDocument` su:

`open` - otvara izlazni tok u koji se upisuju rezultati metoda `write` i `writeln`.

`close` - zatvara izlazni tok.

`write`, `writeln` - upisuju zadati HTML kôd u izlazni tok.

`getElementsByName` - ovaj metod dopunjava funkcionalnost metoda `getElementById` i `getElementsByTagName` definisanih u interfejsu `Document` i prikuplja sve elemente sa datim imenom (tj. za zadatom vrednošću atributa `name`).

Navedimo još nekoliko primera.

```
document.write("Hello, world!")
```

u tekući dokument upisuje tekst `Hello, world!`.

```
document.write(document.lastModified)
```

u tekući dokument upisuje vreme njegove poslednje promene.

```
document.getElementById("div1").innerHTML = "<p>Zdravo</p>"
```

sadržaj elementa čiji je identifikator `div1` postaje element `p` sa tekstom `Zdravo`.

Interfejsi koji odgovaraju HTML elementima. Svi HTML elementi imaju zaseban interfejs koji ih predstavlja. Svi ovi interfejsi nasleđuju interfejs `HTMLElement`, a koji nasleđuje osnovni interfejs `Element`. Interfejs `HTMLElement` nudi svojstva `id`, `title`, `lang`, `dir` i `className` koja predstavljaju direktnu vezu sa generičkim atributima HTML elemenata o kojima je više bilo reči u poglavlju 3.3.4.

Interfejsi koji odgovaraju HTML elementima obično imaju naziv `HTML???Element` i nude svojstva koja direktno oslikavaju standardne atribute pridružene tom

elementu. Tako npr. interfejs `HTMLMetaElement` odgovara `meta` elementu jezika HTML i nudi svojstva `httpEquiv`, `name`, `content` i `scheme` koja direktno odgovaraju atributima elementa `meta`. Neki interfejsi (ili pregledači) uvode dodatna svojstva kao što su npr. `offsetHeight` kojim se određuje visina elementa, `offsetWidth` kojim se određuje širina elementa prilikom prikaza i slično. Neki interfejsi pored ovakvih atributa definišu i metode kao što su `focus()` kojima se element postavlja u fokus tj. proglašava za trenutno aktivan dokument na stranici, zatim `blur()` koji uklanja fokus sa elementa, `click()` kojim se simulira klik mišem na element i slično.

Pojedinačni interfejsi HTML elemenata neće biti prikazivani u ovom tekstu, već se čitaoci upućuju na dokumenta DOM standarda.

4.1.3 DOM pristup pregledaču

Iako DOM standard definiše isključivo objekte kojima se predstavljaju dokumenti (objekat `document`) i njihovi delovi, većina pregledača nudi podršku za nestandardizovane objekte koji skriptovima omogućavaju pristup različitim delovima samog pregledača i delimičnu kontrolu nad njegovim ponašanjem. Opišimo ukratko ove objekte.

Objekat `window`. Ovaj objekat predstavlja otvoreni prozor u okviru pregledača. Osnovna svojstva ovog objekta su:

`document` - pristup dokumentu koji je otvoren u okviru prozora.

`navigator` - pristup objektu koji sadrži informacije o pregledaču.

`screen` - pristup objektu koji sadrži informacije o ekranu klijenta.

`history` - pristup objektu koji brine o istoriji posećenih dokumenata u okviru pregledača.

`location` - pristup objektu koji sadrži informacije o tekućoj URI adresi.

`status` - tekst u okviru statusne linije (eng. statusbar) prozora.

`name` - ime prozora

`innerWidth`, `innerHeight` - unutrašnja (bez okvira, menija i slično) širina i visina prozora pregledača.

`outerWidth`, `outerHeight` - spoljašnja (sa okvirom, menijem i slično) širina i visina prozora pregledača.

`pageXOffset`, `pageXOffset` - broj piksela koliko je dokument skrolovan po X ili Y koordinati.

`screenX` **(ili** `screenLeft`**)**, `screnY` **(ili** `screenTop`**)** - koordinate položaja prozora u odnosu na ekran.

Osnovni metodi objekta `window` su:

`open(url, name, options, replace)` - otvara novi prozor i učitava dokument u njega. Npr.

```
window.open("http://www.matf.bg.ac.rs", "Matf",
            "width=500,height=500", false)
```

`close()` - zatvara tekući prozor.

