UVOD

Ljudi koji su stvarali internet:

- **Claude Shannon** - „Matematička teorija komunikacija“, predstavljanje informacija bitovima;
- **Vannevar Bush** - uspostavio suradnju sa sveučilištima, postavio ideju Interneta i pristupa različitim sadržajima;
- **J. C. R. Licklider** – razvio ideju univerzalne mreže i učestovao u njenom stvaranju kao direktor IPTO-a;
- **Paul Baran** – definirao mrežu s prospajanjem paketa;
- **Leonard Kleinrock** – na UCLA surađivao u razvoju ARPANET-a;
- **Lawrence Roberts** – ARPANET voditelj projekta;
- **Vinton Cerf/Robert Khan** – definirali TCP/IP protokol;
- **Tim Berners-Lee** – izumio www;

HTTP

HyperText Transfer Protocol je protokol aplikacijske razine koji upravlja komunikacijom između Web (HTTP) servera i Web (HTTP) klijenta.

HTTP protokol (server/klijent tipa) se obično prenosi preko TCP protokola. Protokoli niže razine trebaju osigurati siguran i pouzdan prijenos između HTTP servera i klijenta.
HTTP server po defaultu sluša na portu 80, ali bilo koji drugi port se također može koristiti.
Često korištene aplikacije HTTP servera su Apache, IIS (Internet Information Server), ….

HTTP server (ili Web server ili servis) je aplikacija koja radi prema HTTP standardu. HTTP klijent i HTTP server razmjenjuju HTTP poruke preko TCP konekcije. Format HTTP poruka je ASCII niz karaktera odnosno tekst. Svaka linija HTTP poruke mora završavati sa dva ASCII karaktera - **CRLF** “\r\n” (**line feed** (LF, \n, 10, HEX 0A) i **carriage return** (CR, \r, 13, HEX 0D)) – tim se karakterima formatira i kraj linije tekstualne datoteke na Windows OS-ovima.

HTTP cookie

HTTP protokol je stateless i connectionless protokol. Cookie je jedna od **metoda praćenja komunikacije** između klijenta i servera (sesija) implementirana u HTTP protokolu. HTTP cookie ili Web cookie je skup informacija u tekstualnom obliku koje server pošalje klijentu (IE, Firefox,…) i koje klijent onda šalje nazad tom istom serveru u sklopu svakog budućeg zahtjeva.

HTTP zahtjev

Klijent serveru šalje HTTP zahtjev.
HTTP protokol točno definira kakav format treba imati ispravni HTTP zahtjev.
Format HTTP zahtjeva je:

- Linija zahtjeva
- Zaglavlja (nijedno, jedno, više)
- Prazna linija
- Sadržaj poruke

Linija zahtjeva sadrži HTTP metodu, prazno mjesto (*space*), URL koji se dohvaća, prazno mjesto i verziju HTTP protokola po kojoj je zahtjev složen.

primjer: POST /cgi-bin/post-query HTTP/1.0

Kako je HTTP protokol ASCII niz znakova kao delimiteri između dijelova HTTP poruke koriste se ASCII znakovi (*space*, *carriage return* , *line feed*). HTTP 1.0 zahtjev ne treba imati niti jedno zaglavlje, dok HTTP 1.1 zahtjev treba imati barem zaglavlje "Host".

URI (URL)

URL (*Uniform Resource Locator*) je pojam koji se vrlo često koristi umjesto pojma **URI** mada to nisu istoznačni pojmovi. URL je ustvari samo dio URI-ja (*Universal Resource Identifier*).
URI je standard koji definira jednoznačnu sintaksu za definiranje globalnih identifikatora dokumenata koji se nalaze na mreži kako bi im se preko tog jedinstvenog identifikatora moglo pristupiti.
Globalni identifikator je string koji specificira način pristupa nekom resursu na mreži odnosno jednoznačno identificira resurs na mreži kao što je na primjer HTML dokument.

Premda u početku Weba nema formalne specifikacije termin *Uniform Resource Locator* označava **adresu** dokumenta dok se termin URN (*Uniform Resource Name*) koristi za **ime** resursa. (http://www.fesb.hr/index.html ↔ index.html)
Tim Berners-Lee uvodi termin URI koji najbolje definira svojstvo jedinstvenog identifikatora, ali je termin URL i dalje uglavnom u upotrebi.

Naime URL je omogućava pristup resursima sa različitima načinima imenovanja (*naming schemes*) i načinima pristupa (HTTP, FTP, E-mail) na jednostavan način.
Shema je navedena prije znaka:
Format URL-a <protokol>:<adresa>

mailto:maja.stula@fesb.hr

Za FTP, HTTP, HTTPS

<protokol>://<server>:[<port>]/<put do dokumenta>
http://laris.fesb.hr/index.html
http://laris.fesb.hr:80/index.html

URI referenca

URI referenca je jedan **tip stringa koji predstavlja URI**. Kod HTML dokumenta vrijednost *src* atributa HTML *img* elementa je URI referenca. Kod XML dokumenta nakon ključne riječi SYSTEM u DTD je URI referenca. Kod XSLT vrijednost *href* atributa xsl:import elementa je URI reference.

URI referenca može imati oblik čitavog URI, ili dijela specifikacije sheme ili čak praznog stringa.

- http://example/resource.txt#frag01
- http://somehost/absolute/URI/with/absolute/path/to/resource.txt
- /relative/URI/with/absolute/path/to/resource.txt
- relative/path/to/resource.txt
- ../../../resource.txt
- resource.txt
- /resource.txt#frag01
- #frag01
- *(empty string)*

Apsolutni ↔ Relativni URI

Relativni URL (odnosno danas URI) **specificira samo dio URL-a**, a ostatak URL se nasljeđuje od ostatka dokumenta**. Apsolutni** URL sadrži **potpunu adresu dokumenta**.

Apsolutni URI

- http://somehost/absolute/URI/with/absolute/path/to/resource.txt
- ftp://somehost/resource.txt
- urn:issn:1535-

Relativni URI

- /relative/URI/with/absolute/path/to/resource.txt
- relative/path/to/resource.txt
- ../../../resource.txt
- resource.txt

HTTP metode

Polje metode u HTTP liniji zahtjeva **označava koja će se metoda primijeniti na objektu** identificiranom sa URL-om. GET metoda je uvijek podržana na svim resursima identificiranim navedenim URL-om (za sve objekte). Imena metoda (GET, POST, …) su *case sensitive* – ne može se pisati Get i sl.

HTTP standardne metode trenutno definirane su:

- OPTIONS
- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE

OPTIONS

OPTIONS metoda je **zahtjev za informacijama o HTTP metodama** koje su na raspolaganju za određeni URI. Ukoliko odgovor servera nije greška, odgovor NE SMIJE uključivati nikakvu informaciju o objektu osim informacije koja se odnosi na komunikaciju.
Ako je URI asterisk ("*"), OPTIONS se odnosi na cijeli server.
Ako HTTP zahtjev sa OPTIONS prođe kroz HTTP *proxy*, onda HTTP *proxy* MORA izmijeniti odgovor od servera da isključi opcije koje nisu dozvoljene preko *proxy*-ja.
Primjer: OPTIONS http://www.fesb.hr/~kiki/index.htm HTTP/1.0

HTTP proxy (posrednik)

Proxy je **posrednički program** koji istovremeno djeluje i kao HTTP server i kao HTTP klijent u svrhu postavljanja zahtjeva u ime drugih klijenta. Zahtjevi se obrađuju ili unutar *proxy*-ja ili ih se prosljeđuje drugim serverima uz moguće izmjene zahtjeva.

Implementacija HTTP proxy-ja može biti na više načina:

- **Presretački *proxy*** (*intercepting proxy*, *transparent proxy*) je kombinacija HTTP *proxy*-ja i NAT (*Network Address Translation*). Konekcije klijenta se preko NAT se prosljeđuju *proxy*-ju bez znanja klijenta. Često ga koriste ISP (*Internet Service Provider*) kako bi smanjili promet koristeći zajednički keš.
- **Otvoreni *proxy*** (*open proxy*) je *proxy* koji prihvaća konekcije klijenata sa bilo koje IP adrese prema bilo kojem resursu (URI-ju). Zbog nedovoljne zaštite (nema ograničenja tko postavlja zahtjev i prema kome) često se zloupotrebljavaju.
- **Obrnuti *proxy*** (*reverse proxy*) je proxy server instaliran uz HTTP server. Sav promet prema HTTP serveru ide preko *proxy*-ja. Omogućava dodatnu zaštitu web servera, keširanje sadržaja sa web servera na *proxy*-ju čime se smanjuje opterećenje web servera, može komprimirati sadržaj koji se prenosi u HTTP porukama …
- **Razdijeljeni *proxy*** (*split proxy*) je par *proxy*-ja instaliranih na dva računala koji mogu međusobno komunicirati brže nego sa web serverom i klijentom. Obično se koriste za kompresiju/dekompresiju podataka preko sporih veza poput bežične ili mobilne konekcije.

