Prirodno-matematički fakultet - Kragujevac
Institut za matematiku i informatiku

Diplomski rad

Programiranje ActionScripta i veza sa eksternim konekcijama XML - PHP - MySQL

Student:
Aleksandar Antić
Br. Indeksa 3/88

Mentor:
dr Boban Stojanović

Kragujevac 2009.

Sadržaj

I - Teorijski deo

1. UVOD

Pojava servisa World Wide Web proteklih godina uvela je revoluciju u korišćenju Interneta i kompjutera uopšte. Od tih momenata kompjuter sve više prestaje da bude alatka koju koriste visokostručni profesionalci i postaje još jedan zabavan i koristan aparat u domaćinstvu. Masovnim korišćenjem povećali su se i zahtevi za mogućnostima koje servis treba da obezbedi. Najveći nedostatak Interneta, protok podataka, uslovio je, s jedne strane, velika ulaganja u razvoj mrežne infrastrukture i povećanje propusnog kapaciteta same mreže i, s druge strane, razvoj i primenu raznih tehnika komprimovanja podataka i steaming formata, kako bi postojeći resursi mogli da obezbede što veću iskoristivost, pre svega, multimedijalnih mogućnosti Interneta.

Flash, proizvod kompanije Adobe (nekada Macromedia), je jedna tehnologija posebno prilagođena sukobljenim zahtevima Interneta. Grafički je zasnovan na vektorskom opisu objekata sa animacijom vođenom opisom promena na samim objektima, uz mogućnost steaminga podataka. U ovom trenutku predstavlja najefikasnije rešenje na webu iz domena animacije. Mogućnost dizajniranja korisničkog interfejsa i programiranja njegovih elemenata za interakciju sa korisnikom čini ga takođe i liderom interaktivnosti na mreži.

Cilj ovog rada jeste upoznavanje sa svetom koji Flash može da predstavi, sa akcentom na mogućnosti programiranja u ActionScriptu, Flashovom internom skript jeziku i povezivanju sa bazama podataka naročito PHP i MySQL.

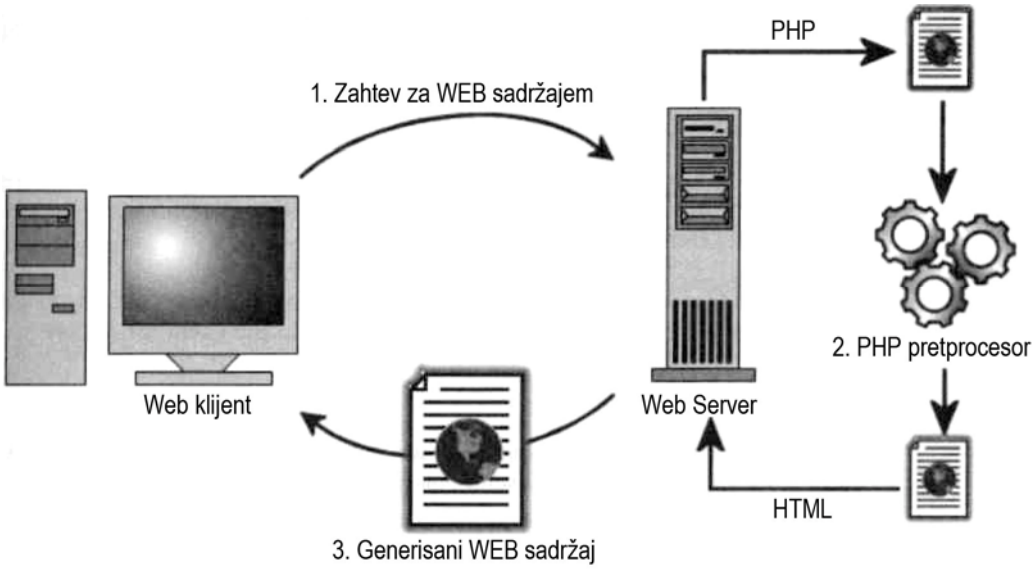
Drugo poglavlje predstavlja podsećanje na glavne prednosti korišćenja PHP i MySQL-a kao i na način njihovog povezivanja, zatim u trećem poglavlju se bavimo XML-om i njegovom primenom. Četrvrti deo posvećen je Flashu i načinu korišćenja sa aspekta intefejsa i rada. Peti deo pripada ActionScriptu njegovoj sintaksi i načinu programiranja. Šesti i sedmi deo posvećen je komunikaciji ActionSripta sa spoljnim svetom tačnije konekcija ka XML formatu i prenosu varijabli HTTP protokolom. Osmi deo posvećen je osvrtu na integraciju gotovog Flash filma u HTML kod.

2. PHP, MYSQL

2.1. PHP

PHP je široko rasprostranjen Open Source skript jezik koji se izvršava na strani web servera, a namenjen je za kreiranje dinamičkih web stranica, i uopšte različitih Internet aplikacija. PHP je nastao 1994. godine od strane Pasmus Ledorfa, a inače je skraćenica od "Personal Home Page Tools". Autor je jezik kreirao radi održavanja vlastitih web stranica, a na bazi tada jako zastupljenog Perl jezika. Inače, sintaksa samog jezika je vrlo slična sintaksama jezika C i Perl. Nakon njegovog pojavljivanja, veliki broj programera i dizajnera je uzeo učešća u njegovom daljem razvoju (pošto se radi o Open Source projektu), tako da je jezik sada "dogurao" do verzije 5.3. U najavi je i verzija 6 mada datum njenog izlaska nije još definisan.

PHP je podržan od strane velikog broja platformi (gotovo svih), ali se posebno odomaćio na Unix/Linux platformi. Direktan konkurent PHP-Linux-Apache platformi je ASP-WindowsNT-IIS platforma, ali po većini Internet marketing agencija ubedljivo vodi PHP platforma, uglavnom zato što je besplatna i što je otvorenog koda. Pored svega navedenog, svoju popularnost duguje svojoj sposobnosti da podržava veliki broj sistema za upravljanje bazama podataka (RDBMS), kao što je na prvom mestu MySQL, pa zatim i ostali: MS SQL server, Oracle, Postgre SQL, MS Access i još mnogi drugi.



Slika 1 - Rad PHP pretprocesora i generisanje traženog HTML sadržaja

PHP na strani web servera predstavlja pretprocesor kome se prosljeđuju PHP skripte. U HTML stranice dodaju se PHP skripte i stranice obavezno imaju ekstenziju "*.php". Kada ih postavite na web server i korisnik ih zatraži putem svog browser-a, web server će na osnovu ekstenzije prepoznati da se radi o PHP stranicama i proslediće ih instaliranom PHP pretprocesoru. Potom će pretprocesor izvršiti programski kod i rezultat vratiti web serveru, koji nakon toga sve šalje browser-u. Rezultat procesiranja su najčešće dinamički kreirane HTML stranice, koje se zasnivaju na podacima iz neke od baza podataka, najčešće MySQL.

2.2. MySQL

MySQL predstavlja relacioni sistem za upravljanje bazama podataka (RSUBP) koji omogućava da čuvate podatke, da im pristupate i da ih organizujete na najbolji mogući način. Pri tome, ovaj RSUBP može da služi u organizaciji podataka i to od nivoa obične liste podataka, pa sve do velike kolekcije tabela sa milionskim brojem slogova.

Inače, relacione baze podataka su vrlo moćna alatka koja osigurava da se nalaze informacije iz višestrukih izvora i pri tome se vrši njihovo poređenje, kombinovanje i obrada radi dobijanja potrebnih informacija. Na primer, možete dobiti jednostavnu listu klijenata, ili iskombinovati klijente sa njihovim narudžbinama i stavkama koje su naručili. Baze podataka osiguravaju potrebnu strukturu i organizaciju koja je potrebna radi operacija za efikasan pristup podacima, čak i kada ovo podrazumeva veliku količinu informacija i veliki broj tabela (entiteta).

Ipak, šta je to što MySQL čini tako posebnim:

- **Open Source** - Dostupan je na Internetu i pri tome je besplatan. Ovo je veliki kontrast drugim komercijalnim sistemima za baze podataka (kao što su Oracle, MS SQL, Informix i sl.), kreiranih od strane velikih kompanija, koji su pri tome i veoma skupi. Međutim, za one koji žele da naruče MySQL sistem na CD-u, i pri tome prime odgovarajuću štampanu dokumentaciju, veliki broj firmi osigurava ovu opciju za malu sumu. Pored toga, neke od ovih firmi nude i tehničku podršku i trening kurseve.
- **Brzina** - Svaki od sistema za baze podataka ima područja u kojima se posebno ističe. Priznato je da su odgovori MySQL sistema brži nego kod drugih sistema. Upravo zbog brzine, MySQL je sistem izbora za Internet aplikacije, gde se zbog velikog saobraćaja zahteva velika brzina.
- **SQL-orijentacija** - MySQL podržava standardni Structured Query Language (SQL), najkorišćeniji jezik za definisanje i korišćenje podataka.
- **Lakoća upotrebe -** Distribucija MySQL je relativno mali paket, koji ne zahteva stotine i stotine megabajta kao drugi SUBP. Razvojna filozofija ovog sistema fokusirana je na široku i laku upotrebljivost, gde se u paket ubacuju samo neophodne funkcije. Ovo MySQL čini lakim za razumevanje, lakim za instaliranje, podešavanje i administraciju.
- **Prenosivost** - MySQL može da se pokrene na brojnim platformama, a najvažnije su UNIX, Linux i Windows.
- **Ubrzani razvoj -** Savremeni dodaci za MySQL uključuju podršku za transakcije, replikaciju, tekstualno pretraživanje i RAID fajl-sisteme. Pored toga, može se uključiti i podrška za zaključavanje na nivou sloga.
- **Lako programiranje** - Ukoliko postojeći softver ne odgovara vašim potrebama, možete kreirati vlastiti. Postoje interfejsi za veliki broj programskih jezika, kao što su: C, C++, Perl, PHP, Python, Java, Ruby, itd. U poslednje vreme osim MySQL-a dosta se koristi i SQL Lite.

2.3. Povezivanje PHP, MySQL

Povezivanje sa bazama podataka predstavlja i osnovu za izgradnju kompleksnih mrežnih aplikacija. Inače, uz pomoć PHP-a možete se povezati sa bilo kojim sistemom za upravljanje bazama podataka (DBMS), a među PHP programerima su svakako MySQL baze podataka najomiljeniji tip. Na vaš budući izbor baza podataka sigurno će uticati i vaš web provajder, tj. tip baza podataka koje on podržava (tj. instalirani server baza podataka).

Kada je reč o upotrebi MySQL baza podataka, postoji veći broj dobro dokumentovanih PHP funkcija koje će vam pomoći u uspostavljanju veze sa podacima. Međutim, treba će vam samo nekoliko ovih funkcija u cilju postizanja jednostavne konekcije i selekcije nekih podataka:

- **mysql_connect** - funkcija kojom ostvarujemo vezu sa MySQL serverom zahteva naziv hosta, korisničko ime i lozinku.
- **mysql_select_db** - funkcija za izbor baze podataka od mnogobrojnih koje se nalaze na MySQL serveru.
- **mysql_query** - funkcija za postavljanje SQL naredbe.
- **mysql_fetch_array** - funkcija za smeštanje rezultata SQL upita u niz.
- **mysql_free_result** - funkcija za oslobađanje resursa zauzetih trenutnom konekcijom.
- **mysql_close** - funkcija za prekid veze ka bazi podataka.

Nakon upoznavanja sa osnovnim funkcijama za povezivanje i kreiranje baze, kao i tabela potrebno je da se zna naziv servera na kome se nalazi baza podataka (ukoliko se radi u lokalu, naziv će biti *localhost*), validno korisničko ime i lozinka za taj server. Zatim počinjemo sa PHP kodom, kroz primer, kreiranjem konekcije ka serveru:

```
<php>
    $konekcija = mysql_connect ("naziv_servera","korisnik","lozinka")
        or die("Povezivanje nije moguće.");
```

Funkcija **die()** koristi se za prekidanje skripte i ispisivanje poruke o grešci ukoliko prethodna konekcija nije uspostavljena. Kada uspostavite konekciju, sledeći korak predstavlja izbor baze podataka i kreiranje SQL upita. Pretpostavimo da tabela *Proizvodi* već postoji u MySQL bazi podataka koja se npr. zove *mojaBaza*. Potrebno je da izaberete bazu podataka na serveru na koju ćete se povezati:

```
    $db = mysql_select_db("mojaBaza", $konekcija) or die("Baza nije dostupna.");
```

Ovde smo PHP parseru zadali da se poveže sa MySQL serverom i odabere bazu podataka. Ako je sve u redu možemo poslati SQL upit i nadati se vraćanju nekog rezultata, tj. seta slogova. Kreiraćemo upit koji se zasniva na tabeli *Proizvodi*, a koji treba da vrati polja sa nazivima proizvoda, njihovim tipom i količinom proizvoda, s tim da se podaci poređaju po najvećim količinama. Kreiraćemo promenljivu koja će sadržati ovu našu SQL naredbu:

```
    $sql = "SELECT Naziv, Tip, Kolicina FROM Proizvodi ORDER BY Kolicina  DESC";
```

Zatim ćemo kreirati promenljivu koja će prihvatiti rezultate prethodnog upita, a koji će se dobiti uz pomoć **mysql_query** funkcije. Funkcija zahteva dva argumenta i to SQL promenljivu i link na izabranu bazu, koje smo prethodno kreirali:

```
$sql_rezultat = mysql_query($sql,$db) or die("Upit nije izvršen");
```

Ovim bi trebalo da uspostavimo konekciju, selektujemo tabelu, postavimo upit i smestimo rezultate u promenljivu.

```
while ($red = mysql_fetch_array($sql_rezultat)) {
    //progamski kod
    }
```

Kao što se vidi uz pomoć **while** petlje kreiraćemo niz koji smo nazvali *red*, i to za svaki slog u skupu slogova rezultata. Da bi smo dobili pojedinačne elemente sloga, odnosno polja *Naziv*, *Tip*, *Kolicina*, kreiraćemo potrebne varijable:

```
$naziv_proiz = $red["Naziv"];
$tip_proiz = $red["Tip"];
$kolicina = $red["Kolicina"];
```

Pošto je osnovna svrha bilo kog upita prikaz i nekakva analiza podataka, smestićemo dobijene podatke u HTML tabelu, tj. prikazaćemo rezultat u takvom obliku da može da ga pročita bilo koji web browser. Da bi se ovo uradilo sledeći HTML kod se unese ispred **while** petlje, kako bi se kreirao naslovni red tabele za ispis:

```
echo "<TABLE BORDER=1>";
echo "<TR>
    <TH>Naziv proizvoda</TH>
    <TH>Tip</TH>
    <TH>Kolicina</TH>
</TR>";
```

Nakon definisanja promenljivih unutar **while** petlje, ispisaćemo ih u tabeli i prema tome, konačna **while** petlja izgleda ovako:

```
while ($red = mysql_fetch_array($sql_rezultat)) {
    $naziv_proiz = $red["Naziv"];
    $tip_proiz = $red["Tip"];
    $kolicina = $red["Kolicina"];
    echo "<TR>
        <TD>$naziv_proiz</TD>
        <TD>$tip_proiz</TD>
        <TD>$kolicina</TD>
    </TR>";
}
```

Nakon završetka petlje, sli1edi i zatvaranje HTML taga tabele:

```
echo "</TABLE>";
```

I konačno, na kraju je potrebno da oslobodimo sve resurse zauzete izvršavanjem upita, i zatvorimo konekciju ka bazi podataka. Ako ovo ne uradimo, možemo izazvati neželjene probleme sa memorijom ili slične probleme sa resursima.

```
mysql_free_result($sql_rezultat);
mysql_close($konekcija);
?>
```

3. XML

XML (Extensible Markup Language – proširiv jezik za označavanje) je u svojoj osnovi informacija o informaciji. Više nije dovoljno imati samo informaciju jer to u današnjim uslovima znači tražiti i pronaći istu, već je potreban način da opišemo informaciju, a da taj opis informacije upotrebimo dalje za pronalaženje iste i za njenu dalju obradu.

Prvo se u IBM-u pojavio SGML (Standard Generalized Markup Language) kao odgovor na problem prebacivanja dokumenata sa jedne na drugu platformu. Zatim se krajem osamdesetih u CERN-u (evropskoj laboratoriji za fiziku atomskih čestica) pojavio HTML, a kada je posle uspešne promocije na internetu postalo očigledno da HTML ne može baš sve rodila se ideja o XML-u negde 1996 godine. Međutim XML nije evoluirani HTML. On je komplement HTML-u i dizajniran je ne da zameni HTML već da ponudi ono što HTML ne može.

3.1. XML činjenice

- XML je metod za smeštanje struktuiranih podataka u tekstualni fajl. XML ima podatke i strukturu u tekstualnom fajlu. To ga kvalifikuje za rad sa bazama podataka.
- XML liči na HTML ali nije HTML. Iako koristi tagove kao i HTML on se suštinski razlikuje od HTML-a jer proizvoljno proširiv tagovima koje sami izmišljate.
- XML je tekst ali nije namenjen čitanju od strane ljudi već mašina. Tekst nije namenjen čitanju već parsiranju od strane računara.
- XML je porodica tehnologija XML čini čitava porodica tehnologija i on sam po sebi ne predstavlja posebno mnogo funkcionalnosti ali u sadejstvu sa ostalim tehnologijama (CSS, XLink, XPointer, XFragments, XSL ...itd) daje odlične rezultate.
- XML za posledicu ima obiman fajl ali to nije problem. Iako je XML fajl obiman zbog upotrebe oznaka to nije problem. Stvar se kompenzuje time što dobijate fleksibilnost u primeni.
- XML je relativno nov ali njegovi koreni sežu u početke osamdesetih. On predstavlja evoluciju ideje a ne evoluciju jezika.
- XML se može koristiti za razvoj novih jezika. XML je preteča WAP-a i WML-a. Wireless Markup Language (WML), koji se koristi na primer u obeležavanju Internet aplikacija za mobilne telefone, je zapravo usko specijalizovani XML. Slična stvar je i sa WAP-om.
- XML ne pripada nikome, ne zavisi od platforme i dobro je podržan. XML je samo specifikacija W3 konzorcijuma. Ujedno je i preporuka priznatog autoriteta. Nema profitnu pozadinu i slobodan je za upotrebu. Da biste ga koristili niste ničim obavezani, ni platformom, ni proizvođačima, ni licencama i ugovorima. Shvatite njegovu suštinu i koristite ga onako kako vama odgovara.

