SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1664

Razvojna okolina za paralelno programiranje zasnovano na tabličnom programatoru

Tomislav Homan
Zagreb, rujan 2007.

Zahvaljujem mentoru prof.dr.sc. Siniši Srbljiću na pruženoj potpori u stručnom i osobnom razvoju.
Također zahvaljujem mr.sc. Dejanu Škvorcu na brojnim korisnim savjetima, primjedbama i pomoći pri pisanju diplomskog rada

Sadržaj

1. Uvod

Cilj ovog diplomskog rada je izrada razvojne okoline za paralelno programiranje zasnovane na tabličnom programatoru. Razvojem globalne mreže Internet i povećanjem dostupnosti web usluga, uloga krajnjeg korisnika se mijenja iz uloge korisnika ponuđenog sadržaja prema ulozi aktivnog sudionika u razvoju poosobljenih primjenskih programa. Kao jedan od najraširenijih oblika programiranja bliskih krajnjim korisnicima u posljednjih se nekoliko godina nametnula tehnika tabličnog programiranja. Osim primjene za obavljanje složenih matematičkih izračuna, tablično programiranje je u posljednje vrijeme prepoznato i kao pogodna tehnika za razvoj raspodijeljenih računalnih sustava zasnovanih na globalnoj mreži Internet.

Drugo poglavlje diplomskog rada donosi pregled arhitektura raspodijeljenih sustava s naglaskom na tehnikama programiranja paralelnih i raspodijeljenih primjenskih programa. Nakon uvodne riječi o raspodijeljenim računalnim sustavima, ciljevima njihova korištenja te prednostima i nedostacima, dana je razredba raspodijeljenih primjenskih programa s obzirom na raspodijeljenost programske logike i uvišestručenost podataka. Na kraju su opisane dvije razvojne okoline za izgradnju paralelnih primjenskih programa.

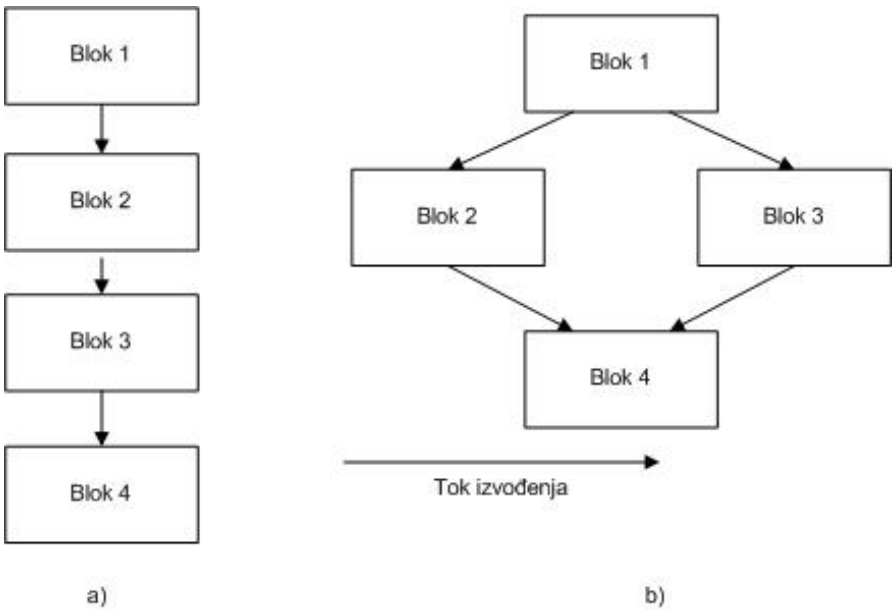
Treće poglavlje posvećeno je razvojnim okolinama zasnovanim na proračunskim tablicama, odnosno tabličnim programatorima. Opisano je nekoliko primjera razvojnih okolina zasnovanih na tabličnom programatoru te su uspoređene prednosti i nedostaci tabličnog programiranja. Naposljetku je dan primjer primjene tabličnog programiranja za razvoj raspodijeljenih programa.

Četvrto i peto poglavlje posvećeno je pregledu funkcionalnosti i programskog ostvarenja razvojne okoline zasnovane na tabličnom programatoru koja je razvijena u okviru ovog diplomskog rada. Šesto poglavlje donosi pregled tehnologija i razvojnih alata korištenih za programsko ostvarenje razvojne okoline. Završna riječ sa zaključcima iznesena je u sedmom poglavlju.

2. Raspodijeljeni sustavi i paralelno programiranje

Raspodijeljeno računarstvo je grana računarske znanosti koja proučava metode oblikovanja, programskog ostvarivanja i izvršavanja računalnih programa pri čemu se različiti dijelovi programa izvršavaju na dva ili više računala. Računala pritom međusobno komuniciraju putem računalne mreže, a moraju se uzeti u obzir i različita okruženja na kojima će se izvršavati različiti dijelovi programa. Na primjer dva različita računala mogu imati različite operacijske i datotečne sustave te različito sklopovlje [1].

Paralelno računarstvo je grana računarske znanosti koja proučava metode oblikovanja, programskog ostvarivanja i izvršavanja računalnih programa pri čemu se različiti dijelovi programa izvode istodobno na više procesora. Kako bi se program mogao istodobno izvršavati na više procesora mora biti paraleliziran, odnosno podijeljen na dijelove koji se mogu izvršavati nezavisno jedni od drugih. Slika 2 prikazuje primjer računalnog programa koji se sastoji od četiri osnovna programska bloka. Slika 2.1a prikazuje slijedno izvođenje programa u kojem se pojedini blokovi izvršavaju slijedno jedan iza drugog. Slika 2.1b prikazuje paralelno izvođenje programa u kojem se programski blokovi 2 i 3 izvršavaju istodobno [2].



Slika 2.1. Primjeri slijednog i paraleliziranog programa

Prema podjeli Michaela J. Flynna, postoje četiri osnovna tipa računalnih arhitektura s obzirom na raspodijeljenost i uvišestručenost instrukcija i podataka. To

su SISD (engl. *Single Instruction, Single Data*), SIMD (engl. *Single Instruction. Multiple Data*), MISD (engl. *Multiple Instruction, Single Data*) i MIMD (engl. *Multiple Instruction, Multiple Data*) [3].

SISD arhitektura se odnosi na jedan slijed instrukcija koji djeluje na jednom podatku u određenom trenutku. Tipični predstavnik SISD arhitekture je jednoprocesorski računalni sustav [3, 4]. SIMD arhitektura se odnosi na jedan tok instrukcija koji djeluje na skup podataka u isto vrijeme. Primjerice, instrukcija može biti zbrajanje, a skup podataka dva polja cijelih brojeva. Takve instrukcije se zovu još vektorske instrukcije, a računala koja ih podržavaju vektorska računala. Vektorski procesori koriste se za posebne namjene poput obrade slike i zvuke te za simulacije fizičkih modela kao što su simulacije vremenskih promjena i reakcije u nuklearnim generatorima struje. Današnji procesori opće namjene, uz skalarne, mogu sadržavati i vektorske instrukcije [3, 4]. MISD arhitektura se odnosi na više sljedova instrukcija koji u određenom trenutku djeluju na jednom podatku. Današnje arhitekture sa protočnom strukturom (engl. *pipeline*) se smatraju MISD arhitekturom. Protočna struktura je struktura sa više dijelova kroz koju prolazi instrukcija i u svakom dijelu se izvršava drugi dio instrukcije. Pošto se čitava instrukcija smatra jednim podatkom to znači da prolaskom kroz protočnu strukturu više procesuirajućih jedinica obrađuje jedan podatak, a upravo to je definicija MISD arhitekture [3, 4]. MIMD arhitektura se odnosi na više sljedova instrukcija koji djeluju na više podataka, odnosno na skupu podataka [3, 4]. U praksi to znači da postoji više procesora u sustavu. Oni mogu dijeliti istu memoriju te je tada to SMP (engl. *Shared-memory MultiProcessor*) arhitektura [4]. U tom slučaju su procesori pomoću sabirnice spojeni međusobno i sa zajedničkom memorijom. Ako je memorija raspodijeljena, tada je riječ o DSM (engl. *Distrubuted Shared Memory*) arhitekturi [4]. Iako je memorija fizički raspodijeljena, procesori mogu dijeliti i komunicirati preko istog logičkog adresnog prostora. Iako su to odvojena računala, pogoni ih isti operacijski sustav, a povezana su pomoću vrlo brze računalne mreže. To je slučaj kod računalnih grozdova (engl. *cluster*) [4] računala koji se koriste u znanstvene svrhe za složena računanja. Ako procesori ne dijele isti logički adresni prostor, tada je moguće na različitim računalima koristiti različite operacijske sustave i povezati ih putem proizvoljne računalne mreže. Ovo svojstvo MIMD arhitekture koriste računalni spletovi (engl. *grid*) [4] koji su u načelu skup običnih komercijalnih računala povezanih putem globalne mreže Internet. Računala u spletu su upravljana posredničkim sustavom (engl. *Grid Middleware*) [4]

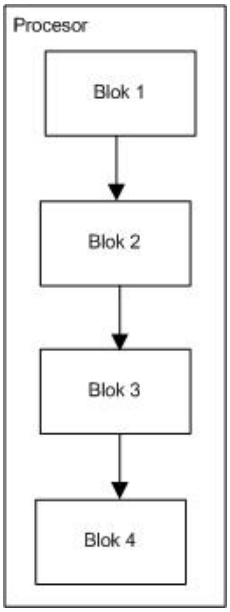
ili raspodijeljenim operacijskim sustavom (engl. *DOS – Distributed Operating System*) [4].

Tablica 2.1. prikazuje arhitekture računala prema podjeli Michaela J. Flynna za sve četiri kombinacije slijednog, odnosno paralelnog izvođenja i raspodijeljenog, odnosno monolitnog izvođenja.

Tablica 2.1.

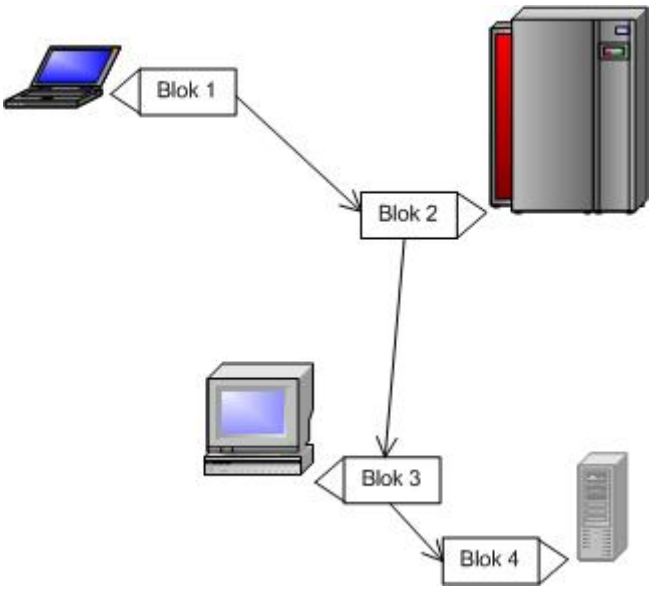
	monolitno izvođenje	raspodijeljeno izvođenje
slijedno izvođenje	SISD	
paralelno izvođenje	SIMD, MIMD	MIMD

Slika 2.2. prikazuje primjer računalne aplikacije koja je slijedna i nije raspodijeljena. Četiri bloka aplikacije izvršavaju se na jednom procesoru unutar istog računala.



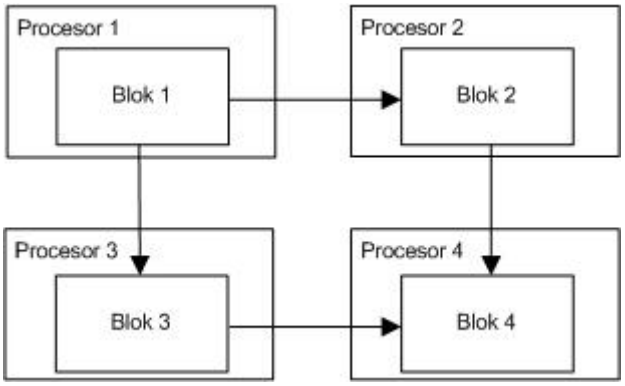
Slika 2.2. Primjer slijednog izvođenja monolitne aplikacije

Slika 2.3. prikazuje primjer računalne aplikacije koja je raspodijeljena, ali nije paralelna. Četiri bloka raspodijeljene aplikacije slijedno se na izvršavaju na četiri različita računala.



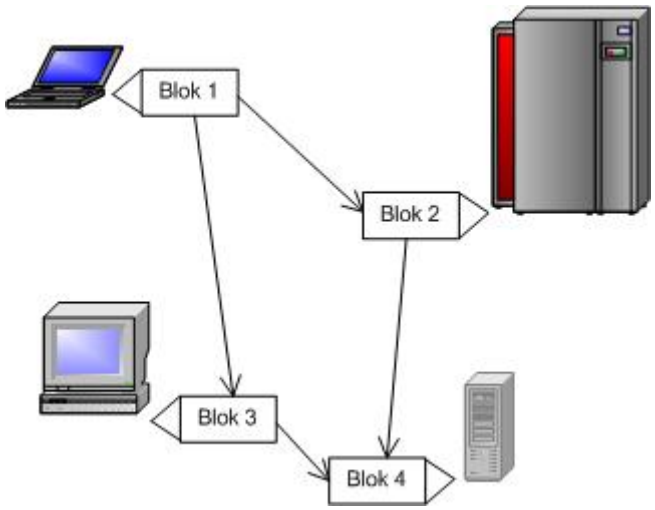
Slika 2.3. Primjer slijednog izvođenja raspodijeljene aplikacije

Slika 2.4. prikazuje primjer računalne aplikacije koja je paralelna, ali nije raspodijeljena. Četiri bloka paralelne aplikacije izvršavaju se na četiri procesora koji se nalaze unutar istog računala.



Slika 2.4. Primjer paralelnog izvođenja monolitne aplikacije

Slika 2.5. prikazuje primjer računalne aplikacije koja je paralelna i raspodijeljena. Četiri bloka paralelne aplikacije izvršavaju na četiri različita računala.



Slika 2.5. Primjer paralelnog izvođenja raspodijeljene aplikacije

2.1. Arhitekture raspodijeljenih sustava

Osnovna zadaća raspodijeljenog računalnog sustava je povezati korisnike i računalna sredstva na transparentan, otvoren i podesiv način. U idealnom slučaju je raspodijeljeni računalni sustav puno otporniji na pogreške i mnogo moćniji u računanju nego pojedinačna računala [1].

Raspodijeljene računalne arhitekture se obično dijele na nekoliko skupina, ovisno o tome koji se kriterij raspodijele gleda. Ako je kriterij način povezivanja pojedinih dijelova raspodijeljene aplikacije, odnosno topologija raspodijeljenog sustava, tada postoje korisnik–poslužitelj arhitektura ili mreža ravnopravnih sudionika [1]. Ako je kriterij razdvajanje aplikacijskih funkcionalnosti, tada postoje arhitekture sa različitim brojem slojeva, gdje svaki sloj ima određenu funkciju u aplikaciji [1]. Zatim još postoji kriterij koji se odnosi na stupanj povezanosti komponenata u sustavu. Ako se sustav sastoji od puno istih, visoko integriranih računala, koja su povezana vrlo brzom mrežom, tada je to čvrsto povezana arhitektura, a nasuprot tome, ako se sustav sastoji od računala različite sklopovske konfiguracije koja su povezana globalnom mrežom Internet tada se govori o slabo povezanoj arhitekturi [1]. Jedna aplikacija može spadati u više skupina. Primjerice troslojna arhitektura može biti ostvarena i kao kombinacija korisnika i poslužitelja, ali i kao mreža ravnopravnih sudionika. U ovome radu se opisuju programske okoline za razvoj raspodijeljenih programa neovisno o tome jesu li slabo ili čvrsto povezani. Što se tiče topologije sustava, neki

su primjeri ostvareni korisnik–poslužitelj arhitekturom, a neki mrežom ravnopravnih sudionika.

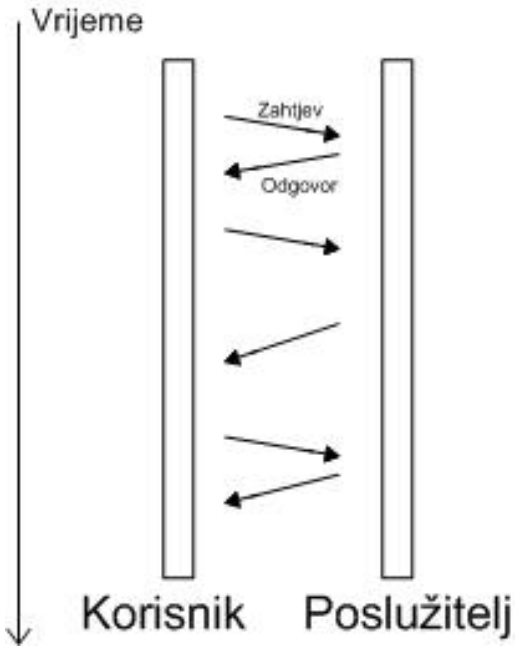
2.1.1. Podjela s obzirom na topologiju raspodijeljenog sustava

S obzirom na topologiju raspodijeljenog sustava postoje korisnik-poslužitelj arhitektura i mreža ravnopravnih sudionika.

Osnovni dijelovi korisnik-poslužitelj arhitekture su korisničke i poslužiteljske komponente. Komponente su odvojene i povezane putem računalne mreže. Svaki korisnik ili poslužitelj spojen na mrežu naziva se čvorom. Korisnički programi šalju zahtjeve za obradu računalnih resursa jednom ili više poslužitelja. Poslužitelji prihvaćaju zahtjeve, obrađuju ih te vraćaju zahtijevanu informaciju korisniku. Korisnik-poslužitelj arhitektura prikazana je na slici 2.6. [5].

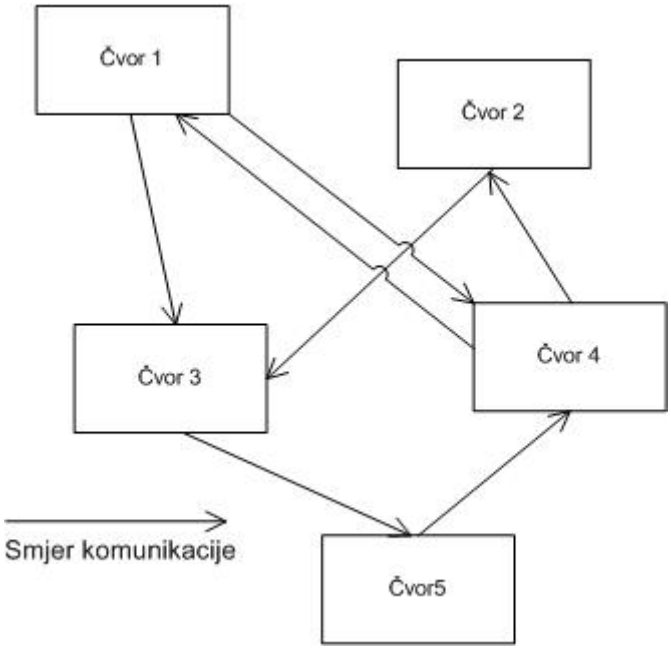
Slika 2.6. Korisnik–poslužitelj arhitektura

Iako koncept korisnik-poslužitelj arhitekture može biti primijenjen na mnoštvo različitih tipova aplikacija, arhitektura uvijek ostaje ista. Danas su korisnici obično Internet preglednici. Primjeri poslužitelja su Internet poslužitelji, poslužitelji baza podataka i poslužitelji za elektronsku poštu. Slika 2.7. prikazuje vremenski dijagram međudjelovanja korisnika i poslužitelja.



Slika 2.7. Primjer sekvencijalnog dijagrama

Karakteristike korisnika su da je aktivan (engl. *master*), inicira zahtjeve, čeka i prima odgovore, obično se spaja na mali broj poslužitelja u isto vrijeme, komunicira direktno s čovjekom koristeći grafičko korisničko sučelje. Karakteristike poslužitelja su da je pasivan (engl. *slave*), čeka zahtjeve od korisnika, nakon primitka zahtjeva obrađuje ih i odgovara, prihvaća veze velikog broja klijenata, ne komunicira izravno s čovjekom, može pamtiti ili ne pamtiti stanje [5].



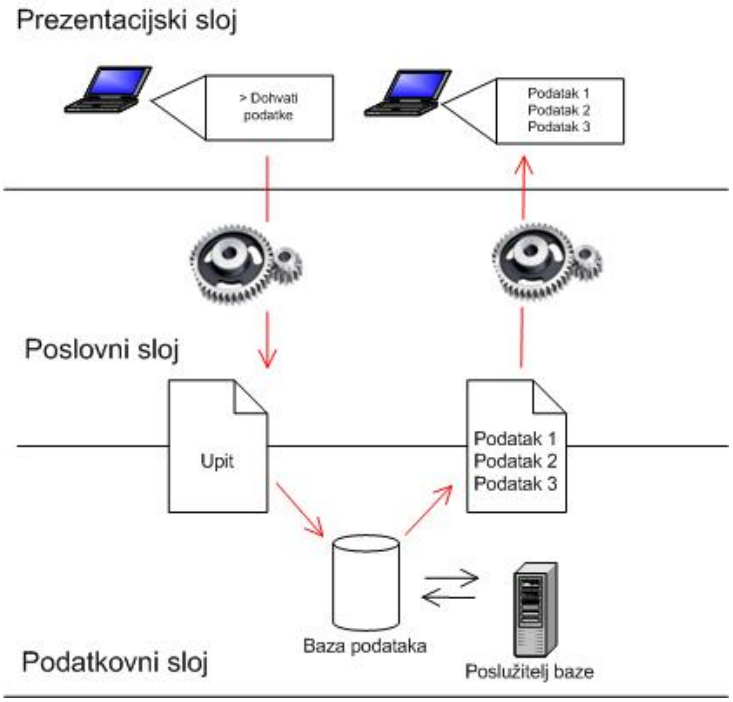
Slika 2.8. Izgled mreže ravnopravnih sudionika

Mreža ravnopravnih sudionika se oslanja na računalnu moć sudionika mreže, prije nego na relativno mali broj poslužitelja. Koriste se u mnoge svrhe, primjerice za dijeljenje tekstualnih datoteka, audio i video sadržaja te u telefoniji. Prava mreža ravnopravnih sudionika nema odijeljene korisnike i poslužitelje, nego jednake čvorove koji su u isto vrijeme i korisnici i poslužitelji. Izgled mreže ravnopravnih sudionika dan je na slici 2.8. [6].

2.1.2. Podjela s obzirom na razdvajanje aplikacijske funkcionalnosti

S obzirom na razdvajanje aplikacijske funkcionalnosti postoje dvoslojne, troslojne te višeslojne arhitekture [1].

Troslojna arhitektura pomiče logiku korisnika u srednji, tj. posrednički sloj, tako da korisnici ne moraju pamtiti stanje, nego samo služe za prezentaciju. Ovo olakšava distribuciju aplikacije. Većina Internet aplikacija su troslojne. Na slici 2.9. prikazan je primjer troslojne aplikacije [7].



Slika 2.9. Primjer troslojne aplikacije

Prvi sloj obično služi za prezentaciju, srednji služi za ostvarenje poslovne logike i može se nalaziti na aplikacijskom poslužitelju, dok donji sloj može služiti kao sloj za pristup podacima i uključuje poslužitelja baze podataka.

2.1.3. Podjela s obzirom na stupanj povezanosti dijelova sustava

Čvrsto povezane arhitekture se odnose na skup visoko integriranih računala koja izvode paralelno jedan proces podjeljujući zadatak na više dijelova koji se kasnije sastavljaju kako bi se dobio konačni rezultat. Računala su obično, ali ne uvijek međusobno povezana preko vrlo brze računalne mreže. Taj skup računala se još zove grozd. Jedan grozd se od strane korisnika ne bi smio razlikovati od pojedinačnog računala. U odnosu na troškove pojedinačnog računala iste snage, grozdovi su puno isplativiji i dostupniji [8].