`alert(message)` - prikazuje prozorčić koji sadrži neku poruku i OK dugme.

```
window.alert("Hello, world!")
```

`confirm()` - prikazuje prozorčić koji sadrži neku poruku, OK i Cancel dugme kojim se od korisnika zahteva da potvrdi neku akciju.

```
c = window.confirm("Da li ste sigurni?")
```

`prompt(msg, defaultText)` - prikazuje prozorčić preko kojeg korisnik može da unese neki podatak.

```
b = window.prompt("Unesite neki broj: ", "5")
```

`focus()` - fokusira trenutni prozor.

`blur()` - uklanja fokus sa trenutnog prozora.

`moveBy(x, y)` - pomera prozor, relativno, u odnosu na tekuću poziciju.

`moveTo(x, y)` - pomera prozor, apsolutno, na zadatu poziciju na ekranu.

`resizeBy(w, h)` - menja dimenzije prozora, relativno, u odnosu na tekuće dimenzije.

`resizeTo(w, h)` - menja dimenzije prozora, apsolutno, i postavlja ih na zadate dimenzije.

`scrollBy(x, y)` - skroluje prozor, relativno, u odnosu na tekuću poziciju.

`scrollTo(x, y)` - skroluje prozor, apsolutno, na zadatu poziciju.

`setInterval(code, milisec, lang)` - u pravilnim vremenskim intervalima (čija se dužina zadaje) poziva zadatu funkciju. Funkcija se zadaje kao niska koja sadrži skript kôd.

```
i = window.setInterval("clock()", 500, "JavaScript")
```

Ovim se obezbeđuje da se nakon svakih pola sekunde poziva JavaScript funkcija `clock()`.

`clearInterval(id_of_setInterval)` - prekida pozivanje funkcije u pravilnim vremenskim intervalima.

```
window.clearInterval(i)
```

`setTimeout(code, milisec, lang)` - nakon određenog vremena (čija se dužina zadaje) poziva zadatu funkciju. Funkcija se zadaje kao niska koja sadrži skript kôd.

`clearTimeout(id_of_setTimeout)` - otkazuje automatsko pozivanje funkcije nakon određenog vremena.

Objekat `navigator`**.** Ovaj objekat sadrži neke informacije o pregledaču. Osnovna svojstva su:

`appCodeName` - kodno ime pregledača.

`appName` - ime pregledača.

`appVersion` - verzija pregledača.

`platform` - određuje platformu na kojoj radi pregledač.

`userAgent` - određuje identifikaciju koju pregledač šalje na server prilikom slanja HTTP zahteva.

`cookieEnabled` - određuje da li je dopušteno korišćenje kolačića.

Objekat `screen`**.** Ovaj objekat sadrži informacije o ekranu klijentske mašine. Osnovna svojstva su:

`width`, `height` - ukupna širina i visina ekrana.

`availWidth`, `availHeight` - širina i visina ekrana raspoložive za prikaz prozora pregledača.

`pixelDepth`, `colorDepth` - broj bita koji se koriste za kodiranje boje piksela ili slika (određuje ukupan broj raspoloživih boja).

Objekat `history`**.** Ovaj objekat sadrži informacije o istoriji posećenih dokumenata u okviru pregledača. Osnovno svojstvo je `length` koje određuje broj posećenih dokumenata u istoriji. Osnovni metodi su:

`back()` - učitava prethodni dokument u istoriji.

`forward()` - učitava naredni dokument u istoriji.

`go(num|URI)` - učitava određeni dokument iz istorije.

Objekat `location`**.** Ovaj objekat sadrži informaciju o tekućoj URI adresi. Osnovna svojstva su:

`href` - ceo URI

`protocol` - protokol deo URI

`host` - ime hosta i port iz URI

`hostname` - ime hosta iz URI

`port` - port iz URI

`pathname` - putanja iz URI

`hash` - fragment iz URI

`search` - upit (GET parametri) iz URI

Osnovni metodi su:

`reload()` - ponovno učitava tekući dokument

`assign(URI)`, `replace(URI)` - učitavaju novi dokument

4.2 Javascript/ECMAScript

JavaScript je svakako najpopularniji jezik za pisanje klijentskih Veb skriptova. Razvijen 1996. u kompaniji Netscape, JavaScript je veoma brzo počeo da se intenzivno koristi za dodavanje dinamičnosti Veb stranicama. Microsoft odmah ugrađuje podršku za jezik u Internet Explorer, ali zbog pravnih pitanja svoj dijalekt naziva *JScript*. Veoma je važno na početku naglasiti da je jezik JavaScript potpuno različit od jezika Java. Prvo ime jezika u Netscape-u, bilo je *LiveScript* što je trebalo da ukaže na njegovu namenu — pravljenje dinamičkih (živih) Veb stranica. U to doba, u kompaniji Sun Microsystems pojavio se jezik Java i veoma brzo stekla popularnost na internetu pošto je omogućila kreiranje apleta — aplikacija koje se preko Veba distribuiraju korisnicima i Netscape ubacuje podršku za aplete u njihov pregledač. Novi Netscape-ov jezik se između ostalog mogao koristiti za kreiranje neke interakcije između Java apleta i Veb stranicu u koju je aplet ubačen, te tako, uz dozvolu kompanije Sun, Netscape se odlučuje na marketinški potez — svoj jezik preimenuje u *JavaScript* kako bi iskoristio tadašnju popularnost jezika Java. JavaScript je zaštićeno kompanije ime Sun Microsystems.