3.2. Sintaksa XML-a

Sintaksna pravila XML-a su veoma jednostavna i striktna. Lako se uče i još lakše primenjuju. Zbog toga je kreiranje aplikacija koje čitaju i manipulišu XML-om relativno jednostavno. Pogledajmo opet primer:

```
<?xml version="1.0" ?>
   <poruka>
      <za>Pera</za>
      <od>Mika</od>
      <tema>pozdrav</tema>
      <tekst>Puno pozdrava iz Kraljeva</tekst>
   </poruka>
```

Prva linija XML dokumenta - XML deklaracija - određuje XML verziju dokumenta. U ovom slučaju dokument poštuje specifikaciju 1.0 XML-a koju propisuje W3 konzorcijum. Ovaj red ujedno i govori Internet Exploreru da parsira (rasčlani) dokument XML parserom odnosno da dokument tretira kao XML fajl, a ne kao HTML fajl. Bez ove linije dobili bismo poruku o grešci od IE. Ova linija nema svoj zatvarajući ekvivalent jer ona nije deo XML dokumenta već njegova deklaracija. Sledeće je osnovni tag koji dokument formira kao poruku **<poruka>**. Moguć je samo jedan osnovni tag inače opet dobijamo poruku o grešci. Sledeće četiri linije opisuju četiri podčlana osnovnog člana **<za>**, **<od>**, **<tema>**, i **<tekst>**. Poslednja linija zatvara osnovni tag **</poruka>**.

Svi XML elementi moraju da budu zatvoreni u XML-u, izostavljanje završnog taga vodi u grešku. Dok je u HTML-u prolazilo:

```
<p>ovo je paragraf<p>ovo je još jedan paragraf
```

u XML-u ovo ne bi bilo ispravno, već bi ispravan dokument izgledao ovako:

```
<p>ovo je paragraf</p><p>ovo je još jedan paragraf</p>
```

XML tagovi razlikuju mala i velika slova. Za razliku od HTML-a, XML tagovi su osetljivi na mala i velika slova pa zato treba voditi računa da otvarajući i zatvarajući tagovi budu potpuno identični i po nazivu i po upotrebljenim karakterima:

```
<Poruka>Ovo je neispravno</poruka>
<poruka>Ovo je ispravno</poruka>
```

Svi XML elementi moraju biti propisno ugnežđeni. Neispravno ugnežđeni elementi nemaju smisla u XML-u. Dok se u HTML-u elementi mogu preklapati u XML to nikako nije slučaj. Pogledajmo sledeći primer:

HTML ispravno

```
<b><i>Ovo je tekst</b></i>
```

XML ispravno

```
<b><i>Ovo je tekst</i></b>
```

Svi XML dokumenti moraju da imaju osnovni (top level) ili startni tag. Prvi tag u XML dokumentu je osnovni tag. Svi XML dokumenti moraju da imaju jedan par tagova koji definiše osnovni tag. Svi ostali elementi su ugnežđeni u osnovni tag. Gnežđenje u dubinu je

neograničeno. Znači element može imati neograničen broj elemenata-dece. Odnos koji vlada je takozvani roditelj-dete odnos.

```
<poruka>
   <za>Pera</za>
   <od>Mika</od>
   <tema>pozdrav</tema>
   <tekst>Puno pozdrava iz Kraljeva</tekst>
</poruka>
```

Ovde je par osnovnih tagova **<poruka>** i **</poruka>** dok su podčlanovi parovi: **<za>** i **</za>**, **<od>** i **</od>** , **<tema>** i **</tema>**, **<tekst>** i **</tekst>**.

Vrednosti atributa moraju biti pod znacima navoda. U XML-u se vrednosti atributa moraju uokviriti znacima navoda. XML elementi mogu imati atribute i formi tj. ime=vrednost parova (kao i u HTML). Pogledajmo ova dva XML dokumenta.

Prvi je neispravan:

```
<?xml version="1.0"?>
   <poruka datum=10/06/2000>
   <poruka></poruka>
```

Drugi je ispravan:

```
<?xml version="1.0"?>
   <poruka datum="10/06/2001">
   <poruka></poruka>
```

U XML-u je sačuvan prazan prostor. Korišćenjem XML-a prazan prostor je prikazan u parsiranom dokumentu. Na primer:

```
<body>Puno pozdrava iz Kraljeva</body>
```

će u parseru biti: Puno pozdrava iz Kraljeva, dok to sa HTML-om nije slučaj.

U XML-u, CR / LF karakteri se pretvaraju u LF karakter, U XML-u, nov red u tekstu je uvek sačuvan kao LF (line feed). U Windows aplikacijama nov red je par CR (carriage return) i LF (line feed) karaktera. Kod UNIX sistema karakter za nov red je LF mada neke aplikacije koriste i samo CR. Ova razlika među operativnim sistemima često za posledicu ima da se podaci vraćaju u obliku strima (engl. toka) a ne u željenom formatu.

XML nije nešto specijalno ali ima svoje male tajne. XML je zapravu samo tekst dizajniran tako da ga čita mašina odnosno softver. Softver koji podržava čisti tekst može da obrađuje XML. Na primer, u Notepadu se može obrađivati XML dokument. XML može da sadrži ne-engleske karaktere (č,ć,ž,đ...) međutim tada je potrebno dokument sačuvati u Unicode formatu što nije moguće u nekim verzijama Windows na primer u 95/98 dok je pod Windows 2000 i Windows XP operativnim sistemima to moguće. Stoga je u deklaraciju XML fajla uveden i atribut encoding (engl. dešifrovanje) što zapravo govori browseru koju kodnu stranu da koristi.

```
<?xml version="1.0" encoding="windows-1252"?>
```

Međutim ovde je potrebno obratiti pažnju. Fajlovi sačuvani kao Unicode ne mogu imati i encoding atribut inače se pojavljuje greška u Internet Exploreru.

4. FLASH

Flash je proizvod kompanije Adobe (nekada Macromedia) čiji se fajlovi (filmovi) sa ekstenzijom .swf prikazuju u web-browseru putem plug-in dodatka. Flash plug-in player je razvijen i razvija se za sve vodeće platforme (procesor + operativni sistem + browser). Format .swf je u formi open-source što omogućava njegovo brže i raširenije prisustvo u svetskim razmerama. Na naslovnoj stranici prezentacije firme Adobe najsvežiji podatak je da 99.3% korisnika Interneta poseduje ovaj dodatak, a da je od njih 44-55% na najnovijoj verziji. Ovo praktično govori da Flash jeste postao standard u svetu globalne mreže. Većina novijih browsera pri samoj instalaciji postavljaju ovaj plug-in u svoje okruženje, dok onaj "ostatak" uglavnom predstavljaju korisnisnici zastarelih verzija programa koji ionako ne mogu da udovolje zahtevima savremenih prezentacija. Najnovija verzija programa nosi naziv Adobe Flash CS4 Professional i to je deseta verzija ovog programa.

4.1. Prednosti

Neke od stvari koje su flesh učinile jako popularnim su njegova nezavisnost od platforme, vektorska grafika, animacija i interakcija.

4.1.1. Prednost nezavisnosti od platforme

Svaki posetilac koji poseduje Flash Player ima mogućnost da pogleda .swf film jer ne zavisi od platforme sa koje pristupa fajlu. Dalje, postoji mogućnost da se svi fontovi potrebni za prezentaciju u Flashu uključe u dokument, tako da izgled ne zavisi ni od fontova instaliranih na računaru. Svi grafički elementi u Flashu, pa i fontovi, su vektorski opisani (ako se izuzme mogućnost uvoza bitmapiranih slika) tako da je moguće skaliranje celog filma prema prozoru browsera odnosno – nema zavisnosti od ekranske rezolucije. Kako Flash ne zavisi od platforme, konstrukcije ispitivanja okruženja zbog prilagođenja prikaza nisu potrebne – svaka realizacija prikaza radi se tačno jednom.

4.1.2. Vektorska grafika

Opis vektorske grafike je jednostavan i nema gubitka podataka jer nema ni kompresije. Film je u jednom fajlu tako da server obrađuje samo jedan zahtev pri slanju filma klijentu. Ukupna veličina .swf fajla je manja od analogno tome rađene HTML stranice. I sledeće: Flash film može da počne da se prikazuje odmah po učitavanju prvih kadrova filma, pre nego što se kompletno učita kod klijenta.

4.1.3. Animacija i interakcija

Flash fajl naziva se film zbog mogućnosti animacije, osnovne prednosti na Internetu u odnosu na sva druga ponuđena rešenja.

Primer: animacija kvadrata koji se rotira po nekoj putanji i pritom menja svoju veličinu, pamti se kao objekat opisan pozicijom nekoliko tačaka, početnom i krajnjom pozicijom objekta, faktorom slaliranja objekta i parametrom broja okretaja tokom animacije. Ceo taj opis može biti smešten u desetak bajtova, bez obzira na veličinu kvadrata i broj slika koje učestvuju u celoj animaciji. Realizacija iste ove animacije u bitmapiranom svetu je niz potrebnog broja slika (dužina animacije), od kojih svaka slika kompletno opisuje sadržaj boje svojih tačaka (broj

tačaka zavisi od dimenzija – širina puta visina) ili promenu sadržaja u relaciji na prethodnu sliku. Znači, dimenzije animacije i njena dužina značajno utiču na veličinu zapisa animacije.

Jedan od elemenata Flasha su objekati koji mogu da reaguju na određene događaje, među kojima i su korisnikove akcije unosa putem tastature ili akcije miša, koji dalje mogu da uslovljavaju dalje ponašanje u filmu. Kompletan sadržaj filma u Flashu može se menjati, kao odgovor na korisničke akcije, bez potrebe za dovlačenjem sadržaja sa servera, jer se celokupni sadržaj može se nalaziti u okviru samog filma. U okviru Flasha se za upravljanje kontrolom filma koristi jezik ActionScript, kome je u okviru ovog rada posvećena posebna pažnja.

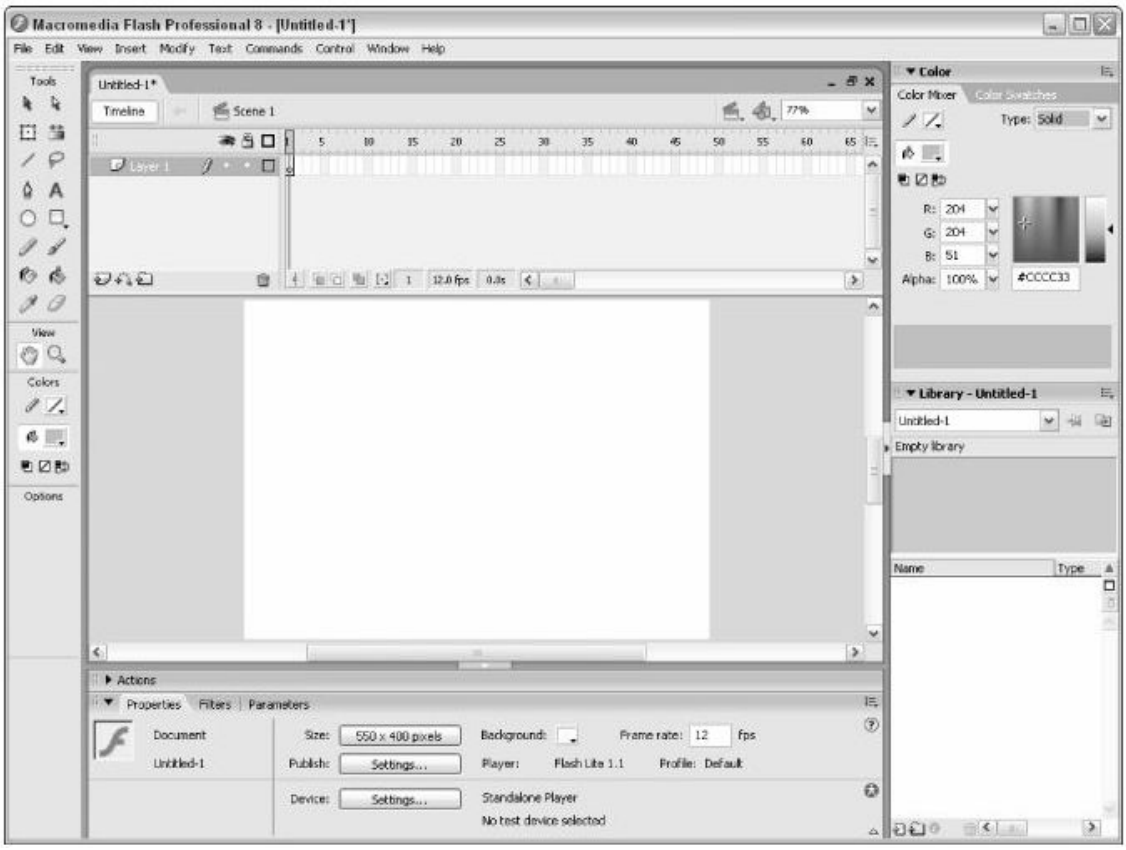
4.2. Radni interfejs

Flash film koji se pravi u Flashu snima se kao .fla dokument. Radno okruženje je podešeno za kreiranje grafičkih elemenata i njihovu animaciju u vremenu. Elementi okruženja su:

- Pozornica
- Vremenska osa
- Kutija alata
- Biblioteka simbola
- Paneli

4.2.1. Pozornica

Centralni deo radnog okruženja dodeljen je pozornici (stage). Ona predstavlja "svet" dvodimenzionalnog koordinatnog sistema sa početkom u gornjem levom uglu, čije se dimenzije (u pikselima) određuju kao globalne za ceo film. Na slici dole, pozornica je centralni beli pravougaonik.



Slika 2 – Flash radni interfejs

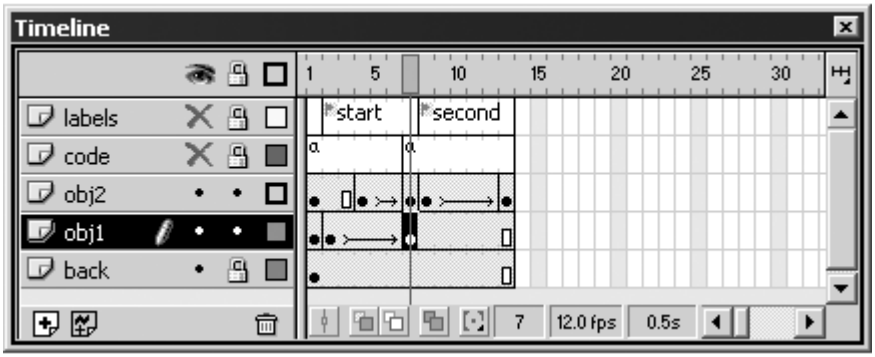
Sam film može da se organizuje po scenama. Tako je pri kreiranju praznog dokumenta kreirana i aktivna prva scena (Scene 1). Slika 2 - prikazuje u interfejsu Flasha pri gornjem levom uglu, ispod menija aktivnu prvu scenu (slika filmske "klape" i natpis "Scene 1").

4.2.2. Vremenska osa

Svakoj sceni pridružuje se vremenska osa (timeline). Ona dodaje treću prostornu dimenziju kroz koncept lejera (leyers) i vremensku komponentu, preko frejmova.

Lejeri se mogu posmatrati kao providne folije naslagane jedna na drugu, određujući tako šta se nalazi "ispod" a šta "iznad". Primer: ako imamo dva lejera sa nekim sadržajem koji se na nekom delu scene preklapa, u preklopljenom delu biće vidljiv sadržaj gornjeg lejera jer on je "iznad" donjeg.

Pojam frejma poznat je iz filmske terminologije, predstavlja stanje u nekom posmatranom trenutku. Vremenska komponenta je diskretizovana na frekvenciju frejmova po sekundi, što se takođe globalno određuje na nivou filma. Ključni frejm je frejm sa nekim definisanim stanjem elemenata određenog lejera. Između dva ključna frejma u lejeru može se zadati animacija elemenata lejera, odnosno distribucija promena po frejmovima između uočenih ključnih frejmova. Ovaj pojam koristi se pri izradi animiranih filmova, gde glavni crtači crtaju ključne slike (frejmove), a animatori dovršavaju posao crtanjem međuslika.



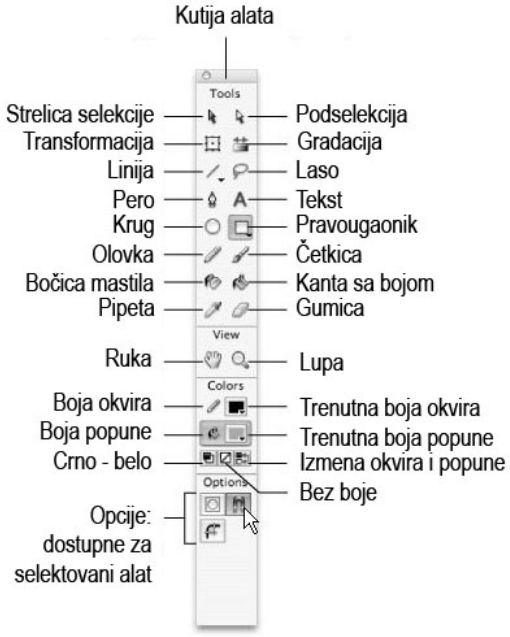
Slika 3 – Prozor vremenske ose

Za ilustraciju ove priče primer je slika iznad. Postoji pet lejera, nazvani *back*, *obj1*, *obj2*, *code* i *labels*. Lejer *back* je najdonji, iznad njega je *obj1*, i tako redom do lejera *labels*, koji je najgornji. Selektovan je lejer *obj1*. Lejeri *labels* i *code* nisu vidljivi (oznaka "x" ispod "oka") i zaključani su (katanac), kao i lejer *back*. Kako se lejer *obj2* nalazi iznad *obj1*, čiji elementi trenutno dizajniraju, to se kod lejera *obj2* prikazuju samo obrisi oblika elemenata (kvadratić isključuje vidljivost popune) kako bi elementi lejera *obj1* mogli biti vidljivi i pored eventualnih preklapanja. Film zauzima 13 frejmova, a u donjem redu očitava se da je aktivan sedmi frejm, da se film podešen za prikazivanje 12 frejmova u sekundi (fps) i da trenutna pozicija predstavlja 0.5 sekundi od početka filma. Na samom sedmom frejmu prisutna je vertikalna crta koja označava selektovani frejm. U preseku selektovanog lejera i frejma je selektovan ključni frejm (popunjen kružić) lejera obj1. Između nekih ključnih frejmova nalazi se strelica, koja predstavlja animaciju elemenata lejera između uočenih frejmova (program u frejmovima strelice izračunava promenu elemenata lejera u odnosu na ključne frejmove). U lejeru *code* vide se ključni frejmovi sa oznakom "a", što ukazuje da ti frejmovi sadrže neke akcije ActionScripta. U lejeru *labels* dva ključna frejma imaju zastavice i tekst koji predstavlja labelu (oznaku) frejma, koja olakšava prepoznavanje delova filma i čije ime se može koristiti pri zadavanju komandi ActionScripta za kontrolu toka filma (na primer, može se narediti skok na poziciju imenovanog frejma).