2.2. Razvojne okoline za izgradnju paralelnih primjenskih programa

U ovom poglavlju se uspoređuju svojstva razvojnih okolina za izgradnju paralelnih primjenskih programa, s naglaskom na svojstvima izražajnosti, području primjene, korisničkom sučelju te razini potrebnih znanja koja se zahtijevaju od korisnika.

Poželjna svojstva razvojne okoline za izgradnju paralelnih primjenskih programa su predočavanje grafa zadataka, preslikavanje grafa zadataka na prividni i stvarni sustav za izvođenje te mogućnost praćenja izvođenja završnog paralelnog programa. Osim pojednostavljenja procesa paralelizacije slijednog programa, dobra razvojna okolina bi trebala voditi brigu o računalnim sredstvima koje koristi napisani primjenski program. Time se programera paralelne aplikacije oslobađa brige o raspoloživosti sredstava tijekom izvođenja programa te mu se omogućuje poklanjanje više pažnje rješavanju samog problema.

Još jedan zahtjev koji bi razvojne okoline za izgradnju paralelnih primjenskih programa trebale zadovoljavati je mogućnost stvaranja omotača oko postojećih programskih modula koji rješavaju neki problem, kako bi se oni mogli iskoristiti u paralelnom programu. Bilo bi idealno kada bi taj omotač bio ostvaren kao web usluga

(engl. *Web Service*) [9], jer su web usluge široko prihvaćen standard te bi takvi moduli mogli sa grafa zadataka biti preslikani na raznorodnu okolinu, neovisno o sklopovskoj konfiguraciji pojedinog čvora i neovisno o operacijskom sustavu koji je na njemu instaliran. Ponovna iskoristivost modula omogućuje programeru da jednostavno paralelizira problem na različite načine, ovisno o tome kakva svojstva očekuje od napisanog paralelnog programa. Primjerice, način paralelizacije programa kojim se postiže najkraće vrijeme izvršavanja može se razlikovati od načina paralelizacije kojim se postiže najveća preciznost dobivenih rezultata.

U slijedeća dva odjeljka biti će opisani konkretni primjeri programskih okruženja za razvoj paralelnih primjenskih programa.

2.2.1. Programsko okruženje IPPE

Programsko okruženje IPPE (engl. *Interactive Parallel Programming Environment*) [10] je projekt vođen od strane Sjeveroistočnog Centra za Paralelne Arhitekture pri Sveučilištu u Sirakuzi (engl. *Northeast Parallel Architectures Center at Syracuse University*) [11]. Cilj projekta je bio ostvariti razvojnu okolinu za izgradnju paralelnih programa. Cilj okoline IPPE je generiranje i izvođenje paralelnih programa za istorodne i raznorodne računalne platforme. Paralelni programi su predočeni grafičkim objektima. Razvojna okolina koristi grafički editor. Dijelovi programa koji se mogu ponovno iskoristiti se spremaju u programske knjižice. Radna svojstva programa tijekom izvođenja na raznim platformama se mjere za vrijeme izvođenja, a rezultati mjerenja se spremaju u bazu podataka. Okolina je ispitana na FDDA (engl. *Four Dimensional Data Assimilation*) problemu predviđanja vremenske prognoze [12] američke agencije NASA (engl. *National Aeronautics and Space Administration*) [13].

Postoji nekoliko pristupa paralelizaciji slijednih programa. Za paralelizaciju FDDA problema koristi se pristup od vrha prema dnu. Tipični pristup od vrha prema dnu uključuje određivanje glavnih programskih blokova, a zatim odluka koja je paradigma za paralelno programiranje najpogodnija za paralelizaciju problema. Ovdje se koristi paradigma za paralelno programiranje zasnovana na zadacima. Pretvorba slijednog programa u paralelni program zasnovan na zadacima slijedi nekoliko koraka. Prvi je odrediti strukturu slijednog programa i podijeliti program u manje potprograme. Zatim se izvrši strukturalna analiza programa te se nacrta kao graf zadataka koji uključuje i ovisnost podataka. Nakon toga se maksimizira paralelnost grafa. Onda se odrede zadaci koji nužno ostaju slijedni te se ti dijelovi probavaju ponovno iskoristiti što je

više moguće. Slijedi ponovno pisanje dijelova programa koji se daju paralelno izvršavati, koji se zatim preslikaju prvo na virtualnu računalnu platformu, a zatim i na stvarnu hardversku platformu. Na kraju se paralelni program izvrši te se izmjere performanse [10].

Na primjer za rješavanje FDDA problema koji je spomenut ranije, koristi se algoritam optimalne interpolacije u čije se detalje neće ulaziti. Taj algoritam je dobro razrađen i mnogo se koristi u praksi [10]. Problem je što prije nije bilo paralelnih računala tako da je programski kod koji ostvaruje taj algoritam napisan slijedno u programskom jeziku Fortran. Ponovno ostvarenje algoritma od nule koristeći neki programski jezik za paralelno programiranje je preskupo većini institucija, tako da bi središnji dio programa trebao ostati napisan u programskom jeziku Fortran, a ostali moduli koji se daju izvršavati paralelno bi trebali dobiti omotač koji prilagođava modul razvojnoj okolini za izradu paralelnih programa, u ovome primjeru razvojnoj okolini IPPE. Isto tako većina znanstvenika, za koje su razvojne okoline izrađene, zna koristiti programski jezik Fortran, tako da bi im bio veći problem iznova učiti neki novi programski jezik koji je namijenjen specijalno za izradu paralelnih primjenskih programa.

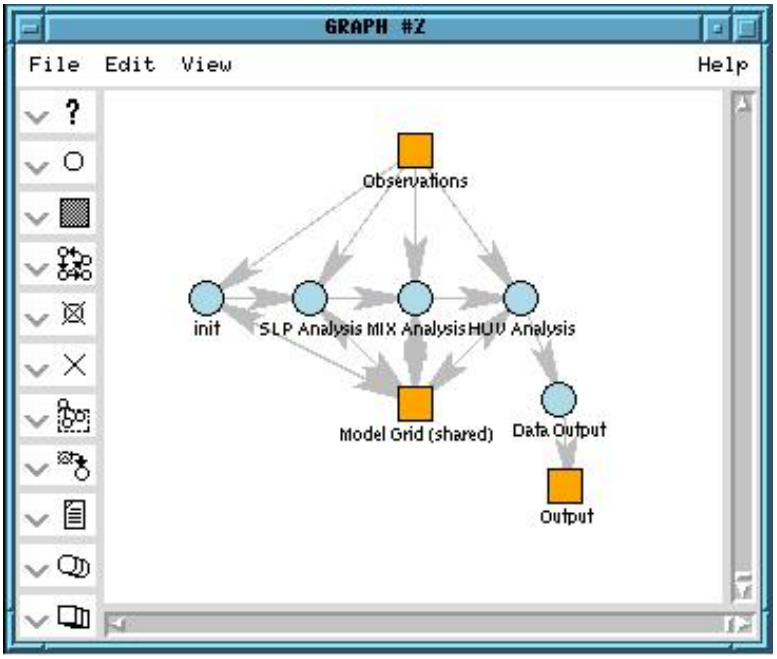
Kako bi ispunila neke od tih mogućnosti koje se zahtijevaju od dobre razvojne okoline za izgradnju paralelnih primjenskih programa, IPPE razvojna okolina koristi koncept protoka podataka, a kako bi se omogućila prenosivost koristi se paradigma prijenosa poruka (MPI, engl. *Message Passing Interface*) [10]. Nakon što se generira paralelni program koji sadrži procedure za slanje i primanje poruka, izvršni program se pokreće na skupu statički ili dinamički odabranih čvorova [10].

Gledano iznutra IPPE se sastoji od dva sloja. Prvi je sučelje za razmjenjivanje poruka čiji je cilj omogućiti izvršavanje paralelnog programa na raznim platformama. Sloj iznad prethodnog je programska knjižnica za potporu paralelnom programiranju [10].

MPI je komunikacijski protokol koji se koristi za paralelno programiranje paralelnih računala. Iako službeno pripada petom i višim slojevima OSI modela, trenutna ostvarenja obuhvaćaju većinu slojeva, koristeći socket-e i TCP/IP protokole kao transportni sloj. Ciljevi MPI protokola su visoke performanse, podesivost i prenosivost. Generalno gledajući MPI protokol je uspio ostvariti te ciljeve, no zamjera se što radi na preniskoj razini i što je dosta kompliciran. No, unatoč kritikama, nema

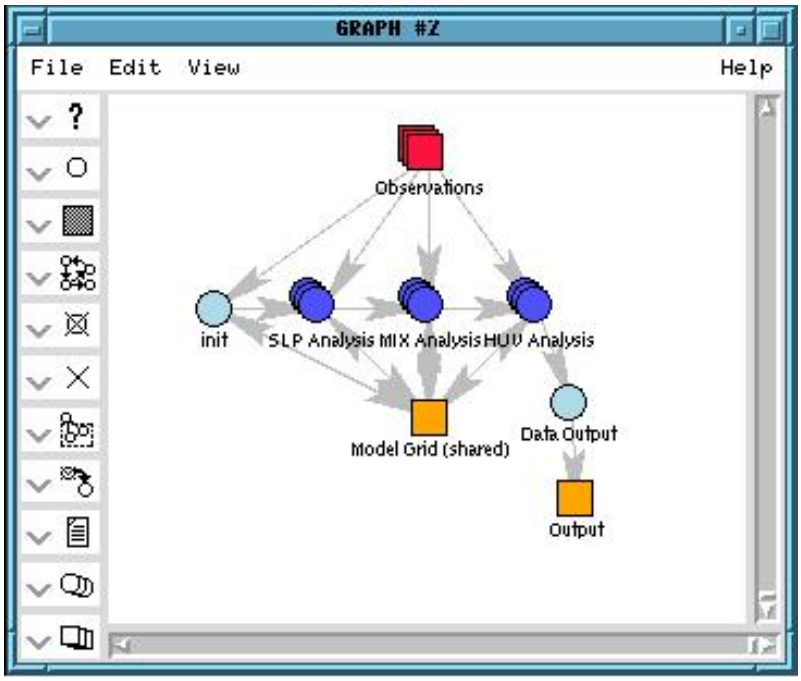
prave alternative pa je on postao „de facto“ standard za komunikacije među procesima od kojih se sastoji paralelni program [14].

Slijedeće slike prikazuju kako izgleda razvoj jednog paralelnog programa u razvojnoj okolini IPPE. Slika 2.10. prikazuje logičku podjelu slijednog programa. Podaci su prikazani kvadratima, a procesi krugovima. Ovisnosti među njima su prikazane strelicama.



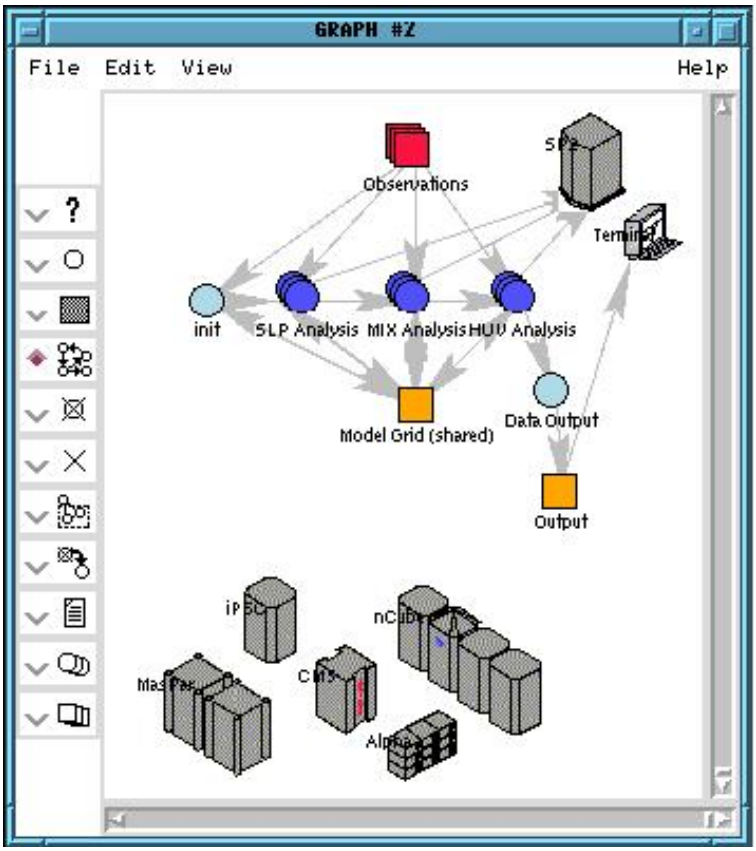
Slika 2.10. Logička podjela slijednog programa prikazana u razvojnoj okolini IPPE

Kada su definirani procesi koji se mogu paralelno izvršavati uvrštavaju se u graf zadataka. Slika 2.11. prikazuje graf zadataka u razvojnoj okolini IPPE. Proces koji se može paralelno izvršavati je prikazan sa više krugova, a raspodijeljeni podaci su prikazani sa više kvadrata.



Slika 2.11. Graf zadataka prikazan u razvojnoj okolini IPPE

Podatkovni objekt koji se kreće između procesa je definiran na jednostavan način sličan zapisu u programskom jeziku Fortran, odnosno strukturi u programskom jeziku C. Definicija podatkovnog objekta omogućuje da se generiraju odgovarajuće procedure za razmjenu poruka te tako omogućujući komunikaciju između podatkovnih i procesuirajućih objekata. Zatim se definiraju procesi u nekom od visokih programskih jezika koji su prošireni tako da omogućuju korištenje raspodijeljenih podataka. Tok podataka je označen ključnim riječima IN DATA i OUT DATA. Nakon što su definirani podatkovni i procesuirajući objekti, oni moraju biti preslikani na određenu mrežu računala. Preslikavanje može biti statičko, tj. raspored paralelnih procesa je unaprijed definiran, ili može biti dinamičko. Kod dinamičkog preslikavanja ovisno o opterećenju pojedinog čvora i predviđanju opterećenja na temelju već pohranjenih mjerenja u bazi podataka, paralelni proces se može ponovno prevesti i učitati na neki drugi čvor [10]. Slika 2.12. prikazuje jedan odabir preslikavanja.



Slika 2.12. Preslikavanje paralelnih procesa na stvarnu mrežu računala

2.2.2. Programsko okruženje ASSIST

ASSIST je skraćenica od engleskog naziva „A Software development System based on Integrated Skeleton Technology“ [15]. To je programsko okruženje s korisničkim sučeljem koje nudi programerima paralelnih sustava učinkovit način ostvarenja paralelnih programa. Zajednički je projekt od tijela ASI (engl. *Italian National Space Agency*) [16] i CNR (engl. *Italian National Research Council*) [17].

Uključuje programski jezik ASSISTcl, gdje „cl“ na engleskom jeziku znači „coordination language“ te skup alata za prevođenje i programske knjižnica. Paralelni programi napisani u ASSISTcl-u se izvršavaju u mrežnom okruženju koje podržava POSIX (engl. *Portable Operating System Interface*) [18] i ACE (engl. *Adaptive Communication Enviroment*) [19].

POSIX je skup standarda definiranih od strane tijela IEEE koji definiraju API (engl. *Application Programming Interface*) [20] za programe kompatibilne sa raznim verzijama UNIX operacijskog sustava. Operacijski sustav koji zadovoljava POSIX standard mora ispunjavati razne uvjete koji se tiču primjerice kreiranja procesa,

signala, iznimaka, cjevovoda, pristupanja datotečnom sustavu, itd... POSIX standard zadovoljavaju većina operacijskih sustava zasnovanih na UNIX operacijskom sustavu, Windows operacijski sustav te djelomično razne distribucije Linux operacijskog sustava [18].

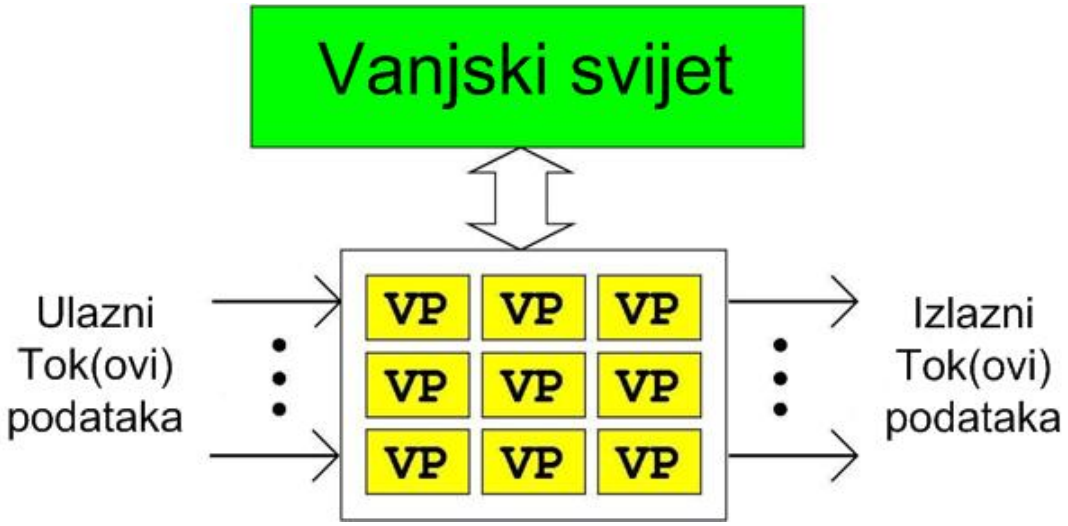
ACE je otvoren, objektno orijentirani okvir za izrađivanje aplikacija, napravljen i dizajniran koristeći veliki broj programskih uzoraka koji se tiču mrežne komunikacije. Služi kao komunikacijski okvir koji olakšava programiranje mrežne komunikacije među raznim programima, procesima ili objektima [19].

Koristeći ASSISTcl programer može struktuirati program grafom slijednih procesa ili paralelnih modula. Čvorovi u grafu su povezani tokovima podataka. Slijedni dijelovi koda mogu biti napisani u programskim jezicima C, C++ ili FORTRAN77 [15].

Sintaksa ASSISTcl-a je djelomično posuđena iz programskih jezika C i Pascal. Program ima više dijelova. Prvi dio opisuje graf procesa odnosno paralelnih aktivnosti. Drugi dio sadržava kod koji opisuje svaki pojedini proces ili paralelnu aktivnost. Većina paralelnih aktivnosti se može opisati već ugrađenim programskim kosturom koji ostvaruje najčešće programske uzorke vezane za paralelno programiranje. Tako se paralelna aktivnost može specijalizirati da se ponaša kako farma, cjevovod, itd... [15]

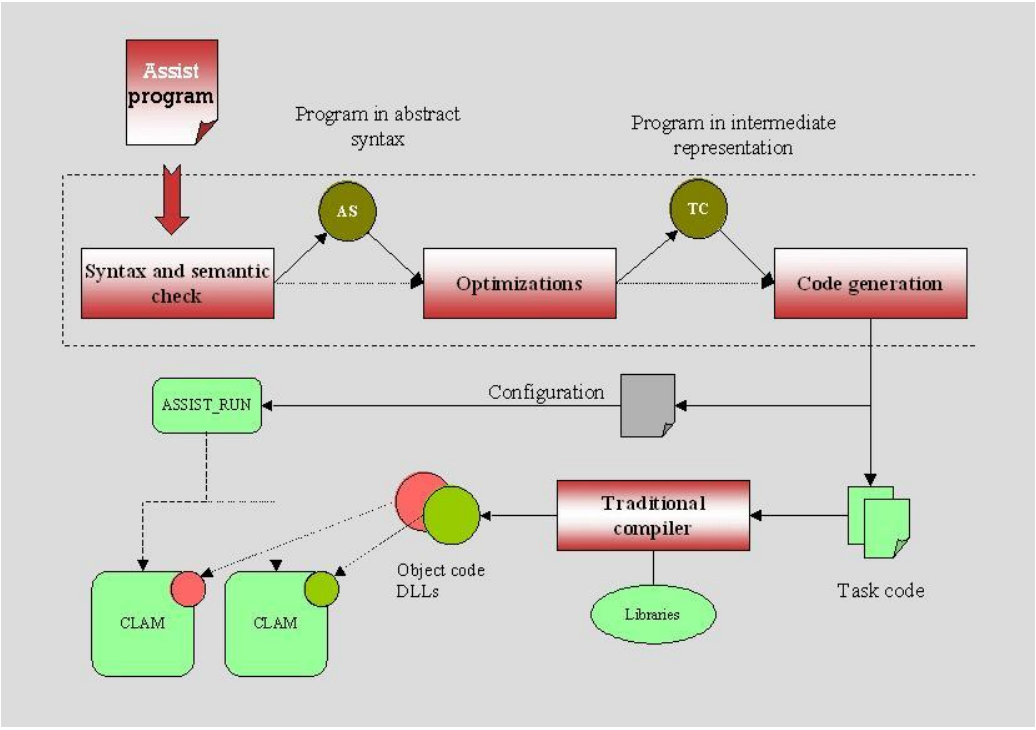
Taj kostur dopušta da se definira skup virtualnih procesora, da im se dodijele zadaci (pojedinačno ili skupu procesora), da rukuju nijednim, jednim ili više ulaznih i/ili izlaznih tokova podataka te komunikaciju sa vanjskim svijetom putem standardnih načina za pristup objektima (npr. CORBA – engl. *Common Object Request Broker Architecture*) [21].

Slike u ovome odjeljku su preuzete sa službene stranice programskog okruženja ASSIST i na engleskom su jeziku. Slika 2.13. prikazuje skicu jedne paralelne aktivnosti izrađene u programskom okruženju POSSIX.



Slika 2.13. Skica paralelne aktivnosti izrađene u programskom okruženju POSSIX

Koordinacijska tehnologija koja se koristi u programskom okruženju ASSIST dopušta programeru da se ne brine za većinu detalja koji mogu dovesti do pogreške, a koji se obično pojavljuju kod paralelnog programiranja, primjerice postavke procesa i komunikacije među njima, vremensko planiranje, preslikavanje procesa na virtualne procesore, itd... Ta tehnologija, dakle, omogućuje programeru da ostvaruje složene paralelne programske uzorke ne brinući se puno za detalje ostvarenja [15]. Slika 2.14. prikazuje proces programiranja u programskom okruženju ASSIST.



Slika 2.14. Proces programiranja u programskom okruženju ASSIST

Dobre strane programskog okruženja ASSIST su te što omogućuje ponovno korištenje koda, budući da su slijedni dijelovi napisani u programskim jezicima C, C++ ili FORTRAN77, zatim što programer može eksperimentirati sa raznim strategijama paraleliziranja tako da promijeni samo par linija koda i ponovno prevede program, što putem CORBA protokola može komunicirati sa vanjskim svijetom, što slijedni dijelovi mogu koristiti već napisane vanjske programske knjižnice te što su performanse tako napisanih programa jako dobre te se na grozdovima na kojima su instalirani operacijski sustavi Linux približavaju učinkovitosti od 99% idealne.

Loše strane su da iako se programer ne treba brinuti i ne treba poznavati detalje paralelnog programiranja, sustav zahtijeva određenu količinu tehničkog znanja i znanja o programiranju, pošto nema razvijeno grafičko sučelje. Isto tako su podržani programski jezici koji se koriste u slijednom dijelu zastarjeli te bi se mogli podržati moderni programski jezici poput programskih jezika Java ili C#. Nema ugrađenih mehanizama za kontrolu i praćenje korisnika i kvalitete usluge. Najveća zamjerka, međutim, je što za komunikaciju sa vanjskim svijetom koristi zastarjeli protokol CORBA, a moglo bi se koristiti moderne, otvorene i od svih podržane protokole zasnovane na Web uslugama.

3. Programski alati zasnovani na proračunskim tablicama

Bit tabličnog programiranja je u tome da pojedinci ili timovi inženjera, ekonomista ili običnih ljudi koriste tablične aplikacije poput Microsoft Excela kako bi izradili i testirali programsku logiku i poslovna pravila, koja se nakon toga ugrade izravno u sustav.