GET

GET metoda **dohvaća objekt identificiran s URI-jem** u HTTP zahtjevu. Objekt identificiran URI-jem ne treba biti isključivo samo statička HTML stranica. Ako se zatraženi URI odnosni na neki proces koji generira podatke, web server zna na koji način pozvati proces, proslijediti mu podatke klijentovog zahtjeva, te vratiti nazad klijentu podatke koje je izgenerirao proces, a ne tekst koji je zapisan u file-u kojim se pokreće proces.
Web server mora razlikovati statički dokument od procesa. Kako to radi nije definirano u HTTP protokolu. Npr. neke implementacije web servera (Apache) definiraju posebni direktorij u koji se smještaju CGI (*Common Gateway Interface*) programi.
PRIMJER:

statička stranica:	http://www.fesb.hr/~kiki/proba.htm
proces:	http://www.fesb.hr/kiki-cgi/proba

UVJETNI GET

Semantika GET metode se mijenja kod uvjetnog GET-a (*conditional* GET) kada HTTP zahtjev sadrži If-Modified-Since, If-Unmodified- Since, If-Match, If-None-Match, ili If-Range polje zaglavlja. Uvjetni GET zahtjev omogućava prijenos objekta definiranog URI-jem samo ako su zadovoljeni uvjeti definirani u zaglavlju.

Format zaglavlja If-Modified-Since:

If-Modified-Since = "If-Modified-Since" ":" HTTP datum (*HTTP–date*)

Format zaglavlja If-Unmodified-Since:

If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP datum

Format zaglavlja If-Match:

If-Match = "If-Match" ":" ("*" | 1#entity-tag)

OZNAKA ENTITETA

Oznaka entiteta (*entity tag*) koristi se za **uspoređivanje dva ili više entiteta** zahtijevanog resursa (URI-ja). Oznaka entiteta koristi se u zaglavljima ETag (koje server šalje klijentu s vrijednošću oznake entiteta), If-Match, If-None-Match i If-Range. Oznaka entiteta je u biti string koji može imati dva formata:

- entity-tag = [weak] opaque-tag
- weak = "W/“ (weakness indicator)
- opaque-tag = quoted-string

Jaku oznaka entiteta (*strong entity tag*) mogu imati samo oni entiteti zahtijevanog resursa (URI-ja) koji su potpuno identični (*octet equality*). Slabu oznaka entiteta (*weak entity tag*), sa "W/" prefiksom, mogu imati samo oni entiteti zahtijevanog resursa (URI-ja) koji se mogu zamijeniti bez gubitka značenja. Oznaka entiteta treba biti jedinstvena za sve entitete određenog resursa.

Kod starijih verzija Apache web servera vrijednost oznake entiteta (ETag zaglavlja) se generirala na osnovu veličine datoteke, vremena zadnje promijene datoteke i sl.

Ukoliko klijent ima više entiteta dohvaćenih sa zahtijevanog resursa (URI-ja) može za svih provjeriti da li su i dalje nepromijenjeni navodeći oznake entiteta u zaglavlju If-None-Match.

Zaglavlje If-Range omogućava klijentu da zahtjeva samo dijelove entiteta. Ukoliko se entitet promijenio klijent očekuje cijeli entitet, a ne samo zahtijevani dio.

PARCIJALNI GET

Semantika GET metode se mijenja kod parcijalnog GET-a (partial GET) kada HTTP zahtjev sadrži Range polje zaglavlja. Parcijalni GET zahtjev **dohvaća samo dio objekta**, a ne čitav objekt.

Njime se također postiže smanjivanje prometa na mreži.

Sadržaj HTTP poruke (entity-body) je “samo” niz bajtova pa se pojam bajta može koristiti za ograničavanje dijelova sadržaja HTTP poruke.

Ako server podržava Range zaglavlje kod običnog GET-a (bez uvjeta) vraća dio entiteta određen u Range zaglavlju uz statusni kod 206 (*Partial Content*) umjesto 200 (*OK*).

HEAD

HEAD metoda je po svemu **identična GET metodi** osim što server u odgovoru **NE SMIJE vratiti tijelo poruke** (to je sve ono što ide iza zaglavlja). Meta podaci (*meta information*) iz HTTP zaglavlja odgovora trebaju biti isti kod HEAD i kod GET zahtjeva.
Ova se metoda koristi kada se žele dohvatiti meta podaci o objektu bez prenošenja cijelog objekta. Često se koristi za testiranje linkova - da li im se može pristupiti, da li postoje, da li su nedavno promijenjeni.

POST

POST metoda se koristi kada želimo **da server prihvati objekt** (sadržaj poruke) HTTP zahtjeva kao dio koji je podređen zahtijevanom resursu identificiranom preko URI-ja. POST je napravljen kao jedinstvena uniformna metoda koja se koristi u sljedećim slučajevima:

- Kada želimo postojećem resursu dodati neku oznaku;
- Kada se šalje poruka na neku mailing listu, ili newsgrupu i sl.;
- Za isporuku bloka podataka isporučenih s forme nekom procesu za obradu tih podataka;
- Dodavanju novih slogova u bazu.

Koja od prethodno navedenih funkcija će se i izvršiti nakon što server dobije HTTP zahtjev s POST metodom određeno je samim serverom i obično ovisi o URI-ju koji se zahtjeva.

Location zaglavlje

Zaglavlje Location koristi se **za redirekciju klijenta** na lokaciju različitu od one navedene u URI-ju zahtjeva. Za odgovore sa statusnim kodom 201 Location je lokacija resursa kreiranog kao odgovor na klijentov zahtjev. Za odgovore sa statusnim kodovima 3xx lokacija bi trebala biti URI na koji server upućuje klijenta umjesto zahtijevanog URI-ja.
Format zaglavlja Location:

Location = "Location" ":" absoluteURI
Primjer: www.fesb.hr

Redirekcija

Redirekcija preko HTML meta taga (http-equiv=" Refresh“) nije implementirana u HTTP protokolu. To je dio HTML standarda i klijenti koji dohvaćaju HTML stranice znaju što napraviti kada u HTML kodu naiđu na ovaj tag.
Redirekciju preko HTTP protokola treba implementirati na strani servera. Način implementacije ovisi o web serveru, o URI-ju koji se redirektira i sl.

SLANJE PODATAKA SA FORME POST METODOM

POST metoda često se koristi za slanje sadržaja forme. Podaci sa forme se šalju u sadržaju poruke (*entity-body*). Na strani servera podaci se procesu (obično se radi o nekakvom procesu koji će podatke obraditi) šalju kao standardni ulaz procesa (stdin).

Primjer:

```
<form METHOD="POST" ACTION="http://www.fesb.hr/kiki-cgi/proba1">
<INPUT TEXT NAME="polje"><br>
<INPUT TEXT NAME="polje2"><br>
<INPUT TEXT NAME="polje3"><br>
<input type="submit" value="POKRENI PROGRAM" size="1" >
</form>
```

Content-length zaglavlje

Zaglavlje Content-Length **sadrži veličinu sadržaja poruke** izraženu u obliku decimalnog broja okteta (bajtova) sadržaja poruke, koja se šalje primatelju poruke ili u slučaju HEAD metode veličinu sadržaja koji bi se poslao da se radi o GET metodi. HTTP aplikacije bi trebale koristiti ovo polje da odrede veličinu sadržaja neovisno o tipu sadržaja (tekst, slika, ….). Primatelj poruke mora biti u stanju odrediti kraj poruke kod HTTP/1.1 protokola.

Format zaglavlja Content-Length:

Content-Length = "Content-Length" ":" 1*DIGIT

Duljina poruke

Kada poruka ima sadržaj duljina sadržaja se određuje na jedan od sljedećih načina:

- Svaki odgovor koji ne smije imati sadržaj (statusni kodovi 1xx, 204 i 304, te odgovor na HEAD zahtjev) završava sa prvom praznom linijom nakon zaglavlja.
- Ukoliko zaglavlje Transfer-Encoding definira da je poruka razbijena na dijelove (Transfer-Encoding: chunked) duljina se dobiva iz oznake dijelova poruke.
- Ako je navedeno zaglavlje Content-Length tada je duljina sadržaja poruke vrijednost Content-Length zaglavlja.
- Ukoliko je tip sadržaja "multipart/byteranges“ (Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES) duljina se određuje delimiterima dijelova poruke.
- Kada server zatvori konekciju. (Time se može definirati samo duljina odgovora, a ne zahtjeva jer server ne smije zatvoriti vezu dok ne pošalje odgovor.)