4.2.3. Kutija alata

Za rad u filmu, sceni, lejeru i frejmu koriste se razne alatke iz kutije alata (slika desno), koja je podeljena na četiri dela: alati za crtanje, vizuelno pozicioniranje, selektori boja i opcije selektovane alatke.

Poput drugih programa za rad sa vektorskom grafikom i u Flashu su "standardne" alatke za crtanje: selektor, linija, pravougaonik, elipsa, slobodno crtanje, gumica... Postoje dva selektora za boje: za boju linije (konture) i za boju popune konture. Za popunu se, pored osnovnih boja, mogu odabrati i radijalno ili linearno nijansirani prelazi boja. Boja pozadine određuje se globalno na nivou filma.



Slika 4 – Kutija alata

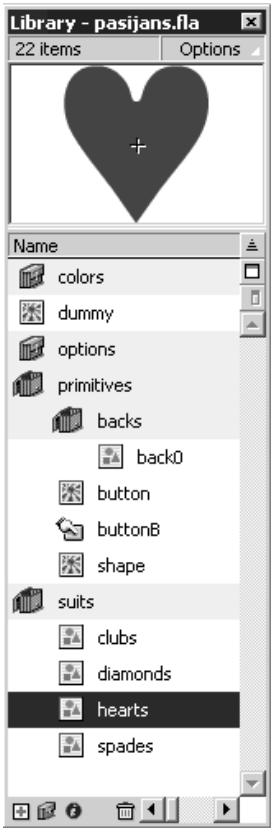
4.2.4. Biblioteka simbola

Crtanje grafičkih elemenata može se izvoditi direktno na sceni i tom prilikom kreiraju se oblici (shapes). Drugi prisup jeste kreiranje simbola. Simboli su elementi koji se mogu više puta koristiti. Poseduju sopstvenu vremensku osu. Opis simbola čuva se i organizuje u okviru biblioteke simbola (Library).

Na slici desno prikazana je biblioteka simbola programa pasijans. Sa slike se vidi da svaki simbol ima svoje ime, kao i da se simboli mogu organizovati po folderima simbola (na slici su neki folderi zatvoreni, npr. colors, dok su neki otvoreni – suits). Različite sličice levo od imena simbola označavaju tip simbola.

Postoji tri osnovna editabilna tipa simbola: filmski klip (Movie Clip), dugme (Button) i grafike (Graphic). Na slici je simbol dummy tipa Movie Clip, buttonB tipa Button, a back tipa Graphic. Postoje još i tipovi uvoznih simbola – slike, zvučni i video zapisi.

Simbol hearts je selektovan, a u gornjem prozoru prikazan je njegov izgled. Postavljanje simbola na scenu u stvari je njegovo instanciranje. Instanci simbola moguće je menjati razne osobine (pozicija, skaliranje, rotacija, ...) bez uticaja na definiciju simbola.

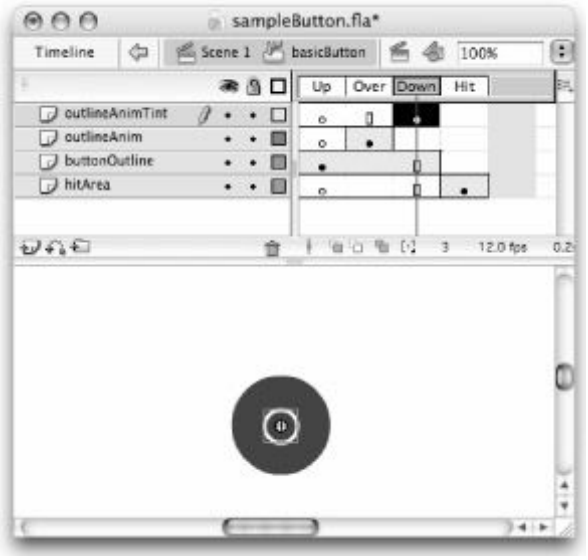


Slika 5 - Biblioteka simbola

Menjanje osobina u definiciji simbola odražava se na svim njegovim instancama.

Simbol tipa dugme reaguje na događaje miša – kada se pokazivač miša nalazi iznad simbola i kada je pritisnuto dugme miša dok je pokazivač iznad simbola. Zato vremenska osa simbola tipa dugme sadrži četiri specijalna frejma:

- **up** – stanje simbola kada se pokazivač miša ne nalazi iznad simbola
- **over** – pokazivač je iznad simbola
- **down** – pokazivač je iznad i pritisnuto je levo dugme miša
- **hit** – ovaj frejm određuje površinu na osnovu koje simbol određuje svoje stanje



Slika 6 – Simbol tipa dugme sa 4 osnovna stanja

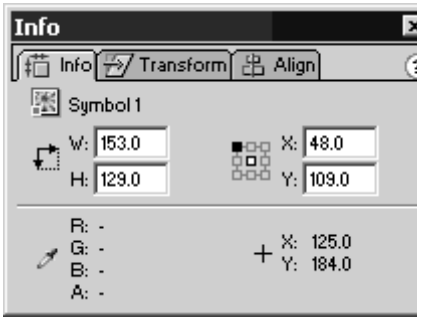
Grafički simbol može biti animiran, ali se njegova vremenska osa vezuje za scenu (ili filmski klip, ako se u njemu instancira), tako da sve promene toka (po frejmovima) roditeljskog elementa utiču i na instancu grafičkog simbola. Tako, na primer, ako se u nekom trenutku zaustavi animacija na sceni, zaustavlja se i animacija svih instanci simbola tipa grafika.

Drugačije ponašanje u odnosu na ovo realizuju simboli tipa filmski klip. Gledano na istom primeru, zaustavljanje animacije na sceni ne utiče na ponašanje instanci filmskih klipova postavljenih na scenu već naprotiv, one nastavljaju tok svoje vremenske ose. Pored ovoga, svakoj instanci simbola filmskog klipa može se dodeliti jedinstveno ime instance, kojim se potom referencira instanca u akcijama ActionScripta.

4.2.5. Paneli

Za pregled, organizaciju i modifikaciju elemenata Flash filma koriste se paneli sa komandama i opcijama vezanim za posmatrani tip elementa. Mogu se modifikovati simboli, instance, boje, tekst, frejmovi i drugi elementi filma.

Slika desno predstavlja jedan panel. Paneli se mogu organizovati u grupe, tako da se na ovoj slici u stvari nalaze tri panela, od kojih je aktivan (jezičak gore) panel Info. Panel svoj prikaz vezuje za selektovani element scene, frejm i slično, i nudi opcije koje se mogu vršiti nad selekcijom.



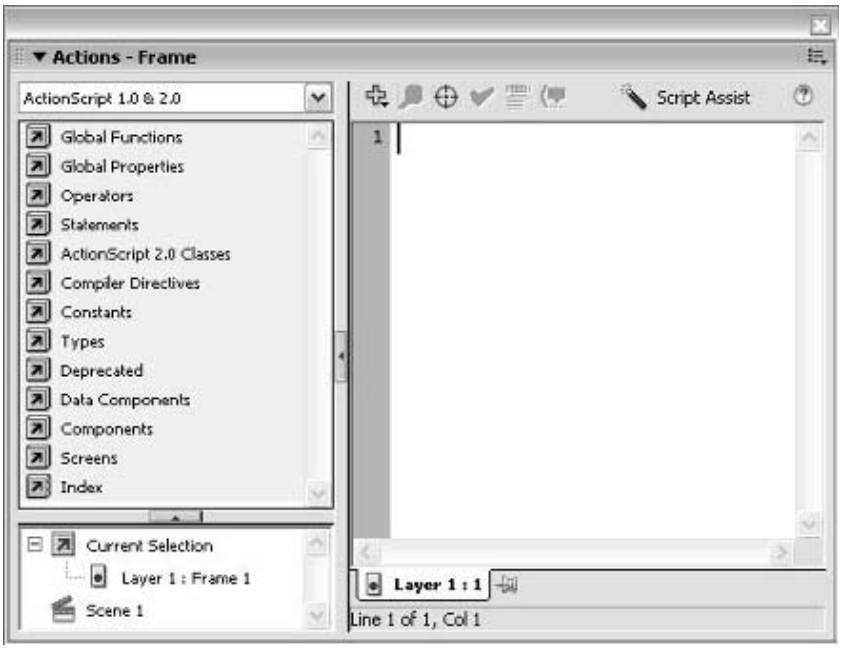
Slika 7 – Info panel

4.3. Interfejs za programiranje

Akcioni panel je glavni deo fleshovog interfejsa zaduženog za programiranje, koji se aktivira i gasi pritiskom na funkcijski taster "F9". Najčešće je otvoren u okviru interfejsa ispod pozornice, a može i da se odvoji kao nezavistan prozor.

Sastoji se od tri dela:

- biblioteke alata
- navigatora skripti
- prozora skripti

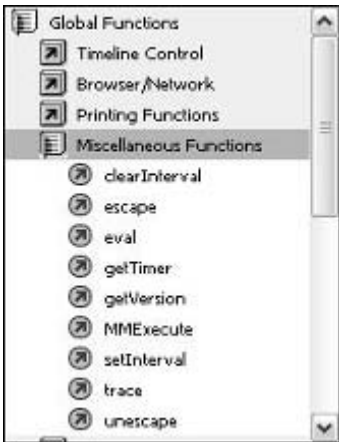


Slika 8 – Akcioni panel sadrži tri jasno odvojena dela

4.3.1. Biblioteka alata

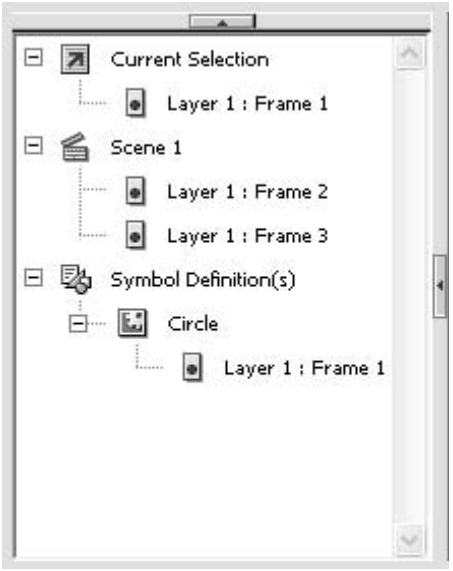
U ovom delu nalaze se sve raspoložive akcije, operatori, objekti i to kategorizovano prema definiciji i korišćenju. Sve ovo je organizovano po delovima knjige nazvanim po oblastima npr: Global Functions, Statements, Operators, i slično. Svaki deo sadrži i detaljnije razvrstane oblasti koje otvaramo i zatvaramo klikom.

Akcije dodajemo u prozor skripti duplim klikom ili prevlačenjem. Ovaj koristan način dolaska do sintakse akcija nije i jedini u Flashu.



Slika 9 – Biblioteka alata

4.3.2. Navigator skripti



Slika 10 – Navigator skripti

Navigator ima nekoliko nivoa.

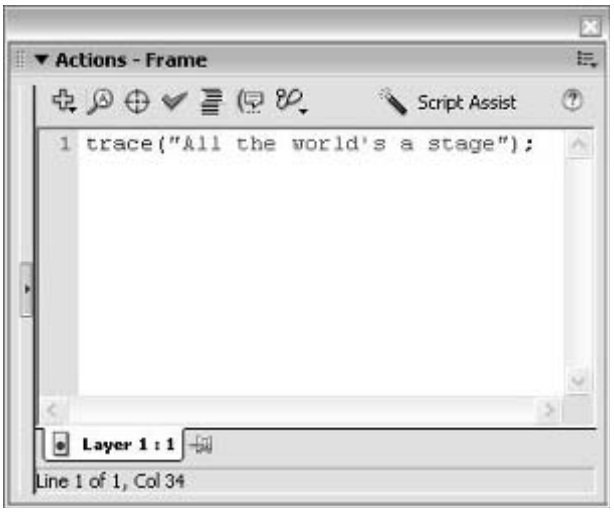
Uvek dostupna opcija je Current Selection i kada je otvorena možete selektovati stavku trenutne selekcije bilo to frejm ili objekat.

Naredni nivo su scene u filmu. Ako se selektuje scena dobija se mogućnost selekcije svih frejmova ili objekata sa kodom na njima.

Poslednji nivo ako ima bilo kog simbola ili instance nekog simbola u filmu je Symbol Definition(s). Klikom dobijamo sve simbole povezane sa skriptama u našem filmu.

Klikom na bilo koju skriptu u bilo kom nivou navigatora skripti ona postaje aktivna u prozoru skripti.

4.3.3. Prozor skripte ili koda



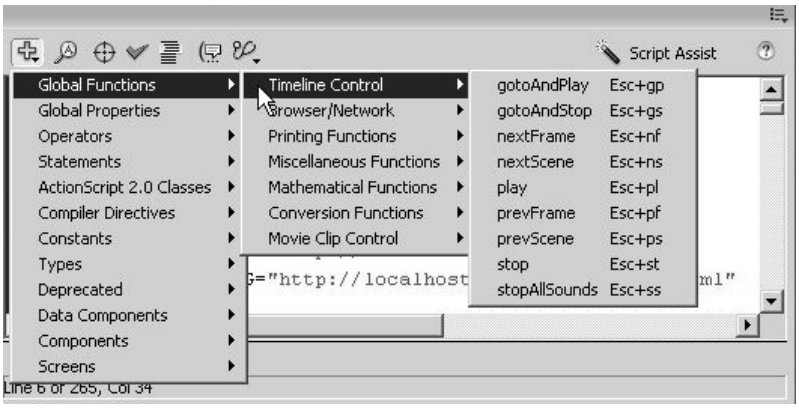
Slika 11 – Prozor koda

Ovo je glavni prozor za programiranje. U njemu se unosi ActionScript kod na film, ili neki njegov deo. Na vrhu prozora se nalazi toolbar koji nudi lak i funkcionalan način provere sintakse i formatiranja skripti.

Glavni deo sadrži deo za tekst gde se unosi kod. Na dnu se nalaze jezičci otvorenih skripti kojima se iste aktiviraju, a postoji i mogućnost fiksiranja skripte koju gledamo radi provere nekog koda na drugom objektu. Na dnu je statusna linija sa brojem linije i reda u aktivnoj skripti.

Iznad koda u toolbaru nalaze se sa leva na desno sledeće opcije:

- **Actions Add Menu** – jos jedan način dolaska do odgovarajuće sintakse, koje su ovde kao i u biblioteci alata razvrstane po glavnim oblastima i primenama. Ikona je u obliku znaka plus, klikom se otvara niz podmenija.
- **Find and Replace** – klasični mehanizam za pronalaženje teksta i zamene istog u kodu skripte.
- **Insert Target Path** – ovom opcijom uz pomoć grafičkog interfejsa biramo putanju MovieClip instanci.
- **Check Syntax** – provera sintakse filma pre pokušaja eksportovanja ili startovanja. Ako neka greška nastane - biće prikazana u izlaznom prozoru.
- **Auto Format** – ova opcija se koristi za automatsko formatiranje koda za optimalno čitanje.
- **Show Code Hint** – ova opcija prikazuje odabrani savet iz baze programa za datu liniju koda.
- **Debug Options** – ova opcija daje meni za asistiranje pri dibagovanju, postavljanju ili brisanju debug tačaka u kodu.
- **Script Assist** – pomoćnik pisanja skripte se koristi ako niste sigurni oko sintakse ili želite da promenite značenje neke linije koda.
- **Help** – otvara klasičan windows mehanizam za pomoć u Flash-u.



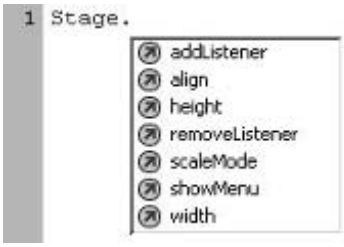
Slika 12 – Prikaz menija "Action Add Menu"

4.3.4. Saveti u kodu

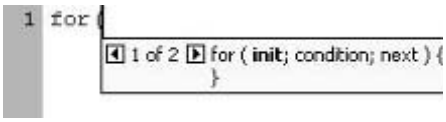
Savetima u kodu flesh daje još jedan način savladavanja sintakse osobina ili metoda objekata ili promenljivih.

Na primerima levo se vidi da se u raznim situacijama javlja prozor sa ponuđenim opcijama. Gore se posle tačke pojavljuju moguće osobine objekta Stage, a dole posle zagrade i započete **for** komande pojavljuje sintaksa iste.

Saveti se pojavljuju na dva načina i to automatski i ručno. Automatski posle prepoznatog objekta ili varijable od strane programa. Ručno se savet poziva iz pomenutog toolbara ili pritiskom na kombinaciju tastera Ctrl i razmak (Space).