Jedan način da se implementira tehnika tabličnog programiranja je integrirati tablični programator u programe koristeći COM (engl. *Component Object Model*) [22] ili VBA (engl. *Visual Basic for Applications*) [23]. Ovaj pristup, međutim, ima problema sa performansama, distribucijom i sposobnošću razmjernog rasta.

Proizvodi poput KDCalc-a [24] ostvaruju tablično programiranje prevodeći tablice u izvršni kod koji se pokreće nezavisno od tablične aplikacije. On u biti pretvara tabličnu aplikaciju u razvojno okruženje za izradu visoko sofisticiranih sustava za procesuiranje podataka sa više ulaza i izlaza.

3.1. Korištenje tabličnog programiranja za brzi razvoj programa

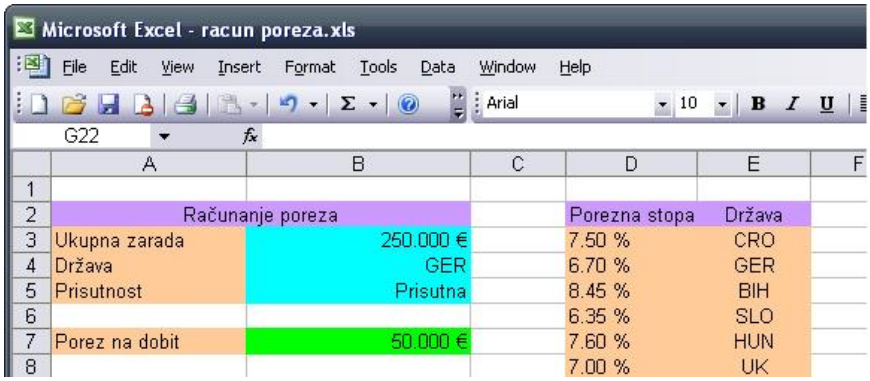
Nema dvojbe da su tablične aplikacije bili jedan od glavnih čimbenika koji su utjecali na prihvaćanje računala za poslovnu i osobnu upotrebu. Lakoća s kojom obični ljudi mogu vizualno kreirati složenu logiku, što-ako scenarije i simulacije je preobrazila poslovni svijet. Tablično programiranje donosi istu tu učinkovitost u razvoj programa. U tabličnom programiranju se koriste tablične aplikacije poput Microsoft Excela kako bi se ostvarili i testirali algoritmi odlučivanja, simulacije, izvještaji ili bilo kakva druga vrsta složenih izračuna i procesuiranja podataka.

Razvijanje poslovne logike u tabličnom programatoru je mnogo učinkovitije nego ručno kodiranje u programskom jeziku. Vizualna priroda tablica i korisničke mogućnosti poput povlačenja i ispuštanja (engl. *drag and drop*) su bile ključne za njihovo rasprostranjeno prihvaćanje. Kada se ta poboljšanja produktivnosti primijene na programiranje, učinci nisu ništa manje dramatični. Programeri mogu pisati proračune i procedure za transformaciju podataka u tablici u djeliću vremena koje bi im bilo potrebno da to isto napišu u nekom programskom jeziku. Programeri početnici mogu biti učinkoviti u kodiranju logike isto kao vrlo vješti programeri [24].

Ljudi koji nisu programeri mogu kontrolirati poslovnu logiku. U većini procesa razvijanja poslovnih programa, poslovni ljudi ili stručnjaci u tom polju su odgovorni za provedbu poslovnih pravila i zahtjeva. Zato što su tablični programatori oduvijek bili ciljani na te korisnike, poslovni ljudi mogu sami ostvarivati poslovna pravila u tablicama, radije nego da predaju specifikacije razvojnom timu programera. To uklanja puno prilika za nesporazume i umanjuje potrebu da programeri budu stručnjaci u poslovnom području.

Ponašanje jezgre sustava može biti provjereno od strane stručnjaka u tom području. Jedna od stvari koje zahtijevaju najviše rada u razvoju programa je testiranje kako bi se provjerilo da li program radi po specifikacijama. U mnogim procesima razvoja programa glavni dijelovi sustava moraju biti izrađeni prije nego što se ijedan drugi dio izradi i testira. To otežava određivanje točnog mjesta pogreške. Ali ako se koristi tablica kako bi se razvila središnja logika sustava, tada je lako napraviti više skupova testnih podataka i koristiti naizmjence te skupove kako bi se istestirala tablica. To znači da se središnja funkcionalnost može provjeriti lokalno, prije distribucije i bez pisanja ikakvog koda.

Sada će to biti pokazano na primjeru aplikacije za izračun poreza. Odnosi se na programere u firmi koja se bavi Internet trgovinom. Prema zakonu firma mora računati i plaćati poreze u bilo kojoj državi u kojoj je fizički prisutna. Programeri imaju zadatak da napišu program koji će računati poreze za narudžbe napravljene preko Interneta. Slika 3.1. prikazuje kako bi izračun poreza mogao biti ostvaren koristeći tablično programiranje.



Slika 3.1. Računanje poreza koristeći tablično programiranje

Sada se jednostavno podaci programatski unesu u tablicu.

poreznaTablica.setValue(3, 2, 250 000);

poreznaTablica.setValue(4, 2, „GER“);

poreznaTablica.setValue(5, 2, „Prisutna“);

Zatim se povuku izlazni podaci.

double porez = poreznaTablica.getValue(7, 2);

Ćeliji *7,2* se prethodno pridijelila funkcija za računanje poreza na dobit pa se čitanjem podataka iz te ćelije u biti izvršava ta funkcija.

3.2. Tablični programator kao razvojno okruženje za programske komponente

Tablično programiranje je komponentno usmjereno programiranje. Njegova osnovna filozofija je da se koristi tablična aplikacija poput MS Excel-a kao razvojno okruženje za kreiranje poslovnih komponenata. Te komponente se tada mogu iskoristiti za kreiranje složenih i sofisticiranih poslovnih sustava koji su laki za korištenje i održavanje. Nadalje, pošto su poslovna pravila razvijena u tabličnom programatoru, poslovni korisnici, koji u tipičnim uvjetima postavljaju zahtjeve, s lakoćom mogu pregledati logiku, provjeriti je i izmijeniti je.

Treba uzeti u obzir da se tablični programator koristi samo kao razvojno okruženje. Za vrijeme izvršavanja pogonski programi mogu koristiti poslovne komponente bez pomoći tabličnog programatora. Nekoliko proizvođača nudi proizvode koji omogućuju korištenje ove tehnike u programskom jeziku Java i razvojnoj okolini .NET.

U slijedećih nekoliko odjeljaka biti će uspoređeno tradicionalno programiranje usmjereno prema komponentama i tablično programiranje. Tablica u tabličnom programatoru sadrži mnogo redaka, stupaca i ćelija. Svaka ćelija može sadržavati podatke (broj ili tekst), ili formulu. Ćelije mogu biti grupirane u raspone, a rasponi se mogu imenovati te im se može pristupiti pomoću tog imena. Pod paradigmom tabličnog programiranja, tablica odgovara poslovnoj komponenti, ćelija koja sadrži podatak odgovara ulazu u poslovnu komponentu, a ćelija sa formulom odgovara izlazu komponente. Slijedeća tablica daje kratku usporedbu dvaju paradigama. Pod kraticom API se podrazumijeva sučelje za izradu programa (engl. *Application Programming Interface*)

Tablica 3.1. Usporedba paradigama

Tradicionalno programiranje usmjereno komponentama	Tablično programiranje
Komponenta	Tablica
Javno ulazno podatkovno polje	Ćelija s podatkom napunjena za vrijeme izvođenja kroz API
Javna izlazna metoda	Ćelija s formulom koja se evaluira za vrijeme izvođenja kroz API
Privatna metoda	Ćelija s formulom koja se ne evaluira za vrijeme izvođenja
Privatno konstantno podatkovno polje	Ćelija s podatkom koja se ne mijenja za vrijeme izvođenja

Za vrijeme dizajna se interaktivno koristi tablični programator kako bi se provjerila funkcionalnost poslovne komponente. Međutim, tablični programator se ne može koristiti na poslužitelju kako bi poslužio mnogo korisnika. Tu kao pomoć dolaze tablični pogonski programi koji za vrijeme izvršavanja aplikacije komuniciraju s tablicama koristeći API. Ulazni parametri se unose koristeći *SetCellValue(x,y)* funkciju ili neku analognu, zatim se evaluiraju funkcije, odnosno pokrene procesuiranje te se na kraju očitaju rezultati koristeći funkciju tipa *GetCellValue(x,y)*.

Najkritičniji zadatak u dizajniranju programa kada se koristi tablično programiranje je određivanje adekvatnih ulaznih i izlaznih ćelija. To definira API za tablicu. Nakon što je API definiran mogu se nastaviti paralelno dizajnirati aplikacija koja poziva API i pozadinska poslovna logika. Tada se fokusira na te dvije stvari. Pozivajuća aplikacija samo treba proslijediti prave podatke u i iz tablice, a tablicu mogu dizajnirati ljudi koji dobro poznaju poslovnu logiku [24].

Ispitivanje je područje gdje je tablično programiranje posebno dobro. Testiranje treba provjeriti dvije stvari. Poslovnu logiku u tablici i komunikaciju između pozivajuće aplikacije i tablice. Kako bi se testirala poslovna logika napravi se posebna tablica koja sadrži mnoštvo parova ulaznih i izlaznih podataka. Zatim se pomoću jednostavne VBA funkcije iterira po tim podacima. Za svaki par se uzme ulazni podatak, proslijedi se poslovnoj logici, očita izlazni podatak te se usporedi sa drugim članom para. Postupak je jednostavan, automatiziran i učinkovit [24].

Slijedeći korak je ispitivanje komunikacije između pozivajuće aplikacije i tablice kako bi bili sigurni da su pravi ulazni podaci uneseni u pravu ćeliju u tablici te da su pravi podaci pročitani iz prave ćelije. Test se može provesti na više načina ovisno o pogonskom programu. Pogonski stroj tabličnog programatora primjerice umeće novu tablicu između tablice i pozivajuće aplikacije te svaki put kada se unosi ili čita podatak iz tablice, on bude upisan i u tu novu tablicu u koju se može tada ručno pogledati je li sve u redu. Postupak je dosta brz jer se ručno može pregledati, za razliku od debuggiranja kada se mora izvršavati linija po linija koda i očitavati je li učitana ili upisana prava vrijednost [24].

Održavanje takvih aplikacija je isto dosta olakšano jer se ta dva dijela mogu odvojeno održavati sve dok se API ne mijenja. Održavanje poslovne logike u tablici koju je neko drugi napravio je puno lakše nego održavanje nečijeg tuđeg koda.

Aplikacije povoljne za izradu ovom paradigmom su računanje poreza, računanje cijene prijevoza na temelju težine tereta, načina prijevoza, rasporeda prijevoza... Zatim računanje koje uključuje dionice, gotovo sve što uključuje poslovno odlučivanje i poslovne izračune se može brže i točnije računati tabličnim programiranjem. U novije vrijeme se tablično programiranje koristi i u raspodijeljenim računalnim sustavima [24].

3.3. Primjena tabličnog programatora za razvoj raspodijeljenih računalnih sustava

U ovom odjeljku opisuje se kako se tablični programator može iskoristiti za programiranje raspodijeljenog sustava, točnije mreže senzora.

3.3.1. Opis arhitekture raspodijeljenog sustava

Danas je jedna od prepreka u ostvarivanju mreže senzora programiranje i pokretanje čitavog sustava. To uključuje programiranje i rukovanje senzorima, programiranje sučelja, interpretaciju i procesuiranje tokova podataka koji dolaze iz senzora i postavljanje poslužitelja koji skladište podatke. Čak je i stručnjacima u tom području ovaj problem dosta težak. Težina proizlazi iz raspodijeljene prirode mreže senzora koja prelazi preko mnogo slojeva računalnih platformi: senzore, sučelja prema njima, poslužitelje te sloj za komunikaciju sa korisnikom. Dodatno još

nepredvidljiva priroda senzora zahtijeva puno očitavanja podataka te njihove analize. Sama priroda senzorskih mreža je takva da raspored same mreže uvelike utječe na proces dizajniranja programa koji će upravljati njome. Općenito gledano determinističko ponašanje je ovdje jako rijetko.

Generalni cilj radova koji se bave programiranjem sustava za upravljanje mrežama senzora je dizajnirati programski jezik visoke razine koji sadrži elemente za apstrakciju detalja niske razine vezanih za pojedine čvorove mreže senzora. Te apstrakcije visoke razine su vrlo korisne u programiranju i održavanju mreža senzora koje se sastoje od puno čvorova, no postoje još neka ograničenja koja sprečavaju da te apstrakcije budu još bogatije. Podrška sustavu mora omogućavati korisnicima da ponovno konfiguriraju i programiraju dio ili čitavu mrežu senzora. Zato okruženje za programiranje mreža senzora mora ispunjavati tri mogućnosti: prikupljanje i procesuiranje podataka, programiranje na licu mjesta i podršku za održavanje i ponovno konfiguriranje sustava za vrijeme dok je pokrenut.

U ovome odjeljku će biti pokazano kako se tablično okruženje može iskoristiti za ispunjenje ta tri uvjeta. Glavna ideja je da se sučelje programa poput Microsoft Excela iskoristi za predočenje podataka, odnosa među čvorovima mreže, za unošenje raznih filtara podataka i skripata koje će biti prevedene u jezik senzora te učitane na neke ili sve od njih.

Slika 3.2. Korištenje tabličnog okruženja za programiranje mreža senzora

Najniži sloj arhitekture prikazane na slici 3.2. se sastoji od samih senzora raspoređenih negdje u prostoru. Senzori se samoorganiziraju u mrežu koja šalje podatke prema poveznicima ili kako se još negdje nazivaju, mikroposlužiteljima.

Poveznici čine slijedeći sloj arhitekture. Oni mogu biti bežični ili ožičeni, ali obično će imati veliki protok mreže i veliku pouzdanost. Mogu biti programirani da dalje usmjeravaju tokove podataka i da služe kao zastupnici viših slojeva koji će ispitivati sloj senzora za podacima. Poveznici, kao što im samo ime kaže, služe kao veza senzora prema globalnoj mreži Internet.

Tokovi podataka iz poveznika se mogu sakupljati i pohranjivati u bazama podataka u većim i jačim poslužiteljima koji sačinjavaju slijedeći sloj. Taj sloj obuhvaća prostorno gledano bilo koje područje gdje je prisutan i Internet. Korisnici komuniciraju sa mrežom senzora indirektno preko gornjeg sloja ili direktno preko srednjeg sloja. Tablica igrata ulogu u pojednostavljenju sakupljanja i organizacije podataka, programiranju i održavanju te ponovnom konfiguriranju sučelja za korisnike srednjeg sloja. Pošto su svakidašnji korisnici već upoznati sa programiranjem i rukovanjem podacima u tabličnim programatorima za znanstvene i poslovne svrhe, primjenjivanje istih principa za rukovanje tokovima podataka koji dolaze iz senzora im ne bi trebalo predstavljati veliki problem, pogotovo zato što se ne moraju opterećivati detaljima niske razine koji se tiču organizacije i funkcioniranja same mreže senzora. Iste ugrađene statističke funkcije i alati za predočenje podataka u tabličnim programatorima mogu se iskoristiti i za analiziranje toka podataka u kontekstu mreže senzora.

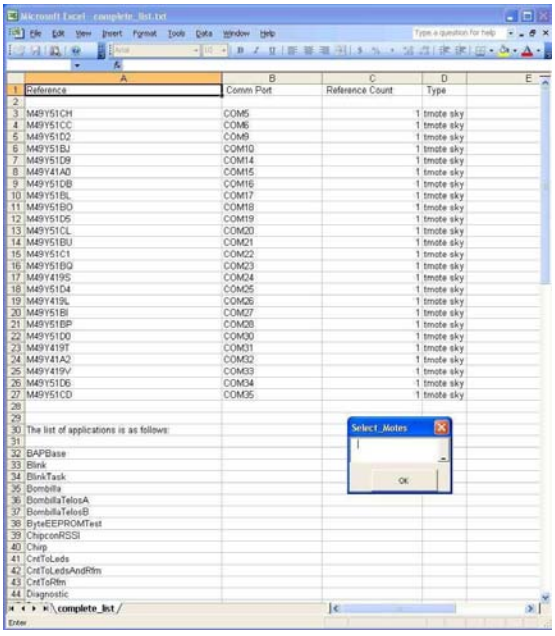
Još jedna prednost korištenja tabličnog modela je lakoća kombiniranja podataka koji se dobivaju u stvarnom vremenu i statičnih podataka poput lokacija čvorova, konfiguracija čvorova i informacija koje nemaju veze s čvorovima. Naprimjer može se povezati tablica sa mapom na kojoj je prikazan raspored čvorova. To omogućuje korisniku da rukuje u isto vrijeme i podacima koje dobiva od čvora i podacima o tome samom čvoru te da primjerice promijenivši vrijednost u ćeliji koja je pridružena tome čvoru može automatski konfigurirati ili čak preprogramirati čvor.

Ova integracija informacija o prostornom rasporedu, analize podataka, programiranja i održavanja sustava je baš ono što treba korisnicima. Tablica se prirodno umeće između korisnika i poveznika, kao što je prikazano na slici 3.2. Nadalje lakoća integracije tabličnih programatora sa mrežnim uslugama i bazama podataka omogućuje da oni budu i sučelje prema najvišem sloju arhitekture.

3.3.2. Ostvarenje raspodijeljenog sustava

U ovome odjeljku će biti opisano ostvarenje jednog takvog sustava. Ono uključuje dvije uloge tabličnog programatora. Prva je korištenje MS Excela za održavanje mreže senzora, a druga je korištenje Excela za rukovanje tokovima podataka koji se primaju iz senzora. Senzori su marke „Tmote sky“ od tvrtke „Moteiv“ [25]. Kao poveznici se koriste osobna računala s instaliranim mikroserverskim izvršnim programima koji se zovu µSEE [26]. Korisnička osobna računala imaju instalirana MS Excel 2003, MS Visual Studio 2005 te MS SQL Server.

Prva uloga Excela koristi njegove ugrađene VBA mogućnosti koje pružaju korisniku sučelje koje olakšava programiranje, tj. učitavanje programa na sloj senzora. Prvi korak je otkrivanje čvorova senzora. Ako su senzori spojeni direktno na USB (engl. *Universal Serial Bus*) [27] pristup računala koje sadrži Excel, otkrivanje jednostavno znači pozvati odgovarajuće „TinyOS“ aplikacije, primjerice „Motelist“ te ju upitati za čvorove senzora. Ako su čvorovi spojeni na Ethernet mrežu, otkrivanje znači pregledavanje tablice koja sadrži podatke o rasporedu čvorova. Kada je otkrivanje gotovo, svaki čvor je prikazan kao jedna ćelija, kao što je prikazano na slici 3.3. [25].



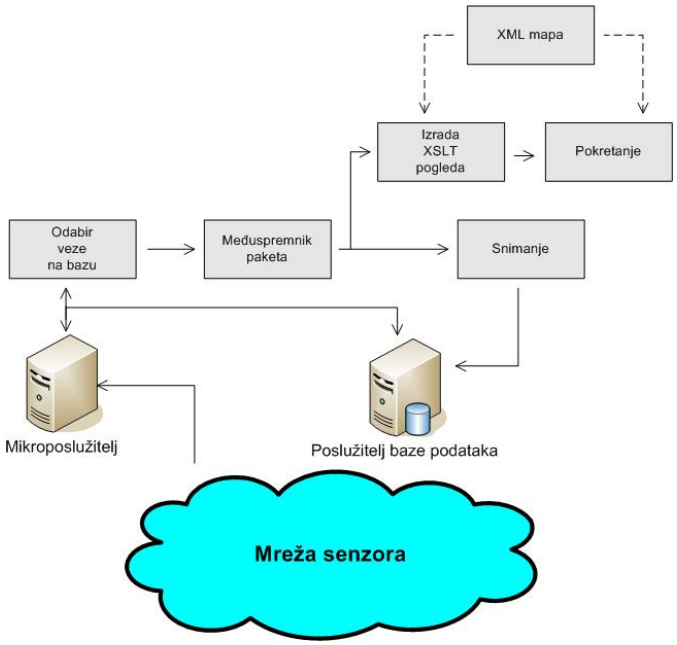
Slika 3.3. Učitavanje Tmote programa na senzore koristeći Excel

Korisnik izabire ćeliju koja predstavlja čvor na koji se treba učitati program. Svi „TinyOS“ programi su prikazani kao lista u Excelu i korisnik ih može vidjeti i izabrati.

Slanje programa na čvor može biti obavljeno preko USB-a, Ethernet-a ili bežično [25].

Kada je odabran čvor i program za učitavanje, pokreće se VBA skripta koja će izvršiti vanjske komande ljuske, kako bi se program preveo, namjestio identifikacijski broj čvora i konačno program učitao na njega [25].

Kao što je rečeno, druga uloga Excela je upit senzora za podacima te prikupljanje, analiziranje i prikaz tokova podataka koji su pristigli sa senzora. Za to se koristi MS Visual Studio 2005 i .NET 2.0, koji imaju ugrađenu vrlo dobru podršku za komunikaciju sa Excelom. Slika 3.4. prikazuje ključne komponente u ostvarenju aplikacije za rukovanje tokovima podataka pristiglim sa senzora [25].



Slika 3.4. Blok dijagram programa za rukovanje tokovima podataka pristiglim sa senzora

Senzori komuniciraju sa mikroserverskim poveznicima koji šalju podatke Excelu. Koristeći kontrole u Excelu korisnik može izabrati odgovarajući izvor podataka, izravno iz mreže senzora, ili arhivirane podatke sa SQL poslužitelja. Usmjerivač paketa međuspremnika prvo tokove podataka stavlja u red. Zatim dretva zadužena za prikaz podataka periodički uzima podatke iz međuspremnika te više njih spaja u oblik pogodan za prikaz koristeći XSL (engl. *eXtensible Stylesheet Language*) [28] transformacije, pri tome dodjeljujući svakom toku podataka jedinstveni prostor imena. Taj pogled je zatim prikazan koristeći ugrađenu mogućnost XML (engl. *Extensible Markup Language*) [29] mapiranja [25].

Poveznik ima pokrenut mikroposlužitelj µSEE koji apstrahira senzorski sloj. Svaki poveznik izvana izgleda kao mrežna usluga i s njim se komunicira pomoću XML protokola. Pošto su tokovi podataka prikazani kao XML, to znači da je njima lako rukovati od strane Excela [25].