SLANJE PODATAKA SA FORME GET METODOM

Sadržaj forme se može slati i korištenjem GET metode. Tada podaci idu kao dio URI u formi *?podatak1=1&podatak2=2&podatak3=3*. Ovaj dio se na strani server sprema u okolinsku varijablu (*environment variable*) koju procesi koji se izvode mogu dohvatiti i pročitati njen sadržaj, ali to nije dio HTTP protokola. URI-ji koji sadrže dio ?... se nazivaju upitni URI-ji (*query* URI) i server je zadužen za to da ih ispravno procesira.

Primjer:

```
<form METHOD="GET" ACTION="http://www.fesb.hr/kiki-cgi/proba2">
<INPUT TEXT NAME="polje"><br>
<INPUT TEXT NAME="polje2"><br>
<INPUT TEXT NAME="polje3"><br>
<input type="submit" value="POKRENI PROGRAM" size="1" >
</form>
```

PUT

PUT metoda **traži od servera da** priložene **podatke pohrani pod zahtijevanim URI-jem**. Ako zahtijevani URI već postoji tada se smatra da je to zahtjev za *update-om* postojećeg URI-ja. Ukoliko resurs nije mogao biti kreiran server treba odgovoriti s određenom greškom.

DELETE

DELETE metoda se koristi da bi se serveru uputio zahtjev za brisanjem resursa identificiranog URI-jem. Klijent ne može garantirati da je akcija izvršena čak i ako je server vratio odgovor u kojem indicira da je akcija izvršena.

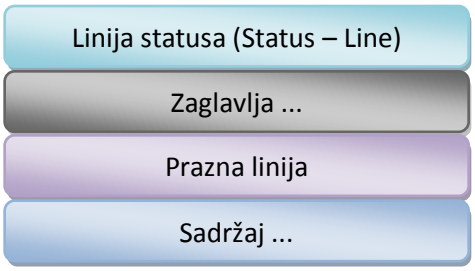
TRACE

TRACE metoda **poziva povratnu petlju poruka** zahtjeva koje je poslao klijent. Konačni primatelj zahtjeva treba primljenu poruku vratiti nazad klijentu kao podatkovni dio 200 (OK) odgovora. TRACE zahtjev omogućava klijentu da vidi što se prima od onoga što je klijent poslao na drugoj strani lanca (*request chain*) tj. što server prima.

HTTP odgovor

Server šalje klijentu HTTP odgovor.
HTTP protokol točno definira kakav format treba imati ispravni HTTP odgovor.
Format HTTP odgovora je:

- Linija statusa
- Zaglavlja
- Prazna linija
- Sadržaj poruke



STATUSNI KODOVI

Linija statusa HTTP odgovora **sadrži statusni kod** koji opisuje odgovor servera na HTTP zahtjev.

primjer: HTTP/1.0 200 OK *(„verzija“ „statusni kod“ „statusni tekstualni odgovor“)*

Format statusnog kod su tri cijela broja (npr. 200) koji identificiraju tip odgovora server. Statusni kod je namijenjen računalu dok je statusni tekstualni odgovor namijenjen ljudskom korisniku stoga web klijent ne mora razumjeti statusni tekstualni odgovor.

Trenutno je definirano 5 klasa odgovora servera (statusni kodovi 1xx, 2xx, 3xx, 4xx i 5xx).

- **1xx** (*Informational*) su Informacijski kodovi i označavaju odgovor koji se sastoji samo od linije statusa i zaglavlja.
- **2xx** (*Successful*) su Uspješni kodovi koji kažu da je zahtjev klijenta uspješno primljen i prihvaćen.
- **3xx** (*Redirection*) su Redirekcijski kodovi koji označavaju da klijent odnosno njegov korisnički agent (*User Agent*) trebaju poduzeti još neke akcije da bi dobili zahtijevani URI.
- **4xx** (*Client Error*) označavaju grešku na strani klijenta.
- **5xx** (*Server Error*) označavaju grešku na strani servera.

HTTP zaglavlja

HTTP zaglavlja se dijele na:

- općenita zaglavlja (*general-header*),
- zaglavlja zahtjeva *(request-header*),
- zaglavlja odgovora (*response-header*),
- zaglavlja entiteta tj. podatkovnog dijela HTTP poruke (*entity-header*)

Sva zaglavlja imaju isti format. Svako zaglavlje se sastoji od imena nakon kojeg ide dvotočka (":") i vrijednost zaglavlja. Imena zaglavlja su case-insensitive.

HTTP općenita zaglavlja

Općenita zaglavlja se odnose i na HTTP zahtjev i na HTTP odgovor, ali nisu vezana uz entitet tj. podatkovni dio HTTP poruke. Odnose se isključivo na HTTP poruku čiji su dio. Ovaj tip HTTP zaglavlja se može proširiti (dodati novo zaglavlje) samo promjenom HTTP protokola.

HTTP zaglavlja zahtjeva

Zaglavlja zahtjeva omogućavaju klijentu da serveru u sklopu zahtjeva pošalje dodatne informacije o zahtjevu ili o samom klijentu. Ovaj tip HTTP zaglavlja se može proširiti (dodati novo zaglavlje) samo promjenom HTTP protokola.

HTTP zaglavlja odgovora

Zaglavlja odgovora omogućavaju serveru da proslijedi dodatne informacije koje se ne mogu smjestiti u liniju statusa. Te informacije su neke informacije o serveru i o mogućnostima pristupa resursu identificiranom URI.

HTTP zaglavlja entiteta

Zaglavlja entiteta tj. podatkovnog dijela HTTP poruke definiraju opcionalne meta podatke o entitetu poruke ili ako HTTP poruka nema entiteta ti podaci su o resursu koji je identificiran URI-jem. Ovaj tip HTTP zaglavlja se može proširiti (dodati novo zaglavlje) i bez promjene HTTP protokola zahvaljujući *extension-header* mehanizmu. Međutim ne podrazumijeva se da su ta polja razumljiva primatelju HTTP poruke tj. da primatelj zna interpretirati sadržaj tog polja

HTTP 1.0 ↔ HTTP 1.1

Glavni nedostatak HTTP 1.0 protokola bila je nepostojana konekcija (*non-persistent connection*) između klijenta i servera. To je značilo da se za svaki pojedini zahtjev otvarala nova konekcija (TCP) preko koje su se slali podaci i čim bi server odgovorio na zahtjev klijent bi zatvorio vezu. Time se opterećivao server (vrijeme potrebno da uspostavi veza, potrebna memorija i sl.), a i promet na mreži je bio veći (TCP FIN i sl.).
Kod HTTP 1.1 moguće je preko jedne konekcije razmijeniti više zahtjev-odgovor parova.

MIME

MIME (*Multipurpose Internet Mail Extensions*) je standard koji se koristi da bi pretraživač znao interpretirati tj. prikazati dohvaćeni resurs. Standard je originalno napravljen za definiranje formata privitka (*attachment*) elektroničke pošte, ali se danas koristi i kod drugih Internet aplikacija. HTTP koristi Internet *Media Types* (noviji naziv za MIME *Content-Types*) (samo jedan mali dio MIME-a) kojim server i klijent dogovaraju tip podataka.
MIME povezuje niz bajtova koje klijent dobiva sa tipom kodiranja odnosno MIME "kaže" klijentu što ti bajtovi znače.

Kada HTTP server šalje informacije nazad klijentu u zaglavlju uključuje i MIME tip po kojem klijent zna o kojem se tipu informacija radi. Zaglavlje Content-Type opisuje tip sadržaja poruke ili u slučaju HEAD metode, tip sadržaja koji bi bio poslan u GET zahtjevu.

Zaglavlje Content-Encoding se koristi kao modifikator tipa sadržaja poruke. Ukoliko je zaglavlje prisutno njegova vrijednost definira dodatna kodiranja sadržaja tj. načine dekodiranja sadržaja kako bi se dobio originalni sadržaj čiji tip je naveden u polju Content-Type.

Format MIME tipova: text/plain, text/html, image/gif, application/pdf, …
Format kodiranja: base64, x-gzip
Dokument se pretraživaču isporuči sa ove dvije informacije.
Za tip imamo tri mogućnosti

- pretraživač zna sam prikazati tip (HTML, GIF,..)
- pretraživač zna koja aplikacija može prikazati dokument te je pozove.
- pretraživač ne zna kako da prikaže određeni tip i ponudi korisniku da dokument spremi na disk.