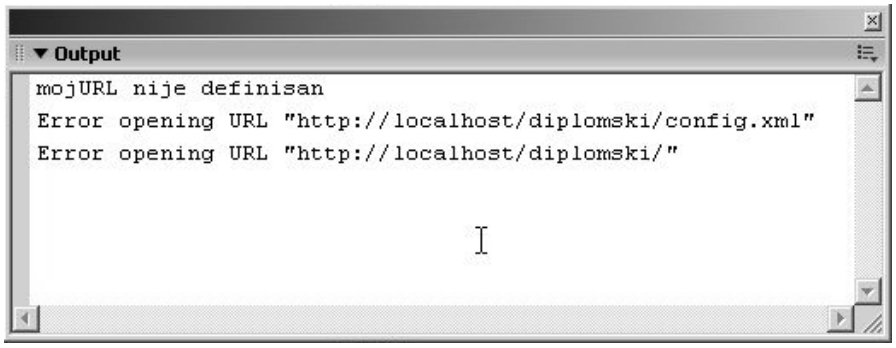
Slika 13 – Savet u kodu posle objekta



Slika 14 – Savet u kodu posle naredbe

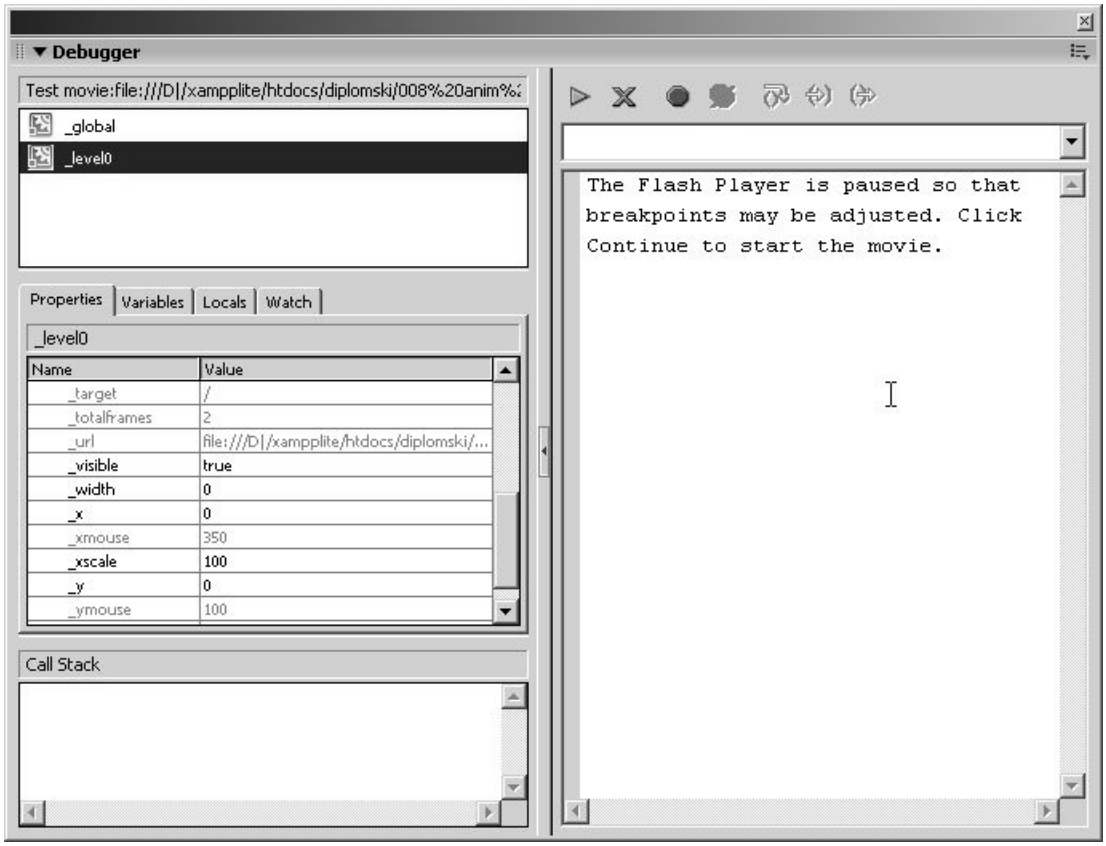
4.4. Prozor izlaza i debagovanje

Za kontrolisanje toka programa i promenljivih koristi se prozor izlaza koji se aktivira na F2 ili prilikom aktiviranja programa i izvršavanja komande **trace()**. Sledeća slika prikazuje rezultat tri **trace()** komande iz diplomskog rada.



Slika 15 – Izvršavanje trace() komande aktivira pojavu Output panela

Pri razvoju većih programa korisiti se dibager prozor uz obavezno postavljanje tačaka prekida programa iz prozora skripte ili koda. Prilikom zaustavljanja izvršavanja programa levo u prozoru će se naći sve trenutno definisane promenljive i objekti, a zatim se po selektovanju istih levo dole mogu videti vrednosti osobina i varijabli razvrstanih po oblastima delovanja. Desno je izlazni prozor dibagera sa kontrolama postepenog prolaza po tačkama prekida.



Slika 16 – Prozor dibagera

4.5. Rad u Flashu

Da bi autor ostvario svoj projekat filma u Flashu, koristi se različitim tehnikama editovanja sadržaja svog rada. U nastavku su predstavljeni osnovni koncepti rada koje nudi okruženje.

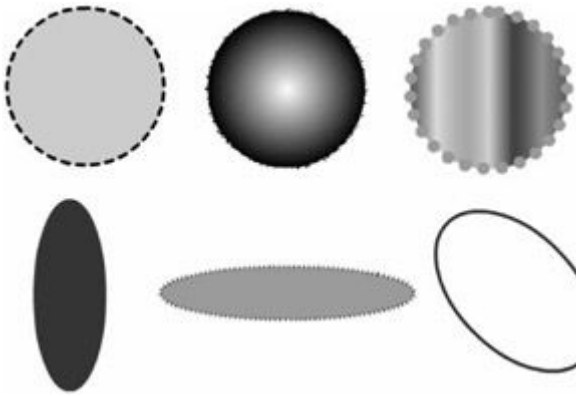
4.5.1. Početak rada

Otvaranjem programa, ili zadavanjem komande za novi projekat, kreira se inicijalni dokument u kome je definisana i aktivna prva **scena** (Scene 1), koji sadrži jedan **lejer** (Layer 1) sa jednim praznim frejmom. Dimenzije scene, broj slika u sekundi i boja pozadine uzimaju vrednosti koje su u programu postavljene za podrazumevane. Ovi parametri mogu se menjati u bilo kom trenutku rada, a od trenutka promene postaju globalni za ceo film.

4.5.2. Crtanje

Grafički elemenati mogu biti kreirani korišćenjem alata za crtanje. Crtanje se odvija u okviru selektovanog ključnog frejma selektovanog lejera, u okviru elementa filma koji se edituje.

Element filma može biti scena ili simbol. Odabirom alatke (linija, elipsa, pravouganik, slobodno crtanje...) određuje se šta će se crtati, selekcijom boja određuju se boje linije i popune elementa koji se crta, a u panelima moguće je odrediti tip i debljinu linije, način popune (čista boja, radijalni ili linearni gradijent, bitmapa) i procenat transparencije boja. Sve osobine moguće je i naknadno menjati selekcijom željenih kontura ili popuna i promenom vrednosti osobine u određenom panelu ili selektoru boje.



Slika 17 – Primeri crtanja alatkom elipsa

Za pomoć pri crtanju koriste se rešetka - **grid**, vođice - **guidelines**, koje se vuku sa lenjira - **ruler**, i već postojeći objekti na sceni (ili u definiciji simbola). Postoji mogućnost vezivanja - **snap** ključnih tačaka elementa koji se crta za neki od pomenutih elemenata, a takođe se određuje i stepen vezivanja (koliko pokazivač miša može biti udaljen od elementa da bi došlo do vezivanja). Parametri rešetke, horizontalni i vertikalni razmak između linija rešetke, menjaju se po potrebi. Pozicioniranje elementa moguće izvodi se poravnavanjem u odnosu na druge elemente ili u odnosu na pozornicu, kao i distribucija elemenata, levo, desno ili centrirano.

Pozicija elementa može se i numerički odrediti, kao i dimenzije, a moguće je izvoditi i relativnu transformaciju dimenzija (procentualno), rotacije i zakošenja - **skew**.

4.5.3. Uvozni elementi

Vektorski elementi pozornice mogu se dobiti i uvozom vektorske grafike urađene u nekom drugom programu. Ovo je naročito korisno pri potrebi za zahtevnijim grafikama, koje je ipak lakše realizovati u specijalizovanim programima (Adobe Illustrator, Macromedia Freehand, Corel Draw i slično). Uvežen vektorski element može se dalje obrađivati na isti način kao i grafika crtana u Flashu.

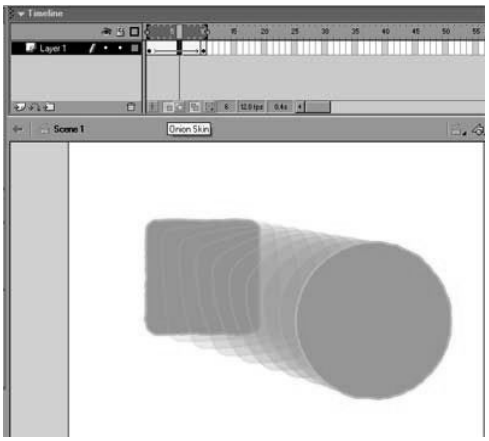
Pored uvoza vektorskih slika, mogu se uvoziti bitmapirane slike koje se, kao bitmapirani simboli, smeštaju pri uvozu u biblioteku, dok se instanca postavlja na pozornicu. Instanci se mogu menjati pozicija, veličina, rotacija i zakošenje. Kao i kod bitmapiranih slika, zvučni i video zapisi takođe uvozom postaju elementi biblioteke simbola.

4.5.4. Animacija

Pri dizajniranju filma mogu se koristi dve vrste animacije: animacija oblika - **shape tween** i instanci simbola ili grupisanih objekata - **motion tween**. Zajedno za obe vrste animacije je da se definiše između dva ključna frejma (početnog i krajnjeg frejma animacije) istog lejera i definisanje dinamike animacije (**easing** – pozitivne vrednosti daju bržu promenu u na početku animacije, negativne obratno).

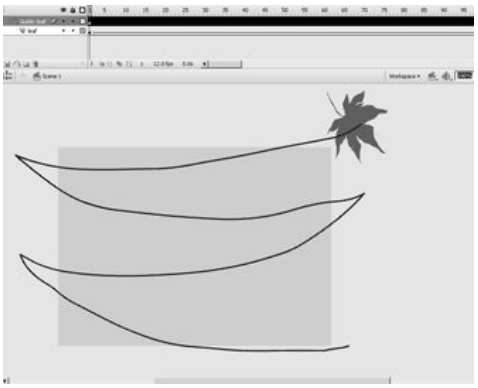
Slika 18- Primer klasične animacije crtanjem slika u različitim frejmovima

Animacija oblika predstavlja jednostavno pretapanje oblika između uočenih ključnih frejmova. Ne može se primenjivati na instance simbola i grupisane objekte, već samo na oblike. Primer bi bio transformacija pravougaonika u elipsu: u prvom ključnom frejmu crta se pravougaonik, u drugom elipsa, i na kraju se kod prvog ključnog frejma zadaje animacija **tweening** postavljanjem na **shape**. O frejmovima između stara se Flash i izračunava međuoblike.



Slika 19 – Animacija Shape tween

Motion tween, slično, zahteva samo jednu (istu) instancu simbola ili grupu simbola prisutnu na oba ključna frejma lejera. Sve promene vrednosti osobina elementa ključnih frejmovima se pri zadavanju animacije (**tweening** na **motion**) distribuiraju po međufrejmovima. Pri zadavanju može se odrediti i dodatna rotacija elementa (smer i broj rotacija) tokom animacije. Naprednija varijanta ove animacije postiže se tako što se lejeru pridruži vodeći lejer - **motion guide**. Na vodećom lejeru iscrtava se samo putanja elementa. Element se u ključnim frejmovima vezuje za početnu i krajnju tačku putanje. Pri ovakvom zadavanju animacije određuje se da li se element tokom animacije orijentiše i po putanji, ili ne. Sadržaj vodećeg lejera se ne prikazuje pri testiranju filma.



Slika 20 – Animacija Motion Guide

4.5.5. Tekst

Na pozornicu se mogu dodavati i tekstualni elementi. Dodaju se kao tekstualna polja koja se definišu kao:

- statička
- dinamička
- ulazna

Statičko polje određuje svoj sadržaj kucanjem potrebnog teksta tokom editovanja u samo polje.

Dinamičko polje vezuje svoj sadržaj za neku promenljivu koja se definiše u kodu ActionScripta, ovo polje prikazuje vrednost promenljive. Polje je definisano kada se odredi ime promenljive kojoj se polje pridružuje.

Slično je i sa ulaznim poljem, sadržaj se vezuje za promenljivu. Razlika što se korisniku dozvoljava da menja sadržaj polja, time i vrednost promenljive.

Tekstualna polja mogu zauzimati jednu liniju ili se prostirati na više linija. Slično kao i u programu Word, moguća su razna podešavanja sadržaja polja po osobinama paragrafa (centriranje, margine, uvlačenje, razmak između redova) i karaktera (tip fonta, veličina, boja, stil, definisanje hiperlinka).

.

5. ACTIONSCRIPT

ActionScript je skript jezik čiji je cilj kontrola objekata u Flash filmovima, kreiranje navigacionih i interaktivnih elementa i za kreiranje filmova visokog stepena interakcije i web aplikacija.

5.1. Skript jezik

Skript jezik je programski jezik koji se koristi za manipulaciju, prilagođenje i automatizaciju postojećeg sistema. U takvim sistemima, funkcionalnost je već obezbeđena putem korisničkog interfejsa, a skript jezik predstavlja mehanizam za proširenje funkcionasti programskom kontrolom. Za postojeći sistem se kaže da obezbeđuje okruženje za objekte i mogućnosti koje kontroliše skript jezik.

Asocijacija evropskih proizvođača kompjutera (ECMA – the European Computers Manufacturers Association, www.ecma.ch) sastavila je dokument pod nazivom ECMA-262 čiji je cilj standardizacija jezika JavaScript. ActionScript zasniva se na specifikaciji ovog dokumenta.(*http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM*)

5.2. Elementi jezika

Sintaksa ActionScripta podseća na Javu. Razlika je u tome što jezik upućuje na jednostavnost u korišćenju. ActionScript se zasniva na objektima: osnovni jezik i okruženje Flasha obezbeđuju se putem objekata, a sam program u ActionScriptu je put komunikaciji među objektima. Sam objekat je kolekcija osobina - **properties** koje mogu imati atribute koji određuju način korišćenja osobine (na primer – ReadOnly). Osobine su kontejneri koji sadržavaju druge objekte, primitivne vrednosti ili metode. Objekat u ovom smislu predstavlja referencu na objekat, primitivna vrednost je vrednost nekog primitivnog tipa podataka, a metod je funkcija koja se s objektom povezuje putem osobine.

5.3. Tipovi podataka i promenljive

Tip podataka predstavlja vrstu informacije koju promenljiva ili element ActionScripta može da sadrži. Tipovi podataka dele se na primitivne i referentne tipove, a postoje i specijalni tipovi podataka.

Primitivni tipovi su **string**, **number** i **boolean.**

Referentni tipovi su **object**, **function** i **movieClip.**

Specijalni tipovi podataka: **null** i **Undefined.**

Referentni tipovi kao vrednost imaju adresu svog sadržaja. Tip **movieClip** je grafička reprezentacija elemenata Flasha.

Flash 8 i ActionScript 2.0 deklarišu varijable koristeći “jaki stil” tako da kad se varijabla deklariše njoj se određuje i tip koji će sadržati. Kada se film eksportuje Flash vodi računa da u promenljivama čuvaju ispravni tipovi podataka. Ako se detektuje bilo kakvo pogrešno prosleđivanje promenljivama, generiše se greška. Da bi definisali promenljivu koristimo sledeći šablon:

var *promenljivaIme*:*TipPodataka*;

Po službenoj reči **var** znamo da deklarišemo promenljivu. Iza dve tačke po imenu promenljive *promenljivaIme* navodimo *TipPodataka*. Pomoć oko ovoga deklarisanja daju i saveti u kodu. Primeri:

```
var nKolicina:Number;
var nKolicina:Number = 6;
nKolicina = 5;
```

U prvom primeru definišemo brojnu promenljivu, a u drugom imamo i njeno inicijalizovanje sa vrednosti **undefined** na vrednost 6. Dobra navika je i da se promenljiva koja je definisana, a nije joj dodeljena vrednost da se postavi na vrednost **null.** Po prvom definisanju korišćenjem službene reči **var** ista se više ne koristi već se dodeljivanje vrši postojećem imenu promenljive npr. *nKolicina*.

```
var nJedan:Number = Number("468");      // 468
var nDva:Number = Number("23.45");      // 23.45
var nTri:Number = parseInt("54");       // 54
var nCetiri:Number = parseFloat("54");  // 54
```

Za konvertovanje stringova u brojeve koristi se funkcija **Number(),** a za konkretnije konvertovanje korisite se funkvije **parseInt()** i **parseFloat()**. Prva konvertuje string u broj i to promenljive *nJedan* u **integer**, a *nDva* u **float**. Treća funkcija **parseInt**() sigurno formira **integer**, a **parseFloat**() promenljivu tipa **float.**

5.4. Operatori

ActionScript definiše skup ugrađenih operatora koji se mogu podeliti u sledeće grupe: numerički operatori, operatori poređenja, String operatori, logički operatori, bit-orijentisani operatori, operatori jednakosti i operatori dodele. Kao i u drugim programskim jezicima, operatori su hijerarhijski određeni po prioritetu izvršavanja operacija.

Zanimljivi operatori su, recimo, operatori dodele, što ilustruje sledeći primer:

```
var myString = "moj";
myString += " string";
trace ("myString: " + myString);  //  "myString: moj string"
```

Prva linija deklariše promenljivu myString i dodeljuje mu ("obična" dodela) string vrednost "moj". U drugoj linija koda takođe je dodela, ali sa operatorom +=, što semantički predstavlja sledeće: saberi vrednost operanda s leve strane operatora (promenljivu myString) sa operandom s desne strane (" string") i rezultat sabiranja dodeli promenljivoj s leve strane.

Drugi primer ilustruje operatore inkrementacije (++):

```
var myVar = 0;
trace ("myVar: " + myVar);          //  "myVar: 0"
trace ("myVar++: " + myVar++);      //  "myVar++: 0"
trace ("myVar: " + myVar);          //  "myVar: 1"
trace ("++myVar: " + ++myVar);      //  "++myVar: 2"
trace ("myVar: " + myVar);          //  "myVar: 2"
```

Post-inkrementacija povećava vrednost promenljive nakon njenog učestvovanja u izrazu, a pre-inkrementacija prvo povećava vrednost promenljive koja daje svoju novu vrednost za izračunavanje izraza.