4. Okolina za razvoj paralelnih primjenskih programa zasnovana na tabličnom programatoru

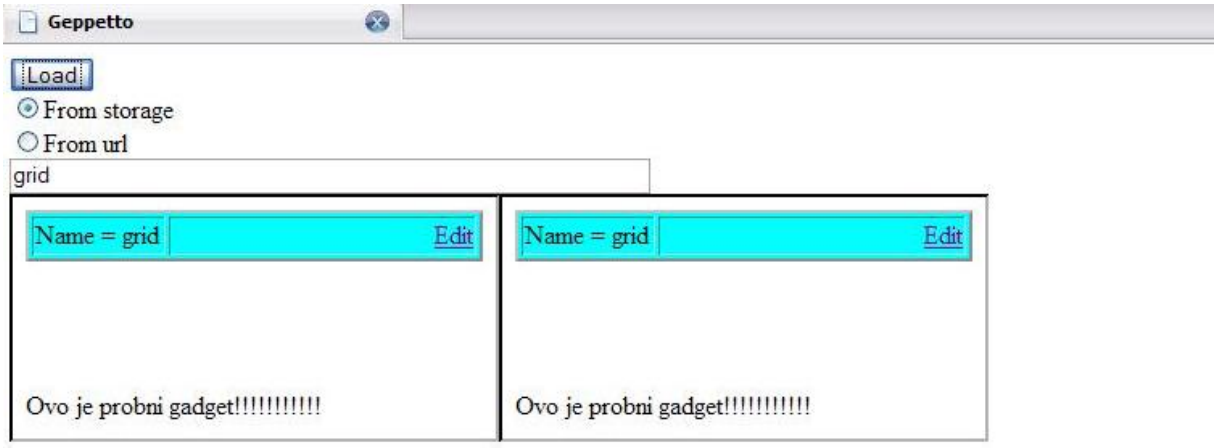
Cilj ovog diplomskog rada je programsko ostvarenje tabličnog programatora koji će se koristiti za razvijanje paralelnih programa. Naglasak razvijenog tabličnog programatora je na jednostavnosti korištenja od strane krajnjeg korisnika. Njegova primjena biti će pokazana na primjeru gadgeta. Gadgeti su mali programčići koji mogu prikazivati raznovrsne informacije korisniku. Kompozitni gadget je spoj nekoliko gadgeta. Programator treba moći dohvatiti kompozitne gadgete iz spremišta gadgeta na čvrstom disku, ili iz nekog mrežnog izvora, izmijeniti ga vizualno ili logički i naposljetku sačuvati ga nazad odakle ga je i uzeo. Upravo kompozitni gadget je raspodijeljena aplikacija te je izmjenjivanje njegove pozadinske logike paralelno programiranje. Raspravom i istraživanjem se došlo do zaključka da bi programiranje kompozitnih gadgeta bilo praktično činiti kroz tablični programator. Korisničko sučelje se sastoji od dvije web stranice. Prva služi za dohvat gadgeta, bilo iz spremišta gadgeta, bilo sa nekog mrežnog izvora te njihovo prikazivanje, a druga služi za izmjenjivanje pozadinske logike kompozitnih gadgeta. Spremište gadgeta je ostvareno kao web usluga. Funkcionalnost sučelja za učitavanje, izmjenu i spremanje gadgeta te spremište gadgeta biti će opisano u ovome poglavlju.

4.1. Web stranica za učitavanje gadgeta

Prva web stranica služi za učitavanje kompozitnih gadgeta. Omogućuje odabir želi li se gadget dohvatiti iz spremnika gadgeta ili iz mrežnog izvora. U polje za unos teksta se upisuje ime gadgeta za slučaj da se dohvaća iz spremnika, odnosno mrežni izvor za slučaj da se dohvaća preko mrežnog izvora. Slika 4.1. pokazuje početnu web stranicu sa odabranim dohvatom iz spremnika i upisanim imenom gadgeta u polje za unos teksta.

Slika 4.1. Početna web stranica

Kada korisnik klikne na gumb *load* gadget pod imenom *grid* se dohvaća iz spremnika i učitava na web stranicu. Ako korisnik još jednom klikne na gumb *load* gadget se opet dohvaća i učitava. Slika 4.2. prikazuje rezultat dva pritiska na gumb *load*.

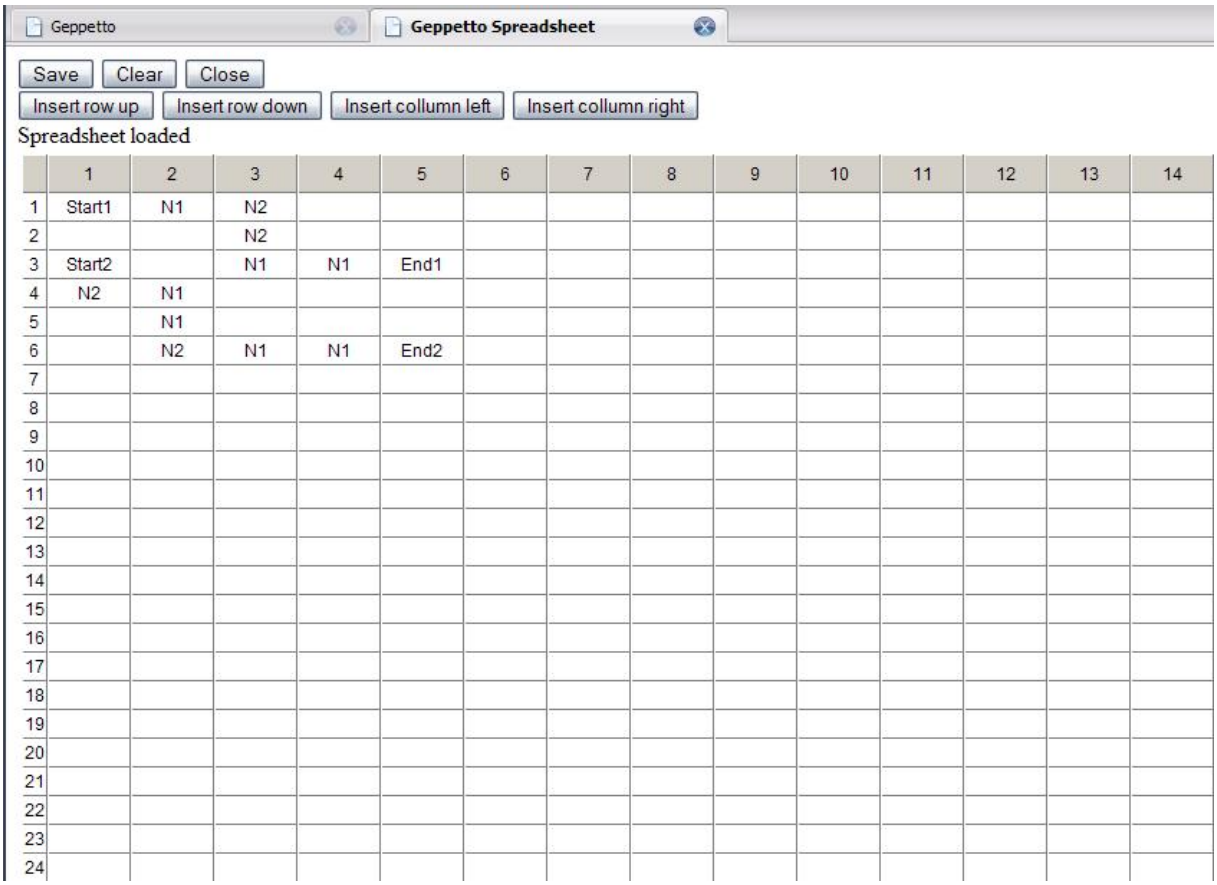


Slika 4.2. Učitani gadgeti

Dohvat i učitavanje ne uzrokuje ponovno učitavanje jer je dohvat gadgeta asinkron. Gadget se učitava na stranicu tako da se prvo dinamički stvori *iframe* a zatim se u njega učita gadget. *Iframe* je u principu web stranica unutar web stranice [30].

4.2. Web stranica za izmjenu i pohranjivanje gadgeta

Kao što se vidi na slikama, svaki gadget ima malo zaglavlje u kojem je ispisano ime samog gadgeta pod kojim je on sačuvan u spremniku te je izložen link *edit*. Pritiskom na taj link otvara se nova stranica koja služi za izmjenu i pohranjivanje gadgeta kojem je ta stranica pridružena. Ona na vrhu sadrži panel sa komandama, ispod kojega je polje koje ispisuje status tabličnog programatora. Većinu prostora zauzima sama tablica. Slika 4.3. prikazuje tablicu pridruženu učitanom gadgetu.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Start1	N1	N2											
2			N2											
3	Start2		N1	N1	End1									
4	N2	N1												
5		N1												
6		N2	N1	N1	End2									
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														

Slika 4.3. Web stranica pridružena učitanom gadgetu

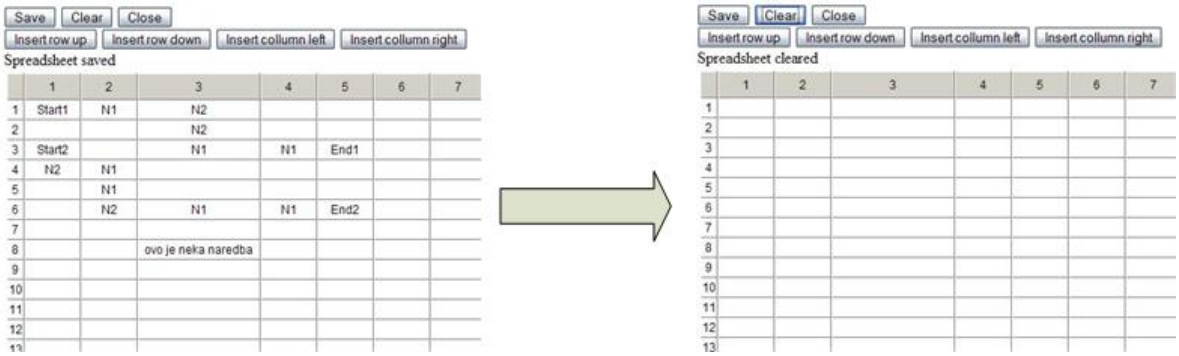
Tablica sadrži pozadinsku logiku kompozitnog gadgeta i opisuje na koji način su povezani ulazi i izlazi njegovih komponenata. U ovome primjeru su naredbe programskog jezika koji služi za definiranje pozadinske logike izmišljene. Konstrukti su samo dvije izmišljene naredbe *N1* i *N2* te naredbe za početak i kraj pojedinih procesa, odnosno slijednih dijelova, *StartN* i *EndN*, gdje je N redni broj slijeda naredaba. To je samo radi primjera, a zadiranje u detalje programskog jezika nije tema ovog diplomskog rada. Vremenska dimenzija ide odozgo prema gore i slijeva prema desno. Znači naredba u ćeliji koja se nalazi dolje ili desno od trenutne će se izvršiti nakon naredbe koja se nalazi u trenutnoj ćeliji. Sljedovi naredaba koji nemaju ćelija koje se dodiruju su neovisni i izvršavaju se paralelno. Dvostrukim klikom na pojedino polje se omogućuje njegova izmjena, tj. izmjena naredbe pridružene tome polju. Ako dužina teksta naredbe pridružene tome polje prelazi širinu stupca, on se automatski proširuje na dovoljnu širinu. Slika 4.4. prikazuje primjer unosa naredbe u neko polje.

Slika 4.4. Unos naredbe u polje tabličnog programatora

Panel sa naredbama sastoji se od linije za prikaz statusa razvojne okoline i upravljačkog dijela. Upravljački dio omogućuje korisniku spremanje napisanog programa na disk ili u mrežni izvor, brisanje radne površine, zatvaranje radne površine te ubacivanje redaka i stupaca u tablicu. Pritiskom na gumb *Save* program pridružen gadgetu se pohranjuje u spremnik pod istim imenom kao i gadget kojem je pridružen te se u statusnom polju ispisuje je li program uspješno sačuvan. Ako dođe do ikakvih problema prilikom spremanja programa u spremnik ispisuje se odgovarajuću poruku na ekranu.

Pritiskom na gumb *Save* program pridružen gadgetu se pohranjuje u spremnik pod istim imenom kao i gadget kojem je pridružen te se u statusnom polju ispisuje je li program uspješno sačuvan. Ako dođe do ikakvih problema prilikom spremanja programa u spremnik baca se iznimka koju web stanica hvata i ispisuje odgovarajuću poruku na ekranu.

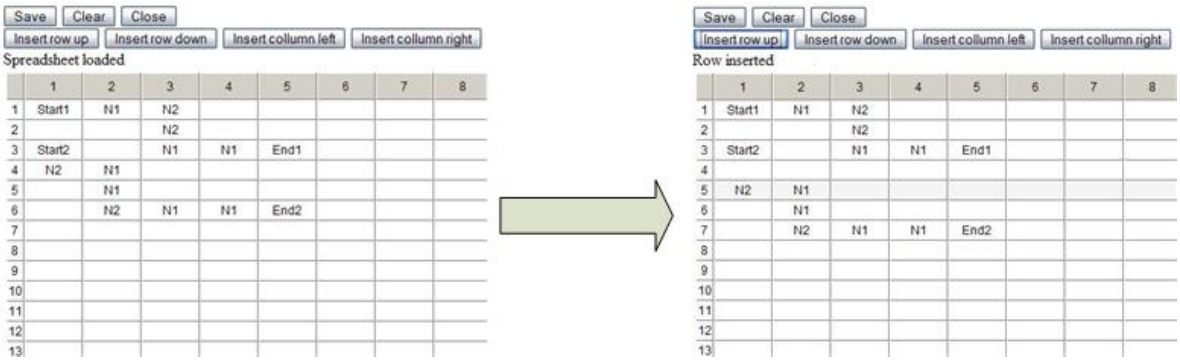
Pritiskom na gumb *Clear* briše se trenutačno učitani program. Slika 4.5. prikazuje rezultat pritiska na gumb *Clear*.



Slika 4.5. Pritisak na gumb *clear*

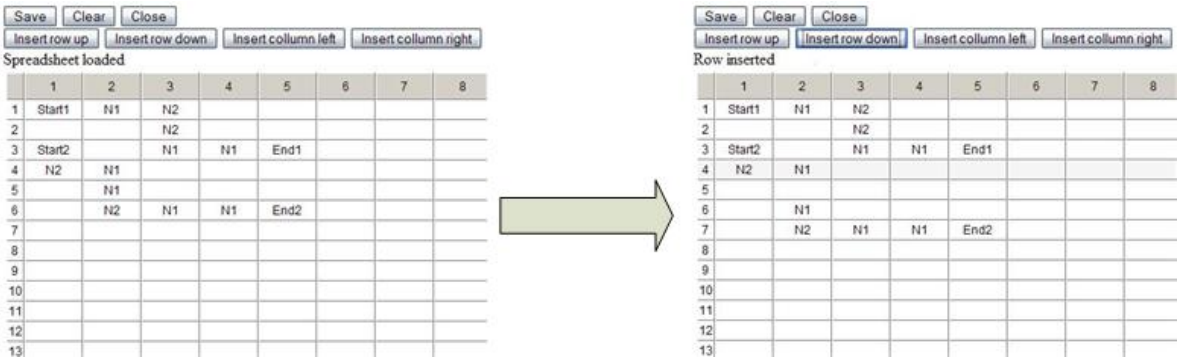
Pritiskom na gumb *Close* zatvara se web stranica koja sadrži tablični programator.

Za slijedećih šest gumbiju potrebno je prvo kliknuti na neku od ćelija kako bi tablica „znala“ u odnosu na koju ćeliju želimo umetnuti redak ili stupac. Nakon što se odabere primjerice ćelija u četvrtom retku i trećem stupcu i pritisne na gumb *Insert row up*, iznad te ćelije se umetne jedan prazan redak. Rezultat je prikazan na slici 4.6.



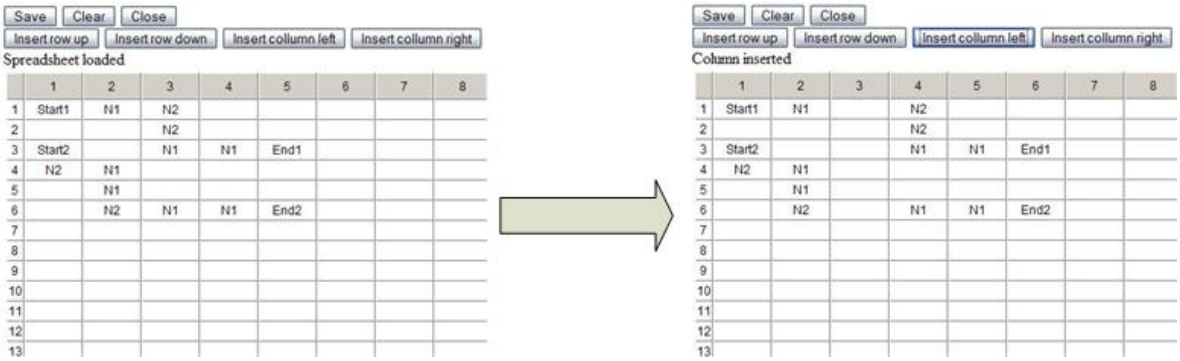
Slika 4.6. Pritisak na gumb *Insert row up*

Odabirom iste ćelije i pritiska na gumb *Insert row down* dolazi do rezultata prikazanog na slici 4.7.

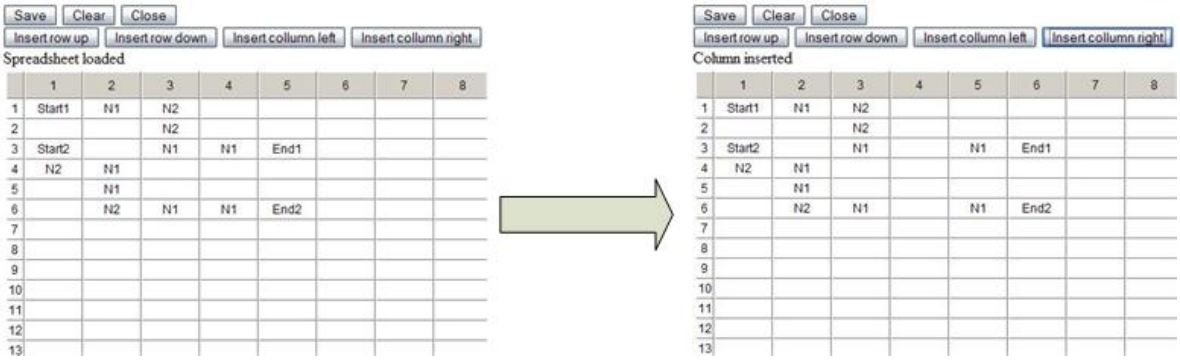


Slika 4.7. Pritisak na gumb *Insert row down*

Rezultati odabira iste ćelije i pritisaka na gumbe *Insert column left*, odnosno *Insert column right* prikazani su na slikama 4.8. i 4.9.



Slika 4.8. Pritisak na gumb *Insert column left*



Slika 4.9. Pritisak na gumb *Insert column right*

4.3. Spremnik gadgeta

Spremnik gadgeta se sastoji od dvije odvojene komponente. Od web usluge koja je sadržana u datotekama *Service.asmx* i *Service.asmx.cs* i koja vrši funkcionalnost spremnika, tj. spremanje i dohvat gadgeta i njima pridruženih paralelnih programa te od zastupnika (engl. *proxy*) [31] za tu web uslugu koji služi kao posrednik između sučelja tabličnog programatora i samog spremnika, kako sučelje ne bi izravno pozivalo spremnik. Sami spremnik ima izloženo nekoliko metoda koje se mogu vidjeti i u Internet pregledniku te „ručno“ pozivati. Tablica 4.1. prikazuje koje metode web usluga spremnika izlaže.

Tablica 4.1.

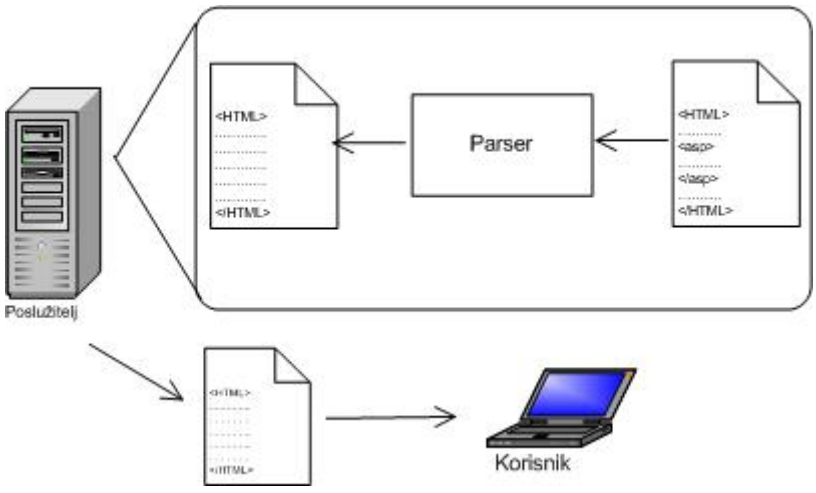
loadGrid	Vraća string koji predstavlja sadržaj tablice
loadGridFromXml	Vraća XML koji predstavlja sadržaj tablice
loadHTML	Vraća mrežni izvor u kojem je izložen HTML kod gadgeta
saveGrid	Sprema sadržaj tablice u datoteku na disku
saveGridToXml	Sprema sadržaj tablice kao XML datoteku u disku

5. Programsko ostvarenje razvojne okoline zasnovane na tabličnom programatoru

Programska okolina izrađena je u programskom paketu „Microsoft Visual Studio 2005“ [32]. Ona se nalazi unutar dvije prividne mape na poslužitelju IIS (engl. *Internet Information Services*) [33]. Unutar jedne se nalazi sučelje programske okoline, a unutar druge se nalazi web usluga spremnika sa svojim pripadnim datotekama. Slijede detalji ostvarenja prvo sučelja, a zatim i spremnika.

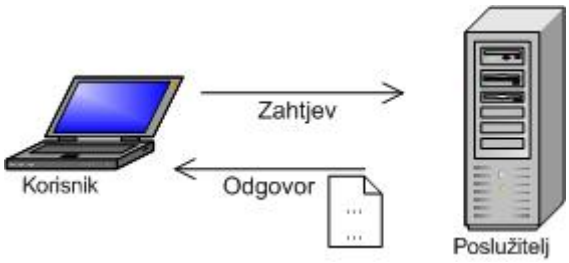
5.1. Programsko ostvarenje korisničkog sučelja

Sučelje programske okoline je ostvareno kao tri web stranice i jedna web usluga. Svaka web stranica sastoji se od dvije datoteke. Jedna određuje izgled stranice u Internet pregledniku i ima nastavak *.aspx* u imenu, a druga sadrži pozadinsku poslužiteljsku logiku i ima nastavak *.cs* u imenu. Kada IIS sazna da datoteka ima nastavak *.aspx* proslijedi ju platformi ASP.NET (engl. *Active Server Pages .NET*) [34] koja proparsira datoteku i napravi iz nje čistu HTML (engl. *HyperText Markup Language*) [35] stranicu izbacivši sve ASP (engl. *Active Server Pages*) [36] tagove. Prije parsiranja *.aspx* datoteka sadrži mješavinu ASP i HTML koda te Javascripta [37]. Rezultat parsiranja se može kontrolirati pozadinskom poslužiteljskom logikom koja se nalazi u *.cs* datoteci. U njoj je napisan kod u programskom jeziku C#. Znači ta datoteka donekle modificira HTML stranicu koju će dobiti korisnički Internet preglednik. Slika 5.1. prikazuje proces slanja web stranice korisniku.



Slika 5.1. Slanje web stranice korisniku

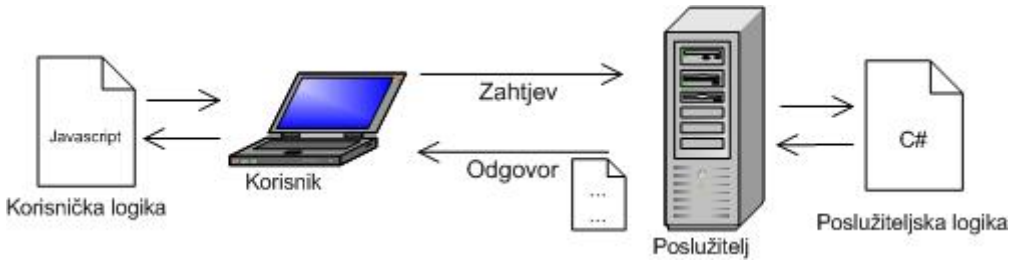
Problem je što korisnički Internet preglednik ne održava stalnu vezu sa poslužiteljem nego svaki puta iznova traži web stranicu, odnosno stanje se ne pamti. Komunikacija djeluje po principu zahtjev – odgovor. Slika 5.2. prikazuje zahtjev – odgovor princip komunikacije.



Slika 5.2. Zahtjev – odgovor princip komunikacije

Prividno održavanje stanja se može ostvariti koristeći skrivena polja za unos teksta, ili Ajax (engl. *Asynchronous JavaScript and XML*) [38] tehnologiju.