HTML, DTD, CSS

HTML

HTML (*HyperText Markup Language*) je jezik koji se koristi za publiciranje informacija na World Wide Web-u. *Markup* jezik je jezik koji kombinira sadržaj koji se objavljuje sa dodatnim podatcima o tom sadržaju(npr. boja teksta, veličina slova, položaj informacije u pregledniku i sl.)
Najpoznatiji *markup* jezik je HTML za publiciranje na Web-u. *[XML, XHTML, LaTeX….]*
Trenutna verzija HTML standarda je 4.01, a standardizacijom HTML-a bavi se W3 konzorcijum.
HTML je "izmislio" **Tim Berners-Lee** postavljajući temelje www-a.

W3C (*World Wide Web Consortium*) ima za cilj razvoj Web-a i otvorenost Web-a.

SGML

SGML (*Standard Generalized Markup Language*) je markup jezik koji se koristi za definiranje drugih markup jezika. HTML je definiran preko SGML-a. Markup jezik definiran u SGML-u zove se SGML aplikacija, koja sadržava:

- **SGML deklaraciju** – SGML deklaracija sadrži leksičku osnovu SGML dokumenta (koji se karakteri mogu pojavljivati u SGML aplikaciji, kako se označavaju markup delimiteri i sl.).
- **DTD (*Document Type Definition*)** – DTD definira sintaksu markup jezika tj. skup pravila koja dokument treba poštivati da bi bio ispravan (*valid*) prema shemi određenoj DTD-om.
- **Specifikaciju** koja opisuje semantiku jezika.
- **Instance dokumenta** koji sadrže podatke i oznake (*markup*). Svaka instanca sadrži referencu na DTD koji se koristi za interpretiranje dokumenta.

SGML ↔ DTD

Na početku svakog dokumenta trebalo bi specificirati DTD prema kojem je taj dokument napisan. DTD (*Document Type Definition*) je jedan od SGML i XML shema jezika kojim se opisuje dokument. Na taj način SGML parser zna prema kojem DTD-u treba provjeriti da li je dokument ispravno napisan. Sintaktičkom analizom ili *parsiranjem* tokeni (znakovi dobiveni leksičkom analizom) se organiziraju (obično u formi stabla) u gramatički ispravnu strukturu.

DTD

DTD se definira obično u jednoj liniji na početku dokumenta u DOCTYPE (*document type*) deklaraciji. Tipična deklaracija izgleda:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">

- Znak <! se koristi za označavanje SGML deklaracije.
- DOCTYPE opisuje da se radi o SGML deklaraciji tipa dokumenta.
- Oznaka HTML se odnosi na tip dokumenta i oznaku prvog elementa u dokumentu.
- PUBLIC je način pristupa DTD-u koji se koristi u dokumentu.
- "-//W3C//DTD HTML 4.0//EN" je FPI (*Formal Public Identifier*) koji parser koristi za pronalaženje odgovarajućeg DTD-a prilikom obrade dokumenta (FPI format "*Owner//Keyword Description//Language*").

Ispravan (*valid*) HTML dokument sadrži deklaraciju verzije HTML korištene u dokumentu. Deklaracija tipa dokumenta sadrži DTD koji se koristi u stranici. HTML 4.01 specificira tri DTD.

- HTML 4.01 Strict DTD
- HTML 4.01 Transitional DTD, tj. LOOSE
- HTML 4.01 Frameset DTD

HTML

HTML dokument sastoji se od tri dijela:

- **deklaracije tipa dokumenta**,
- deklarativnog zaglavlja ograničenog elementom **HEAD**,
- tijela - **BODY** ili **FRAMESET**

Prazna mjesta i linije dozvoljeni su bilo gdje u HTML dokumentu.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>My first HTML document</TITLE>
</HEAD>
<BODY>
<P>Hello world!
</BODY>
</HTML>
```

Linkovi

Link uvijek ima dva kraja. Počinje od izvorišnog sidra i upućuje na odredišno sidro što može biti bilo koji web resurs (HTML stranica, slika, …) adresiran URI-jem. Osnovno ponašanje linka se očituje kao dohvaćanje resursa na koje link upućuje.
Tipovi linkova mogu biti: Alternate, Stylesheet, Start, Next, Prev, Contents, Indeks, Glossary, Copyright, Chapter, Section, Subsection, Appendix, Help, Bookmark.
Npr. <link rel="stylesheet" href="boja_pozadina.css" type="text/css">
Atributi rel i rev koriste se za navođenje tipa linka.

HTML elementi LINK i A omogućavaju povezivanje dokumenata odnosno kreiranje hiperteksta (premda se link definira i preko IMG elementa i preko FORM elementa).
Atribut href (i LINK i A elementa) sadrži vrijednost odredišta linka tj. URI.
<A href="http://www.fesb.hr/">FESB</A>

URI može biti apsolutan URI (http://www.fesb.hr/~kiki/moja_stranica.htm), relativan URI (/~kiki/moja_stranica.htm), kao i link definiran unutar iste stranice (URI fragment #nešto).

Veličina

HTML specificira tri tipa vrijednosti veličine HTML elemenata:

- **Piksel (*pixels*):** Vrijednost (%Pixels; u DTD-u) je cjeli broj koji predstavlja broj piksela kojim će biti iscrtan element. (npr. font-size: 19px;)
- **Duljina (*length*):** Vrijednost (%Length; u DTD-u) može biti ili piksel ili postotak vertikalne ili horizontalne dimenzije. Tako vrijednost "50%" znači da će veličina elementa biti pola od raspoložive veličine.
- **Višestruka duljine (*multilength*):** Vrijednost (%MultiLength; u DTD-u) može biti ili duljina ili relativna duljina u formi zapisa "i*", gdje je "i" cijeli broj. Prilikom iscrtavanja elementima se veličina određuje na osnovu raspoloživog prostora * i. Npr. 20 piksela * 2 = 40 piksela.

Boje

Vrijednost atributa tipa boja (%Color;) odnosi se na boju specificiranu u "A Standard Default color Space for the Internet - sRGB". Prostor boja (*color space*) je model za predstavljanje boja numerički sa tri ili više koordinata npr. RGB prostor boja je određen sa Crvenom, Zelenom i Plavom koordinatom. Vrijednost boje se zapisuje kao heksadecimalni broj (na početku s hash znakom) ili nekim od 16 definiranih imena boja. Imena boja su *caseinsensitive*.

Imena okvira

Kada u stranici koristite okvire (*frame*) kako biste mogli pristupati pojedinom okviru trebate atributu NAME dodijeliti vrijednost preko koje možete pristupati tom okviru.
Ime okvira (%FrameTarget; u DTD-u) mora počinjati sa slovima (a-z, A-Z). Inače ga korisnički agent treba ignorirati. Ova se vrijednost također koristi u HTML elementima A, LINK, AREA, FORM, … kao vrijednost atributa TARGET kako bi se definiralo u kojem okviru ili prozoru se dohvaća informacijski resurs identificiran u elementu.
Sljedeća imena su rezervirana i imaju specijalno značenje:

- _blank
- _self
- _parent
- _top

XHTML

XHTML (*Extensible HyperText Markup Language*) je skup trenutnih i budućih specifikacija koji proširuju HTML tipove dokumenata i redefiniraju ih pomoću XML-a. XHTML se sastoji od svih elemenata iz HTML 4.01 kombiniranih sa sintaksom XML-a (*eXtensible Markup Language*). XHTML je baziran na XML-u (nasljednik HTML-a). XHTML postavlja stroža pravila nego HTML.

CSS

CSS (*Cascading Style Sheets*) je jezik stila (*stylesheet language*) koji se koristi za opisivanje prezentacije tj. izgleda dokumenta napisanog u *markup* jeziku.
Jezik stila je računalni jezik za opisivanje stila elemenata u dokumentu napisanom u *markup* jeziku. (XSL – *Extensible StylSheet Language*, …) CSS se obično koristi s HTML-om i XHTML-om za opisivanje stila stranice, ali može se primijeniti i na dokumente napisane u drugim *markup* jezicima npr. XML-u. Svi glavni pretraživači podržavaju CSS.

Stilovi se obično spremaju u vanjske .css datoteke.Vanjske stranice stilova omogućavaju korisnicima upravljanje izgledom više web stranica preko samo jednog CSS dokumenta.
Uključivanje CSS dokumenta u HTML dokument:

```
<link rel="stylesheet" href="example.css" type="text/css">
ili
<style type="text/css">
@import "example.css";
</style>
```

CSS kod se može umetati u HTML kod na dva načina.

- Jedan je ograničavanje CSS koda unutar **HEAD** elemenata tagovima **<style>**.