Treći primer, uslovna dodela:

```
var myBool = true;
var notBool = !myBool;
trace (“myBool: “ + ((myBool) ? myBool : notBool));          // “myBool: true”
trace (“notBool: “ + ((notBool) ? myBool : notBool));        // “notBool: false”
```

Uslovna dodela je ternarni operator. Prvi operand je logički izraz koji se izračunava. Ako je izračunao **true**, operator računa vrednost drugog operanda (iza znaka pitanja) i to vraća kao rezultat, inače (izračunato **false**) se računa vrednost trećeg operanda (iza dvotačke) i to postaje rezultat. U ovom primeru prvi poziv trace ispisuje “**myBool: true**”, jer se ispituje uslov myBool **true** i vraća vrednost drugog operanda *myBool* ima vrednost **true**, dok drugi poziv ispisuje “**notBool: false**” jer je uslov netačan *notBool*, pri dodeli je !*myBool*, negacija od **true** pa se vraća vrednost trećeg operanda *notBool* je **false**).

5.5. Grananje i petlje

Naredba **if** u ActionScriptu ima sledeću sintaksu:

```
if ( uslov ) {
      //  linije koda
   } else {
      //  linije koda
   }
```

Ovde je *uslov* - logički izraz, obavezno naveden u zagradama, na osnovu kojeg se donosi odluka da li se izvršava kod bloka naredbi unutar vitičastih zagrada. Blok naredbi se uvek navodi unutar vitičastih zagrada, a pojedine naredbe u bloku razdvajaju se tačka-zarezom (**;**). U slučaju samo jedne naredbe nije neophodno navođenje vitičastih zagrada. Druga grana **else** takođe nije obavezna.

Ponavljanje dela koda obezbeđeno je petljama **for**, **while**, **do-while** i **for-in**. U svim petljama mogu se koristiti komande **continue**, za skok na sledeću iteraciju, i **break**, za napuštanje petlje.

```
for ( start; uslov; sledeci ) {
   //  linije koda
   }
```

Slično kao i u jeziku Java, **for** petlja zadaje se s tri parametra: startna vrednost kontrolne promenljive, izlazni uslov i izraz za izračunavanje pre naredne iteracije. Po definiciji je dozvoljeno izostavljanje bilo kog od ova tri dela (ostaje tačka-zarez). Izraz *start* se izračunava samo jednom, pre početka prve iteracije, izraz *uslov* mora biti logički i izračunava se pre svake iteracije i odlučuje da li se nastavlja sa iteracijom (izračunato **true**) ili ne (**false**), izraz *sledeci* izračunava se na kraju svake iteracije. Ovakva konstrukcija dozvoljava mnogo veću slobodu pri baratanju petljom u odnosu na, recimo, paskaloliku konstrukciju. Ipak, uobičajeno se koristi baš simulacija te osnovne konstrukcije, postavljanjem inicijalne vrednosti kontrolne promenljive u prvom delu, poređenje kontrolne promenljive s nekom izlaznom vrednošću u drugom delu i inkrementiranje ili dekrementiranje kontrolne promenljive u trećem delu.

```
while ( uslov ) {
   //  linije koda
   }
```

Dogod je *uslov* – logički izraz ispunjen izvršava se blok naredbi petlje **while**. Ovo ispitivanje može biti i numeričko, u kom slučaju se ispituje različitost od vrednosti 0.

```
do {
    //  linije koda
} while ( uslov )
```

Slično je i kod petlje **do-while**, s razlikom što se prvo izvršava blok naredbi, pa tek onda proverava uslov (što znači, kod bloka izvršava se bar jednom). Ovo podseća na paskalsko **repeat-until**, a razlika je kod uslova – **do-while** radi dok je ispunjen uslov, a **repeat-until** prekida po ispunjenju uslova.

```
for (osobina in object){
    //  linije koda
}
```

Petlja **for-in** koristi se osobinama objekta.

Primer:

```
myObject = {ime:'Mira', godine:27, grad:'Kragujevac'};
for (vrednost in myObject) {
    trace ("myObject." + vrednost + " = " + myObject [vrednost]);
    }
// myObject.ime=Mira
// myObject.godine=27
// myObject.grad=Kragujevac
```

Prva linija koda daje jedan način zadavanja objekta. Objekat *myObject* ima 3 osobine, svakoj je dodeljena određena vrednost. Promenljivoj *vrednost* u **for-in** petlji pridružuju se imena osobina objekta *myObject*, redom kroz iteracije. Primer takođe prikazuje dva načina pristupa vrednostima osobina objekta. U izlaznom prozoru prikazan je prikaz referenciranja "tačkom", odnosno navodi se ime objekta, tačka i sledi ime osobine. To i jeste uobičajen način referenciranja, ali nije bilo moguće unutar **for-in** petlje pozvati *myObject.vrednost*, jer bi se to odnosilo na osobinu vrednost koju objekat nema. Stoga je korišćena referenca *myObject*[vrednost], koja izračunava izraz u uglastim zagradama (vrednost promenljive vrednost) i potom traži vrednost odgovarajuće osobine objekta.

Kada treba ispitati neku vrednost više puta umesto korišćenaja **if** više puta koristimo **switch case** konstrukciju gde u narednom opštem primeru testiramo *uslov* i ukolikoje *uslov* izraz *testExp1* izvršavaju se komande i iskače sa break iz **switcha**. **Case** stavki može biti jedna, opciono možemonavesti više, a dozvoljeno je i opciono korišćenje ako nije ništa od ponuđenih opcija onda se izvrši **default** linije koda.

```
switch( uslov ){
    case testExp1:
        //  linije koda
    break;
    case testExp2:
        //  linije koda
    break;
    default:
        //  linije koda
    }
```

5.6. Funkcije

Definicija funkcije:

```
function myFunc [ ( arg0, arg1,...,argN ] ):tipFunkcije {
        //   linije koda
    }
```

ili

```
function [ ( arg0, arg1,...argN ) ]:tipFunkcije {
        //   linije koda
    }
```

Funkcija je blok koda koji se može više puta koristiti pozivom funkcije. Primer poziva bio bi *myFunc()*. U zagradi se nalaze argumenti funkcije, koje ona pri izvršavanju prima. Uglaste zagrade su u primeru navedene u smislu BNF notacije, odnosno ukazuju na opcionu pojavu parametara i argumenata. Funkcija ove parametre vidi kao vrednosne parametre, nema varijabilnih parametara, te se ne mogu menjati vrednosti ovih parametara van lokalnosti funkcije, uz dužnu pažnju referentnim podacima, gde se može direktno pristupati osobinama objekta, na primer.

Ulazne parametre funkcija vidi kao niz argumenata, broj deklarisanih imenovanih parametara funkcije ne mora da se poklapa sa brojem argumenata koji se prosleđuju funkciji. Pristup imenovanim parametrima u funkciji moguć je direktno navođenjem imena parametra, a pristup proizvoljnom parametru ostvaruje se kao pristup elementu niza arguments na određenom indeksu. Iza dve tačke navodi se *tipFunkcije* u najvećem broju slučajeva, ako funkcija ne vraća vrednost je **Void.** Ako je vrednost određene vrednosti stavlja se **Number** ili **String**.

Primer:

```
function myFunc ():Void {
    var myVar;
    for (var i = 0; i < arguments.length; i++) {
        myVar += arguments [i];
        }
    trace ("myVar: " + myVar);
}

myFunc (3, 4, "to", 1, 1);              //   "myVar: 7to11"
```

U primeru se lokalna promenjiva *myVar* samo deklariše, inicijalno nema vrednost, znači tipa je **undefined**. Prvo sabiranje je sa argumentom tipa number, pa se promenljiva konvertuje u vrednost 0, sabira sa 3 i dobija vrednost 3. Sabiranje sa drugim argumentom je sabiranje dva broja, pa *myVar* dobija vrednost 7. Treći argument je tipa string, što povlači konverziju *myVar* u string. Četvrti i peti argument su brojevi, ali se konvertuju u **string**. Tako *myVar* na kraju dobija vrednost "7to11".

Funkcija može da vrati vrednost korišćenjem naredbe **return** *returnValue*, ovde *returnValue* predstavlja izraz čiju vrednost funkcija vraća. Ako funkcija vraća **String** ili **Number** treba se i navesti pridefinisanju funkcije kao u primeru.

```
function dajPovrsinu(nA:Number, nB:Number):Number {
    var nPovrsina:Number = nA * nB;
    return nPovrsina;
}
```

U ActionScriptu postoje predefinisane funkcije, a mogu se definisati i nove. Često se funkcija pridružuje nekom objektu, tada ona postaje metod objekta.

5.7. Objekti

Objekat je skup osobina koje mogu imati vrednosti nekog tipa. Svaki objekat jedinstveno se identifikuje svojim imenom. Već pomenuti primer:

```
myObject = {ime:'Mira', godine:27, grad:'Kragujevac'};
```

Objekat *myObject* ima tri osobine i svakoj osobini dodeljena je vrednost. Pristup osobini godine, na primer, može se ostvariti sa *myObject.godine* ili *myObject*[“*godine*”]. Dodavanje nove osobine moguće je: *myObject.drzava*=‘Srbija’, kao i modifikacija postojeće osobine: *myObject.age* = 28.

```
function persona (ime, godine, grad) {
        this.ime = ime;
        this.godine = godine;
        this.grad = grad;
}
myObject = new person (‘Mira’, 27, ‘Kragujevac’);
```

Ovaj kod kao krajnji rezultat daje takođe objekat *myObject* sa potpuno istim osobinama i vrednostima osobina kao i prethodni primer. Razlika postoji. Sada je *myObject* instanca klase *persona*, a funkcija *persona* naziva se konstruktorom klase *person*. Objekat se instancira pozivanjem ključne reči **new** i konstruktora. Ključna reč **this** u konstruktoru referencira na objekat koji se kreira. Sledeći primer dodaje metod klasi *person*:

```
function showPerson () {
        var myString = this.ime + “, “ + this.godine + “, “ + this.grad;
        trace (myString);
}
person.prototype.show = showPerson;
myObject.show ();
//  “Mira, 27, Kragujevac”
```

Sve funkcije poseduju osobinu **prototype** koja se automatski kreira pri definiciji funkcije. Pri kreiranju novog objekta konstruktorom, sve osobine i metode konstruktorske osobine prototype postaju osobine i metode osobine **__proto__** novokreiranog objekta. Kada se referencira neka osobina ili metoda objekta, ActionScript traži postoji li takva osobina ili metoda, ako ne postoji, traži se u okviru osobina i metoda njegovog **__proto__** objekta (i dalje **__proto__.__proto__**, ...). Ovo je nasleđivanje u ActionScriptu. Uobičajena praksa je dodeljivanje metoda osobini prototype konstruktora, kao što i stoji u navedenom primeru.

ActionScript već dolazi sa dobrim skupom predefinisanih objekata.

Postoji skup ugrađenih objekata preuzetih iz ECMA specifikacije:

- **Object** – najopštija klasa objekata
- **Array** – niz, sa metodama manipulaciju nizom, dodavanje i uzimanje elementa niza sa početka ili kraja, sortiranje itd, osobina length je dužina niza...
- **String** – metode za manipulaciju stringovima, podniz, konkatenacija...
- **Boolean**
- **Number** – konstante, najmanja i najveća vrednost, beskonačnost...
- **Math** – aritmetičke i trigonometrijske funkcije, matematičke konstante...
- **Date** – datum i vreme

Pored ovih, ugrađeni su i bitni objekti za Flashovo okruženje:

- **MovieClip** – osobine od značaja za vizuelnu reprezentaciju objekta, širina, visina, horizontalna i vertikalna pozicija, vidljivost, providnost, rotacija...; metode za kontrolu osobina, za kretanje po vremenskoj osi, kreiranje i uništavanje instanci, prevlačenje objekata...
- **Selection** – kontrola tekstualnih polja
- **Mouse** – skrivanje i pokazivanje pokazivača miša
- **Key** – konstante specijalnih tastera tastature, kodovi karaktera...
- **Color** – transformacije boje MovieClip objekata...
- **Sound** – kontrola zvuka i zvučnih efekata
- **XML** – učitavanje, slanje, parsiranje, izgradnja XML dokumenata...
- **XMLSocket** – stalna veza sa serverom, uspostavljanje i prekid veze...

5.8. Događaji

Programski elementi, akcije, u Flashovom filmu mogu se nalaziti na više mesta. Kako se film odvija po vremenskoj osi podeljenoj na frejmove, delovi koda mogu biti dodeljeni frejmovima. Taj kod izvršava se kada se film nađe na određenom frejmu. Pored toga, instance simbola tipa dugme takođe se mogu programirati za izvođenje određenih akcija pri pojavi nekog događaja, poput pritiska na dugme. Instance simbola tipa filmski klip takođe mogu reagovati na neke događaje ako im se pridruže potrebne akcije. Uz to, simbol tipa filmski klip je takođe film po svojoj definiciji, poseduje svoju vremensku osu, a može sadržavati i druge instance simbola tipa dugme ili filmski klip, pa se cela priča može ponoviti u kontekstu definicije simbola. Ovo daje mogućnost velike zbrke i raštrkavanja koda na sve moguće strane, što s druge strane onemogućava korektno praćenje toka filma, nalaženje grešaka, i slično. Stoga je preporuka držanje koda na manje mesta, pogotovo definicije promenljivih, funkcija i objekata, recimo u prvom frejmu filma ili definicije simbola, dok se za obradu događaja može samo navesti poziv već definisane funkcije ili metoda nekog objekta.

5.8.1. Događaji instanci simbola tipa dugme

Kod koji se pridružuje instanci simbola tipa dugme nalazi se u hendleru on:

```
on (mouseEvent) {
    // linije koda
}
```

Za jedno instancu može se postaviti više hendlera, zavisno od događaja *mouseEvent*. Jedan hendler može opsluživati više događaja razdvojenih zarezom u zaglavlju hendlera. Kod pridružen hendleru izvršava se kada se desi neki od događaja navedenih u zaglavlju hendlera. Sledi opis događaja koje prepoznaje objekat tipa button:

- **press** – dugme miša je pritisnuto dok je pokazivač iznad dugmeta
- **release** – dugme miša je otpušteno dok je pokazivač iznad dugmeta
- **releaseOutside** – dugme miša je otpušteno dok je pokazivač van dugmeta
- **rollOver** – pokazivač miša kreće se iznad dugmeta
- **rollOut** – pokazivač miša je napustio površinu dugmeta
- **dragOver** – dok je pokazivač iznad dugmeta, dugme miša je pritisnuto, pokazivač pomeren van površine dugmeta i potom vraćen iznad
- **dragOut** – dok je pokazivač iznad dugmeta, dugme miša je pritisnuto i potom pokazivač pomeren van površine dugmeta
- **keyPress** (“key”) – taster key je pritisnut

5.8.2. Događaji instanci simbola tipa filmski klip

Instanca simbola tipa filmski klip takođe reaguje na događaje.

```
onClipEvent (movieEvent) {
    // linije koda
}
```

Ovde je *movieEvent* neki od sledećih događaja:

- **load** – akcija se inicira instanciranjem objekta njegovom pojavom u vremenskoj osi
- **unload** – akcija se inicira u prvom frejmu nakon uklanjanja objekta sa vremenske ose, a izvodi se pre bilo koje akcije vezane za posmatrani frejm
- **enterFrame** – akcija se inicira pri svakom frejmu, a izvodi se nakon svih akcija vezanih za posmatrani frejm
- **mouseMove** – akcija se inicira pri svakom pomeranju miša
- **mouseDown** – akcija se inicira pritiskom na levo dugme miša
- **mouseUp** – akcija se inicira otpuštanjem levog dugmeta miša
- **keyDown** – akcija se inicira pritiskom na neki taster
- **keyUp** – akcija se inicira otpuštanjem tastera
- **data** – akcija se inicira prijemom podataka pri učitavanju

6. PROGRAMIRANJE ACTIONSCRIPTA: LOADVARS KLASA

6.1. Slanje i prihvatanje podataka sa LoadVars klasom

LoadVars klasa omogućava slanje i učitavanje promenljivih sa njihovim vrednostima u Flash aplikaciju. Postoje velike sličnosti između XML i LoadVars objekta. Razlika je u osnovi to da XML objekt radi sa XML formatiranim podacima, a LoadVars objekat koristi imenovane varijable sa njihovim vrednostima.

6.2. Kreiranje LoadVars objekta

Prva stvar u korišćenju LoadVars objekta je da se ista i kreira. Način za kreiranje je jednostavno korišćenje funkcije konstruktora u liniji koda. Funkcija ne zahteva parametre:

```
var lvPodaci:LoadVars = new LoadVars();
```

Dodavanje potrebnih varijabli radi se potpuno automatizovano navođenjem imena promenljive kao atributa promenljive LoadVars i zadavanjem vrednosti npr.

```
lvPodaci.omiljenaBoja = “crvena”;
lvPodaci.omiljenaPesma = “Dobro jutro”;
```

6.3. Učitavanje podataka

Učitavanje podataka korišćenjem LoadVars objekta sastoji se u definisanju objekta, a zatim u pozivanju metoda **load()** kome je prosleđen parametar URL-a sa podacima. Kao i kod XML objekta proces učitavanja je asinhron, tako da treba definisati metod **onLoad()** koji će biti pozvan kada se podaci učitaju. Podaci koji se prosleđuju moraju biti u URL enkodiranom formatu, ime=vrednost formatu. Drugim rečima, svako ime i vrednost moraju biti odvojeni znakom jednakosti (=), a svaki par ime=vrednost biće odvojeno (**&**). Svaki ne alfa-numerički karakter biće preskočen. Npr:

```
ime=Haklberi%20Fin&autor=Mark%20Tven
```

Podaci mugu biti snimljeni u tekst fajl ili generisani serverskom skriptom. U svakom slučaju moraju se vratiti u odgovarajućem formatu.

```
var lvData:LoadVars = new LoadVars();
lvData.load(“bookResult.txt”);
lvData.onLoad = function(bSuccess:Boolean):Void {
if(bSuccess){
    trace(this.ime);
    trace(this.autor);
    }
};
```

U zavisnosti od sadržaja *bookResult.txt* program će izbaciti odgovarajuće podatke.

6.4. Slanje podataka

Isto kao i XML objekat podaci će biti poslati iz LoadVars objekta sa jednom od dve postojeće metode **send()** ili **senaAndLoad()**. Oba metoda šalju sve unete atribute LoadVars objekta na URL adresu navedenu kao parametar. U obe metode, varijable se šalju POST metodom dok nije drugačije definisano.