JavaScript je danas jezik koji implementira (i donekle proširuje) standard poznat kao *ECMAScript* definisan u dokumentima ECMA-262 i ISO/IEC 16262. Naime, radi postizanja kompatibilnosti pregledača, kreće se na standardizaciji jezika, koja je urađena pod okriljem asocijacije ECMA. *European Computer Manufacturers Association (ECMA)* je industrijska asocijacija osnovana 1961. posvećena standardizaciji u oblasti Information and Communication Technology (ICT) i Consumer Electronics (CE). 1994. osnovana je *ECMA International - European association for standardizing information and communication systems.*

Na razvoju JavaScript-a danas radi Mozilla Foundation, koja se smatra naslednikom Netscape-a.

4.2.1 Osnovne karakteristike jezika

Sintaksa jezika JavaScript, je u velikoj meri inspirisana sintaksom jezika Java, a koja je inspirisana jezicima C++ i C. Pored ovoga, veliki broj standardnih imena je preuzet iz jezika Java kao i konvencije o imenovanju (koristi se notacija kamilinihGrba). Jezik razlikuje velika i mala slova. Jezik je dinamički i slabo tipiziran, i kombinuje nekoliko programskih paradigmi: pre svega proceduralnu i objektno-orijentisanu, međutim, iako se programiranje u JavaScript-u u velikoj meri zasniva na korišćenju objekata (kako objekata iz standardne biblioteke jezika, tako i DOM objekta), JavaScript nije klasičan objektno-orijentisan jezik jer korisnicima ne nudi mogućnost kreiranja sopstvenih klasa, već se proširivanje objekata zasniva na tzv. programiranju zasnovanom na prototipovima (eng. prototype based programming). Pored ovoga, JavaScript nudi i neke mogućnosti funkcionalnih programskih jezika (npr. funkcije višeg reda (eng. higher order functions), zatvorenja (eng. closures)). JavaScript kôd se u potpunosti odvija u okviru procesa Veb pregledača i iz njega nije moguće pristupati okruženju van samog pregledača. Ovo je veoma važno iz bezbednosnih razloga kako bi se sprečili zlonamerni skriptovi da pristupe delovima datotečkog sistema klijentske mašine.

Umetanje skriptova u Veb stranice Podsetimo se, postoji za umetanje skript koda u Veb stranice postoji nekoliko načina. Najčešće korišćeni način je korišćenje `script` elementa, već opisanog u poglavlju 3.3.4. Ovaj element može da se javi bilo u zaglavlju, bilo u telu HTML dokumenta. Kôd skripta može da se navede bilo kao sadržaj `script` elementa, bilo u spoljašnjoj datoteci na koju se ukazuje korišćenjem `src` atributa. Atributom `type` navodi se tip skripta (pošto HTML dozvoljava različite skript jezike) i vrednost koju potrebno navesti za JavaScript je `text/javascript`. Npr.

```
<script type="text/javascript">
  document.write("Hello, world!");
</script>
```

ili

```
<script type="text/javascript" src="skript.js"></script>
```

Skript kôd se takođe navodi kao vrednost nekog atributa koji služe da opišu reakcije na događaje tj. akcije korisnika (npr. `onclick`, `onkeydown`, ...), koji su već ranije pominjani u poglavlju 3.3.4. Npr.

```
<input type="button" value="izracunaj" onClick="izracunaj()" />
```

kada se pritisne dugme, poziva se funkcija izračunaj.

S obzirom da ovaj put nije moguće koristiti `type` kojim bi se naveo skript jezik koji je korišćen, poželjno je kroz HTTP polje `Content-Script-Type` navesti podrazumevani skript jezik korišćen u dokumentu. Npr.

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

Ako se ovo i izostavi, korisnički agenti obično za podrazumevani jezik obično koriste JavaScript. Bez obzira da li je podrazumevani programski jezik postavljen, navođenje atributa `type` u okviru elementa `script` je obavezno.