```
<style type="text/css">
 p {color: red; }
 body {background-color: green; }
</style>
```

- Drugi način je umetnuti CSS koji se može preko **atributa style** umetati u bilo koji HTML element.

```
<p style="background: blue; color: white;">Umetnuti CSS</p>
```

CSS primjenjuje pravilo kaskade, te nasljeđivanje, specifičnost i lokaciju da bi odredio koje će među sukobljenim pravilima pobijediti. Osnovno je pravilo da je među ravnopravnim stilovima najjači onaj koji je na posljednjem mjestu.
Sve u svemu, u odsutnosti pravila, djeca nasljeđuju od roditelja brojne stilove. Pri srazu dva pravila, specifičnije je pravilo važnije – bez obzira na lokaciju. Ako su oba pravila jednako specifična, posljednje pravilo pobjeđuje.
Općenito, svi stilovi će se kaskadno povezati u novu virtualnu stranicu stilova prema sljedećim pravilima, gdje broj četiri ima najveći prioritet:

- unaprijed zadano u pretraživaču;
- vanjska stranica stilova;
- unutarnja stranica stilova (unutar *<head>* taga);
- lokalna primjena stilova (unutar HTML elementa).

CSS ima jednostavno sintaksu, koristi riječi engleskog jezika za definiranje vrijednosti stila.
CSS sadrži niz pravila. Svako pravilo sadrži jedan ili više selektora (*selectors*) međusobno razdvojenih zarezom i deklaracijskih blokova (*declaration block*). Deklaracijski blok se sastoji od liste deklaracija ograničenih zagradama ({}) i odvojenih točkazarezom
(;). Svaka deklaracija se sastoji od svojstva, znaka (:) i vrijednosti svojstva.

Prednost korištenja CSS: Prezentacijske informacije o kolekciji stranica su na jednom mjestu i lako se mogu mijenjati, za različite korisnike može se koristiti različiti CSS, veličina i složenost HTML dokumenta su smanjeni jer je prezentacijska informacija pohranjena u zasebnom dokumentu.

DHTML

DHTML (*DynamicHTML*) nije W3 konzorcijum standard već je to pojam koji su uveli Netscape i Microsoft za označavanje kombinacija novih tehnologija za kreiranje dinamičkih Web stranica na strani klijenta. DHTML se odnosi na kombinaciju HTML 4.0, CSS-a, DOM-a i JavaScript-a.

DOM

DOM (*DocumentObjectModel*) je W3 konzorcijum standard koji definira sučelje neovisno o platformi i jeziku koje omogućava programima i skriptama da dinamički pristupaju i mijenjaju sadržaj, strukturu i stil dokumenta.
DOM pruža standardni skup objekata za predstavljanje HTML i XML dokumenata i standardno sučelje za pristup i manipulaciju njima.
Preko DOM-a dokumentu se pristupa u formi stabla. Takva implementacija zahtjeva da čitav dokument bude učitan u memoriju i parsiran.

DOM se sastoji od nekoliko dijelova:

- **Core DOM** –definira standardni set objekata za bilo koji strukturirani dokument
- **HTML DOM** -definira standardni set objekata za HTML dokumente
- **XML DOM** – definira standardni set objekata za XML dokumente
- **DOM** događaji (***events***) – definira standardni set događaja
- **DOM CSS** –ovo sučelje je definirano kako bi se omogućio pristup CSS elementima

DOM razine:

1. se odnosi na samu osnovu HTML modela dokumenata. Sadrži funkcionalnost za navigaciju i manipulaciju dokumentima..
2. uključuje „style sheet object model“, i definira funkcionalnost za mijenjanje informacije o stilu pridijeljenom dokumentu. Definira model događaja i omogućava podršku za XML.
3. adresira učitavanje i spremanje dokumenta, kao i sadržaj modela s podrškom za potvrdu dokumenta. Također adresira pogled dokumenta (*document view*), formatiranje dokumenta.

Mehanizam prikaza (*layout engine, rendering engine*) je softver koji dohvaća web sadržaj (HTML, XML, slike,…) i prikaz sadržaja (CSS, XSL,…) i prikazuje formatirani sadržaj na ekranu. Mehanizam prikaza “iscrtava” sadržaj prozora prikazanog na ekranu ili printeru (Trident, Gecko…)

CoreDOM

DOM predstavlja dokument kao hijerarhijski organizirane objekte čvorova (*nodes*) koji mogu implementirati određena sučelja. Neki tipovi čvorova mogu imati čvorove djecu, a neki ne. Za HTML i XML tipovi čvorova su:
Dokument (*Document*) –je osnovni čvor XML ili HTML dokumenta koji predstavlja čitav dokument, Fragment dokumenta (*DocumentFragment*), Tip dokumenta (*DocumentType*), Referenca entiteta (*EntityReference*), Element, Atribut (*Attr*), Naredba (*ProcessingInstruction*), Komentar (*Comment*), Tekst, CDATASection, Entitet (*Entity*), Notacija (*Notation*).

HTML DOM

HTML DOM definira HTML dokument kao kolekciju objekata. Objekti imaju svojstva i metode. HTML DOM vidi HTML dokument kao stablastu strukturu elemenata ugrađenih unutar drugih elemenata. Svim elementima, tekstu koji sadrže i njihovim atributima, može se pristupiti preko DOM stabla. Njihov sadržaj može biti mijenjan i izbrisan, te se mogu kreirati novi elementi. Svi čvorovi u HTML dokumentu tvore stablo dokumenta (ili stablo čvorova).

XML DOM

XML DOM definira standardni način pristupa i manipulacije XML dokumenta. XML DOM vidi HTML dokument kao stablastu strukturu elemenata ugrađenih unutar drugih elemenata. Svim elementima, tekstu koji sadrže i njihovim atributima, može se pristupiti preko DOM stabla. Njihov sadržaj može biti mijenjan i izbrisan, i mogu se kreirati novi elementi. Stablo ima jedan *root* čvor. Svi čvorovi osim *root* čvora imaju jedan čvor roditelj. Svaki čvor može imati više čvorova djece. I atributi se promatraju kao čvorovi XML DOM stabla.

JAVASCRIPT

JavaScript je klijentski skriptni objektno-orijentirani programski jezik. Koristi se kako bi se HTML stranicama dodala interaktivnost i dinamika:

- JS može reagirati na događaje
- JS može čitati i mijenjati sadržaj HTML elemenata
- JS se može koristiti za analizu (*validate*) forme podataka

Za uključivanje JS u kod HTML stranice koristi se tag *<script>*. Unutar stranice može se nalaziti proizvoljan broj skripti. Skripte se mogu uključivati unutar HEAD dijela HTML stranice ili unutar BODY dijela stranice. Skripta se može smjestiti i u zasebnu datoteku s ekstenzijom .js.

```
<html>
<head>
<script type="text/javascript">
 JavaScriptnaredbe
</script>
</head>
```

```
<body>
<script type="text/javascript">
JavaScriptnaredbe
</script>
</body>
</html>

<script src="xxxx.js"></script>
```

U JavaScriptu je moguće kreirati **tri vrste privremenih prozora**:

- ***AlertBox***(prozor za upozoravanje, OK) Sintaksa: alert("sometext")
- ***ConfirmBox***(prozor za potvrdu, OK i Cancel) Sintaksa: confirm("sometext")
- ***PromptBox***(prozor za unos, OK i Cancel) Sintaksa: prompt("sometext","defaultvalue")

Funkcije se definiraju na početku dokumenta, u HEAD dijelu, na taj način osiguravamo da se funkcija učita prije njenog poziva. Definiranje funkcija vrši se tako da se funkciji dodjeli ime, definiraju argumenti i u tijelu funkcije napišu sve naredbe za koje želimo da se izvrše kada se funkcija pozove.

XML

XML (*eXtensibleMarkup Language*) – je *markup* jezik (jezik koji kombinira sadržaj sa dodatnim podatcima o tome sadržaju) koji se koristi za strukturiranje, pohranjivanje i slanje podataka. XML standard je razvio W3C da bi omogućio jednostavan, otvoren i standardiziran način pohrane samo-opisujućih podatka (*self-describing data*) (podaci opisuju i svoj sadržaj i svoju strukturu).

XML format zapisa podržavaju gotovo sve aplikacije i implementiran je u gotovo svim programskim jezicima. Do XML-a nije postojao format zapisa podataka koji je bio toliko proširen i prihvaćen.

Prednosti XML-a: Format je prilagođen čovjeku, podržava Unicode pa se može koristiti sa bilo kojim ljudskim jezikom, može sadržavati podatkovne strukture poput liste i stabla, stroga sintaksa čini parsiranje jednostavnim i efikasnim, proširen je, baziran je na međunarodnim standardima, XML je tekstualna datoteka (takav format je manje restriktivan od binarnog), neovisan je o platformi, pa ga je lakše prenijeti na nove tehnologije (npr. mobitel).

Nedostatci XML-a: Podaci zapisani prema XML sintaksi su redundantni i zauzimaju više mjesta u odnosu na zapis u binarnom obliku, XML sintaksa je preopširna, ne postoji notacija za označavanje tipa podatka poput datuma, stringa i sl., hijerarhijski model XML je ograničen u odnosu na relacijski model podataka, XML imenski prostori su složeni za implementaciju u parseru, izražavanje veza među XML čvorovima koje nisu hijerarhijske nije definirano u XML-u.

Ispravnost XML dokumenta definira se na dva načina:

- **Ispravno formatiran (*Well-formed*)** XML dokument prilagođen je sintaktičkim pravilima XML. (tagovi pravilno otvoreni i zatvoreni)
- **Ispravan (*Valid*)** XML dokument osim poštivanja sintaktičkih pravila poštuje i semantička pravila koja su obično uključena u XML shemu dokumenta. (nedozvoljeni tagovi koji nisu definirani u XML shemi)