Metod **send()** zahteva jedan parametar i to URL na koji će se varijable poslati. Na primer LaodVars objekt šalje varijable na server:

```
var lvPodaci:LoadVars = new LoadVars();
lvPodaci.omiljenaBoja = “crvena”;
lvPodaci.omiljenaPesma = “Dobro jutro”;
lvPodaci.omiljeniAuto = “Yugo Tempo 1.1”;
lvPodaci.send(“http://www.myserver.com/cgi-bin/surveyResults.cgi”);
```

Postoji mogućnost za navođenje opcionih parametara pri korišćenju **send()** metoda. Prvi opcioni je ciljni prozor browsera za rezultat send i HTTP send metoda. Ako ciljni prozor nije naveden ili je ostavljen na vrednosti null,neće biti prikazanih rezultata. Ako skripta na serveru očekuje podatke poslane GET metodom, moguće je navesti GET za drugi opcioni parametar.

```
lvPodaci.send(“http://www.myserver.com/cgi-bin/surveyResults.cgi”,“_blank”, “GET”);
```

Metod **sendAndLoad()** funkcioniše na sličan način, mada zahteva dva parametra URL na koji će biti poslane varijable i LoadVars objekat kome će biti prosleđen rezultat. Kao i kod **send()** metoda moguće je proslediti opcioni parametar za HTTP metod slanja.

```
var lvPodaci:LoadVars = new LoadVars();
var lvPrimljeno:LoadVars = new LoadVars();
lvPodaci.omiljenaBoja = “crvena”;
lvPodaci.omiljenaPesma = “Dobro jutro”;
lvPodaci.omiljeniAuto = “Yugo Tempo 1.1”;
lvPodaci.sendAndLoad(“http://www.myserver.com/cgi-bin/surveyResults.cgi”, lvPrimljeno);

lvPrimljeno.onLoad = function(bSuccess:Boolean):Void {
if(bSuccess){
    trace(this.responseMessage);
    trace(this.processTime);
    }
};
```

7. PROGRAMIRANJE ACTIONSCRIPTA: XML KLASA

Objekat XML omogućava komunikaciju kroz slanje i prijem poruka definisanim u jeziku XML, opšte prihvaćenom standardu za razmenu podataka. Ulazno-izlazne metode objekta su **send()** (slanje XML poruke na određenu adresu), **load()** (učitavanje sa specifirane adrese) i **sendAndLoad()** (slanje poruke na adresu i učitavanje sa adrese u neki XML objekat). Za indikaciju prijema poruke koristi se logička osobina objekta **loaded** (samo za čitanje – postavlja se na true po kompletiranju prijema). Takođe se može koristiti hendler **onLoad**, koji program poziva po prijemu podataka, za definisanje akcija u zavisnosti od uspeha prijema poruke (hendleru se dodeljuje potrebna funkcija).

7.1. Korišćenje XML objekta

Svaki put kada se radi sa XML podacima u ActionSriptu moramo da inicijalizujemo XML objekat, i to korišćenjem poziva XML konstruktor funkcije u novoj liniji:

```
var xmlData:XML = new XML();
```

Ova linija kreira prazan XML objekat, koji može da se koristi da učita podatke iz eksternog izvora ili se može koristiti za eksportovanje ličnog XML dokumenta iz Fleša. Zapravo, ako se planira kreiranje sopstvenog XML dokumenta, u najvećem broju slučajeva lakše je prvo kreirati string koji sadrži vrednost XML dokumenta i onda ga proslediti kao parametar XML konstruktoru:

```
var sXml:String = "<?xml version=\"1.0\"?><test>Pozdrav</test>";
var xmlData:XML = new XML(sXml);
```

Alternativa ovoj tehnici je popunjavanje XML objekta korišćenjem **parseXML()** metode. Ova metoda uzima jedan string objekat koji biva parsovana u XML objekat. Bilo koje postojeće stablo u objektu je izgubljeno i zamenjeno sa XML stablom koje nastaje parsovanjem zadatog stringa.

```
var sXml:String = "<nesto>Pocetak</nesto>";
var xmlData:XML = new XML();
xmlData.parseXML(sXml);
```

7.2. Učitavanje XML-a

Najbolja karakteristika XML-a je da je nezavistan od platforme pri razmeni podataka. Iako XML može striktno da se koristi unutar Flash filma, jedna od namena XML objekta je mogućnost za učitavanje XML podataka iz spoljnih izvora, kao slanje XML podataka ka spoljnim izvorima. Često se bitni podaci prenose između servera pomoću XML-a. Učitavanje XML-a radi odlično uz pomoć parsovanja XML stringa unutar ActionScripta. Ako se izlazi van okvira Flash aplikacije prolazi se kroz malo izmenjen proces učitavanja u film. Za učitavanje je zadužena **load()** metoda koja učitava XML dokument iz eksternog izvora u Flash film i parsuje ga do kreiranja XML stabla za naš XML objekat.

XML **load()** metod uzima jedan parametar URL kao string imena XML dokumenta koji učitavamo. Sledeći kod prikazuje XML objekat koji učitava dokument koji se zove *data.xml* iz istog direktorijuma odakle se film emituje.

```
var xmlVal:XML = new XML();
xmlVal.load("data.xml");
```

Referencni string može biti i potpuni URL. Treba zapamtiti da flesh film radi na računaru klienta i nema pristup serverovom local file sistemu. Tačnije film koji se učitava sa *www.pmfkg.edu*, **load()** metoda bi izgledala:

```
xmlVal.load("http:// www.pmfkg.edu /data.xml");
```

Ovde dolazimo do bitnog mesta pri radu sa učitavanjem XML-a u Flash. ActionScript dozvoljava učitavanje XML dokumenta samo sa istog domena sa kojega i film radi. Ovo je iz sigurnosnog razloga. Spoljni podaci ne moraju da budu iz statičnog XML fajla sa XML ekstenzjom, već mogu da nastanu kao dinamički pozivanjem CGI, JSP ili neke druge vrste skripte ili programa koji generiše dinamički XML.

7.3. Primanje učitanih XML podataka

XML se učitava asinhrono, tj. **load()** metoda inicira učitavanje, ali ne čeka da se ceo XML dokument učita pre nego pređe na sledeću liniju koda. Drugim rečima Flash aplikacija neće čekati par sekundi ili nekoliko minuta u zavisnosti od količine podataka i konekcije. Umesto toga XML podaci se učitavaju u pozadini, a ostatak aplikacije nastavlja dalje. Zato moramo da se pobrinemo za podatke kada konačno budu učitani. sa **onLoad()** metodom XML objekta. Sve što treba definisati je metod **onLoad()** XML objekta koji prosleđuje parametar **boolean** tipa koji nam je indikator kada su podaci uspešno učitani i prosleđeni. Sledeći primer prikazuje prost primer **onLoad()** metoda.

```
xmlVal.onLoad = function(bUspeh:Boolean):Void {
    if (bUspeh){
        trace(this.toString());
    }
    else{
        trace("Dokument se nije ucitao ili parsovao.");
    }
};
```

U predhodnom primeru funkcija proverava da li je *bUspeh* jednako **true**. Ako jeste dokument je pravilno učitan i podaci su prosleđeni trace funkciji.

7.4. Rad sa "belim razmacima"

Konvertovanje i korišćenje belina, tabova i novih linija može da napravi čitav niz neočekivanih nodova pri radu sa XML dokumentom koji se učitava. Na sreću možemo koristiti **ignoreWhite** parametar za rešavanje ovog problema. Svi XML objekti podrazumevano ne ignorišu bele razmake. Ako se ovaj parametar postavi na logičku vrednost tačno – **true**, beli razmaci biće odbačeni prilikom parsovanja XML-a. Ovaj parametar mora biti postavljen na istinitu vrednost pre bilo kakvog učitavanja ili parsovanja.

```
xmlVal.ignoreWhitespace=true;
```

7.5. Čitanje nodova XML stabla

Podaci su konačno učitani i XML stablo je prosleđeno. XML dokument je organizovan kao grane na stablu. Na vrhu hijerarhije je uvek stablo, pa glavne grane, pa listovi na granama. Ova struktura se poredi sa odnosima dete i roditelj. Osobine XML objekta koje su za čitanje su **firstChild**, **lastChild**, **nextSibling**, **previousSibling**, and **childNodes**. Kada se kreće po strukturi

XML-a najkorisnije je primiti ili prebaciti u promenljive nodove dece od roditelja. Na primer ako se radi na sledećem primeru:

```
<book>
    <autor>Homer</autor>
    <naslov>Odiseja</naslov>
</book>
```

sigurno ćemo imati potrebu da pročitamo imena i vrednosti nodova dece. Jedan način da se ovo uradi je korišćenje **firstChild** i **lastChild** osobina, a onda koračanje kroz nodove dece pristupom sa **nextSibling** ili **previousSibling** osobinama nodova dece. Sibling elementi bi jednostavno bili elementi sa istom hijerarhijom u stablu i ugnježđeni sa istim roditeljem.

Prvo dete ili **firstChild** osobina XML objekta koja je parsovana iz stringa ili dokumenta je početni nod. Sledeći primer daje objašnjenje pomenutih osobina

```
var sXml:String = "<knjiga><autor>Homer</autor><naslov>Odiseja</naslov></knjiga>";
var xmlVal:XML = new XML(sXml);
var xnPocetniNod:XMLNode = xmlVal.firstChild;          // <knjiga />
var xnAutorTag:XMLNode = xnPocetniNod.firstChild;      // <autor />
var xnAutor:XMLNode = xnAutorTag.firstChild;           // Homer
var xnNaslovTag:XMLNode = xnAutorTag.nextSibling;      // <naslov />
var xnTitle:XMLNode = xnNaslovTag.firstChild;          // Odiseja
```

Pristup tagu *xnNaslovTag* je moguć direktno iz roditeljskog noda *knjiga*, kao **lastChild** osobina roditelja.

```
xnNaslovTag = xnPocetniNod.lastChild;
```

U tom slučaju referenciramo nod autor relativno *xnNaslovTag* objektu kao **previousSibling** osobinu *xnNaslovTag* objekta.

```
xnAutorTag = xnNaslovTag.previousSibling;
```

Treba primetiti iz predhodnog da su tekst nodovi tretirani kao deca elemenata koji im daju kontekst. U ovom slučaju tekst nodovi sa vrednostima Homer i Odiseja su **firstChild** (i na slučan način **lastChild**) osobina od *xnAutorTag* and *xnNaslovTag* objekata, resprektivno.

Osobina **childNodes** je kolekcija ili niz referenci ka svim objektima dece XML objekta. Ovoje alternativa korišćenju **firstChild** i **nextSibling** pristupu XML stablu. Sledeći kod pokazuje korišćenje **childNodes** osobine kreiranja niza XML node objekata, od kojih svaki počinje detetom roditelja.

```
var sXml:String = "<?xml version=\"1.0\"?><modeli><auto><marka>Honda</marka>"
    sXml = sXml + "<model>Accord</model><godina>2001</godina></auto></modeli>"
    var xmlVal:XML = new XML(sXml);
    var xnPocetniNod:XMLNode = xmlVal.firstChild;          // <modeli>
    var xnAuto:XMLNode = PocetniNod.firstChild;            // <auto>
    var aDete:Array = xnAuto.childNodes;                   // deca noda <auto>
    for(var i:Number = 0; i < aDete.length; i++){
        trace(aDete[i].toString());
    }
```

Ovaj primer daje sledeći rezultat:

```
<marka>Honda</marka>
<model>Accord</model>
<godina>2007</godina>
```

Svaki element niza kreiran je kao childNodes osobina je **XMLNode** objekata i potrebno je pristupiti **firstChild** osobini za prikazivanje vrednosti tekst noda. Nova petlja bi izgledala ovako:

```
for(var i:Number = 0; i < aDete.length; i++){
    trace(aDete[i].firstChild.toString());
}
```

Rezultat bi bio sledeći:

```
Honda
Accord
2007
```

Kada se radi sa elementom možem jednostavno da proverimo da li je element dete pre nego uradimo nešto sa njim. To radimo na sledeći način:

```
if(xmlVal.firstChild.hasChildNodes()){
    trace("Pocetni element ima dete!");
}
```

7.6. Čitanje osobina elemenata

U mnogim XML dokumentima, imaćemo elemente sa atributima. Atribut je osobina blizu taga npr.

```
<crayon boja="plava"/>
```

U primeru sa childNodes osobinom imali smo XML sa sledećom strukturom:

```
<?xml version=\"1.0\"?>
<modeli>
    <auto>
        <marka>Honda</marka>
        <model>Accord</model>
        <godina>2007</godina>
    </auto>
</modeli>
```

Ipak lako možemo strukturu da definišemo i sa atributima:

```
<auto marka="Honda" model="Accord" godina="2007"/>
```

Pristup kolekciji nodova dece ostvaruje se **childNodes** osobinom koja kreira asocijativni niz.

```
var sXml:String = "<?xml version=\"1.0\"?><modeli>";
sXml = sXml + "<auto marka=\"Honda\" model=\"Accord\" godina=\"2007\"/></modeli>";

var xmlVal:XML = new XML(sXml);
var xnPocetak:XMLNode = xmlVal.firstChild;          // <modeli>
var xnAutoTag:XMLNode = xnPocetak.firstChild;       // <auto>
var oAttribs:Object = xnAutoTag.attributes;         // atributi od <auto>
```

```
for(var sAttrib:String in oAttribs){
    trace(sAttrib + ":" + oAttribs[sAttrib]);
}
```

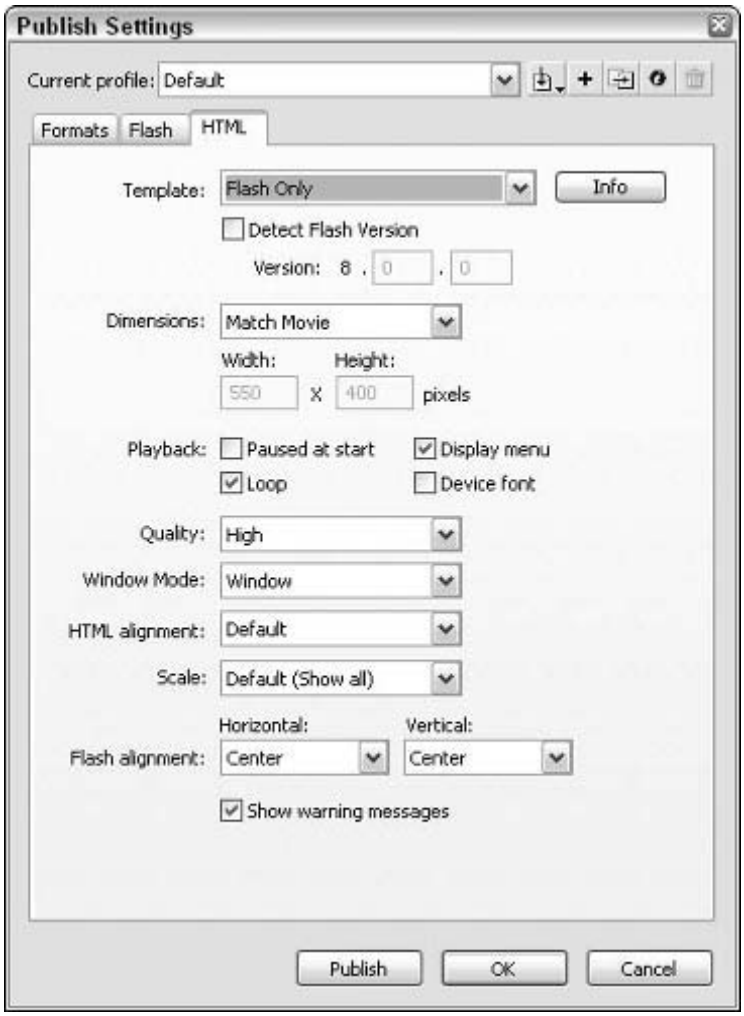
Izlaz za predhodni primer je sledeći:

```
marka:Honda
model:Accord
godina:2007
```

8. ODNOS FLASHA I HTML-A

8.1. Publikovanje Flasha u HTML

Kada objavljujemo ili pablikujemo aplikaciju iz Flasha imamo opciju da napravimo takođe i HTML stranu. Biranjem iz menija **File – Publish Settings** i čekiranjem HTML opcije na format tabu (na slici) . Ova opcija skraćuje vreme ali nije uvek sve ono što je potrebno da bi se fleš integrisao u HTML.



Slika 21 – HTML podešavanje pri eksportovanju iz Flasha

Ako se koristi **Flash Only** template za publikovajne HTML strane i otvori se kod u nekom editoru, dobiće se nešto poput ovog koda što sledi na sledećoj strani.