Nekada je poželjno i eksplicitno u kodu naglasiti da se radi o JavaScript kodu, korišćenjem modifikatora `javascript:`.

```
<input type="button" value="izracunaj" onClick="javascript:izracunaj()" />
```

Na ovaj način, moguće je pozivati JavaSkript kôd i kroz hiperveze. Npr.

```
<a href="javascript:izracunaj()">Izracunaj</a>
```

Sakrivanje skript koda od pregledača koji ga ne razumeju. U davna vremena, nisu svi pregledači imali mogućnost interpretacije skriptova. Dešavalo bi se, da prilikom pokušaja otvaranja stranice koja sadrži skript, oni tekst skripta jednostavno prikažu na ekran, što svakako nije poželjno. U cilju sprečavanja ove pojave, jezik JavaScript je dizajniran tako da pored standardnih C/C++ komentara (`/* */` i `//`) zanemaruje i linije koje počinju započetim HTML komentarima (`<!--`). Tako je poželjna praksa bila da se kôd skripta ogradi HTML komentarima, kako bi ga pregledači koji ne razumeju JavaScript jednostavno preskočili. Npr.

```
<script type="text/javascript">
<!--
  document.write("Hello, world!");
// -->
</script>
```

Naredbe. JavaScript programi/skriptovi su sastavljeni od naredbi. Osnovne naredbe uključuju pozivanje metoda na određenim objektima, pri čemu ti objekti mogu biti DOM objekti (npr. `window`, `document`, ...), standardni JavaScript objekti (npr. `Math`, `Date`, `String`, ...) ili neki korisnički definisani objekti. Naravno, podržane su i naredba dodele, uobičajene kontrolne strukture, sekvencijalno nizanje naredbi i slično. Za razliku od jezika C/C++, naredbu nije neophodno terminisati karakterom `;`. Ukoliko ovaj karakter nije naveden krajem naredbe se smatra kraj reda. Ipak, korišćenje terminatora `;` česta je praksa. Ovim se omogućava da i više naredbi bude navedeno u istom redu. Više naredbi je moguće grupisati u složenu naredbu (blok naredbi) korišćenjem vitičastih zagrada `{` i `}`.

Promenljive, elementarni tipovi, operatori. Promenljive se uvode korišćenjem `var` naredbe. Promenljive mogu biti samo deklarisane ili im ujedno može biti dodeljena vrednost, u kom slučaju se ključna reč `var` može izostaviti. Npr.

```
var i = 2, j; k = "zdravo";
```

Prilikom uvođenja promenljivih nema potrebe za navođenjem njihovog tipa. JavaScript je slabo, dinamički tipiziran jezik. Osnovni tipovi su brojevni tip (celobrojni i realni), niske karaktera (predstavljene klasom `String`), Bulovski tip (predstavljen klasom `Booolean`), itd. Brojevne konstante se navode po uzoru na jezik C (oktalne sa vodećim karakterom `0`, heksadekadne sa vodećim `0x` ili `0X`). Karakterske niske se navode između jednostrukih (`'`) ili dvostrukih navodnika (`"`). Sekvence se koriste za predstavljanje specijalnih karaktera (npr. `\t`, `\n`, `\\`, ...).

Konverziju u Bulovski i tip kao i u tip niske je moguće eksplicitno uraditi korišćenjem konstruktora `Boolean()` i `String()`, dok je konverziju niske u brojevni tip moguće eksplicitno izvršiti korišćenjem funkcija `parseInt()` i `parseFloat()`.

Niz je moguće navesti navođenjem njegovih elemenata u zagradama `[]`. Slično, pristup elementima niza vrši se korišćenjem navođenja indeksa (brojanje kreće od 0) u okviru zagrada `[]`. Npr.

```
var predmeti = ["P1", "UVIT", "P2"];
var p = predmeti[1];   // vrednost je "UVIT"
```

Veliki broj operatora nasleđen je iz jezika C/C++/Java i pravila prioriteta su veoma slična. Aritmetički operatori su npr. `+`, `*`, `/`, `%`, `++` u prefiksnoj i postfiksnoj formi, `--` u prefiksnoj i postfiksnoj formi, itd. Naglasimo da, s obzirom na slabu tipiziranost jezika, ovi operatori deluju na operande različitog tipa, pri čemu se često automatski ostvarjuju implicitne konverzije. Tako npr. operator `+` može da se primeni na brojeve kada označava sabiranje, ali može da se primeni i na niske kada označava nadovezivanje (konkatenaciju) niski. Npr.

```
i = 3 + 3;     // ima vrednost 6
j = "3" + "3"; // ima vrednost 33
k = 3 + "3";   // ima vrednost 33
```

Ostali aritmetički operatori konvertuju niske u brojeve pre primene.