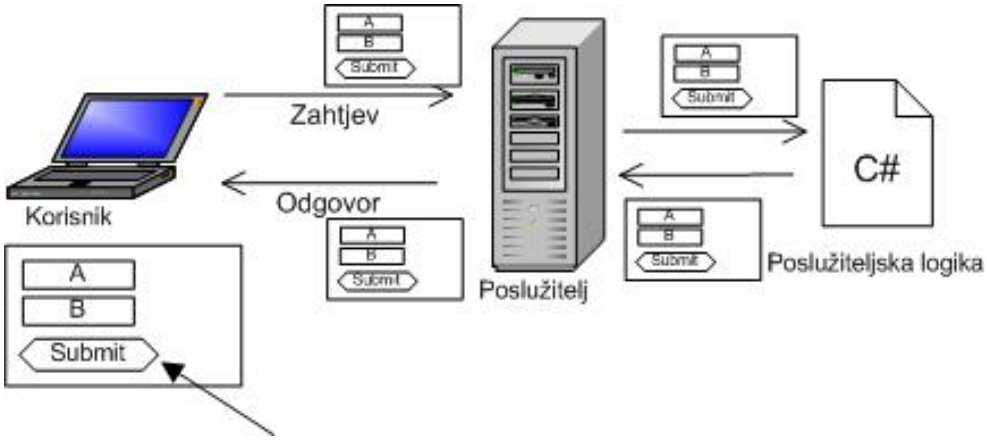
S druge strane nije dovoljna samo komunikacija sa poslužiteljem i poslužiteljska pozadinska logika, nego je potrebna i pozadinska logika koja će se izvršavati u samom Internet pregledniku, tj. korisnička pozadinska logika. Za razliku od poslužiteljske logike koja je napisana u programskom jeziku C#, korisnička logika je napisana u programskom jeziku Javascript. Pomoću njega se mogu dinamički stvarati, mijenjati i brisati HTML tagovi, a samim time se modificira i izgled web stranice te se mogu vršiti neka računanja na samom korisničkom računalu kako se ne bi zauzimala veza prema poslužitelju bez potrebe. Skripte napisane u programskom jeziku Javascript se mogu nalaziti u samoj HTML datoteci, a mogu se i nalaziti u posebnim datotekama te se u tom slučaju u glavnoj HTML stranici navode reference na nju. U ovome slučaju, glavnina skripata je stavljena u zasebne datoteke u mapu pod nazivom „scripts“ Slika 5.3. prikazuje izvršavanje korisničke i poslužiteljske logike.



Slika 5.3. Korisnička i poslužiteljska logika

5.1.1. Početna stranica

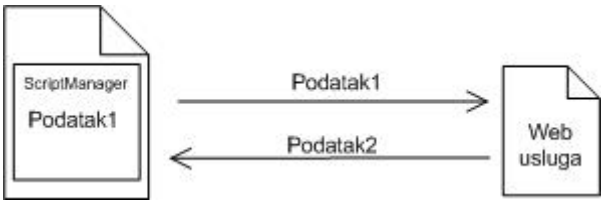
Početna stranica sadrži panel za učitavanje gadgeta. Na panelu se nalazi nekoliko HTML kontrola. To su polje za unos teksta, gumb i dva gumba za izbor. ASP.NET radi sa web formama. Web forma sadrži razne kontrole poput polja za unos teksta, gumbiju, izbornika i padajućih izbornika, itd... Kada korisnik popuni formu tada može pokrenuti njen *submit* događaj. To čini tako da klikne na gumb *Submit* ili bilo koji drugi gumb koji ima funkciju *submit* gumba, a može i programatski pokrenuti *submit()* metodu forme pomoću Javascript naredbe. Pokrenuvši *submit* događaj Internet preglednik sakuplja stanje forme, posprema ga u posebne GET [39] ili POST [39] varijable te šalje formu nazad na poslužitelj. Na poslužitelju se pročita stanje forme i ovisno o njemu čine određene akcije te se forma šalje nazad korisniku. Problem je što poslužitelj ne pamti stanje, nego eventualno može to isto stanje vratiti nazad korisniku. Korisnik, međutim, ne zna ništa o tome da je poslao formu poslužitelju. Forma se nanovo učitava i što se korisnika tiče ona se prvi puta učitala. Slika 5.4. prikazuje kako to izgleda.



Slika 5.4. Slanje forme na poslužitelj

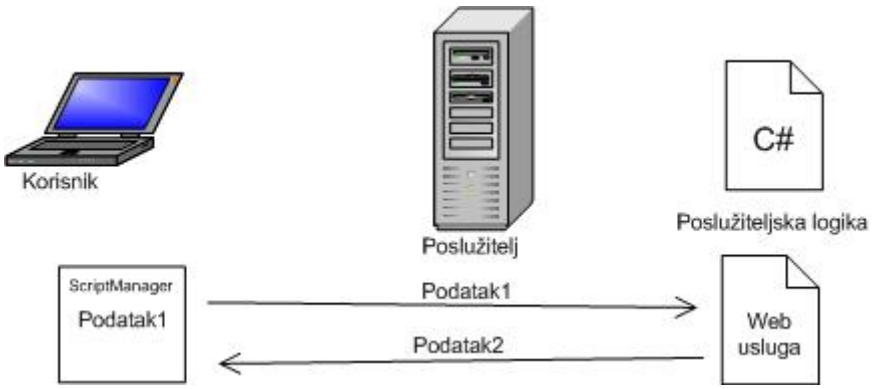
Prividno to izgleda dobro pošto je poslužitelj vratio stanje, tj. vrijednosti kontrola na formi, nazad, ali neke druge stvari ne ostaju sačuvane. Primjerice neki elementi koji su klijentskom pozadinskom logikom dinamički dodani na stranicu ne bivaju sačuvani. A upravo su gadgeti elementi koji se dinamički učitavaju na stranicu. Prilikom učitavanja novog gadgeta stari bi trebali ostati sačuvani, a to nije moguće bez naprednijih tehnologija. Tehnologija koja se ovdje koristi zove se Ajax. Ona omogućuje tri stvari. Prva je da se samo dijelovi stranice šalju nazad na poslužitelj, a ostali dijelovi ostaju učitani na korisničkom Internet pregledniku, druga stvar je

pozivanje poslužiteljskih metoda iz programskog jezika Javascript bez slanja čitave stranice, a treća je korištenje posebnih Ajax kontrola koje se same brinu za svoje stanje, tj. da ono ostane očuvano. Prvu stvar omogućuje Ajax *UpdatePanel* kontrola, drugu Ajax *ScriptManager* kontrola, a treću korisnički napisane Ajax kontrole. U ovome slučaju je korištena *ScriptManager* kontrola. Unutar nje se definira referenca na web uslugu koja se želi koristiti. Tada se mogu pozivati metode te web usluge bez slanja forme poslužitelju. Slika 5.5. prikazuje kako djeluje *ScriptManager* kontrola.



Slika 5.5. Djelovanje *ScriptManager* kontrole

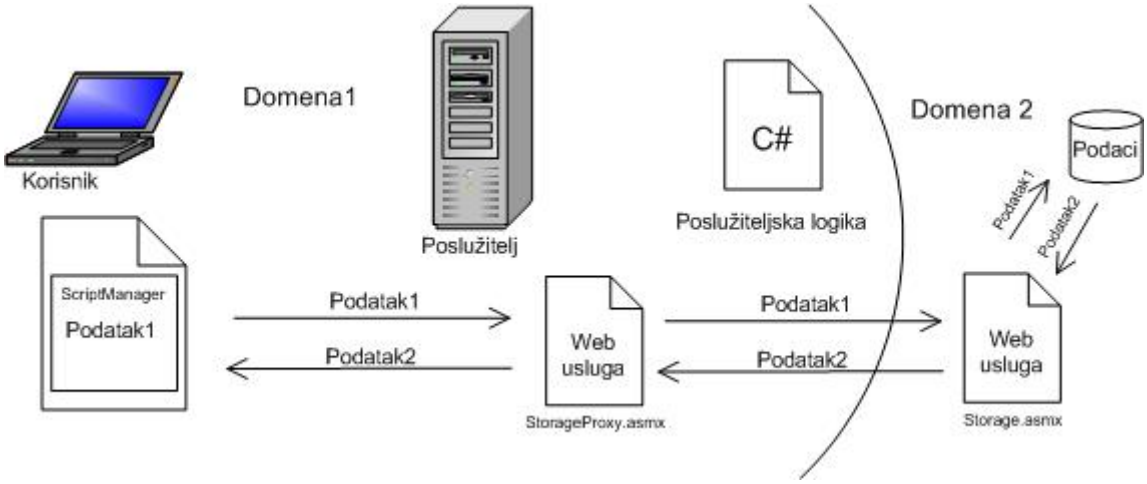
Znači pomoću *ScriptManager* kontrole poslužiteljska pozadinska logika se seli iz *.cs* datoteke pridružene stranici na web uslugu čija je referenca navedena u kontroli. Slika 5.6. prikazuje situaciju kada se koristi *ScriptManager* kontrola.



Slika 5.6. Poslužiteljska pozadinska logika i *ScriptManager* kontrola

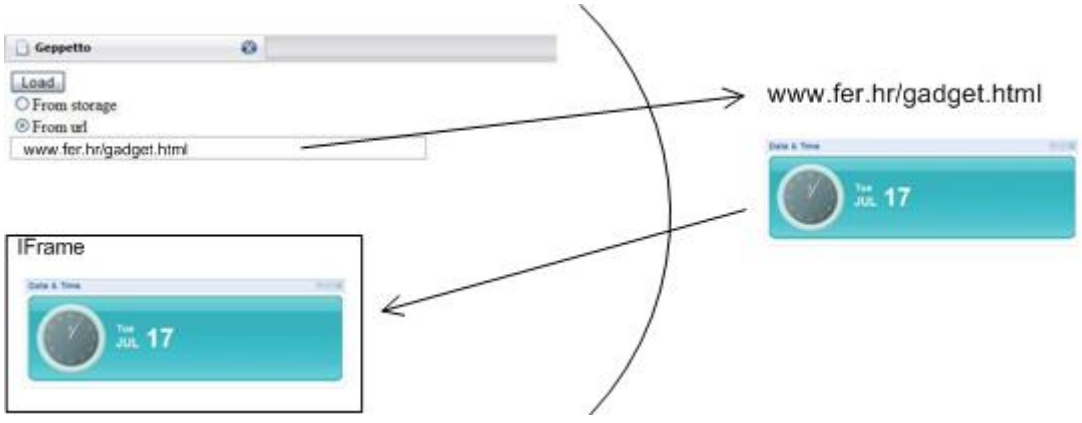
Međutim zbog sigurnosnih razloga web usluga koja se koristi za izvršavanje poslužiteljske pozadinske logike mora biti u istoj domeni kao i sama web stranica. Zato se ne može jednostavno web usluga spremnika navesti kao referenca u *ScriptManager* kontroli, pošto se nalazi u drugoj prividnoj mapi na poslužitelju, tj. u drugoj domeni. Rješenje je napraviti web uslugu koja je nazvana *StorageProxy* i koja se nalazi u istoj domeni kao i web stranica. Ona je navedena kao referenca u *ScriptManager* kontroli i njene metode se zovu sa početne web stranice. *StorageProxy* web usluga tada normalno poziva metode *Storage* web usluge pošto komunikacija između dvije web usluge nije toliko strogo ograničena. Ovaj način

programiranja se još zove *Proxy programski uzorak* [31]. Slika 5.7. prikazuje korištenje *StorageProxy* web usluge.



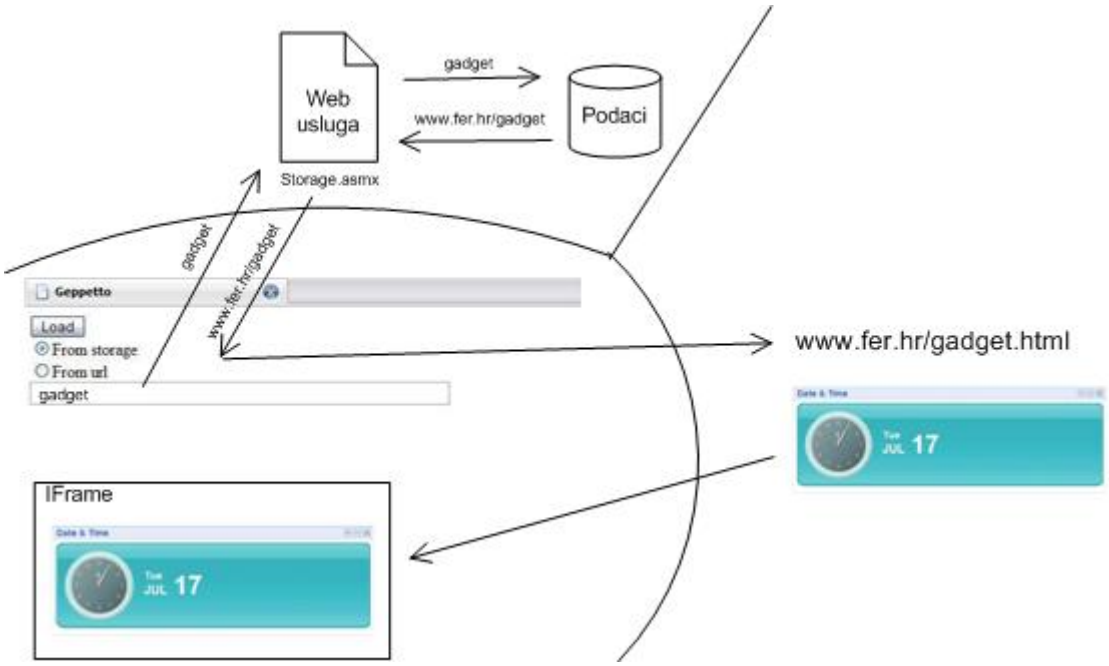
Slika 5.7. Proxy programski uzorak primijenjen na spremnik gadgeta

Konkretno u ovome slučaju, pritiskom na gumb *Load*, pokreće se Javascript funkcija koja obrađuje *Load* događaj toga gumba. Unutar te funkcije provjerava se što je izabrano gumbima za izbor. Mogućnosti su da se gadget učitava direktno iz mrežnog izvora, ili iz spremnika. Ako je izabrano da se učitava direktno iz mrežnog izvora, iz polja za unos teksta se čita mrežni izvor te se poziva Javascript funkcija *LoadGadget* s mrežnim izvorom kao parametrom. Ta funkcija dinamički stvara *iframe* HTML element i predaje mu mrežni izvor kao *src* atribut. To znači da će se unutar *iframe* elementa učitati web stranica iz mrežnog izvora navedenog kao *src* atribut. Gadgeti nisu ništa drugo nego male web stranice učitane unutar *iframe* elementa tako da je postignuto što se htjelo. Unutar *LoadGadget* funkcije još se određuju neke manje bitne stvari poput veličine *iframe* elementa. Slika 5.8. prikazuje učitavanje gadgeta direktno s mrežnog izvora.



Slika 5.8. Učitavanje gadgeta direktno s mrežnog izvora

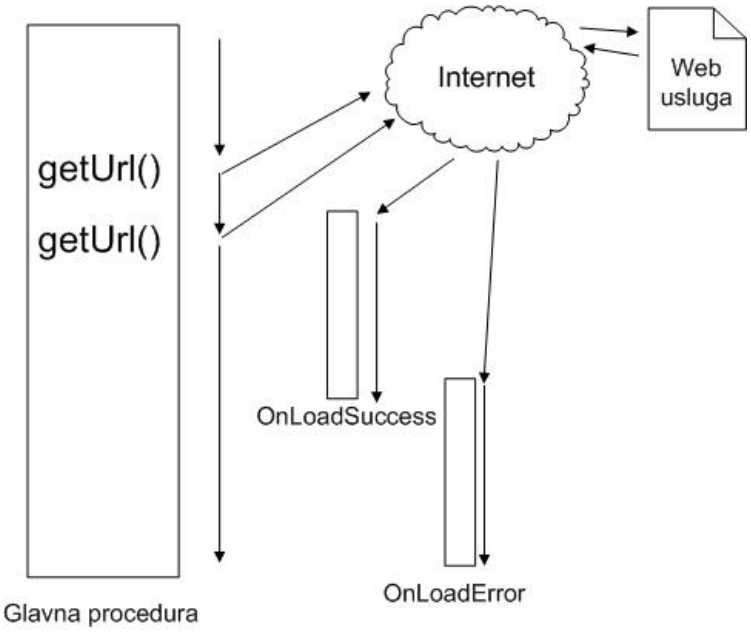
U slučaju da je odabrano da se gadget učitava iz spremnika, iz polja za unos teksta se čita ime gadgeta. Zatim se poziva *getUrl* metoda *StorageProxy* web usluge s imenom gadgeta kao parametrom. Ona poziva *loadHTML* metodu *Storage* web usluge s imenom kao parametrom. Unutar nje spremnik provjerava postoji li gadget sa tim imenom. Ako ne postoji vraća grešku, a inače generira mrežni izvor pod kojim se nalazi taj gadget unutar spremnika i vraća taj mrežni izvor sve do mjesta unutar glavne stranice gdje je pozvana metoda zastupnika. Nakon što se sazna mrežni izvor, poziva se ista funkcija *LoadGadget* kao i u prethodnom slučaju sa mrežnim izvorom kao parametrom i ta funkcija učitava gadget iz zadanog mrežnog izvora u dinamički stvoreni *iframe* element. Pošto se koristi *ScriptManager* kontrola, svaki put kada se klikne gumb *Load* stvara se novi *iframe* element sa učitanim gadgetom bez da su prethodno učitani izgubljeni. Slika 5.9. prikazuje učitavanje gadgeta iz spremnika.



Slika 5.9. Učitavanje gadgeta iz spremnika

Može se primijetiti da kada se gadget učitava iz spremnika, da se *LoadGadget* funkcija ne poziva odmah nakon poziva *getUrl* metode *StorageProxy* web usluge nego unutar jedne druge funkcije koja se zove *onLoadSuccess* Ta se druga funkcija poziva ako je poziv zastupničke metode uspješan. Postoji još jedna funkcija koja se zove *OnLoadError*. Može se isto tako primijetiti da *getUrl* metoda koja se poziva unutar programskog jezika Javascript nema isti potpis kao ista ta metoda unutar

StorageProxy web usluge. Metoda unutar programskog jezika Javascript ima još tri dodatna parametra koji su pokazivači na funkcije. Prvi je pokazivač na funkciju koja se poziva ako je poziv metode uspješan, drugi je pokazivač na funkciju koja se poziva ako je poziv metode neuspješan, a treći je pokazivač na funkciju koja se poziva ako je došlo do isteka regularnog vremena čekanja na rezultat metode (engl. *Timeout*). U ovome slučaju je pokazivač na zadnju metodu null pošto taj slučaj nije bilo potrebno obraditi. Prvi je pokazivač, pokazivač na već spomenutu funkciju *OnLoadSuccess*, a drugi je pokazivač, pokazivač na isto tako već spomenutu funkciju *OnLoadError*. Takav sustav omogućuje asinkronost poziva metode. Pošto je Internet okruženje jako nestabilno i puno se poziva gubi, asinkronost je nužna kako se ne bi bez razloga čekao rezultat poziva metode koji neće stići jer je izgubljen na Internetu. Slika 5.10. prikazuje princip asinkronih poziva.



Slika 5.10. Asinkroni poziv metode web usluge

5.1.2. Stranica sa radnom okolinom

Kada su svi željeni gadgeti učitani tada se mogu izmjenjivati. Svaki od gadgeta je napravljen tako da pamti svoje ime u posebnoj varijabli i ima link *Edit*. Pritiskom na link *Edit* zove se funkcija *_load*. Unutar nje se stvara mrežni izvor sa dvije GET varijable. To su varijabla *action* i varijabla *name*. Vrijednost mrežnog izvora je putanja na drugu stranicu koja čini ovo korisničko sučelje. Ta stranica sadrži radnu okolinu koja je u biti tablični programator. Kao vrijednost varijable *action* se stavlja „load“, a

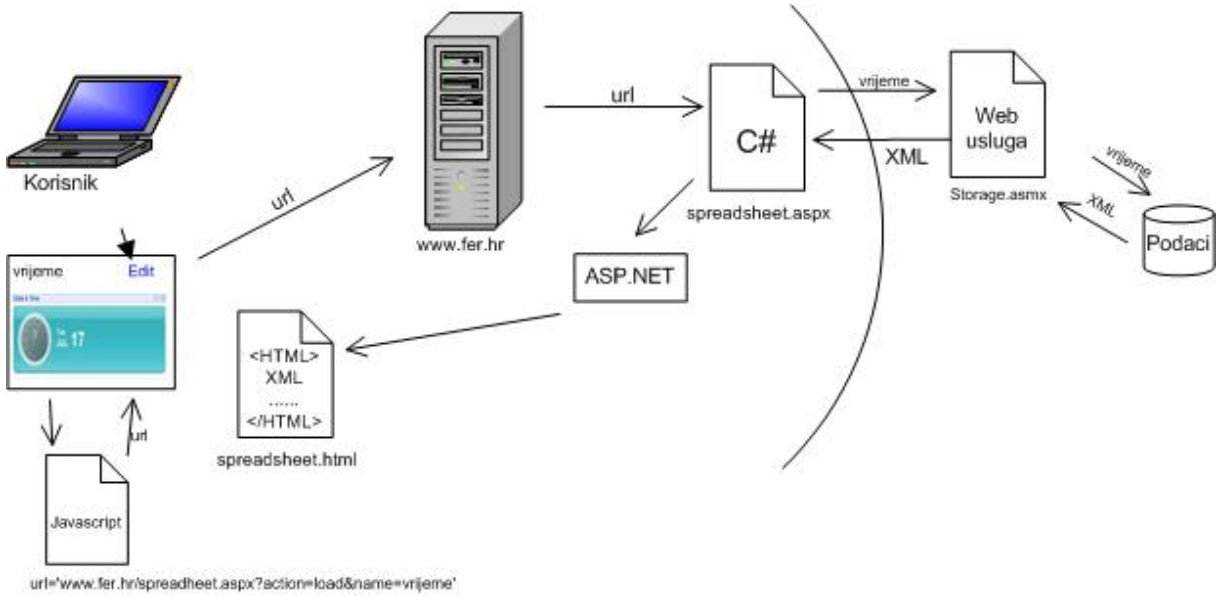
kao vrijednost varijable *name* se stavlja ime gadgeta, što znači da se želi od programatora da učita program gadgeta pod imenom koje je zadano u varijabli *name*. Kada se stvori mrežni izvor sa tim varijablama, tada se pokreće učitavanje stranice sa radnom okolinom.

Poslužiteljska pozadinska logika stranice sa radnom okolinom će se odvijati unutar pripadne *.cs* datoteke. Nije potrebno koristiti web uslugu kao zastupnika niti Ajax tehnologiju jer osim forme nema drugih elemenata koji ovise o učitavanju stranice i koji moraju biti sačuvani nego se prividno očuvanje stanja forme može ostvariti metodom skrivenih polja.

Stranica sa radnom okolinom je kompliciranija od početne stranice i sadrži već prije navedene gumbe *Save*, *Clear*, *Close*, *Insert row up*, *Insert row down*, *Insert column left*, *Insert column right*, polje sa statusom i tablicu. Osim prije navedenih sadrži i tri skrivena polja za unos teksta koja prije nisu navedena jer ih korisnik ne vidi. To su polja s imenima *txtCmd*, *txtData* i *txtName* i ona se nalaze unutar forme *storageProxy* koja se je isto sakrivena.