```
<?xmlversion="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE note [
        <!ELEMENT STUDENTI (OSOBA)>
        <!ELEMENT OSOBA (IME, GODINA_STUDIRANJA)>
        <!ELEMENT IME (#PCDATA)>
        <!ELEMENT GODINA_STUDIRANJA(#PCDATA)>
]>
<STUDENTI>
 <OSOBA>
(ostatak xml-a)...
```

Uključivanje vanjskog DTD dokumenta

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Binarni podaci u XML-u

XML dokument je tekstualni dokument. Kada se u XML dokument uključuju binarni podaci potrebno je koristiti **base64** shemu kodiranja. Shema koristi skup od 64 karaktera (A-Z a-z 0-9 +/) za predstavljanje binarnih podataka plus znak "=" za nadopunu (*padding*).

Ukoliko se radi o slici, slika se može prikazati u pretraživaču korištenjem **XSL (*Extensible Stylesheet Language*)** jezika za formatiranje XML podataka u HTML format ili neki drugi format. Naime u XSL dokumentu, u *img* tagu kao izvor slike navodi podatkovni URI (*dataschemeURI*). **Podatkovni URI** je specijalni oblik URI-a kod kojeg se podatci sadržani u resursu koji se dohvaća navode u samom URI-ju. Kada se koristi podatkovni URI i ako pretraživač podržava podatkovni URI (većina pretraživača da, ali ne i IE) ponašanje je jednako kao da je resurs uključen običnim URI-jem. Prvi dio URI-ja je **shema**, pa opcionalno **MIME tip**, pa opcionalno **kodiranje** i na kraju **podaci**.

primjer: …

Dizajniranje XML jezika ili "dijalekata" je jednostavno. Primjeri formalno definiranih jezika baziranih na XML-u su **RSS (*Really Simple Syndication*)** jezik za opisivanje sadržaja web stranica, **MathML** (*Mathematical Markup Language*) za predstavljanje matematičkih izraza, XHML, …
Za definiranje sintakse XML dokumenta, tj. **za provjeru ispravnosti** (validacija) XML dokumenta razvijen je niz shema jezika, ali se najčešće koriste **DTD (*DocumentTypeDefinition*)**, **W3C XML Shema (WXS)** (drugi naziv je **XSD (*XML SchemaDefinition*)**) i **RELAX NG**.
XSD je novija verzija DTD-a, koristi se za opisivanje strukture i sadržaja XML dokumenata.

XML prolog

Primjeri:

- *<?xml version="1.0" encoding="ISO-8859-1"standalone="no" ?>*
- *<?xmlversion="1.0"?><!DOCTYPE htmlPUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">*

XML dokument započinje sa dijelom koji se zove XML prolog. Prolog se sastoji od XML deklaracije i/ili (opcionalno) DTD. XML deklaracija sadrži **verziju XML-a** te opcionalno **tip kodiranja** dokumenta i ***standalone* deklaraciju** (da li se ovaj dokument veže na neki način sa drugim dokumentom).
Tagovi u HTML-u su predefinirani, dok **kod XML-a korisnik sam definira *tagove*** koji su mu potrebni za strukturiranje podataka.

XML imenski prostor

Zbog mogućnosti kolizije među oznakama elemenata koriste se XML imenski prostori. To je poseban tip rezerviranog atributa koji se smješta u XML tag.
primjer: <nekitag xmlns:moj="Neki moj prostor">

XSL

XSL (*EXtensibleStylesheetLanguage*) je XML *Style Sheets* i opisuje kako XML dokument treba izgledati. XSL je jezik za prevođenje XML-a u XHTML, kojim se mogu filtrirati, sortirati i formatirati XML podatci.

Uključivanje XSL u XML dokument radi se na početku XML dokumenta iza XML prologa:
<?xmlversion="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheettype="text/xsl" href="ime.xsl"?>

XSL *stylesheet* procesor prihvaća dokument ili podatke u XML-u i jedan XSL *stylesheet* i stvara prezentaciju XML izvornog sadržaja na način definiran u XSL *stylesheetu*. Proces stvaranja prezentacije se sastoji od dva dijela:

- Prvi dio je kreiranje rezultirajućeg stabla na osnovu izvornog XML stabla.
- Drugi dio je formatiranje rezultirajućeg stabla u prezentaciju prilagođenu mediju prezentacije (na ekran, na papir, u govorni oblik ili drugi medij).

XSL se sastoji od tri dijela:

- XSLT – jezik za prevođenje XML dokumenata
- XPath – jezik za navigaciju u XML dokumentima
- XSL-FO (xsl formatting) - sintaksa i semantika XSL-a za formatiranje XML dokumenata

XPath

XPath je W3C standard sa sintaksom za definiranje određenih dijelova XML dokumenta. Koristi izraze koji pokazuju putanju za navigaciju unutar XML dokumenta. Pomoću tih izraza selektira čvorove i njihove podskupove unutar XML dokumenta.

XPath predikati

Predikati se koriste za dohvaćanje specifičnog čvora ili čvora koji sadrži specifični vrijednost. Uvijek se navode u uglatim zagradama.

Npr. [last()] –dohvaćanje zadnjeg čvora, [ime='Toni'] – dohvaćanje čvora koji u čvoru ime sadrži vrijednost Toni, [@date] –dohvaćanje čvora koji sadrži atribut date,…

Uz Xpath su usko vezana tri jezika:

- **XQuery(*XML Query Language*)** - jezik za izvršavanje upita nad XML dokumentima.
- **XLink(*Xml Linking Language*)** - jezik za kreiranje hiperlinkovau XML dokumentima.
- **XPointer(*XML Pointer Language*)** - jezik koji omogućava kreiranje hiperlinkova koji pokazuju na točno određene dijelove XML dokumenta.

SOKET

Soket je apstrakcija krajnje komunikacijske točke. Omogućava dvosmjernu komunikaciju od točke do točke (point-to-point) između dva programa. To je metoda komunikacije između programa klijenta i programa servera na mreži.

Soket je **TCP/IP API**. Podržan je od većine OS-ova i programskih jezika.
Nedostatci: složenost, podložnost koda greškama, svaki dio komunikacije treba biti eksplicitno programiran

TCP/IP API

TCP/IP protokol softverski je sastavni dio operacijskog sustava. Operacijski sustav programeru pruža API preko kojeg programer koristit TCP/IP protokol. TCP/IP standard ne specificira kako se aplikacijski softver povezuje sa softverom TCP/IP protokola već samo predlaže funkcionalnost koju treba imati sučelje između aplikacije i softvera TCP/IP protokola.

Prednost: fleksibilnost i prilagodljivost različitim OS-ovima
Nedostatak: Detalji sučelja prema TCP/IP softveru razlikuju se od jednog do drugog OS-a.

Često korišteni API za TCP/IP:

- Berkeley UNIX socket Interface
- System V UNIX Transport Layer Interface (TLI)
- Windows Sockets Interface
- MacTCP

Soket

Soket sučelje za TCP/IP protokol je razvijeno na Berkeley-u za UNIX. Od 1993. se koristi i na Windows operacijskim sustavima pod imenom WinSock. To je samo implementacija Berkeley soketa na Windows OS.

Tipovi soketa:

- **TCP** soket (*stream socket*)
- **UDP** soket (*datagram socket*)
- **IP** soket (*row socket*) (ne na Winsock-u)

Soket je posebni tip *handla* na file koji proces koristi da bi zatražio neku mrežnu uslugu od operacijskog sustava.

Funkcije soketa

Da bi aplikacija mogla koristiti soket treba ga kreirati **funkcijom**:

*int **socket**(int family,int type,int proto); UNIX*
*SOCKET **socket**(int af, int type, int protocol); Windows*

Prvi argument određuje "obitelj" protokola, sljedeći argument određuje tip soketa(SOCK_STREAM, SOCK_DGRAM, SOCK_RAW(samo Unix)). Zadnji argument definira protokol i obično se postavlja u 0 koji znači korištenje *defaultnog* protokola.

Sljedeći korak na strani servera (soketi su server/klijent metoda mrežnog IPC-a) je vezivanje **(*binding*)** soketa na određenu IP adresu i port na kojem će soket biti otvoren:

*int **bind**(int sockfd, const struct sockaddr* name, int localaddrlen);*
*int **bind**(SOCKET s, const struct sockaddr* name, int namelen);*