Relacioni operatori su `==`, `!=`, `<`, `>`, `<=`, `>=`. Naglasimo da je, za razliku od jezika C, ove operatore moguće primenjivati i na niske. Operator `==` proverava da li dva operanda imaju jednake vrednosti, pri je čemu je pre upoređivanja moguća implicitna konverzija. Operator `===` ne vrši konverzije već proverava da li su operandi identične kako vrednosti, tako i tipa.

Logički operatori su `&&`, `||` i `!`.

Bitski operatori su `&`, `|`, `~`, `^`, `<<`, `>>` i `>>>` koji predstavlja logičko pomeranje ulevo (upražnjena mesta se uvek popunjavaju nulama).

Operatori dodele su npr. `=`, `+=`, `-=`, `*=`, `/=`, `%=`, ...

Podržan je i ternarni uslovni operator `?:`.

Kontrolne strukture Jezik JavaScript poseduje uglavnom standardne kontrolne strukture i to u sintaksi koja odgovara jezicima C/C++/Java. Npr.

```
if (uslov)
   naredba
else
   naredba


for (inicijalizacija; uslov; korak)
  naredba

while (uslov)
  naredba

do
  naredba
while (uslov)

switch(izraz) {
  case vrednost1:
    nareba1
    [break]
  ...
  [default:
    naredbak]
}
```

Prekid iteracije vrši se naredbama `break` i `continue`.

Pomenimo još i naredbu `with` i `for(.. in ..)` koje ne postoje u jeziku C. Naredba `with` služi da korisnicima olakša pristup poljima nekog objekta, bez stalne potrebe da navode samo ime objekta. Npr.

```
with(document) {
   write("Zdravo!");
   write("Svete!");
}
```

S obzirom da se oba poziva metoda `write` nalaze u okviru `with` naredbe, jasno je da se oni odnose na objekat `document` te njegovo ime nije neophodno navoditi.

Petlja `for(.. in ..)` se najčešće koristi za iteraciju kroz nizove. Npr.

```
for (i in predmeti)
   document.write(predmeti[i] + "<br />");
```

Funkcije Funkcije se definišu ključnom rečču `function`. Za vraćanje izračunate vrednosti funkcije pozivaocu, koristi se naredba `return`. Npr.

```
function saberi(x, y) { return x + y; }

document.write(saberi(3, 5));
document.write(saberi("ab", "cd"));
```

Funkcije su u jeziku JavaScript „građani prvog reda" (eng. first class citizens), što znači da ih je moguće dodeljivati promenljivima, prosleđivati kao parametre drugim funkcijama, kreirati dinamički tokom izvršavanja programa i slično. Funkcije mogu da budu i anonimne (kada im se ne navede ime već samo ključna reč `function`, lista parametara i telo). Na primer, nema nikakve razlike između gore navedene definicije funkcije saberi i njene naredne definicije (funkcija je u stvari promenljiva čija je vrednost funkcijskog tipa).

```
var saberi = function(x, y) { return x + y; }
```

I naredni primer je sličan prethodnom (postavljamo polje `callback` objekta `o`, koje je takođe funkcijskog tipa i koje se obično koristi da se zada tj. zapamti funkcija koja treba biti pozvana u nekom kasnijem trenutku, obično kada se desi neki događaj.

```
o.callback = function () { window.alert("done"); };
```

I parametar, ali i povratna vrednost funkcije može biti neka druga funkcija. Prilikom prosledjivanja funkcije kao parametra navodi se ili njeno ime (bez zagrada, jer bi se tada funkcija pozvala i umesto nje, prosledila bi se njena vrednost) ili njena anonimna definicija. Na primer:

```
function handleClick() {
   ...
}
button.click(handleClick);
```

ili

```
button.click(function() {
  ...
});
```

Promenljive koje su lokalne za funkciju uvode se ključnom rečju `var`. Funkcije mogu da koriste i svoje lokalne promenljive, ali i globalne promenljive. Lokalna promenljiva može imati isto ime kao i globalna i tada je „sakriva". U jeziku JavaScript funkcije mogu da budu i ugnežđene i tada je u okviru unutrašnje funkcije moguće pristupiti lokalnim promenljivama spoljašnje funkcije.

```
var x = 1; // globalna promenljiva

function() {
   var y = 2; // lokalna promenljiva
   function g() {  // lokalna (unutrasnja) funkcija
       var z = 3;  // lokalna promenljiva (za funkciju g)
       document.write(x + y + z);
   }
}
```