Deo koji je napočetku pripada standardno HTML kodiranju:

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”;
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml” xml:lang=”en” lang=”en”>
<head>
   <meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1” />
   <title>publishTest</title>
</head>
<body bgcolor=”#ffffff”>
<!--url’s used in the movie-->
<!--text used in the movie-->
```

Deo koji je naveden u sledećimlinijama je delo Flash čarobnjaka:

```
   <object classid=”clsid:d27cdb6e-ae6d-11cf-96b8-444553540000” ;
      codebase=”http://download.macromedia.com/pub/shockwave/cabs/flash/swflash. ;
      cab#version=7,0,0,0” width=”550” height=”400” id=”movieName” align=”middle”>
      <param name=”allowScriptAccess” value=”sameDomain” />
      <param name=”movie” value=”movieName.swf” />
      <param name=”quality” value=”high” />
      <param name=”bgcolor” value=”#ffffff” />
      <embed src=”movieName.swf” quality=”high” bgcolor=”#ffffff” width=”550” ;
      height=”400” name=”movieName” align=”middle” allowScriptAccess=”sameDomain” ;
      type=”application/x-shockwave-flash” ;
      pluginspage=”http://www.macromedia.com/go/getflashplayer” />
   </object>
```

Ovome bi sledilo zatvaranje HTML tagova:

```
</body>
</html>
```

Za sveku promenu koja se napravi tagu <**param**> ili atributu taga <**object**>, potrebno je promeniti odgovarajući <**embed**> tag. Oba taga <**object**> i <**embed**> omogućavaju širinu i visinu SWF fajla koji će biti skaliran da stane u ove dimenzije. Tag ID je bitan zbog JavaScript kontrole i potrebna je provera njegovog postojanja. **Align** osobina bitna je zbog poravnanja filma u browseru. Osnovni parametar je **movie**, jer govori browseru gde da nađe SWF fajl. Kvalitet i pozadinska boja mogu da se izostave i da se koriste vrednosti navedene u SWF fajlu, kao i da se vrednosti iz filma prevaziđu postavljanjem novih vrednosti. Ako se value od **allowScriptAccess** postavi na “**never**” biće sprečen bilo kakva skript komunikacija.

Opcije za skaliranje najviše utiču kako će film biti prikazan u HTML-u. Osnovna postavka je “**Show all**" koja znači da se film povećava ili smanjuje da stane u navedene dimenzije sa time da zadržava odnos visine u širini, takozvani aspekt. Treba voditi računa da se od navedenih dimenzija oduzima po 25 pixela koji otpadaju na border levo i desno u odnosu na film.

Ako ne želimo da se pojavi border opciju **scale** postavljamo na opciju "**No border**". Ovde se takođe film skalira bez promene aspekta pa treba voditi računa o veličini filma i dizajnu u koji se smešta. Pošto se aspekt ne menja ukoliko nam je film različitog aspekta može doći do odsecanja po visini ili širini pri prikazivanju.

Donekle se ovo prevazilazi opcijom "**Exact fit**" koja skalira film sa promenom aspekta prikazivanja.

Poslednja opcija je "**No scale**" koja ne dozvoljava skaliranje postojećeg filma, ali omogućava uz poravnanje gore i levo da se film pravilno odseče a da ostane iste veličine kao u Flashu.

8.2. Prosleđivanje podataka za inicijalizaciju Flash-a iz HTML-a

Ovo prosleđivanje je moguće uz pomoć **FlashVars** atributa <**object**> i <**embed**> taga. Ovo se događa samo kada se Flash film učitava. Ovo znači da čak iako se promenljive za **FlashVars** promene korišćenjem JavaScript ili VBScript, nove izmenjene vrednosti neće biti poslane u Flash film. **FlashVars** se u osnovi koristi za inicijalizovanje Flash filma sa određenim vrednostima. Ovo je veoma korisna tehnika, potrebna u različitim scenarijima, uključujući sledeće:

Određivanje vrednosti promenljivih Flash filma koje mogu da se menjaju relativno često umesto da se kodiraju unutar samog filma.

Inicijalizovanje filma sa vrednostima primljenim iz baze ili serverskog izvora je moguće. Na primer ako se koristi ColdFusion ili PHP stranica kojoj je dodat Flash film. Njoj možemo proslediti rezultate upita nad bazom i Flash će ih učitati. Ova tehnika nije dobra za prosleđivanje celog niza rezultata ili velikog broja kompleksnih podataka. Da bi koristili **FlashVars** dodaćemo **FlashVars** <**param**> objekat ugnježden u <**object**> tag, i dodaćemo **FlashVars** osobinu sa <**embed**> tagom.

Vrednost za **FlashVars** treba da bude u URL enkodiranom formatu. Primer:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" ;
    codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash. ;
    cab#version=7,0,0,0" width="550" height="400" id="usingFlashVars" ;
    align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="movie" value="usingFlashVars.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<param name="FlashVars" value="sLabel=circle&sDescription=a%20circle" />
<embed src="usingFlashVars.swf" quality="high" bgcolor="#ffffff" ;
    width="550" height="400" name="usingFlashVars" align="middle" ;
    allowScriptAccess="sameDomain" type="application/x-shockwave-flash" ;
    pluginspage="http://www.macromedia.com/go/getflashplayer" ;
    FlashVars="sLabel=circle&sDescription=a%20circle" />
</object>
```

Predhodni kod prosleđuje dve varijable Flash filmu: *sLabel* sa vrednošću "circle" i *sDescription* sa vrednošću "circle". Ako prosleđujemo vrednost koja nije string , potrebno je koristiti ActionScript za konvertovanje vrednosti iz string tipa u odgovarajući tip. Npr.

```
nQuantity = parseInt(nQuantity);
```

Ako se radi o prosleđivanju varijable tipa **boolean** npr. *blsVisible* upotrebite sledeći kod

```
blsVisible = (blsVisible == "true") ? true : false;
```

8.3. Detektovanje verzije Flash plejera korisnika

Od prve verzije Flasha jedan od bitnih problema je bio koju verziju plejera poseduje korisnik na svom računaru. Pristup ovom problemu srećom od verzije 8 je jako olakšan automatskom procedurom dodavanja koda za detekciju plejera prilikom publikovanja filma.

Na slici ispod je prikazan postupak aktiviranja ovog postupka i izbora verzije plejera sa kojom je potrebno raditi.

Slika 22 – Čekiranje detektovanja verzije Flash plejera pri pravljenju HTML koda

9. ZAKLJUČAK

Činjenica da predstavljeni primeri imaju veličinu korisničkog fajla (.swf filma) od tridesetak KB, a da u potpunosti definišu ponašanje aplikacije i realizuju kompletan korisnički interfejs, nedvosmisleno govori u prilog tvrdnji da Flash filmovi jesu pravo rešenje u svetu Interneta.

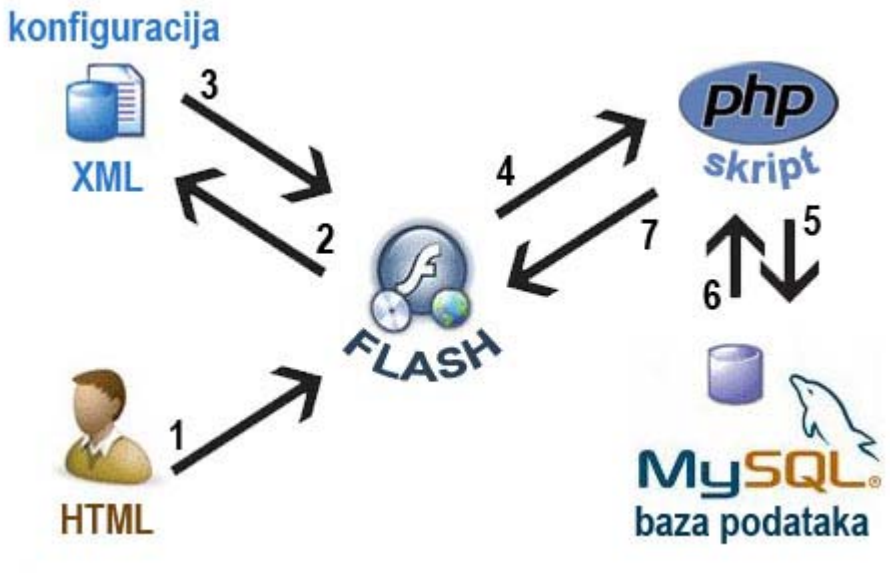
Vektorska grafika i animacija Flasha i jezik ActionScript svojom raznovrsnošću omogućavaju rešavanje širokog spektra web problematike. Od kreiranja navigacionih rešenja u okviru HTML stranica, animiranih reklamnih sličica, do izrade kompletnih web-prezentacija u Flashu i direktnu razmenu podataka sa web serverom i njihovu dalju obradu.

Svojom rasprostranjenošću Flash već danas jeste standard, a sutra je već počelo, jer tu je Adobe Flex i Adobe AIR, koji definitivno zaokružuju priču koju je započeo Flash 5 uvodeći uvodeći pojam Rich Internet Application. Veza Flasha i spoljnih konekcija dovela je do razvoja web aplikacija koje imaju sve karakteristike desktop aplikacija, a prenose se internetom, platformski su nezavisne sa mogućnošću sigurnog korišćenja i automatskog unapređivanja.

II - Implementacija

1. ZADATAK ZA IMPLEMENTACIJU

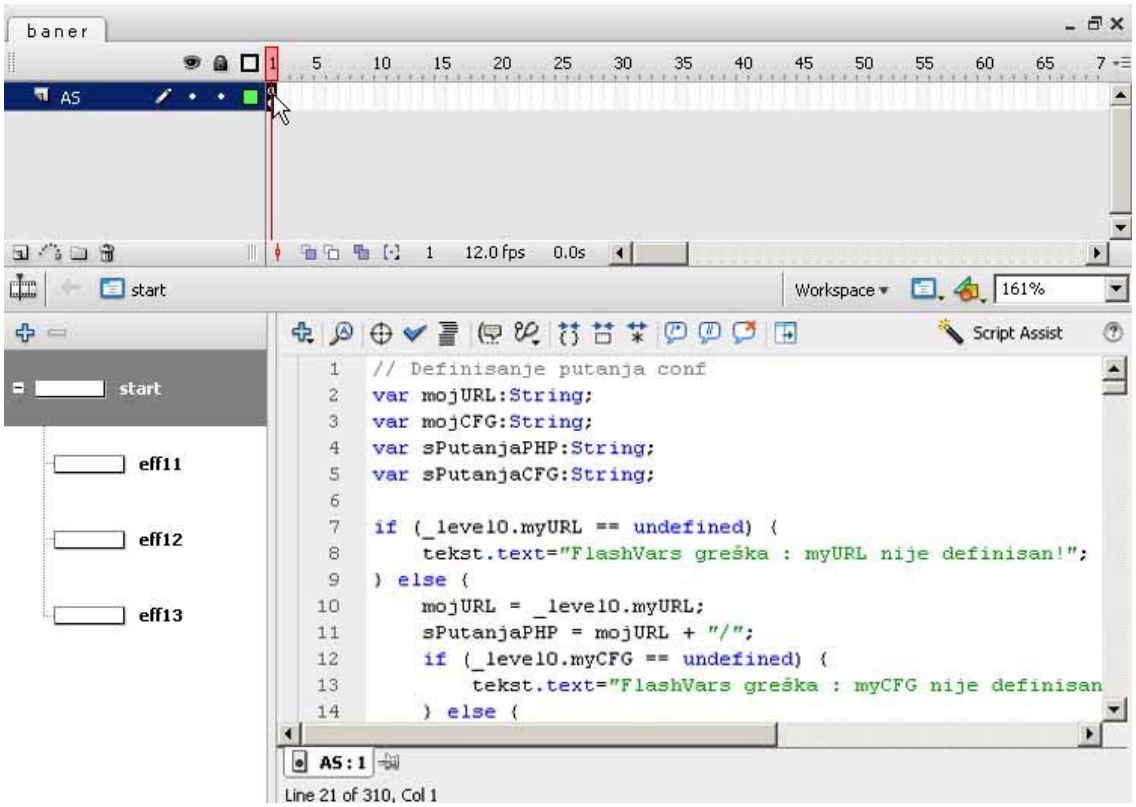
U okviru diplomskog potrebno je programiranjem implementirati konekciju Flasha sa XML, PHP, MySQL i to u obliku banera za prikazivanje vesti na HTML sajtu. Baner bio bi konfigurisan XML konfiguracionim fajlom, koji sadrži potrebne podatke za komunikaciju sa MySQL serverom. Flash baner uz pomoć PHP skripte ostvari komunikaciju sa MySQL serverom i dobijene podatke prosledi nazad. Uz pomoć XML konfiguracionog fajla konfiguriše se i ispis u Flashu.



Slika 23 – Šema toka podataka od HTML zahteva do prikaza podataka iz Flasha fajla

2. FLASH I ACTIONSCRIPT

Baner u Flashu je napravljen kao **Screen aplikacija**, osnovni skrin se zove s*tart*, a dodatni skrinovi su slojevi za tri uneta efekta *eff11*, *eff12* i *eff13*. Za osnovno rešenje ispisa bez efekata potreban je samo kod 2.1. koji se unosi u prvi frejm skrina *Start* i to u lejer AS vremenske ose kao na slici.



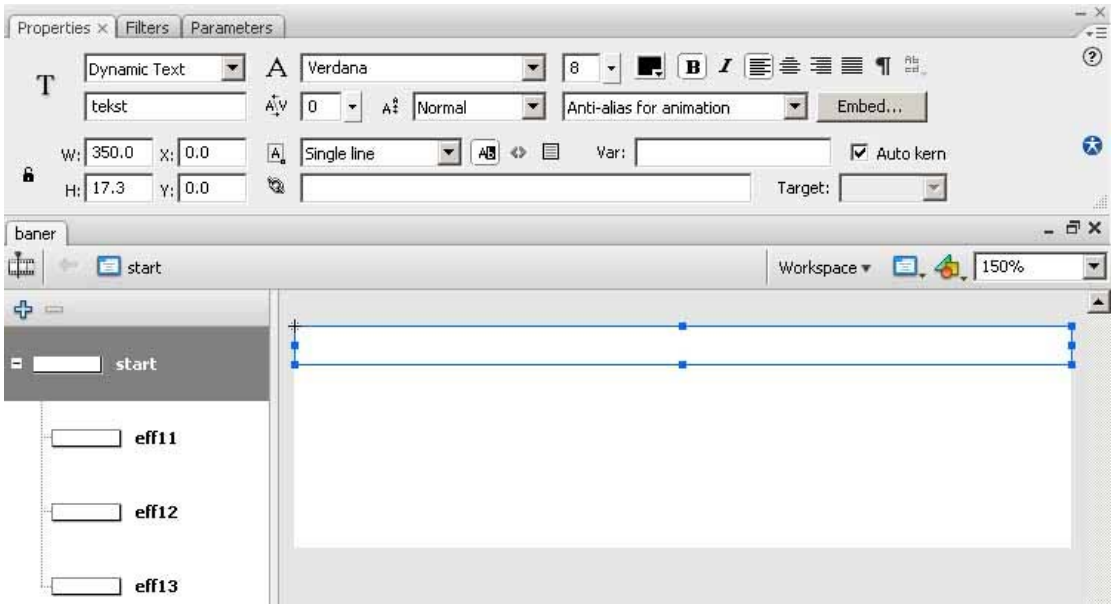
Slika 24 – U prvom frejmu skrina start nalazi se osnovni kod 2.1.

Na skrinu *Start* nalazi se još i dinamičko tekst polje sa imenom *tekst* koje ime namenu statusne linije koja obaveštava korisnika banera o lošim parametrima pri postavljanju istog u HTML, tačnije o nedefinisanom sajtu ili URL-u na kome je baner postavljen i o nedefinisanom imenu konfiguracionog XML fajla.

Tok ActionScripta koda je sledeći: prvo se proveri postojanje URL adrese sa koje se kod aktivira, a zatim se proveri da li je definisana XML konfiguraciona datoteka i da li ista postoji. Ovo se radi proverom definisanosti vredosti promenljivih *_level0.myURL* i *_level0.myCFG* koje se prosleđuju iz HTML koda.

Ukoliko je sve u redu počinje se sa kreiranjem XML objekta *myXML* i sa učitavanjem XML podataka iz konfiguracionog fajla. Za parsiranje XML-a zadužena je metoda *myXML.onLoad()*.

Odmah po inicijalizaciji učitavanja konfiguracionog fajla generiše se paralelno tekst polje *my_txt* uz pomoć skripte. Po završetku učitavanja konfiguracionog fajla, porede se parametri dimenzija Flash banera definisanih HTML kodom i XML konfiguracionim fajlom, pa se radi njihovo usklađivanje radi pravilnog iscrtavanja na ekranu.



Slika 25 – Tekst polje *tekst* je statusna linija u programu

Startuje se prva provera novih vesti funkcijom *proveriVesti()*, a zatim se inicijalizuje tajmer za proveru novih vesti u definisanom intervalu *myTimerSQL* definisanog u XML fajlu.

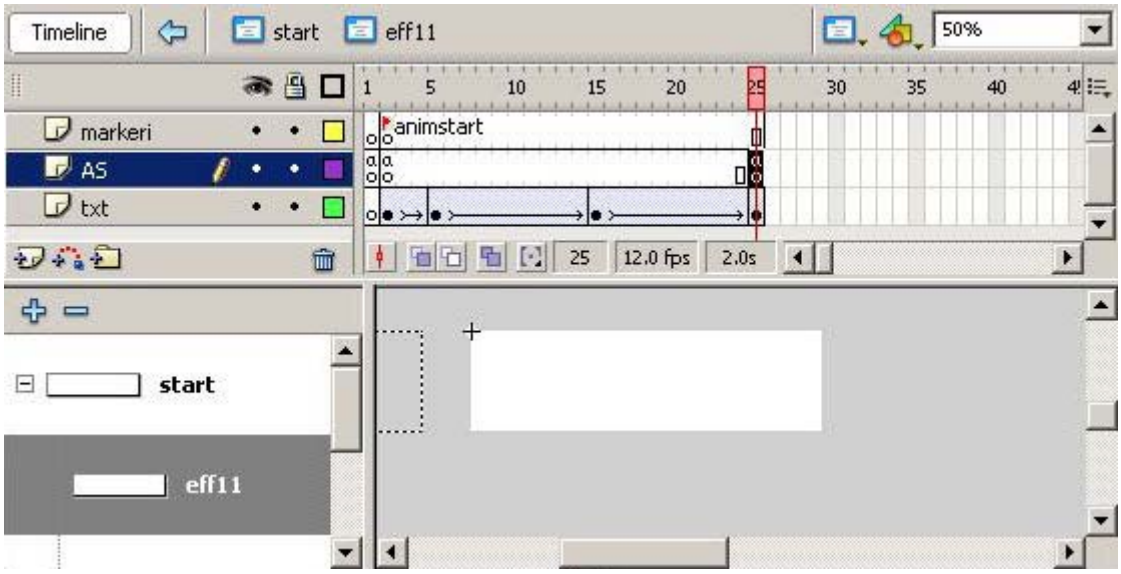
Slanjem podataka definisanoj PHP skripti, počinje deo u kome se PHP konektuje na MySQL bazu i vraća podatke. Objekat za slanje podataka je *sendLV*, sa metodom *sendAndLoad()*,a objekat za povratak podataka u Flash je *loadLV*.

Ukoliko su podaci ispravno vraćeni, metoda *loadLV.onLoad()* pruzima stringove sa vestima i smešta ih u niz za ispis *aNizIspis*. Zatim se inicijalizuje tajmer vesti – *nTajmerVesti()*, koji pokreće ispisivanje vesti u datom intervalu. XML fajlom je opisan i format ispisa teksta i on se po učitavanju konfiguracije dodeljuje objektu *my_fmt* tipa TextFormat, a on se pri ispisu dodeljuje tekst polju *my_txt* kao format ispisa.