Kada gadget pozove stranicu s radnom okolinom poziva se *Page_Load* metoda unutar *spreadsheet.cs* datoteke. Unutar te metode se obrađuje stranica prije slanja korisniku. Provjerava se zahtijeva li se stranica prvi puta ili je već bila učitana u korisnički Internet preglednik pa se sada opet zahtijeva. Pošto je gadget zatražio učitavanje, u ovom slučaju se učitava prvi puta. Unutar programskog bloka koji obrađuje prvo učitavanje se čita vrijednost *action* i *name* GET varijabli. Već je prije rečeno da je vrijednost varijable *action* „load“, a u varijabli *name* je ime gadgeta. Za svaki slučaj se provjerava je li u varijabli *action* vrijednost „load“. Ako nije, tada se u skriveno polje za unos teksta pod imenom *txtCmd* stavlja vrijednost „init“ te se stranica šalje korisniku. Ako je u GET varijabli *action* vrijednost „load“, tada se poziva *loadGrid* metoda web usluge spremnika sa imenom gadgeta kao parametrom koja bi trebala vratiti XML string u kojem je pohranjen sadržaj tablice. Ako web usluga ne uspije iz nekog razloga učitati XML, desi se iznimka te se u polje za unos teksta *txtCmd* upisuje vrijednost „iznimka“, a u polje za unos teksta *txtData* se upisuje tekst iznimke te se tako inicijalizirana stranica šalje korisničkom Internet pregledniku. Ako je XML uspješno učitan, tada se u polje za unos teksta *txtCmd* upisuje vrijednost „storageLoadSpreadsheetResponse“, a u polje za unos teksta *txtData* se upisuje XML string dobiven od strane spremnika te se stranica šalje korisničkom Internet

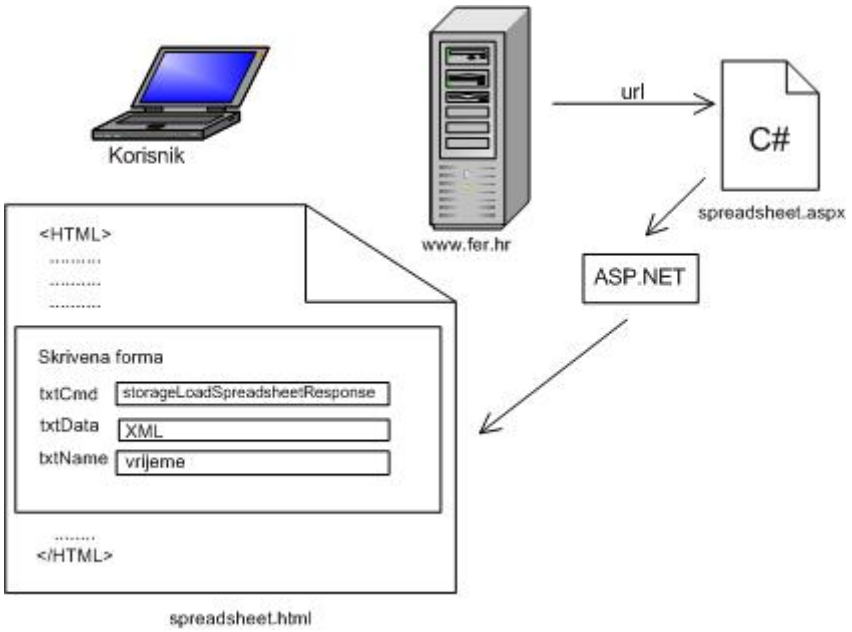
pregledniku. Slika 5.11. prikazuje proces zahtjeva za stranicom s radnom okolinom od strane gadgeta te ispunjenje tog zahtjeva.



Slika 5.11. Zahtjev gadgeta za web stranicom *spreadsheet.aspx*

Kada korisnik dobije stranicu od poslužitelja poziva se njegova funkcija za obradu događaja učitavanja. U ovom slučaju zove se funkcija *loadanje*. Unutar nje se provjeravaju vrijednosti skrivenih polja za unos teksta i na temelju njihovih vrijednosti se odlučuje što dalje. Prvo se provjerava vrijednost polja za unos teksta *txtCmd*. Ono može sadržavati pet vrijednosti: „storageLoadSpreadsheetResponse“, „storageSaveSpreadsheetResponse“, „init“, „storageException“ i „spreadsheetException“. Ako je vrijednost „storageLoadSpreadsheetResponse“, tada korisnički Internet preglednik zna da je se u polju za unos teksta *txtData* nalazi XML string sa sadržajem tablice kojeg treba učitati u tablicu. Ako je vrijednost „storageSaveSpreadsheetResponse“ , tada korisnički Internet preglednik zna da se u polju za unos teksta *txtData* nalazi XML string sa sadržajem tablice koji u biti nije izmijenjen i da ga treba ponovno učitati u tablicu. Ponovno učitavanje je nužno pošto se ne koristi Ajax tehnologija pa se stranica, a samim time i tablica, svaki puta ponovno učitava. Ako se u polju za unos teksta *txtCmd* nalaze vrijednosti „storageException“, ili „spreadsheetException“, tada se zna da je došlo do iznimke ili u spremniku ili negdje u datoteci *spreadsheet.cs*, tj. unutar poslužiteljske logike stranice s radnom okolinom. Ako je vrijednost „init“, tada se polje za unos teksta *txtData* ne čita i tablica se inicijalizira prazna. Polje za unos teksta *txtName* se uvijek čita kako bi korisnički Internet preglednik znao ime gadgeta. Kao što je rečeno, u ovoj

fazi se u polju za unos teksta *txtcmd* nalazi vrijednost „storageLoadSpreadsheetResponse“, a u polju za unos teksta *txtData* se nalazi XML string sa sadržajem tablice koji je poslužiteljska strana dohvatila iz spremnika i na ovaj način poslala korisničkoj strani. Slika 5.12. prikazuje proces slanja sadržaja tablice od poslužitelja prema korisniku.



Slika 5.12. Slanje sadržaja tablice od poslužitelja prema korisniku

U oba slučaja „storageLoadSpreadsheetResponse“ i „storageSaveSpreadsheetResponse“, sadržaj tablice je prikazan kao XML string kojeg treba prepisati u samu tablicu. Za to se koristi programsko sučelje izrađeno za manipulaciju tablicom. Naime ova tablica nije samo obična HTML tablica nego posebna web kontrola. Ona ima dosta funkcionalnosti, ali opet nema sve što je potrebno. Iz tog razloga je izrađeno programsko sučelje koje omotava tablicu i u kojem su dodane neke stvari koje su potrebne, a sama tablica ih nema, a maknute su stvari koje sama kontrola ima, a nisu potrebne. U tablici 5.1. dan je popis funkcija koje čine sučelje.

Tablica 5.1.

grid2string	Kodira sadržaj tablice u posebno formatirani string
grid2xml	Kodira sadržaj tablice kao XML string
string2grid	Dekodira posebno formatirani string kao sadržaj tablice i upisuje sadržaj u tablicu
xml2grid	Dekodira XML string kao sadržaj tablice i upisuje sadržaj u tablicu
sastaviInitGridFunkciju	Na temelju programatski zadanih parametara sastavlja funkciju koja služi za inicijalizaciju tablice
sastaviInitGridFunkcijuIzXmla	Na temelju pročitanog XML stringa zaključuje koja je početna veličina tablice te na temelju tih podataka sastavlja funkciju koja služi za inicijalizaciju tablice
getCell	Za zadani redak i stupac vraća vrijednost ćelije
setCell	Za zadani redak i stupac postavlja vrijednost ćelije
getNumRows	Vraća broj redaka
getNumColls	Vraća broj stupaca
clearGrid	Briše sve ćelije u tablici
resizeAllColumns	Mijenja širinu svih ćelija u skladu sa veličinom teksta koji je u njima pohranjen
insertRow	Umeće prazan redak na zadanu poziciju
insertColumnIntoXml	Umeće prazan stupac u XML string koji predstavlja sadržaj tablice
setHeaderAndInit	Inicijalizira zaglavlje tablice
setRows	Inicijalizira retke tablice koji imaju neki sadržaj

Za slučaj prevođenja sadržaja tablice iz XML stringa u samu tablicu koristi se funkcija *xml2grid*. Slijedi objašnjenje funkcije. Prvo se provjeri koji je Internet preglednik, a na temelju toga se odredi koji API za manipulaciju XML-om treba koristiti. Zatim se koristeći odgovarajući API parsira XML string. Slika 5.13. prikazuje strukturu XML dokumenta koji se koristi za spremanje sadržaja tablice.

```
<?xml version="1.0" encoding="UTF-8"?>
<rows>
  <row id="1">
    <cell>..</cell>
    <cell>..</cell>
      ......
    <cell>..</cell>
  </row>
  ........
  <row id="N">
    <cell>..</cell>
    <cell>..</cell>
      ......
    <cell>..</cell>
  </row>
</rows>
```

Slika 5.13. Struktura XML dokumenta u kojem je pohranjen sadržaj tablice

Za vrijeme parsiranja pamti se broj retka i stupca u kojem se trenutno nalazi te kada se naiđe na čvor „cell“, vrijednost čvora se pomoću funkcije *setCell* zapiše u tablicu.

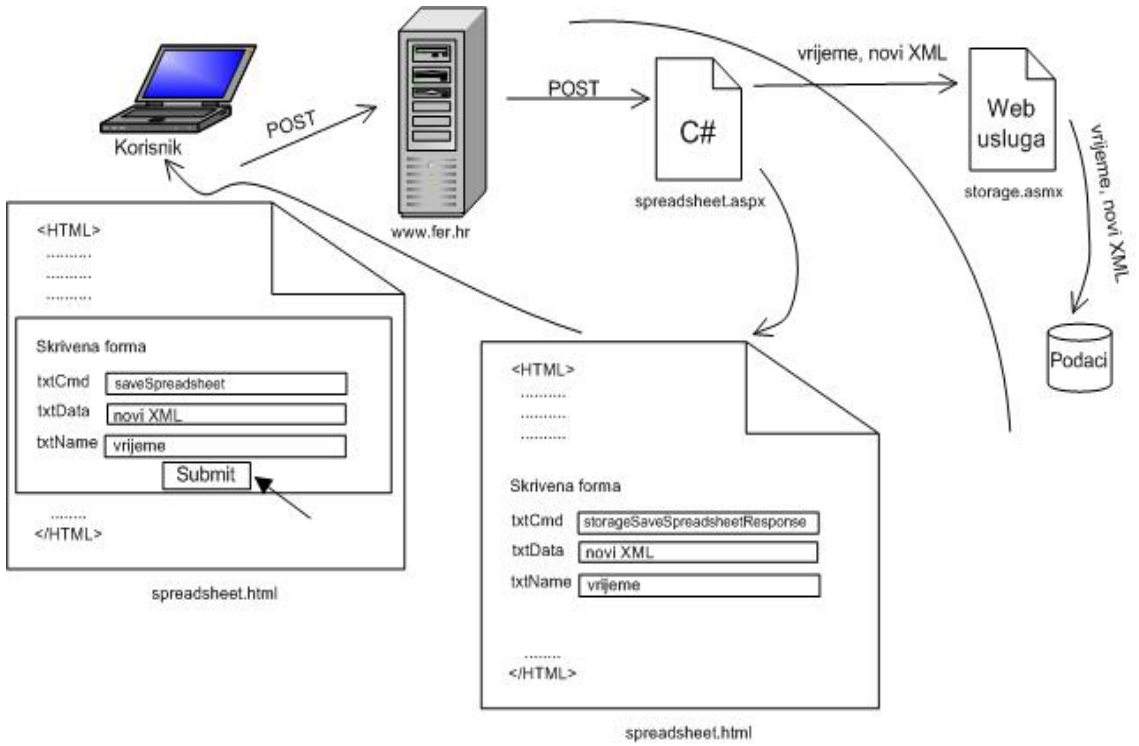
Sada je razumljiv cjelokupni proces što se dešava prilikom klika na link *Edit* na gadgetu: Sastavlja se mrežni izvor u kojem su pohranjene GET varijabla *action* sa vrijednošću „load“ i GET varijabla *name* u kojem je pohranjeno ime gadgeta koji želi otvoriti tablični programator. S tim mrežnim izvorom se dohvaća stranica s radnom okolinom. Prije slanja korisniku izvršava se poslužiteljska pozadinska logika u datoteci *spreadsheet.cs*. Tamo se vidi da je ovo prvi zahtjev za stranicom, odnosno da je zahtijeva gadget te da nije slučaj da stranica samu sebe šalje nazad na poslužitelj radi modifikacije. Poslužitelj zatim iz spremnika dohvaća XML string sa sadržajem tablice za zadano ime, posprema ga u skriveno polje za unos teksta *txtData* te u skriveno polje za unos teksta *txtCmd* stavlja vrijednost „storageLoadSpreadsheetResponse“. Zatim šalje stranicu korisniku. Korisnik pri učitavanju stranice izvršava korisničku pozadinsku logiku: Čita skrivena polja, vidi da se u polju *txtCmd* nalazi vrijednost „storageLoadSpreadsheetResponse“ te zna da se u polju *txtData* nalazi XML string. Čita taj string i koristeći sučelje za manipulaciju tablicom prenosi sadržaj u samu tablicu.

Sve ovo dosad je bio slučaj komunikacije poslužiteljske logike prema korisničkoj logici. Kada korisnička logika želi komunicirati sa poslužiteljskom situacija je slična. Proces te komunikacije biti će objašnjen na primjeru klika na gumb *Save*.

Klikom na gumb *Save*, zove se *_save* funkcija unutar stranice s radnom okolinom. Ona radi vizualne pripreme. Primjerice u polje za prikaz statusa upisuje vrijednost „Please wait..“ te nakon toga zove funkciju *saveSheet* koja se nalazi unutar datoteke *GeppettoLibrary.js* sa parametrima koji su pokazivač na tablicu, pokazivač na formu sa skrivenim poljima te ime gadgeta čiji je program učitan u tablicu. Funkcija *saveSheet* pomoću pokazivača na tablicu ima pristup svim poljima tablice, tako da može iskoristiti funkciju *grid2xml* sučelja za manipulaciju tablicom. Ta funkcija iterira od polja do polja tablice te sastavlja XML string strukture koja je navedena. Zatim se pomoću pokazivača na *storageProxy* formu koja sadrži skrivena polja u polje za unos teksta *txtCmd* upisuje vrijednost „saveSpreadsheet“, u polje za unos teksta *txtData* se upisuje prethodno sastavljeni XML string, a u polje *txtName* se upisuje ime gadgeta. Na kraju se poziva *submit* metoda *storageProxy* forme što uzrokuje

spremanje njenih polja u POST varijable te slanje stranice na poslužitelj i njen ponovni zahtjev.

Poslužitelj prije no što će vratiti stranicu poziva *Page_Load* metodu, isto kao i kada je gadget poslao zahtjev za stranicom. Međutim poslužiteljska logika sada vidi da se stranica ne zahtijeva prvi puta, nego da je već bila jednom poslana. Znači da korisnička logika želi komunikaciju sa poslužiteljskom. Čita se polje za unos teksta *txtCmd* i u ovisnosti o vrijednosti se odlučuje o daljnjoj akciji. Mogućnosti su „saveSpreadsheet“ i „loadSpreadsheet“. U ovome slučaju je vrijednost „saveSpreadsheet“, što govori poslužiteljskoj logici da iz polja *txtData* treba pročitati XML string, iz polja *txtName* ime gadgeta te pozvati *saveGrid* metodu web usluge spremnika sa parametrima koje čine XML string i ime gadgeta. Spremnik tada sačuva taj XML string kao datoteku na disku pod imenom *imeGadgeta.XML*. Polja za unos teksta *txtCmd*, *txtData* i *txtName* su dostupna poslužiteljskoj logici pošto su bila sačuvana u POST varijablama, jer je forma kojoj pripadaju ta polja podnijela zahtjev za stranicom. Proces komunikacije od korisnika prema poslužitelju je prikazan na slici 5.14.



Slika 5.14. Proces komunikacije od korisnika prema poslužitelju

Korisnička pozadinska logika se izvršava kada se kliknu ostali gumbi osim *Save* gumba. Ostali gumbi koriste funkcije programskog sučelja tablice. Pritiskom na gumb

Clear poziva se funkcija *_clr* koja se nalazi u datoteci *spreadsheet.aspx*. Ona manipulira sučeljem, odnosno poljem za prikaz statusa. Upisuje u njega „Please wait..“, zatim zove funkciju sučelja tablice *clearGrid*, a kao parametar predaje pokazivač na tablicu te na kraju u polje za prikaz statusa upisuje „Spreadsheet cleared“. Funkcija *clearGrid* pomoću funkcija sučelja *getNumRows* i *getnumColls* saznaje koliko redaka, odnosno stupaca ima tablica, a zatim u petlji iterira po tablici te pomoću funkcije sučelja tablice *setCell* upisuje u svaku ćeliju vrijednost „ “, što označava prazninu ćeliju u prikazu HTML tablice (engl. *Non Breakable SPace*). Nije dovoljno kao vrijednost upisati prazan string „“, ili razmak „ “ , jer neki Internet preglednici zahtijevaju da prazna ćelija u tablici bude označena sa „ “, inače je neće dobro prikazati.

Pritiskom na gumb *Close* zove se funkcija *_close* koja poziva *close* metodu *window* objekta. *window* objekt je standardan objekt DOM-a (engl. *Document Object Model*) [40] i dopušta manipulaciju prozorom Internet preglednika.

Pritiskom na gumb *Insert row up* poziva se funkcija *_insup* u datoteci *spreadsheet.aspx*. Ona u polje za prikaz statusa upisuje „Please wait...“, zatim poziva *insertRow* funkciju sučelja tablice sa parametrima koji su redni broj retka ispod kojeg se želi ubaciti novi redak i pokazivač na tablicu, a na kraju u polje za prikaz statusa upisuje „Row inserted“. Primijetite da funkcija sučelja za manipulaciju tablicom ubacuje redak ispod trenutnog, tako da bi redak ubacio iznad morao joj se kao parametar predati redni broj trenutnog retka umanjen za jedan.

Funkcija insertRow koristi već postojeću funkciju kontrole *addRow*, međutim *insertRow* je omotač oko te funkcije koji prvo inicijalizira redak praznim ćelijama, doda ga koristeći funkciju *addRow*, a zatim nanovo označi prvi stupac sa rednim brojevima redaka. To nije kompliciranо, ali je ovaj način puno transparentniji za korisnika.

Pritiskom na gumb *Insert row down* poziva se funkcija *_insdown* koja se nalazi u datoteci *spreadsheet.aspx* i koja radi sve isto kao i funkcija *_insup*, samo što funkciju *addRow* zove s trenutnim rednim brojem retka, a ne s trenutnim brojem retka umanjenim za jedan.

Pritiskom na gumb *Insert column left* i *Insert column right* poziva se funkcija *_insleft* u datoteci *spreadsheet.aspx*. Ona pomoću metode same kontrole tablice *getSelectedCellIndex* saznaje u kojem je stupcu trenutno odabrana ćelija. Ako je trenutni stupac krajnje lijevi, a pritisnut je gumb *Insert column left*, ništa se ne

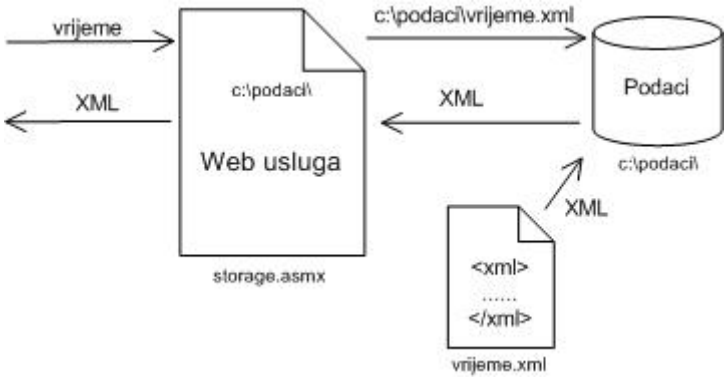
događa. Isto tako ako je trenutni stupac krajnje desni a, pritisnut je gumb *Insert column right*, opet se ništa ne događa. Inače se u polje za prikaz statusa upisuje vrijednost „Please wait...“, zatim se poziva funkcija sučelja tablice *insertColumn*, a na kraju se u polje za prikaz statusa upisuje vrijednost „Column inserted“. Funkcija *insertColumn* kao parametar prima redni broj stupca s čije će se desne strane umetnuti novi stupac, tako da ako je pritisnut gumb *Insert column left*, tada se ta funkcija poziva s parametrom koji je vrijednost trenutno odabrane ćelije umanjena za jedan.

Funkcija *insertColumn* djeluje drugačije nego funkcija *insertRow*, jer ne postoji funkcija kontrole koja umeće stupac, za razliku od umetanja retka, za što postoji funkcija kontrole koja to radi. Unutar funkcije *insertColumn* prvo se sadržaj tablice pretvara u XML string pomoću funkcije sučelja tablice *grid2xml* te se taj sadržaj pamti. Zatim se tablica uništava te nanovo stvara, ali ovaj put sa jednim stupcem više. Stvara se funkcijom sučelja tablice *sastaviInitGridFunkciju*, koja se koristi za stvaranje tablice i na početku pri učitavanju stranice. Zatim se funkcijom *insertColumnIntoXml*, koja kao argument prima XML string i mjesto na koje se želi umetnuti stupac, taj stupac umeće u XML string koji je prethodno sastavljen na temelj u sadržaja tablice. Unutar funkcije *insertColumnIntoXml* prvo se gleda koji se Internet preglednik koristi kako bi se mogao odrediti API koji će se koristiti za manipulaciju XML stringom. Nakon toga se kreće sa parsiranjem XML stringa. Unutar svakog para tagova <*row*> i <*/row*> nalazi se niz čvorova koji označavaju pojedinu ćeliju. Sadržaj ćelije nalazi se unutar tagova <*cell*> i <*/cell*>. Prilikom parsiranja pojedinog *row* čvora broji se koliko *cell* čvorova se prošlo prije zatvarajućeg <*/row*> taga. Kada se brojanjem ustanovi da se došlo do mjesta na koje treba ubaciti stupac, tada se umeće u XML string novi *cell* čvor sa vrijednošću „ “ unutar tagova <*cell*> i <*/cell*>. Kada se dođe do zatvarajućeg <*/row*> taga, tada se brojač ponovno postavlja na nulu te se tako radi za svaki *row* čvor. Umetanjem praznih *cell* čvorova u svakom *row* čvoru na mjesto gdje treba doći novi stupac, postiže se učinak umetanja novog stupca na zadanu poziciju. Kako bi to bilo vidljivo i u tablici, na kraju *insertRow* funkcije se taj novi XML string funkcijom programskog sučelja tablice *xml2grid* preslikava u tablicu.

5.2. Programsko ostvarenje spremnika gadgeta

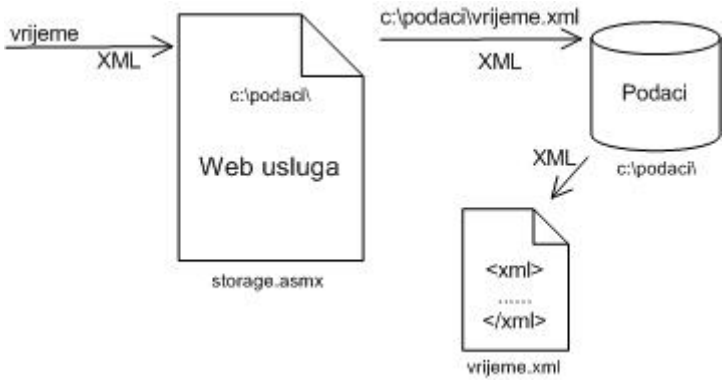
Spremnik gadgeta je ostvaren kao web usluga. Sastoji se od datoteka *Service.asmx* i *Service.cs*. Logika spremnika se nalazi u datoteci *Service.cs*. Ona sadrži kod koji je napisan u programskom jeziku C#. Logiku čini više metoda. Neke su označene atributom *WebMethod*, što znači da će te metode biti vidljive preko Interneta te ih se može pozivati. One čine sučelje web usluge. Neke druge metode nemaju taj atribut i one su privatne metode koje sama usluga koristi za neku svoju unutarnju logiku i operacije. Od nekoliko metoda koje čine sučelje, ovdje je bitno njih tri koje su na kraju iskorištene u ovom diplomskom radu. To su metode *loadGrid*, *saveGrid* i *loadHTML*. Sve tri su vrlo jednostavne i manipuliraju datotekama. Kao putanje do datoteka koriste predefinirane putanje koje se pamte u konstantama unutar samog servise. Makar to može biti drugačije, u ovome slučaju se i HTML kod gadgeta i sadržaj u obliku XML datoteke nalaze u jednoj mapi u korijenskom direktoriju web usluge. Ta mapa se zove *StorageData*.

loadGrid metoda funkcionira tako da iz imena gadgeta koje prima kao parametar i putanje do mape koje služi kao fizičko spremište datoteka na disku sastavlja putanju do XML datoteke unutar koje je spremljen sadržaj tablice programa pripadajućeg gadgeta. Zatim se ta XML datoteka čita te se pročitani sadržaj vraća kao rezultat metode pozivatelju. Slika 5.15. prikazuje djelovanje *loadGrid* metode.