Veoma zanimljivo svojstvo jezika JavaScript je da je moguće da funkcija vrati funkciju koja pristupa njenim lokalnim promenljivima. Iako je u klasičnim jezicima kakav je C uobičajeno da nakon vraćanja vrednosti funkcija prestaje da postoji tj. oslobađa se se okvir na steku pridružen pozivu koji se upravo završio, u jeziku JavaScript to nije slučaj i funkcija koja je vraćena i dalje ima pristup lokalnim promenljivima funkcije koja ju je vratila. Ova pojava naziva se *zatvorenje* (engl. closure). Na primer.

```
function f(x) {
  var y = x + 1;
  return function(z) { return y + z; }
}

var g = f(2); // g ima vrednost funkcije koja svoj argument uvecava za 3
var a = g(5); // ima vrednost 8
```

Pošto se poziv funkcije `f` završilo, nema više načina da se iz glavnog programa pristupi promenljivoj `y`, ali ona i dalje „živi" i posredno se koristiti kroz funkciju koja je vraćena. Napomenimo i da unutrašnja funkcija pristupa promenljivoj `x` po referenci, tj. da se promene promenljive `x` nakon definicije funkcije ogledaju na rad same funkcije. To može biti uzrok grešaka. Razmotrimo naredni kod.

```
function setButtonClicks() {
  for (var i = 0; i < 5; i++)
     buttons[i].click(function() { alert(i); });
}
```

Nizu dugmadi postavljaju se rukovaoci događajem pritiska na dugme i očekuje se da pritisak na i-to dugme ispiše vrednost i (npr. da se pritiskom na dugme 2 ispiše 2), međutim, svi rukovaoci ispisaće vrednost 5 (što je kranja vrednost promenljive i). Primetimo da iako funkcije nisu eksplicitno vraćene iz funkcije `setButtonClicks`, bilo je neophodno napraviti zatvorenje (tj. čuvati vrednost promenljive `i` i nakon završetka poziva funkcije `setButtonClicks`), jer će rukovaoci događajem klika dugmeta biti pozivani u budućnosti, dugo nakon što je poziv funkcije `setButtonClicks` završen. Pošto se iz svih rukovaoca pristupa promenljivoj `i` po referenci, svi će koristiti jednu te istu promenljivu (tj. koristiće njenu vrednost na samom kraju poziva, što je u ovom primeru 5).

Zatvorenja se jako često koriste u realnom programiraju, međutim, u ovom uvodnom, preglednom kursu nemamo prostora za detaljnije analiziranje funkcionalnih svojstava jezika JavaScript.

Izuzeci. Po uzoru na jezike C++ i Java, JavaScript daje mogućnost obrade izuzetaka (eng. exception handling). Ukoliko u nekom delu koda može doći do greške, poželjno je taj deo koda ograditi `try {...} catch(...) {...}` blokom. Npr.

```
try {
  f();   // greška jer funkcija f nije definisana
} catch (err) {
  if (!window.confirm("Došlo je do greške: " + err +
                      "Da li želite da nastavite da gledate ovu stranicu?"))
    window.history.go(0);
}
```

Iz nekog dela koda moguće je eksplicitno podići izuzetak korišćenjem naredbe `throw`. Npr.

```
function koren(a) {
  if (a < 0)
    throw "Pokušali ste da vadite koren iz negativnog broja";
  ...
}
```

4.2.2 JavaScript objekti

Objekti tipa `String`. Niske u jeziku JavaScript su predstavljene objektima tipa `String`.

Svojstvo `length` omogućava da se pročita dužina niske.

```
"Hello, world!".length // vrednost je 13
```

Niske se odlikuju sledećim metodima:

`chareAt(index)` - karakter na datoj poziciji

```
str = "Hello, world!";
str.charAt(0) // vrednost je 'H'
str.charAt(3) // vrednost je 'l'
str.charAt(str.length-1) // vrednost je '!'
```

`chareCodeAt(index)` - UNICODE kôd datog karaktera

```
str = "Hello, world!";
str.charCodeAt(0) // vrednost je 72
```

`concat` - nadovezuje niske (slično kao i operator +)

```
"ABC".concat("DEF")  // vrednost je ABCDEF
```

`fromCharCode(c)` - određuje karakter na osnovu datog UNICODE koda

```
c = String.fromCharCode(72) // vrednost je 'H'
```

`indexOf(str, start)` - određuje poziciju prvog pojavljivanja podniske `str`, pri čemu pretraga počinje od pozicije `start`. Ako se `start` izostavi pretraga kreće od početka. Funkcija vraća -1 ukoliko se podniska ne pronađe.

```
"podniska".indexOf("niska") // vrednost je 3
```

`lastIndexOf(str, start)` - slično prethodnoj, jedino što pretraga vraća poziciju poslednjeg pojavljivanja.

```
"AbcDefAbcDef".indexOf("Abc") // vrednost je 0
"AbcDefAbcDef".lastIndexOf("Abc") // vrednost je 6
```

`match(regexp)`, `search(regexp)` - traži pojavljivanja podniski koje se uklapaju u šablon opisan datim regularnim izrazom i vraća niz svih poklapanja.