Prvi argument funkcije je handle na soket, zatim ide pokazivač na strukturu koja sadrži adresu na koju se soket vezuje i zadnji argument je veličina adresne strukture (obično se samo stavi sizeof(sockaddr)). Funkcija vraća 0 u slučaju uspješnog *binding*, a -1 u slučaju greške.

Struktura ***sockaddr*** je generička struktura soket adrese definirana ANSI C standardom. Tip *sockaddr* automatski određuje strukturu adrese ovisno o postavljenoj "obitelji" adresa.

Mrežni redoslijed bajtova

IP adresa i broj porta u adresnim strukturama soketa trebaju biti zapisani u mrežnom redoslijedu bajtova (*network byte order*). Npr. IP adresa 161.53.168.65 ima 32 bita odnosno 4 bajta i može se u memoriji zapisati na dva načina.

- Prvo se spremi najznačajniji bajt (*MS-most significant*) → ***Big-endian*** zapis.
- Prvo se spremi najmanje značajan bajt (*LS-leastsignificant*) → ***Little-endian*** zapis

PC arhitektura s Intelovim procesorom koristi *little-endian* zapis. RISC arhitekture i *mainframe* računala koriste *big-endian* zapis**. Kao standard na mreži je prihvaćen *big-endian*** zapis i svi paketi koji idu preko mreže trebaju imati zapisanu IP adresu (i port) u ovom obliku.

Adresa i port

Možete zatražiti od OS da procesu dodijeli adresu i slobodni port. Za serverske aplikacije obično je definirano na kojem portu "slušaju" (npr. HTTP server –80) pa trebaju koristiti upravo taj port jer će klijentske aplikacije pokušati otvarati konekciju prema serveru na tom portu.

Klijent

Nakon kreiranja soketa i vezivanja IP adrese i porta uz soket deskriptor, **klijent** za TCP konekciju pokušava uspostaviti konekciju prema nekom serveru funkcijom:

*int **connect**(int sockfd, const struct sockaddr* name, int namelen);*

Prvi argument funkcije je soket deskriptor, drugi argument je pokazivač na strukturu s podacima o serveru (IP adresa i port) prema kojem se uspostavlja konekcija, a zadnji je argument veličina strukture.

Server

Nakon kreiranja soketa i vezivanja IP adrese i porta uz soket handle ili soket deskriptor, **server** za TCP konekciju"osluškuje" dolazne zahtjeve za konekcijama od strane klijenata funkcijom:

*int **listen**(int sockfd, int backlog);*

Prvi argument funkcije je soket deskriptor, a drugi argument je maksimalni broj konekcija koji se može otvoriti na tom soketu.

Nakon što je **server** preko *listen* funkcije dobio zahtjev od klijenta za otvaranje konekcije ukoliko prihvaća klijentov zahtjev konačno se konekcija uspostavlja funkcijom:

int accept(int sockfd, struct sockaddr addr, int* addrlen);*

Prvi argument funkcije je soket deskriptor, drugi argument je pokazivač na strukturu s podacima o klijentu koji se spojio i treći argument je pokazivač na duljinu strukture koji je opcionalan.

Za **zatvaranje soketa** koristi se funkcija: *int **close**(int sockfd);*

MEĐUPROGRAM (*Middleware*)

Međuprogram *(middleware)* je softver koji olakšava izradu distribuiranih aplikacija(aplikacija čije se softverske komponente nalaze na više računala u mreži) pružajući standardne mehanizme kojima distribuirane komponente mogu komunicirati preko mreže. To je program koji djeluje između aplikacije i mreže. To je softver koji omogućava i olakšava komunikaciju između softverskih procesa i pruža programsku apstrakciju te skriva od programera heterogenost mreže.

IPC

Komunikacija dva softverska procesa naziva se interproces komunikacija ***(Interprocess communication – IPC)***. IPC se odnosi i na komunikaciju dva procesa koja se izvršavaju na istom računalu i na dva procesa koja se izvršavaju na različitim računalima.

Interproces komunikacijske paradigme

Komunikacija između distribuiranih procesa zasniva se na modelu prenošenje poruka *(message passing).* Struktura poruke, te način njezine interpretacije određeni su unutar procesa koji rade komunikaciju. Tako poruka može biti samo niz bajtova (soket), a može biti i poziv metode udaljenog objekta.

Osnovna pitanja koja trebaju biti riješena pri komunikaciji dva procesa su:

- Direktno ili indirektno adresiranje procesa
- Blokirajuća ili neblokirajuća komunikacija
- Pouzdana ili nepouzdana komunikacija
- Komunikacija s korištenjem bafera ili bez korištenja bafera

Interproces komunikacijske tehnologije

- Soketi
- Pozivi udaljenih procedura (*RPC – Remote Procedure Call*)
- Distribuirani objekti (*RMI - Remote Method Invocation, DCOM - Distributed component object model, CORBA - Common Object Request Broker Architecture,…*)
- MOM (*message oriented middleware*)
- …..

Klijent-server paradigma

Široko rasprostranjena za mrežne aplikacije i mrežne servise. Jednostavna u suštini: dijelovi sistema se predstavljaju ili kao serveri (oni koji nude servise) ili kao klijenti (oni koji traže servise). Mnogi Internet servisi (FTP, HTTP) su tipične klijent-server aplikacije.

Pozivi udaljenih procedura (RPC)

Pojavljuje se u 80-tima, dva API-ja dominiraju:

- *Open Network Computing* ONC - RPC (Sun)
- *Open Group Distributed Computing Envinronment* DCE - RPC

Pristup koji dozvoljava da se distribuirane aplikacije razvijaju korištenjem koncepta proceduralnog programiranja-uobičajenim apstrahiranjem i komunikacije i sinkronizacije. Operacije na udaljenom računalu izvode se kao lokalne.

S RPC-om proces na jednom računalu može pozvati proceduru procesa na drugom računalu. Sve RPC implementacije imaju jezik kojim se apstraktno, neovisno o jeziku u kojem će biti implementirana, opisuje RPC usluga (procedura,metoda).
Taj jezik se generički naziva definicijski **jezik sučelja IDL** (*interface definition language*) (npr. Microsoft je implementirao DCE RPC IDL s određenim proširenjima pod imenom MIDL – *Microsoft Interface Definition Language*).

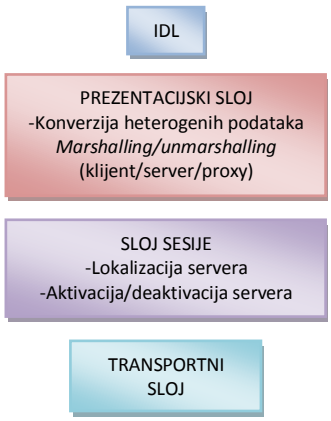
IDL-om se sve usluge (tj. procedure) definiraju preko njihovog imena i ulaznih i izlaznih parametara. Nakon toga se koristi kompajler koji taj IDL (Sunov ili DCE-ov) zna, za programski jezik koji se koristi za pisanje koda, prevesti u tzv. zamjenske programske elemente (*stubs*) na strani klijenta i na strani server koji omogućavaju da klijent poziva udaljene funkcije na serveru kao da su lokalne (*rpcgen* za SUN RPC, *midl.exe* za Microsoft Visual Studio).

RPC mehanizam omogućuje RPC softver koji uključuje sljedeće usluge:

- Lokalizacija serverskih funkcija i zahtjeva različitih klijenata
- Prosljeđivanje parametara i predstavljanje podataka
- Obrada grešaka
- Sigurnosne provjere

Prezentacijski sloj omogućava konverziju i formatiranje podataka (*marshalling/unmarshalling*). Postupak *marshallinga* je konverzija podataka iz lokalnog formata u eksterni format, a postupak *unmarshalling* je konverzija podataka u obrnutom smjeru iz eksternog formata u lokalni format.

- ONC RPC koristi XDR (*External Data Representation*) eksterni format podataka.
- DCE RPC koristi NDR (*Network Data Representation*) eksterni format.
- CORBA koristi *Common Data Representation* (CDR)/IDL eksterni format.
- Java koristi serijalizaciju objekata (*object serialization*) za formatiranje podataka za prijenos preko mreže između dva procesa.



Sloj sesije omogućava klijentima lokalizaciju tj. pronalaženje RPC servera, te aktivaciju RPC servera kada zahtjev pristigne odnosno deaktivaciju RPC servera nakon obrade zahtjeva i odgovora klijentu.