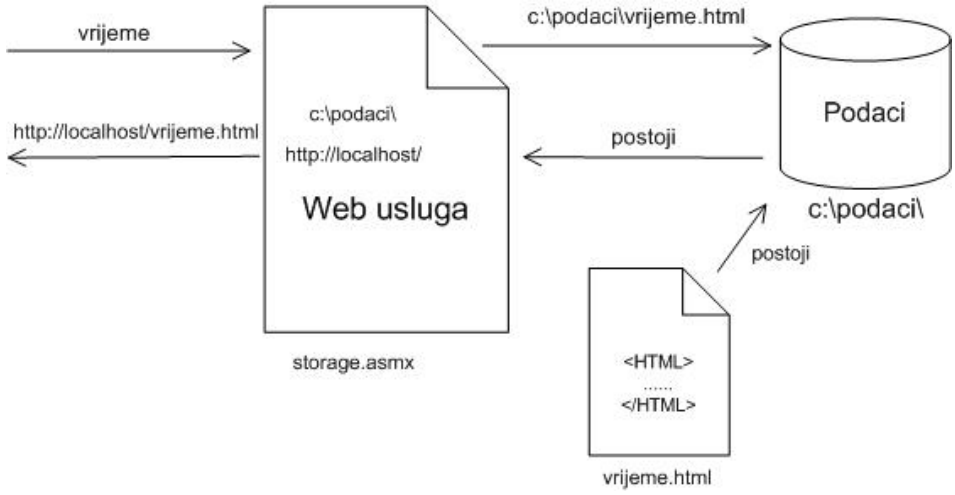
Slika 5.15. Djelovanje *loadGrid* metode

saveGrid metoda djeluje na sličan način kao i *loadGrid* metoda. Razlika je u tome što *saveGrid* metoda kao parametar osim imena gadgeta dobiva i XML string sa sadržajem tablice u kojoj se nalazi njegov program. Isto se generira putanja do datoteke, samo što se njen sadržaj ne čita njego se u nju upisuje XML string koji je dobiven kao parametar metode. Slika 5.16. prikazuje djelovanje *saveGrid* metode.



Slika 5.16. Djelovanje *saveGrid* metode

loadHTML metoda djeluje malo drugačije. Ona isto na temelju imena i putanja do fizičke mape na disku sastavlja putanju do same datoteke unutar koje se nalazi HMTL kod gadgeta, ali ga ne čita nego samo provjerava postoji li on. Ako postoji sastavlja mrežni izvor te vraća pozivatelju mrežni izvor na kojem može dohvatiti gadget i izravno ga iz tog mrežnog izvora učitati u *iframe* element na stranici. Slika 5.17. prikazuje djelovanje *loadHTML* metode.



Slika 5.17. Djelovanje *loadHTML* metode

6. Tehnologije i alati korišteni za ostvarenje razvojne okoline zasnovane na tabličnom programatoru

Kao i svaka moderna tehnologija, tako i web tehnologija svakim danom postaje sve složenija. Web tehnologije su nastale na nekim jednostavnim principima poput čitanja običnih tekstualnih datoteka i izmjenjivanja poruka, da bi s vremenom bili dodavani sve složeniji elementi koji su od tekstualne datoteke napravili HTML stranicu sa statičkim sadržajem pa je zatim došao CGI (engl. *Common Gateway Interface*) [41] koji je HTML stranice napravio dinamičkima, zatim ASP koji je unapređenje CGI-a, preko ASP.NET-a i AJAX-a. U ovom diplomskom radu koriste se HTML, ASP.NET i AJAX tehnologije, programski jezici C# i Javascript te Web usluge.

6.1. HTML, CGI, ASP, ASP.NET, Javascript

Počeci WWW-a (engl. *World Wide Web*) [42] sežu u osamdesete godine prošlog stoljeća kada je Tim Berners-Lee [42] kao suradnik u Cern-u napravio Enquire, osobnu bazu podataka u kojoj su zapisane razne stvari, ali ono što je bitno je da je svaka stranica s podacima bila povezana s drugim stranicama putem hiperteksta, odnosno današnjih linkova.

Dvije godine koje su isto bitne za razvoj WWW-a su 1977. i 1978. kada su Bob Khan i Vint Cerf izmislili TCP (engl. *Transmission Control Protocol*) [43], odnosno IP (engl. *Internet Protocol*) [44]. Kombinacija TCP/IP protokola je 1983. službeno usvojena od strane ARPANET-a (engl. *Advanced Research Projects Agency Network*) [45], prve mreže koja koristi pakete za prijenos podataka te se može nazvati prethodnikom Interneta.

1984. Tim Berners-Lee vratio se u Cern te počeo rješavati problem prikaza i dostupnosti velike količine podataka koje su sakupljali fizičari u Cern-u i koji su trebali biti dostupni svakome tko je radio na problemu. Do Božića 1990. Lee je napravio sve tri stvari koje su bitne za da bi Web radio: prvi Web preglednik, prvi Web poslužitelj te prve Web stranice napisane u HTML-u koje su opisivale sami projekt [42].

Datum 6.8.1991 i objavljivanje World Wide Web projekta na newsgrupi alt.hypertext se smatra prvim danom kada je WWW postala javno dostupna usluga na Internetu [42].

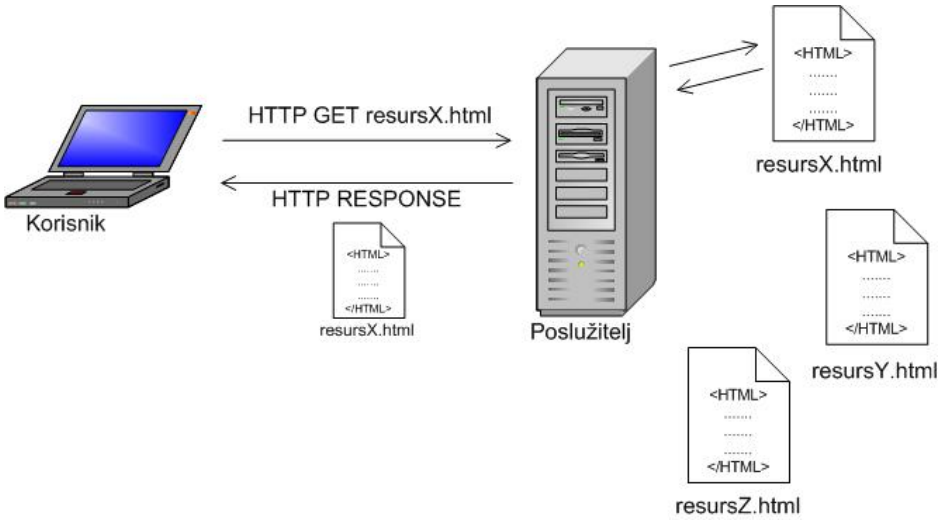
Web još uvijek postoji i ne nazire se njegov pad popularnosti i razvitka, a HTML kao osnovni jezik za definiranje sadržaja na Web-u isto tako još nije izumro, nego doživljava razna unaprjeđenja. U skraćenici HTML „HyperText“, označava tekst koji je povezan s drugim informacijama i stranicama, a „Markup Language“ označava vrstu teksta koji je obogaćen oznakama koje pobliže označuju o kakvoj se vrsti teksta radi. U HMTL jeziku te oznake, ili tagovi, počinju sa znakom „<“, iza kojeg slijedi ime taga, a završava sa znakom „>“. Pojedini element u HTML-u može se sastojati od otvarajućeg taga <*imeTaga*> i zatvarajućeg taga </*imeTaga*>, ili samo od jednog taga koji nije ni zatvarajući ni otvarajući, npr. <*imeTaga* />. Ako se element sastoji od dva taga, tada se može definirati sadržaj elementa kao tekst koji se nalazi između ta dva taga. Bilo koja vrsta elementa može sadržavati atribute elementa koji ga pobliže određuju, a koji se nalaze unutar samoga otvarajućeg taga, npr. <imeTaga atribut=“vrijednost atributa“>. Elementi Internet pregledniku govore što sadržaj elementa predstavlja i kako ga prikazati. Primjerice tekst unutar <B> elementa koji je u HTML datoteci prikazan ovako: „<B>ovo je podebljani tekst</B>“, biti će u Internet pregledniku prikazan ovako:

„**ovo je podebljani tekst**“

Znači Internet preglednik je na temelju <B> elementa zaključio da tekst unutar tagova treba biti podebljan te ga je tako i prikazao, pritom izbacivši sve dijelove datoteke koje se ne prikazuju, a to su tagovi. Oni samo određuju kako će tekst biti prikazan.

Za dohvat Web stranica koristi se HTTP (engl. *Hypertext Transfer Protocol*) protokol [39]. To je komunikacijski protokol koji pripada aplikacijskom sloju ISO-OSI modela [46]. On djeluje na principu zahtjev – odgovor. Klijent podnosi zahtjev za određenim resursom, a web poslužitelj odgovara. Taj odgovor može biti sami resurs, ali i poruka o grešci. HTTP definira osam metoda.

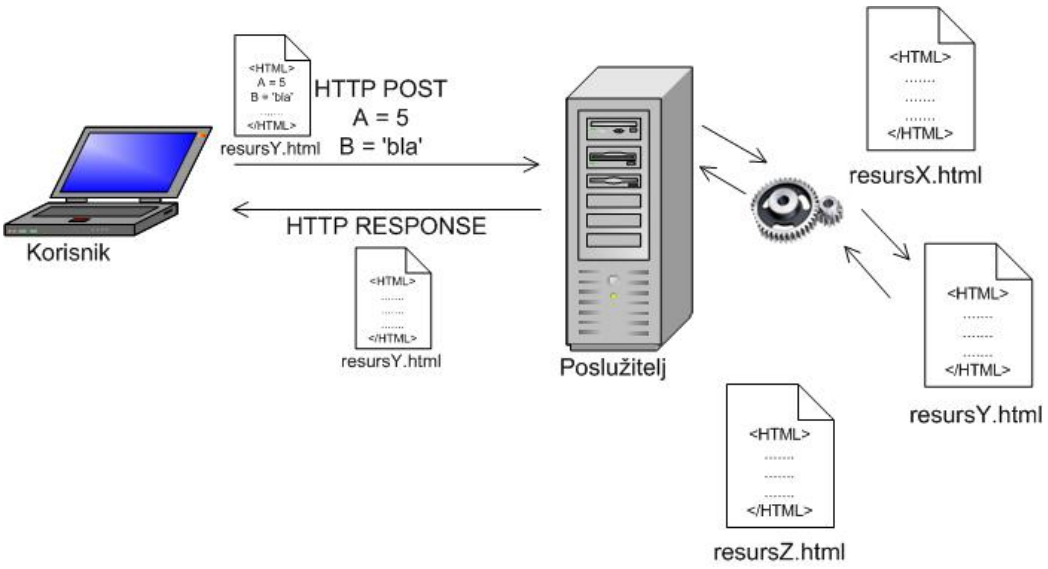
GET metoda označava zahtjev za dohvatom nekog resursa. Mjesto gdje se resurs nalazi je navedeno u mrežnom izvoru. GET može sadržavati još dodatne parametre, koji se još zovu GET varijable, a koji pobliže određuju koji resurs se želi dohvatiti. Slika 6.1. prikazuje funkcionalnost GET metode.



Slika 6.1. Funkcionalnost GET metode

HEAD metoda je slična GET metodi i isto označava zahtjev, ali samo zaglavlja odgovora, a ne i samog resursa [39].

POST metoda šalje podatke poslužitelju. Podaci su obično polja *FORM* elementa HTML-a, a spremljeni su kao POST varijable. Iako se na prvi pogled čini da to nije zahtjev, to u biti je zahtjev poslužitelju da on primi podatke [39]. Slika 6.2. prikazuje funkcionalnost POST metode.



Slika 6.2. Funkcionalnost POST metode

PUT metoda šalje čitav resurs poslužitelju [39].

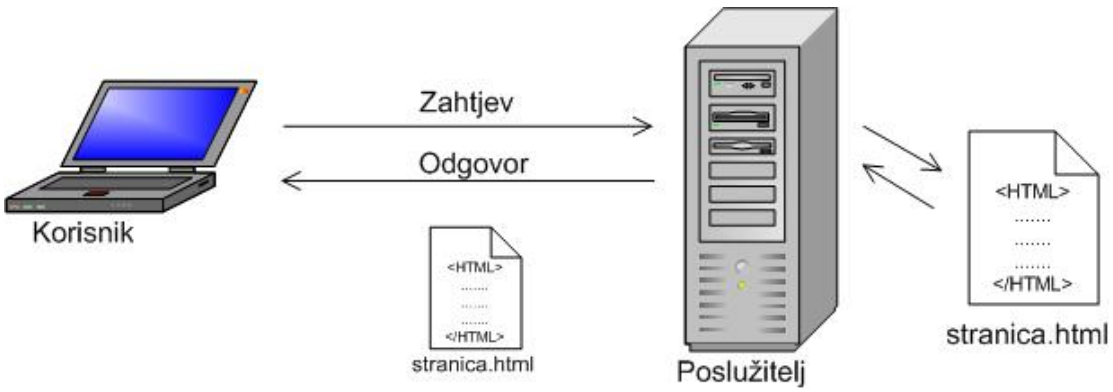
DELETE metoda šalje zahtjev za brisanjem određenog resursa na poslužitelju [39].

TRACE metoda služi kako bi korisnik vidio kojim putem njegom zahtjev sve prolazi [39].

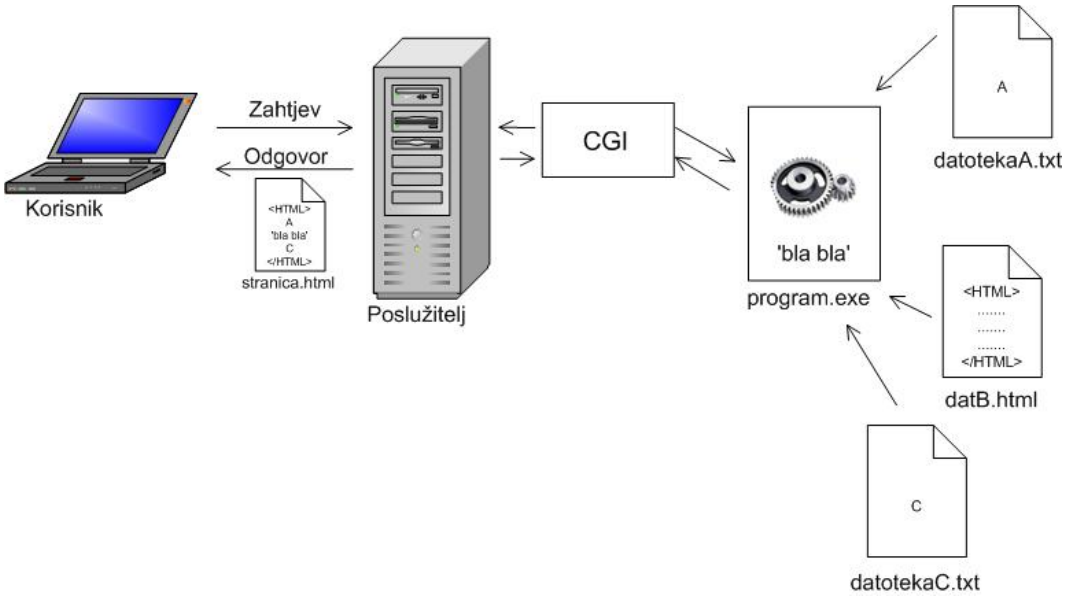
OPTIONS metoda ispituje koje sve metode podržava poslužitelj [39].

CONNECT metoda označava zahtjev za tuneliranjem, obično kako bi se ostvario HTTPS (engl. *HTTP Secure*) protokol preko SSL (engl. *Secure Sockets Layer*) [47] protokola [39].

Daljnji razvoj HTML se tiče dinamičnosti. Naime, web stranica napisana samo u HTML-u je statična, što znači da se njen izgled ne mijenja programatski, nego samo ako netko ručno zamijeni trenutačnu stranicu nekom drugom. Korak dalje u razvoju Web-a je CGI. Ta tehnologija omogućuje da se HTML dokument prije nego što se pošalje korisniku obradi programom. Program primjerice može biti napisan u programskom jeziku C. Znači klikom na određeni link ili upisivanjem adrese u preglednik, ne dohvaća se direktno HTML dokument, nego se pokreće program koji će programatski odlučiti koji HMTL dokument i koji njegovi dijelovi će se prikazati te hoće li uopće ikoji HTML dokument biti pročitan i poslan korisniku. Naime, korisniku se šalje tekst koji CGI program ispisuje, a to ne mora biti neka pročitana datoteka, nego neki tekst pohranjen u izvornom kodu CGI programa, ili kombinacija svega toga. Bitno je da korisnik taj tekst interpretira kao HTML dokument. Taj pristup više nije statičan, nego dinamičan, pošto CGI program može generirati HTML dokument na temelju raznih stvari, među ostalim i na temelju POST, odnosno GET varijabli HTTP protokola. Slika 6.3. prikazuje dohvat statičke stranice, a slika 6.4. prikazuje dohvat dinamičke stranice.

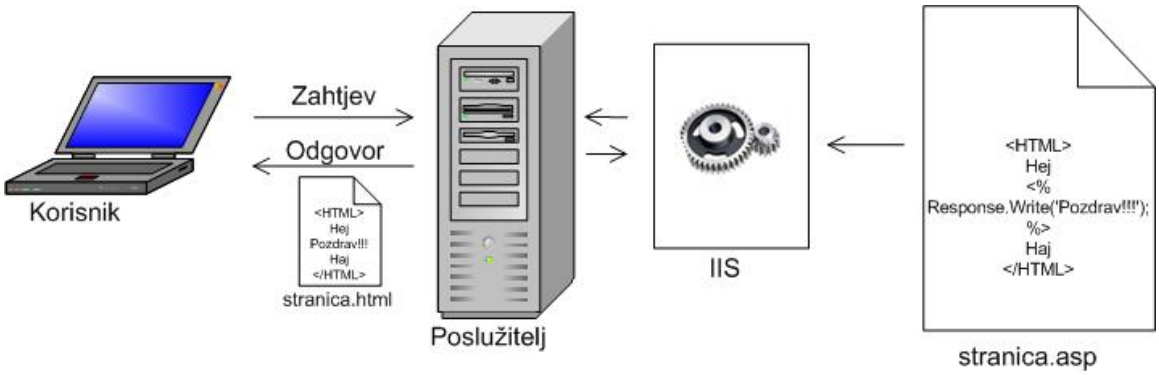


Slika 6.3. Dohvat statičke stranice



Slika 6.4. Dohvat dinamičke stranice

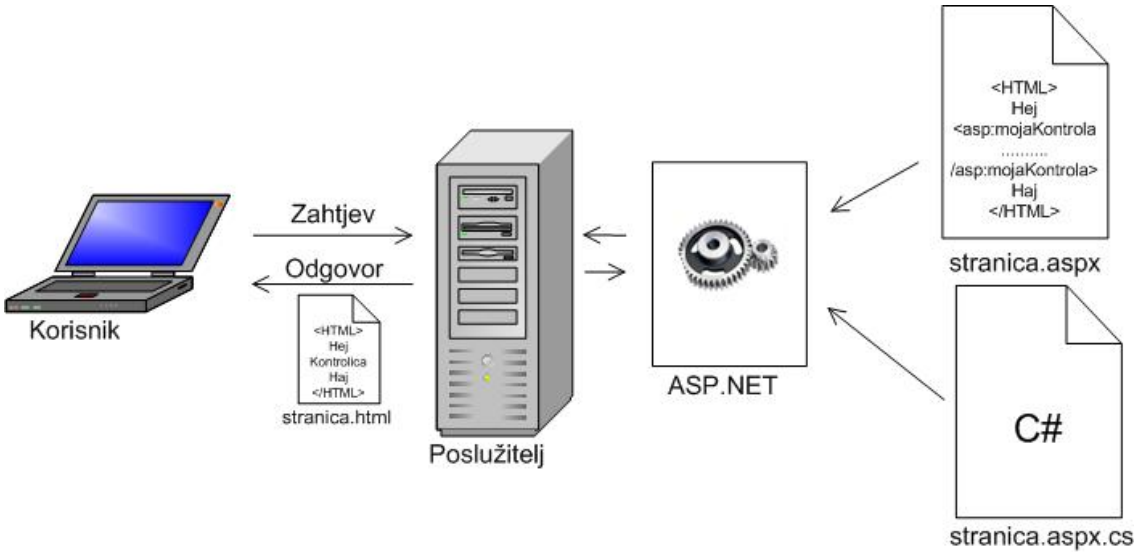
Razlog zašto je spomenut CGI, iako se ovdje direktno ne koristi je taj što se koristi ASP, odnosno jedna njegova verzija, ASP.NET, a on je jedan od nasljednika CGI tehnologije, zajedno sa programskim jezicima PHP (engl. *PHP: Hypertext Preprocessor*) [48], JSP (engl. *JavaServer Pages*) [49], itd... ASP je Microsoft-ovo ISAPI (engl. *Information Server Application Programming Interface*) [50] proširenje IIS poslužitelja. Za razliku od CGI-a koji zahtjeva da se CGI program svaki puta iznova pokrene kada dođe zahtjev za stranicom, što troši puno resursa, ASP je DLL (engl. *Dynamic Link Library*) [51] koji nakon prvog pokretanja ostaje čitavo vrijeme učitan u memoriji. Taj DLL ima objavljeno nekoliko metoda kojima se onda predaju ASP dokumenti kao parametri. Ti dokumenti se zatim parsiraju i obrađuju, da bi se zatim poslali korisniku. ASP dokumenti povrh HTML elemenata sadrže još dodatne elemente koji su zahvaćeni procesom obrade. Pozadinska logika obrade se vrši nekim od programskih jezika, primjerice Visual Basic, C... Obični HTML dokumenti se ne diraju nego se neizmijenjeni šalju korisniku. Slika 6.5. prikazuje proces obrade ASP stranice.



Slika 6.5. Proces obrade ASP stranice

ASP 2.0 sadrži šest ugrađenih elemenata koji u sebi objedinjavaju najčešće korištene funkcionalnosti koje uvelike pomažu u dinamičkom izrađivanju Web stranica. Ti objekti su *Application*, *ASPError*, *Request*, *Response*, *Server* i *Session*. Oni dopuštaju u programatskoj obradi ASP stranice čitanje GET i POST varijabli, korištenje informacija o poslužitelju i samoj Web aplikaciji te uspostavljanje sjednice, odnosno prividne stalne veze sa korisnikom, pošto je HTTP protokol koji ne pamti stanje [36].

ASP.NET je najnovija verzija ASP-a i ta verzija se koristi ovdje. ASP.NET je napravljen na .NET platformi, što znači da skriptni jezik koji provodi pozadinsku poslužiteljsku logiku može biti bilo koji od .NET podržanih programskih jezika. ASP.NET potiče objektni stil programiranja. Stranica se sastoji od raznih ASP.NET Web kontrola koje se renderiraju u HTML elemente te šalju korisniku. Web kontrole su osjetljive na razne događaje, poput kliktanja mišem na nju. Ti događaji, kao i sva pozadinska logika stranice, se obrađuje u posebnoj datoteci u kojoj je smješten kod skriptnog jezika. U ovom projektu se primjerice koristio programski jezik C#. Dakle jedna ASP.NET stranica se sastoji od dvije datoteke. Od *.aspx* datoteke u kojoj se nalaze HTML i ASP.NET elementi koji definiraju izgled stranice i od datoteke u kojoj je smješten kod napisan u nekom od .NET jezika i koji definira pozadinsku logiku stranice. Ekstenzija druge datoteke ovisi o programskom jeziku čiji je kod sadržan u datoteci [34]. Slika 6.6. prikazuje proces obrade ASP.NET stranice.