`replace(regexp|str, newstr)` - zamenjuje sva pojavljivanja niske `str` ili regularnog izraza `regexp` niskom `newstr`.

```
str =  "Zdravo svima. Zdravo i tebi".replace("Zdravo", "Ćao");
// str = "Ćao svima. Zdravo i tebi"
```

`slice(start, end)`, `substring(start, end)` - izdvaja deo niske između dva navedene pozicije. Ukoliko se druga pozicija izostavi izdvajanje se vrši do kraja niske. Negativni brojevi se koriste za označavanje pozicija brojanjem od kraja niske.

```
str = "Hello, world!"
str.substring(7) // vrednost je "world!"
str.slice(-6, -1) // vrednost je "world"
str.substring(2, 8) // vrednost je "llo, w"
```

`substr(start, length)` - izdvaja deo niske od pozicije `start`, dužine `length` karaktera. Ukoliko se dužina izostavi, izdvajanje se vrši do kraja niske. Negativni brojevi se koriste za označavanje pozicija brojanjem od kraja niske.

```
str = "Hello, world!"
str.substr(7) // vrednost je "world!"
str.substr(-6, 5) // vrednost je "world"
str.substr(2, 6) // vrednost je "llo, w"
```

`toLowerCase()` - prevodi velika u mala slova, ostavljajući ostale karaktere nepromenjenim.

```
"Hello, world!".toLowerCase() // vrednost je "hello, world!"
```

`toUpperCase()` - prevodi mala u velika slova, ostavljajući ostale karaktere nepromenjenim.

```
"Hello, world!".toLowerCase() // vrednost je "HELLO, WORLD!"
```

`split(separator, limit)` - deli nisku na delove na osnovu datog karaktera `separator`. Izdvaja se najviše `limit` delova. Ukoliko se `limit` izostavi, izdvajaju se svi delovi.

```
str = "abc def ghi";
delovi = str.split(" "); // vrednost je niz ["abc", "def", "ghi"]
```

`valueOf` - vraća primitivnu nisku od `String` objekta. Retko se javlja potreba za direktnim pozivanjem ovog metoda.

Objekat `Math`. Objekat `Math` sadrži osnovne matematičke konstante i funkcije.

`E` - Ojlerova konstanta e (približno 2.71828183)

`PI` - konstanta π (približno 3.14159265)

`LN2` - $\log_e 2$

`LN10` - $\log_e 10$

`LOG2E` - $\log_2 e$

`LOG10E` - $\log_{10} e$

`SQRT1_2` - $\sqrt{\frac{1}{2}}$

`SQRT2` - $\sqrt{2}$

`abs(x)` - apsolutna vrednost

`cos(x)`, `sin(x)`, `tan(x)` - trigonometrijske funkcije

`acos(x)`, `asin(x)`, `atan(x)` - inverzne trigonometrijske funkcije

`round(x)`, `ceil(x)`, `floor(x)` - zaokruživanje realnog broja na najbliži ceo, prvi veći ceo i prvi manji ceo.

`exp(x)`, `log(x)`, `pow(x, y)` - eksponencijalna, logaritamska i stepena funkcija.

`sqrt(x)` - kvadratni koren broja

`min(x1, ..., xn)`, `max(x1, ..., xn)` - najmanji, odnosno najveći od nekoliko brojeva.

`random()` - slučajni broj iz intervala $[0, 1]$

Na primer, naredni kod ispisuje $\sin(37°)$.

```
document.write(Math.sin(37.0 * Math.PI / 180.0));
```

Objekti tipa `Date`. Objekti tipa `Date` predstavljaju vremenske odrednice (datum i vreme). Postoji nekoliko različitih načina da se konstruiše objekat tipa `Date`. U slučaju da se konstruktoru ne prosledi parametar, konstruiše se objekat koji predstavlja trenutno vreme na klijentskoj mašini. Vremenske odrednice u eri računara se mogu predstaviti brojem milisekundi proteklih od ponoći 1. Januara 1970. godine. Najeksplicitniji je konstruktor koji uzima 7 parametara: godinu, mesec, dan, sat, minut, sekund i milisekund. Ukoliko se neki parametri ovog konstruktora izostave, podrazumeva se njihova vrednost nula. Tako, ako se navedu samo prva tri parametra, smatra se da je konstruisana ponoć tog dana (00:00:00). Objekat tipa `Date` je moguće konstruisati i navođenjem niske koja opisuje vremensku odrednicu. Podržano je nekoliko različitih formata. Npr. `"Jun 3, 2010 14:00:53"`

```
var d = new Date();
var d = new Date(12345678); // milliseconds
var d = new Date("Jun 3, 2010 14:00:53");
var d = new Date(2010, 6, 3, 14, 0, 53, 75);
```