RPC

Prednosti: Jednostavnost, lakše programiranje.
Nedostatci: Zahtjeva dosta procesorske snage, velika pojasna širina komunikacije.

ORPC (Objektni RPC)

Distribuirana aplikacija se strukturira kao kolekcija objekata. Servisi i resursi dohvatljivi su kao objekti. Pozivanjem metoda objekata aplikacije pristupaju servisima.

Razrješavanje složenosti

Složenost vezana za heterogenost programskih jezika, može se pojaviti jer različiti jezici imaju različite konstrukcije i osobine ili im je strojni kod različit. Međuprogramski sloj podržava zajednički model objekta i IDL koji predstavlja ključ za razrješavanje programske heterogenosti. To se postiže odgovarajućim vezama odgovarajućeg programskog jezika s IDL-om. Definiranje veza za neki programski jezik znači specificirati kako se IDL konstrukcije mogu koristiti za serversku ili klijentsku implementaciju i obrnuto.

Jezično vezivanje definira direktno mapiranje (jedan na jedan) između konstrukcija jezika i IDL konstrukcija. Objektni tipovi mapiraju se i na serverskoj i na klijentskoj strani (na *stub-u*). Reference serverskih objekata enkapsulirane su u klijentskom stubu. Operacije se mapiraju u procedure, operacije ili metode u programskom jeziku. Izuzeci se mapiraju u izuzetke ili izlazne parametre koji se moraju vrijednovati. Višestruko nasljeđivanje rješava se ovisno o odgovarajućem modelu nasljeđivanja određenog programskog jezika. Isporučitelji implementiraju povezivanje osiguravanjem API-ja koji se mogu koristiti i na klijentskoj i na serverskoj strani i kompajlere koji generiraju klijentske i serverske *stubove*.

Corba (OMG – Object Management Group)

To je univerzalni, OO međuprogram koji programerima omogućava pisanje objekata koji komuniciraju s drugim objektima bez da znaju gdje se objekti nalaze i kako su implementirani. Podržava izradu i integraciju objektno orijentiranih softverskih komponenti u heterogeno distribuiranim okruženjima.
Zbog toga, okruženje sa samo Windows komponentama možda neće trebati CORBA-u.

IIOP (*Internet Inter-ORB Protocol*) je protokol koji je razvila OMG grupa kako bi se CORBA mogla implementirati preko World Wide Web-a. IIOP omogućava pretraživačima i serverima da razmjenjuju objekte dok HTTP omogućava samo razmjenu teksta.
Spremište sučelja (*Interface repository*) je izvršno spremište meta podataka registriranih IDL-definiranih sučelja koje sadrži imena metoda i tražene parametre. To dinamički vezano sučelje je CORBA sučelje ili API za traženje podataka koji se nalaze u skladištu sučelja. Sučelje se koristi dinamički za generiranje poziva udaljenih metoda.

Implementacijsko spremište (*Implementation repository*) je izvršna baza podataka koja pohranjuje sve registrirane objekte nekog ORB-a, bilo da su aktivirani ili dostupni za aktivaciju nakon klijentskog zahtijeva.
IDL stubovi omogućavaju pristup do IDL definiranih operacija na nekom objektu. Preko ORB sučelja i klijenti i objektne implementacije mogu dosegnuti neke zajedničke operacije. Većina ORB funkcionalnosti ostvaruje se preko IDL stubova ili preko sučelja za dinamičke pozive (*dynamic invocation interface*), okosnica (*skeleton*) ili objektnih prilagodnika (*object adapter*).
Statička IDL okosnica je sučelje prema objektno implementiranim metodama.
Dinamička IDL okosnica omogućava povezivanje tijekom izvođenja i namijenjen je serverima koji moraju voditi računa o ulaznim pozivima za komponente koje nemaju IDL-utemeljene kompajlirane okosnice. Dinamička okosnica uzima vrijednosti parametara neke ulazne poruke kako bi utvrdila ciljni objekt i metodu. ***Basic Object Adapter*** predstavlja način na koji implementacije objekata dohvaćaju servise koje im omogućavaju ORB-ovi.

COM

Microsoftov objektni model distribuiranih komponenti (DCOM) pojavio se 1990. kada su predstavili svoje proizvode nazvane *Dynamic Data Exchange* (DDE) i *Object Linking and Embedding* (OLE). OLE2 nazvan Automatizirani OLE, uslijedio je 1993.
Tom novom komunikacijskom modelu, Microsoft je pridijelio ime **model objektnih komponenti** (**COM - *component object model***), promovirajući ga kao infrastrukturu opće namjene za izgradnju softverskih aplikacija sastavljenih od komponenti namijenjenih istom računalu.

DCOM

DCOM (Distributed Common Object Model) proširuje funkcionalnost COM-ova na mrežu ugradnjom mogućnosti poziva udaljenih metoda, sigurnosnim provjerama, skalabilnošću i lokacijskom transparentnošću. DCOM je u početku bio ograničen na Windows okruženje, no ubrzo se proširio i na druge platforme.

RMI

Java RMI (*Remote Method Invocation*) je Javin RPC. Za razliku od CORBA-e, RMI ORB je potpuno integriran s Java jezikom i njenim okruženjem. Stoga, dok su CORBA sučelja opisana korištenjem arhitekturno neutralnih IDL-ova, sučelja udaljenih Java objekata opisana su korištenjem Java sučelja.

SOAP

SOAP (originalno **Simple Object Access Protocol**, sada **Service Oriented Architecture Protocol)** je protokol za izmjenu XML poruka preko mreže koristeći HTTP protokol. Najvažnija primjena mu je kao RPC protokola. Temelji se na HTTP-u *(HyperText Transfer Protocol)* i XML-u *(eXtensible Markup Language)*.

Paradigma prosljeđivanja poruka (MOM- Message-oriented middleware)

MOM dozvoljava aplikacijama komunikaciju preko poruka koje su pohranjene u redu. Asinkrona (tipično) ili sinkrona tehnika orijentirana na događaje u kojem jedan program drugom šalje poruke (između klijenta i servera).
Primjeri:

- IBM MQ serija (WebSphere)
- Microsoft-ov MSMQ (*Microsoft Message Queuing Services*)
- BEA MessageQ

MOM obično omogućava API-je visokog nivoa do svojih servisa koji se izvršavaju na platformama. MOM reducira uključenost programera u složenost master/slave prirode mehanizma klijent/server. MOM konzorcij formiran je 1993. s ciljem kreiranja standarda za međuprograme s porukama.

Redovi:

- Perzistentni (na disku) ili ne-postojani (u memoriji)
- Lokalni ili udaljeni

Serveri uzimaju poruke sa reda po FIFO principu ili prema definiranoj prioritetnoj shemi. Serveri mogu i filtrirati poruke ili ih prosljeđivati drugim serverima.

MOM aplikacije

Pogodne su za aplikacije čiji je tijek uvjetovan događajima:

> *Kada se događaj pojavi, klijentska aplikacija prepušta middleware aplikaciji s porukama odgovornost za obavještavanje servera da se neke akcije moraju poduzeti*

Aplikacije takvog tipa prilagođene su Internet softverskim rješenjima koja zahtijeva brzu, pouzdanu i prilagodljivu vezu između različitih aplikacija i izvora podataka. Pogodne su za nomadske klijent server sustave koji mogu akumulirati transakcije prema van u redove i napraviti *bulk upload* onda kada se veza može uspostaviti.

MOM ↔ RPC

MOM:

- Slanje poruka je fleksibilnije, slabije povezano i vremenski tolerantnije nego kod RPC-a.
- Poruke oslobađaju potrebu da klijent bude sinkroniziran sa serverom.
- Poruke mogu same stvoriti “nered” (u usporedbi s poštom, pisma se mogu nagomilati, a klijenti mogu kontinuirano prazniti svoje pretince, čekajući na odgovor).
- Olakšan je posao servera na račun klijenta.

RPC:

- Posao se završava kako pristiže; nema gomilanja.
- Klijenti su sretni jer dobivaju trenutni odgovor.
- Na kraju svakog dana svi su poslovi obavljeni.