Slika 6.6. Proces obrade ASP.NET stranice

Za razliku od dinamičnih stranica što se tiče poslužiteljske strane, Web stranice mogu biti dinamične i što se tiče korisničke strane. Tu dinamičnost omogućuju skriptni jezici. Ovdje se koristio skriptni jezik Javascript. Skriptni jezik je mali dio programskog koda koji je ugrađen u samo HMTL stranicu unutar <*script*> elementa i koji dinamički dodaje, izmjenjuje, ili briše razne HTML elemente koji sačinjavaju Web stranicu i to bez posredstva poslužitelja. To se prvenstveno koristi za poboljšanje izgleda stranice, ali ako se ukaže potreba za dodatnim podacima sa poslužitelja, svejedno se mora poslati ponovni HTTP zahtjev za modificiranom stranicom koja sadrži te dodatne podatke. Ponovno učitavanje čitave stranice te obnavljanje stanja, pošto ga sami HTTP protokol ne čuva, troši puno resursa i opterećuje vezu, a sve to koliko god malo dodatnih podataka trebalo. Makar treba samo napuniti listu sa 10 novih podataka. Kao bi se to spriječilo i kako vi odaziv stranica bio brži, razvijena je AJAX tehnologija.

6.2. AJAX

AJAX tehnologija se koristi za izradu interaktivnih Web stranica koje se već jako približavaju desktop aplikacijama. Namjera je da odaziv stranica bude puno brži tako što će se izmjenjivati samo mali dijelovi stranica, a ne čitava stranica. Asinkronost u nazivu AJAX znači da zahtjevi za dijelovima stranice nisu normalni HTTP zahtjevi, tj. ne podnosi se zahtjev za čitavom stranicom, Javascript u nazivu znači da se koristi skriptni jezik Javascript kako bi se obavljali AJAX pozivi, a XML znači da su podaci koji se zahtijevaju formatirani u obliku XML-a. AJAX koristi otvorene standarde i

može se koristiti na raznim operacijskim sustavima, Web preglednicima i različitim platformama te je podržan od strane Microsofta kao proširenje ASP.NET-a. AJAX sam po sebi nije jedna tehnologija nego sadržava skup različitih tehnologija:

XHTML (engl. *eXstensible HTML*) [52], HTML za formatiranje sadržaja stranice te Javascript i DOM za dinamičko formatiranje,

CSS (engl. *Cascading Style Sheet*) [53] za definiranje stilova,

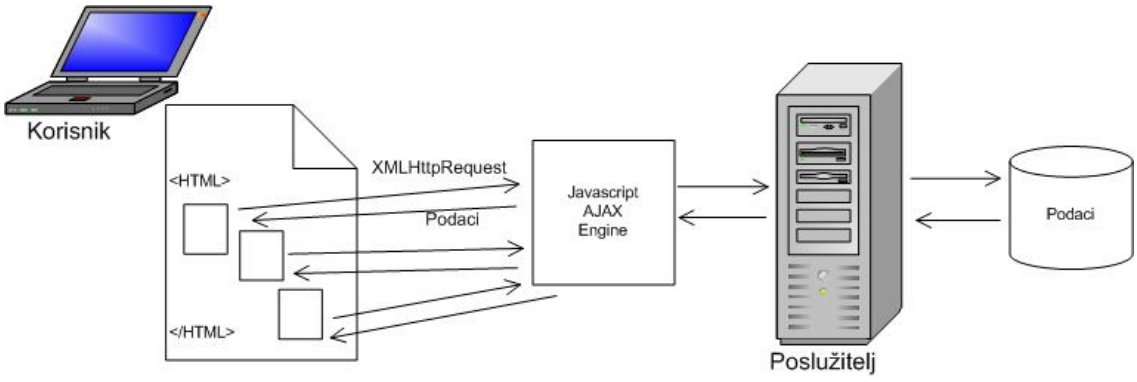
Javascript za interakciju s korisnikom te za izvršavanje korisničke pozadinske logike,

XML, HMTL, običan tekst, ili JSON (engl. *JavaScript Object Notation*) [54] kao format za prijenos informacija,

XMLHttpRequest objekt za asinkronu komunikaciju sa Web poslužiteljem,

te bilo koji od skriptnih jezika za poslužiteljsku pozadinsku logiku.

XMLHttpRequest objekt je glavna tehnologija oko koje se bazira AJAX. On je prvo bio zamišljen kao *ActiveX* [55] kontrola od strane Microsofta, ali je kasnije postao općenito podržani Javascript objekt. AJAX stranica se samo prvi puta učitava čitava. Svaki kasniji zahtjev ne traži čitavu stranicu nego se interpretira kao Javascript poziv AJAX stroju. AJAX stroj tada manipulira *XMLHttpRequest* objektom kako bi slao pojedinačne zahtjeve poslužitelju. *XMLHttpRequest* objekt jednostavno dinamički kreira HTTP GET ili POST zahtjev koji se zatim šalje poslužitelju. Kada se dobije odgovor, on se prosljeđuje glavnoj stranici kao rezultat Javascript poziva funkcije. Taj poziv je, kao što je viđeno u primjeru ovog diplomskog, asinkron i stranica normalno funkcionira dok čeka na odgovor funkcije pa makar on nikad ne stigao. Slika 6.7. prikazuje kako funkcionira AJAX tehnologija.



Slika 6.7. Funkcioniranje AJAX tehnologije

6.3. Web usluge

Dosad su bile opisane tehnologije koje vezane uz Web stranice, njihovo generiranje, dohvat, predstavljanje korisniku te interakciju s korisnikom. Slijedi opis Web usluga. To je relativno nova tehnologija sa dugom poviješću. Njen cilj je omogućiti komunikaciju između dvaju objekta na dva različita računala. To se pokušava već dosta dugo, ali do Web usluga nije bilo tako uspješno riješeno. Pošto se tehnologije koje su prethodile Web uslugama nisu koristile u ovome projektu, za razliku od tehnologija koje su prethodile ASP.NET-u i AJAX-u, one neće biti opisivane nego samo spomenute.

To su CORBA (engl. *Common Object Request Broker Architecture*) [21] – standard koji definira sučelje objekta putem IDL-a (engl. *Interface Definition Language*) [56]. Zatim se to sučelje preslikava u programski jezik koji se koristi za definiciju objekta, primjerice programski jezik C++ te objekti zatim putem tog sučelja međusobno komuniciraju koristeći male programske elemente koji provode poziv metode preko mreže, a na engleskom se zovu stubovi (engl. *stubs*). Sve to je složeno, a samim time i skupo za ostvariti, pošto ne postoji jedno tijelo koje brine o standardu, nego više njih tako da u nedostatku usuglašavanja svi prijedlozi ulaze u standard.

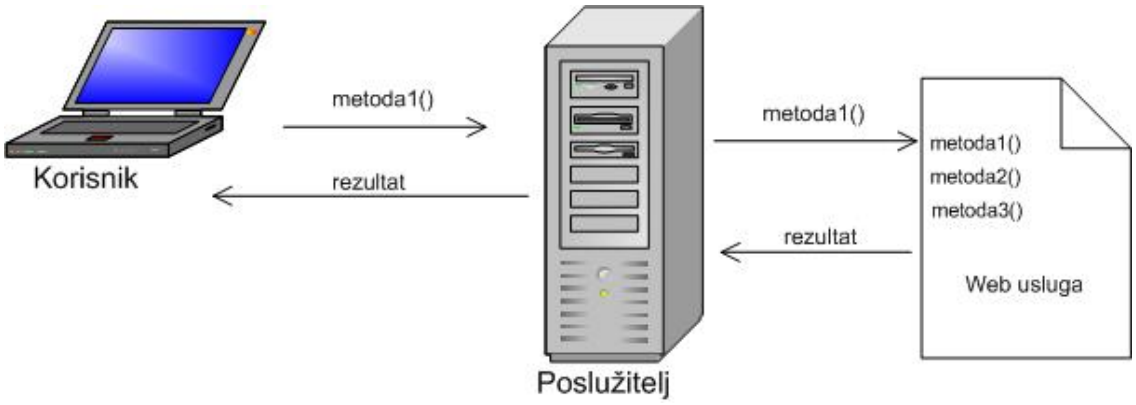
DCOM (engl. *Distributed Component Object Model*) [57] – Vrlo slična tehnologija kao i CORBA, ali razvijana od strane Microsofta. Nije u potpunosti zaživjela jer je imala teškoće u komunikaciji preko vatrozida, za razliku od Web usluga koje komuniciraju putem HTTP protokola koji koristi pristup broj 80, a on je u velikoj većini slučajeva uvijek dostupan.

RPC (engl. *Remote Procedure Call*) [58] – Prethodnik DCOM-a, jako slabo standardiziran sa puno mana, primjerice korisnička strana je blokirana za vrijeme dok se udaljena procedura izvršava na poslužitelju. Ako se koristi objektni model RPC se još naziva RMI (engl. *Remote Method Invocation*).

Java RMI – Dosta dobro ostvarenje RPC–a, ali problem je što ovisi o JVM-u (engl. *Java Virtual Machine*) [59], dok Web usluge ne ovise o platformi na kojoj se izvode.

Svaka od tih tehnologija još uvijek postoji ali nije opće prihvaćena. Neke zbog skupoće, neke zbog složenosti, prilagodljivosti, potpore industrije, a neke jednostavno nisu imale sreće.

Web usluga je učahuren objekt. Taj objekt može počivati na bilo kakvom računalu, na bilo kojem operacijskom sustavu. Bitno je da poštuje skup standarda koji se tiču komunikacije sa svijetom preko mreže. Ti standardi definiraju sučelje Web usluge, način prijenosa podataka između Web usluga te otkrivanje i objavljivanje Web usluga na Internetu. Slika 6.8. prikazuje Web uslugu.



Slika 6.8. Web usluga

Protokoli na kojima počivaju Web usluge su SOAP (engl. *Simple Object Access Protocol*) [60], WSDL (engl. *Web Service Descritpion Language*) [61] te UDDI (engl. *Univesral Discovery, Description and Integration*) [62].

SOAP protokol je format za slanje poruka temeljen na XML bazi. SOAP porukama se šalju parametri metodi na poslužiteljskoj Web usluzi, potiče se njeno izvršavanje te se drugom SOAP porukom dobiva rezultat izvršavanja metode. Slika 6.9. prikazuje primjer SOAP poruka.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <dohvatiDetaljeProizvoda xmlns="htttp://skladište.primjeri.com/ws">
            <idProizvida>7777</idProizvida>
        </dohvatiDetaljeProizvoda>
    </soap:Body>
</soap:Envelope>
```

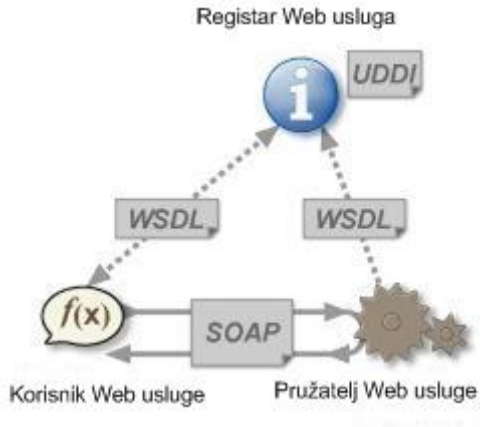
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <dohvatiDetaljePriozvodaResponse xmlns="htttp://skladište.primjeri.com/ws">
            <dohvatiDetaljePriozvodaResult>
                <imeProizvoda>Ribički pribor</imeProizvoda>
                <idProizvoda>7777</idProizvoda>
                <opis>Štap, čizme, set struna</opis>
                <cijena>100</cijena>
                <valuta>HRK</valuta>
            </dohvatiDetaljePriozvodaResult>
        </dohvatiDetaljePriozvodaResponse>
    </soap:Body>
</soap:Envelope>
```

Slika 6.9. Primjer SOAP poruka

Protokol za slanje SOAP poruke je HTTP ili HTTPS, ali su podržani i neki drugi, primjerice SMTP (engl. *Simple Message Transfer Protocol*) [63] i TCP.

WSDL je jezik temeljen na XML-u, a služi kao model za opisivanje Web usluga. Web usluga se definira kao skup pristupnih točaka koje se nalaze u nekom mrežnom izvoru. WSDL opis neke Web usluge je odvojen i neovisan od njenog ostvarenja. Pročitavši WSDL opis Web usluge, njen korisnik zna njeno sučelje te može sastavljati SOAP poruke putem kojih komunicira s Web uslugom.

UDDI je registar raznih usluga na Internetu temeljen na XML-u, među kojima su i Web usluge. Komunikacija sa UDDI registrom se odvija putem SOAP poruka. Korisnik SOAP porukom može upitati registar za lokaciju pojedine Web usluge. Registar mu odgovara gdje je lokacija te mu šalje WSDL opis usluge, tako da sada korisnik ima sve potrebne stvari za komunikaciju s Web uslugom. Slika 6.10. prikazuje korištenje UDDI registra za uspostavljanje komunikacije sa Web uslugom.



Slika 6.10. Uspostavljanje komunikacije s Web uslugom

Ovo su bile osnovne stvari koje se tiču Web usluga i nije potrebno ulaziti u detalje pošto to nije tema ovog diplomskog. Kao što se vidi Web usluge su standardizirana tehnologija, koja nije presložena za ostvariti, podržana je od svih i zaobilazi probleme sa vatrozidima, pošto komunikacijski protokol među Web uslugama, a to je SOAP, počiva na HTTP protokolu koji koristi uvijek slobodan pristup broj 80. Može se koristiti na bilo kojim računalnim platformama, a u najnovijim verzijama podržava i sigurnost, transakcije te preusmjeravanje puta. Jedini problem je zasad što počiva na XML-u koji zauzima dosta prostora, no razvijaju se metode optimizacije XML-a koje bi uskoro trebale ući u primjenu.

6.4. Web 2.0 skup tehnologija

Sve najnovije tehnologije koje su navedene, primjerice AJAX i Web usluge pripadaju onome što se u zadnje vrijeme naziva Web 2.0. To je skup raznih tehnologija i pojava na Internetu a posljedica je sve bržih Internet veza i sve većeg broja ljudi koji koriste Internet. Obuhvaća pojave poput dijeljenja muzike i podataka, blogova, stavljanja svojih filmića na Internet, otvorenih baza znanja poput Wikipedie [64] te sve veće i veće socijalizacije putem Interneta. Što se tiče čisto tehničke strane Web 2.0 obuhvaća sučeljem i funkcionalnošću bogate i brze Web aplikacije koje polako ali sigurno sustižu desktop aplikacije. Sve ih je više, a to je omogućeno korištenjem tehnologija poput AJAX-a, Web usluga, Flash-a [65], Java applet-a, itd, a ovaj projekt je mali dio toga svega koji bi u duhu Web-a 2.0 trebao ponuditi običnim korisnicima mogućnost da na jednostavan, interaktivan način razvijaju bogate raspodijeljene aplikacije.

7. Zaključak

Ovim radom ostvarena je razvojna okolina za izradu paralelnih primjenskih programa. Ona je ostvarena kao web aplikacija i sastoji se od dvije web stranice i dvije web usluge. Za primjenu razvojne okoline su se koristili gadgeti. Prva web stranica je početna web stranica pomoću koje se iz mrežnog izvora ili iz fizičkog spremnika na disku učitavaju gadgeti. Druga web stranica je tablični programator koji služi kao razvojna okolina za izradu paralelnih primjenskih programa. Programiranje tabličnim programatorom se pokazalo kao dobra ideja jer struktura tablice omogućuje programeru izvrstan pregled vremenske dimenzije paralelnog programa i odnos među njegovim dijelovima, tj. jesu li međusobno zavisni ili nezavisni. Vremenska dimenzija teče s lijeva na desno, i odozgo na dolje. To znači da će se naredba koja se nalazi u ćeliji desno ili ispod one koja se promatra izvršiti nakon naredbe koja se nalazi u ćeliji koja se promatra. Sljedovi naredaba koji nemaju dodirnih ćelija se izvršavaju neovisno, odnosno mogu se izvršavati paralelno.

Dobra razvojna okolina bi trebala biti lagana za korištenje od strane stručnjaka i ne stručnjaka u paralelnom programiranju, pojednostaviti dizajniranje paralelnog programa, pružati podršku za dokumentaciju, nadzirati izvršavanje programa, biti proširiva i omogućavati ponovno iskorištavanje postojećih modula preko programskih knjižnica, podržavati programske jezike visokog stupnja, podržavati heterogena računalna okruženja, pružati mogućnost objavljivanja sučelja modula kao web usluge. Ova razvojna okolina ne podržava sve ove zahtjeve, ali to ostaje kao predmet budućeg razvoja i istraživanja.

Zadnji modul razvojne okoline je Web usluga za dohvat i pohranjivanje gadgeta. Web usluzi se predaje ime gadgeta, a Web usluga zatim, ako gadget sa zadanim imenom postoji, vraća mrežni izvor u kojem se gadget nalazi ili dojavu o pogrešci. Predmet budućeg razvoja je poopćiti razvojnu okolinu, da ne radi samo sa gadgetima.

Prilikom razvoja došlo do sigurnosnih problema jer svi najviše korišteni Internet preglednici ne podržavaju izravni pristup Web uslugama koji se ne nalaze na istoj domeni kao i Web stranica koja želi koristiti tu Web uslugu. Problem je uspješno riješen korištenjem zastupnika Web usluge.

8. Literatura

[1] **„Distributed Computing“**, http://en.wikipedia.org/wiki/Distributed_computing

[2] **„Parallel Computing“**, http://en.wikipedia.org/wiki/Parallel_computing

[3] **„Flynn's Taxonomy“**, http://en.wikipedia.org/wiki/Flynn%27s_taxonomy

[4] J. L. Hennessy, D. A. Patterson: **„Computer Architecture: A Quantitative Approach”**, 2. izdanje, 1996

[5] **„Client-server“**, http://en.wikipedia.org/wiki/Client-server

[6] **„Peer-to-peer“**, http://en.wikipedia.org/wiki/Peer_to_peer

[7] **„Multitier architecture“**, http://en.wikipedia.org/wiki/Three-tier_%28computing%29

[8] **„Computer cluster“**, http://en.wikipedia.org/wiki/Computer_cluster

[9] **„Web service“**, http://en.wikipedia.org/wiki/Web_service

[10] **„IPPE“**, www.mcs.anl.gov/~gregor/papers/vonLaszewski--ecwmf-interactive.pdf

[11] **„The Northeast Parallel Architectures Center“**, www.npac.syr.edu/

[12] **„Atmospheric Four Dimensional Data Assimilation: Applications in High Performance Computing“**, http://ct.gsfc.nasa.gov/lys/hpcc96/report.html

[13] **„National Aeronautics and Space Administration“**, http://www.nasa.gov/

[14] **„Message Passing Interface“**, http://en.wikipedia.org/wiki/Message_Passing_Interface

[15] **„Parallel Architecture Lab - ASSIST“**, http://www.di.unipi.it/Assist.html

[16] **„Agenzia Spaziale Italiana“**, http://www.asi.it/

[17] **„Consiglio Nazionale delle Ricerche“**, http://www.cnr.it/sitocnr/home.html

[18] **„POSIX“**, http://en.wikipedia.org/wiki/POSIX

[19] **„Adaptive Communication Environment“**, http://en.wikipedia.org/wiki/Adaptive_Communication_Environment

[20] **„Application Programming Interface“**, http://en.wikipedia.org/wiki/API

[21] **„Common Object Request Broker Architecture“**, http://en.wikipedia.org/wiki/Corba

[22] **„Component object model“**, http://en.wikipedia.org/wiki/Component_object_model

[23] **„Visual Basic for Applications“**, http://en.wikipedia.org/wiki/Visual_Basic_for_Applications

[24] **„KDCalc“**, http://www.kdcalc.com/

[25] **„Moteiv“**, http://www.moteiv.com

[26] **„MSR Networked Embedded Sensing Toolkit (MSR Sense)“**, http://research.microsoft.com/nec/msrsense

[27] **„Universal Serial Bus“**, http://en.wikipedia.org/wiki/Usb

[28] **„Extensible Stylesheet Language“**, http://en.wikipedia.org/wiki/Xsl

[29] **„XML“**, http://en.wikipedia.org/wiki/Xml

[30] **„Iframe“** , http://en.wikipedia.org/wiki/IFrame

[31] **„Proxy pattern“**, http://en.wikipedia.org/wiki/Proxy_pattern

[32] **„Microsoft Visual Studio 2005“**, http://msdn2.microsoft.com/en-us/vstudio/aa973782.aspx

[33] **„Internet Information Services“**, http://www.microsoft.com/WindowsServer2003/iis/default.mspx

[34] **„The Official Microsoft ASP.NET 2.0 Site“** ,http://www.asp.net/

[35] **„HTML“**, http://en.wikipedia.org/wiki/Html

[36] **„Active Server Pages“**, http://en.wikipedia.org/wiki/Active_Server_Pages

[37] **„JavaScript“**, http://en.wikipedia.org/wiki/Javascript

[38] **„Ajax (programming)“**, http://en.wikipedia.org/wiki/Ajax_%28programming%29

[39] **„Hypertext Transfer Protocol“**, http://en.wikipedia.org/wiki/Http

[40] **„Document Object Model“**, http://en.wikipedia.org/wiki/Document_Object_Model

[41] **„Common Gateway Interface“**, http://en.wikipedia.org/wiki/Common_Gateway_Interface

[42] **„World Wide Web“**, http://en.wikipedia.org/wiki/WWW

[43] **„Transmission Control Protocol“**, http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[44] **„Internet Protocol“**, http://en.wikipedia.org/wiki/Internet_Protocol

[45] **„ARPANET“**, http://en.wikipedia.org/wiki/Arpanet

[46] **„OSI model“**, http://en.wikipedia.org/wiki/OSI_model

[47] **„Transport Layer Security“**, http://en.wikipedia.org/wiki/Secure_Sockets_Layer

[48] **„PHP“**, http://en.wikipedia.org/wiki/Php

[49] **„JavaServer Pages“**, http://en.wikipedia.org/wiki/JavaServer_Pages

[50] **„ISAPI“**, http://en.wikipedia.org/wiki/ISAPI

[51] **„Dynamic-link library“**, http://en.wikipedia.org/wiki/Dynamically_linked_library
[52] **„XHTML“**, http://en.wikipedia.org/wiki/XHTML
[53] **„Cascading Style Sheets“**, http://en.wikipedia.org/wiki/CSS
[54] **„JSON“**, http://en.wikipedia.org/wiki/JSON
[55] **„ActiveX“**, http://en.wikipedia.org/wiki/Activex
[56] **„Interface Description Language“**, http://en.wikipedia.org/wiki/Interface_description_language
[57] **„Distributed Component Object Model“**, http://en.wikipedia.org/wiki/Distributed_Component_Object_Model
[58] **„Remote Procedure Call“**, http://en.wikipedia.org/wiki/Rpc
[59] **„Java Virtual Machine“**, http://en.wikipedia.org/wiki/JVM
[60] **„SOAP“**, http://en.wikipedia.org/wiki/SOAP
[61] **„Web Services Description Language“**, http://en.wikipedia.org/wiki/WSDL
[62] **„Universal Description Discovery and Integration“**, http://en.wikipedia.org/wiki/UDDI
[63] **„Simple Mail Transfer Protocol“**, http://en.wikipedia.org/wiki/Smtp
[64] **„Wikipedia“**, http://en.wikipedia.org
[65] **„Adobe Flash“**, http://en.wikipedia.org/wiki/Adobe_Flash