Objekte tipa `Date` je moguće porediti standardnim relacijskim operatorima (< označava pre, a > označava posle). Tip `Date` nudi veliki broj metoda. Izdvojmo samo neke:

`getFullYear()` , `getDate()`, `getDay()`, `getHours()`, `getMinutes()`, `getSeconds()`, `getMilliseconds()` - ovim metodima se izdvajaju određene komponente vremenske odrednice — godina, mesec, dan, sati, minuti, sekundi i milisekundi.

`setFullYear()` , `setDate()`, `setDay()`, `setHours()`, `setMinutes()`, `setSeconds()`, `setMilliseconds()` - ovim metodima se postavljaju određene komponente vremenske odrednice — godina, mesec, dan, sati, minuti, sekundi i milisekundi.

Objekti tipa `Array`. Tip `Array` predstavlja nizove. Prazan niz konstruiše se pozivom konstruktora bez argumenata. Elementima se pristupa indeksno, korišćenjem zagrada `[]`. Pored ovoga, dostupno je nekoliko metoda koje olakšavaju rad sa nizovima.

`concat(a2, ..., an)` - nadovezuje više nizova.

```
var a = [1, 2, 3], b = [4, 5, 6];
a.concat(b) // vrednost je [1, 2, 3, 4, 5, 6]
```

`join(separator)` - Kreira nisku koja se sastoji od elemenata niza između kojih je umetnut navedeni separator.

```
var a = [1, 2, 3];
var s = a.join(" + "); // vrednost je "1 + 2 + 3"
```

`push(x), pop()` - Umeće element na kraj niza, odnosno uklanja element sa kraja niza.

`shift, unshift(x)` - Uklanja element sa početka niza, odnosno umeće element na početak niza.

`reverse()` - Obrće niz.

```
var a = [1, 2, 3];
a.reverse() // vrednsot je [3, 2, 1]
```

`splice(index, numToRemove, add1, ..., addn)` - uklanja `numToRemove` elemenata sa pozicije `index`, a zatim na to mesto umeće redom elemente `add1` do `addn`.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(1, 1, "Kiwi", "Ananas")
  // vrednost je  ["Banana", "Kiwi", "Ananas", "Apple", "Mango"];
```

`slice(start, end)` - izdvaja deo niske između dve date pozicije. Ukoliko se druga pozicija izostavi izdvajanje se vrši do kraja niza. Negativni brojevi se koriste za označavanje pozicija brojanjem od kraja niske.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.slice(1, 2) // vrednost je niz ["Orange", "Apple"]
```

Objekti tipa `RegExp`. Ovaj objekat se koristi za prepoznavanja uzoraka u tekstu koji su opisani formalizmom regularnih izraza. Ovaj formalizam je izrazito koristan, međutim, u ovom trenutku neće biti detaljnije objašnjavan.

4.3 Biblioteka JQuery

Glava 5

Veb serveri - Apache

Glava 6

Serverski skriptovi - PHP

Za razliku od klijentskih skriptova, koji su se u obliku izvršnog ili kompiliranog koda prenosili na klijentsku mašinu i na njoj izvršavali, serverski skriptovi predstavljaju programe koji se izvršavaju na serverskoj mašini i rezultat njihovog rada (najčešće u obliku formirane Veb stranice) se šalje klijentu gde se prikazuje. Za razliku od slučaja klijentskih skriptova kada je klijent morao da bude osposobljen da izvršava preneti program (npr. da pregledač poseduje mogućnost izvršavanja JavaScript koda, da je na klijentu instaliran JRE i slično), da bi klijent mogao da koristi serverske skriptove, potrebno je načešće samo da ima elementarne mogućnosti komuniciranja HTTP protokolom i prikazivanja HTML stranica. Međutim, ovaj put, serverska mašina mora da bude u mogućnosti da izvršava skriptove. U današnje vreme, najčešće korišćeni jezici za pisanje skriptova na strani servera su PHP, ASP.NET, JSP i Python. Nekada je apsolutnu dominaciju na ovom polju imao programski jezik Perl.

Dva su dominantna načina za izvršavanje skriptova od strane Veb servera: CGI i moduli Veb servera.

Common Gateway Interface (CGI).

Apache moduli.

6.1 PHP