Da bi pokazali još neke mogućnosti Flasha, dodali smo u implementaciju skrinove *eff11*, *eff12* i *eff13* i proširili konfiguraciju za mogućnost ispisa uz pomoć efekta. Skrinovi imaju u sebi posebne vremenske linije u kojima je na različit način animiran MovieClip *animtekst* koji je instanca MovieClipa *samoTekst* koji u sebi sadrži dinamičko tekst polje *my_txt*. Ovo je izvedeno zbog posebnih filtera koje je moguće zadati i animirati samo na instancama MovieClipa. Neki od njih su npr. **Blur**, **Glow**, **Drop Shadow**, **Bevel**, **Change Color**, kao i mogućnosti kontrole transparencije – objekat se vidi ili ne vidi na ekranu. Još jedna prednost ovakvog uređenja je i to da je promenljivom *my_txt* definisan tekst koji je ispisuje u bilo kojem od efekata, pa je moguća promena vrednosti iz samog koda, kao i primena Text format objekta.

Svi efekti u vremenskim linijama imaju po tri lejera tačnije *txt*, *AS* i *markeri*. U lejeru markeri nalazi se marker i to najčešće *animstart* potreban za kontrolu iz koda sa naredbama **GotoAndPlay().** U sloju *AS* nalazi se ActionScript kod u različitim frejmovima potreban za pravilno izvršavanje efekta. Vremenske linije za sva tri efekta napravljene su tako da je pozornica prazna u prvom frejmu i da se u AS sloju uvek u prvom frejmu nalazi komanda *stop()*. Startovanje se najčešće izvodi skokom na markere *animstart* iz koda.

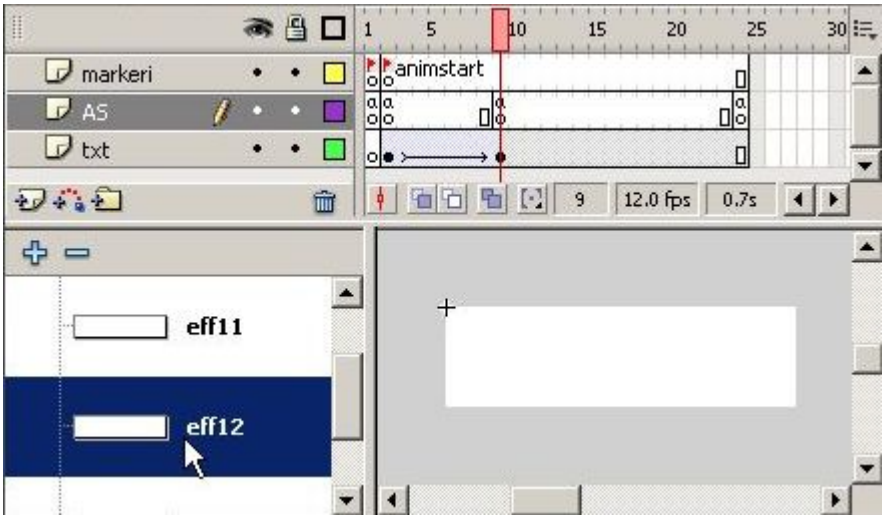
Efekat 11 za dve sekunde koliko traje, uskroluje tekst sa leve strane u pozornicu i ponovo ga skroluje na levo, primer je korišćenja **MotinTwin** tehnike animacije na instanci *animtekst* u frejmovima 2,5,15 i 25.



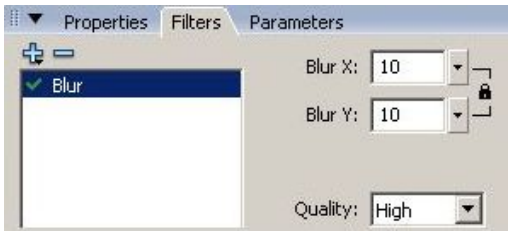
Slika 26 – Vremenska linija eff11 u 25 frejmu

U delu 2.2. je kod za effekat 11 i to po frejmovima. U frejmu 1 je zaustavljeno izvršavanje efekta. Kad se aktivira dati efekat - film je startovan iz skrina *start* skokom na marker *animstart* i potom se *my_txt* iz efekta popuni vrednošću trenutnog ispisa. Ukoliko je vrednost fonta u konfiguraciji velika za pravilan ispis u statusnoj liniji se ispiše potrebna pravilna vrednost za vertikalnu vrednosti fleš banera, a da se zadovolji veličina slova. U frejmu 25 se proverava da li se u međuvremenu promenila vest za ispis, pa ako jeste biva promenjena i efekat je uvek priveden kraju. Na isi način bi bio opisan i kod u 2.3. i 2.4. za efekte *eff12* i *eff13*.

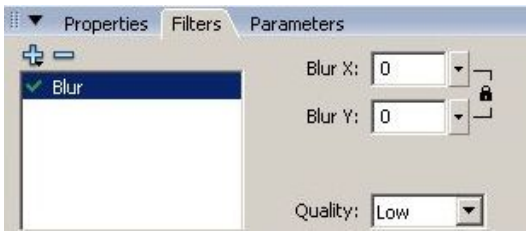
Efekat 12 je primer korišćenja **MotionTwin** tehnike na filteru blur u frejmovima 2 i 9, kao i promene veličine u istim frejmovima. Tačnije instanca *animtekst* je u drugom frejmu 1 procenat svoje veličine, a u devetom je 100 procenata svoje veličine i uz to filter **Blur** je u istim frejmovima respektivno 10 i 0. Animacija **MotionTwin** deluje na oba promenjena paramatra tako što tekst raste i iz stanja neraspoznavanja postaje skroz jasan.



Slika 27 – Vremenska linija eff12 u frejmu 9

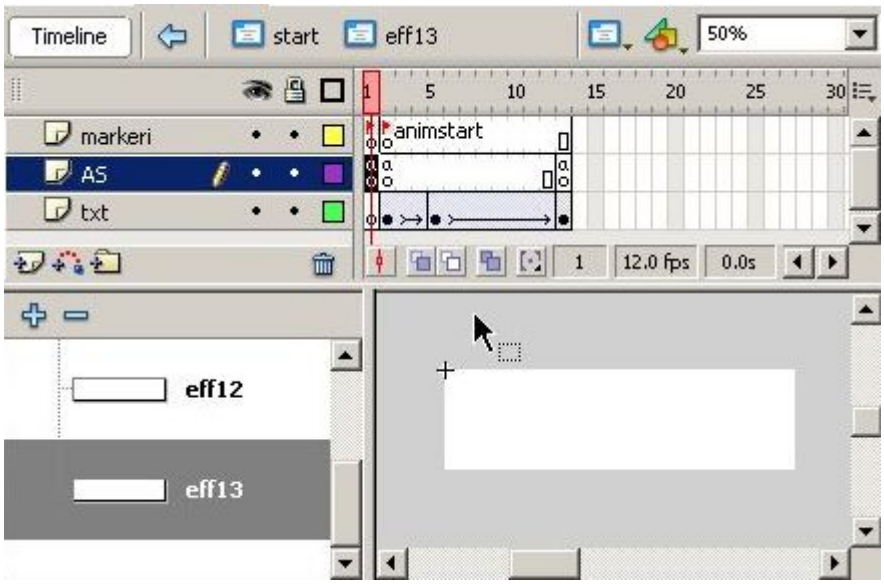


Slika 28 – Filter Blur u frejmu br.2

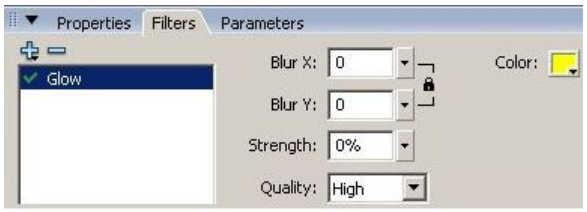


Slika 29 – Filter Blur u frejmu br.9

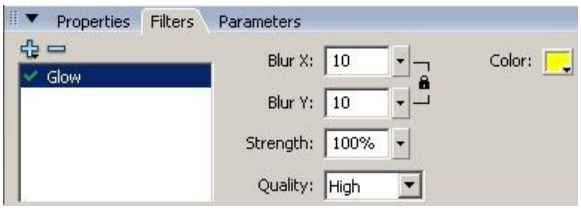
Efekat 13 je primer korišćenja **MotionTwin** tehnike na filteru **Glow** u frejmovima 2, 5 i 14. Tačnije vrednosti **Glow** filtera na instanci *animtekst* su u frejmovima promene respektivno 0, 10, 0 pa je zahvaljujući tajmingu animacija **MotionTwin**-om izgleda kao da slova bljesnu, a zatim bljesak nestane.



Slika 30 – Vremenska linja eff13 u frejmu 1



Slika 31 – Glow filter u frejmu 2 i 13



Slika 32 – Glow filter u frejmu 5

U slučaju da se u konfiguraciji izabere neki od efekata aktiviranje istih se postiže iz skripte *ispisVesti()* pozivanjem funkcije *ispisiEfekat()*.

2.1. ActionScript - screen start: frame 1

```
// Definisanje url-a i putanje conf fajla
var mojURL:String;
var mojCFG:String;
var sPutanjaPHP:String;
var sPutanjaCFG:String;

if (_level0.myURL == undefined) {
    tekst.text="FlashVars greška : myURL nije definisan!";
} else {
    mojURL = _level0.myURL;
    sPutanjaPHP = mojURL + "/";
    if (_level0.myCFG == undefined) {
        tekst.text="FlashVars greška : myCFG nije definisan!";
    } else {
        mojCFG = _level0.myCFG;
        sPutanjaCFG=sPutanjaPHP + mojCFG;
    }
}

tekst.text="";

// Promenljive oko Intervala
var nIntervalSQL:Number;
var nInterval:Number;
var boolSQL:Boolean=Boolean(false);

// Load i Send objekti
var loadLV:LoadVars = new LoadVars();
var sendLV:LoadVars = new LoadVars();

// Promenljive oko ispisa
var nMyX:Number;
var nMyY:Number;
var nMyTx:Number;
var nMyTy:Number;

var nEfekat:Number;
var nTajmerVesti:Number;
var nRbIspis:Number=0;
var my_fmt:TextFormat = new TextFormat();
var trenutanIspis:String;
var menjajIspis:String;

// Promenljive oko konfiguracije
var myXML:XML = new XML();
myXML.ignoreWhite = true;
myXML.load(sPutanjaCFG);

function proveriVesti():Void {
    boolSQL=true;
    sendLV.sendAndLoad(sPutanjaPHP, loadLV, "POST");
}
```

```
function ispisEfekat():Void {
    switch(nEfekat) {
        case 11:
            if(eff11._currentframe==1) {
                trenutanIspis = menjajIspis;
                eff11.gotoAndPlay('animstart');
            }
        break;
        case 12:
            if(eff12._currentframe==1) {
                trenutanIspis = menjajIspis;
                eff12.gotoAndPlay('animstart');
            }
        break;
        case 13:
            if(eff13._currentframe==1) {
                trenutanIspis = menjajIspis;
                eff13.gotoAndPlay('animstart');
            }
        break;
    }
}

function ispisiVest(ispis:TextField, txtNiz:Array, maxbroj:Number):Void {
    if(not boolSQL) {
        if (nRbIspis<maxbroj) {
            menjajIspis=txtNiz[nRbIspis];
            if(nEfekat==0) {
                ispis.text = menjajIspis;
                ispis.setTextFormat(my_fmt);
                if(ispis.textHeight>nMyTy) {
                    tekst.text="Y: "+ispis.textHeight;
                } else {
                    tekst.text="";
                }
            } else {
                ispisEfekat();
            }
            nRbIspis++;
        } else {
            nRbIspis=0;
        }
    }
    else {
    //ispis.text = "Provera podataka u toku";
    }
}

myXML.onData = function (src:String) {
    if (src == undefined) {
        tekst.text="FlashVars greška : Nepostoji: " + mojCFG;
        this.onLoad(false);
    } else {
        this.parseXML(src);
        this.loaded = true;
        this.onLoad(true);
    }
}
```

```
loadLV.onLoad= function(success:Boolean){
    if (success) {
        var aNizIspis:Array = new Array();
        for (var prop in loadLV) {
                aNizIspis[prop]=loadLV[prop];
        }
        if(aNizIspis.length>0) {
            if(boolSQL) {
                clearInterval(nInterval);
            }
            nRbIspis=0;
            nInterval = setInterval(ispisiVest, nTajmerVesti, my_txt, aNizIspis, aNizIspis.length);
            boolSQL=false;
        } else {
            trace("Konekcija sa serverom neuspesna ili nema podataka");
        }
    }
}

myXML.onLoad = function() {
    myConfig=this.firstChild;
    podConfig = myConfig.childNodes;

    // Razvrstavanje XML-a po granama
    for (var i = 0; i < podConfig.length; i++) {
        switch (podConfig[i].nodeName) {
            case "confBaze":
                myConfigBaze=podConfig[i];
            break;
            case "confUpita":
                myConfigUpita=podConfig[i];
            break;
            case "confIspisa":
                myConfigIspisa=podConfig[i];
            break;
        }
    }

    // Obrada konfiguracije baze
    deca = myConfigBaze.childNodes;
    for (var i = 0; i < deca.length; i++) {
        switch (deca[i].nodeName) {
            case "host":
                var myDBhost:String=String(deca[i].firstChild.nodeValue);
            break;
            case "user":
                var myDBuser:String=String(deca[i].firstChild.nodeValue);
            break;
            case "pass":
                var myDBpass:String=String(deca[i].firstChild.nodeValue);
            break;
            case "name":
                var myDBTableName:String=String(deca[i].firstChild.nodeValue);
            break;
        }
    }
```

```
// Obrada konfiguracije upita
deca = myConfigUpita.childNodes;
for (var i = 0; i < deca.length; i++) {
    switch (deca[i].nodeName) {
        case "upitSQL":
            var myUpitSQL:String=String(deca[i].firstChild.nodeValue);
        break;
        case "upitPolje":
            var myUpitPolje:String=String(deca[i].firstChild.nodeValue);
        break;
        case "timerSQL":
            var myTimerSQL:Number=Number(deca[i].firstChild.nodeValue)*1000;
        break;
        case "timerVesti":
            nTajmerVesti=Number(deca[i].firstChild.nodeValue)*1000;
        break;
        case "imePHP":
            var slmePHP:String=String(deca[i].firstChild.nodeValue);
            sPutanjaPHP=sPutanjaPHP+slmePHP;
        break;
    }
}

// Obrada konfiguracije ispisa
deca = myConfigIspisa.childNodes;
    for (var i = 0; i < deca.length; i++) {
        switch (deca[i].nodeName) {
            case "xTbox":
                nMyX=Number(deca[i].firstChild.nodeValue);
            break;
            case "yTbox":
                nMyY=Number(deca[i].firstChild.nodeValue);
            break;
            case "sirinaTbox":
                nMyTx=Number(deca[i].firstChild.nodeValue);
            break;
            case "visinaTbox":
                nMyTy=Number(deca[i].firstChild.nodeValue);
            break;
            case "font":
                my_fmt.font=String(deca[i].firstChild.nodeValue);
            break;
            case "size":
                my_fmt.size=Number(deca[i].firstChild.nodeValue);
            break;
            case "color":
                my_fmt.color=Number(deca[i].firstChild.nodeValue);
            break;
            case "bold":
                my_fmt.bold=Boolean(Number(deca[i].firstChild.nodeValue));
            break;
            case "italic":
                my_fmt.italic=Boolean(Number(deca[i].firstChild.nodeValue));
            break;
            case "underline":
                my_fmt.underline=Boolean(Number(deca[i].firstChild.nodeValue));
            break;
            case "align":
```

```
                    my_fmt.align=String(deca[i].firstChild.nodeValue);
                break;
                case "efekat":
                    nEfekat=Number(deca[i].firstChild.nodeValue);
                break;
            }
        }

    // Uskladjivanje dimenzija ispisa i dimenzija u HTML-u
    if(nMyX+nMyTx>Stage.width) {
        nMyX=0;
        nMyTx=Stage.width;
    }
    if(nMyY+nMyTy>Stage.height) {
        nMyY=0;
        nMyTy=Stage.height;
    }
    if(nEfekat==0) {
        my_txt._x=nMyX;
        my_txt._y=nMyY;
        my_txt._width=nMyTx;
        my_txt._height=nMyTy;
    }

    // Inicijalizovanje sendLV promenljivama
    sendLV.dbhost=myDBhost;
    sendLV.dbuser=myDBuser;
    sendLV.dbpass=myDBpass;
    sendLV.dbname=myDBTableName;
    sendLV.upit=myUpitSQL;
    sendLV.polje=myUpitPolje;

    if(myTimerSQL<10000) {
        myTimerSQL=10000; }
    proveriVesti();
    nIntervalSQL = setInterval(proveriVesti, myTimerSQL);
}
// end config.XML onLoad.function

// Ovo se prvo iscrtava pa se ucitavaju podaci
this.createTextField("my_txt", 0, 0, 0, 0, 0);
my_txt.multiline = false;
my_txt.wordWrap = true;
```

2.2. ActionScript – screen eff11: frame 1, frame 2, frame 25

```
// ActionScript - eff11: frame 1
stop();

// ActionScript - eff11: frame 2
animtxt.my_txt.wordWrap=true;
animtxt.my_txt._width = _parent.nMyTx;
animtxt.my_txt._height = _parent.nMyTy;
animtxt.my_txt.text = _parent.trenutanIspis;
animtxt.my_txt.setTextFormat(_parent.my_fmt);
if (animtxt.my_txt.textHeight>_parent.nMyTy) {
    _parent.tekst.text = "Y: "+ animtxt.my_txt.textHeight;
} else {
    _parent.tekst.text = "";
}

// ActionScript - eff11: frame 25
if(_parent.menjajIspis == _parent.trenutanIspis) {
    gotoAndPlay('animstart');
} else {
    _parent.trenutanIspis=_parent.menjajIspis;
    gotoAndPlay('animstart');
}
```

2.3. ActionScript – screen eff12: frame 1, frame 2, frame 9, frame 24

```
// ActionScript - eff11: frame 1
stop();

// ActionScript - eff11: frame 2
animtxt.my_txt._width = _parent.nMyTx;
animtxt.my_txt._height = _parent.nMyTy;
animtxt.my_txt.text = _parent.trenutanIspis;
animtxt.my_txt.setTextFormat(_parent.my_fmt);
animtxt.my_txt.wordWrap=true;

// ActionScript - eff11: frame 9
animtxt.my_txt.wordWrap=true;
animtxt.my_txt._width=_parent.nMyTx;
animtxt.my_txt._height=_parent.nMyTy;
animtxt.my_txt.text = _parent.trenutanIspis;
animtxt.my_txt.setTextFormat(_parent.my_fmt);
if (animtxt.my_txt.textHeight>_parent.nMyTy) {
    _parent.tekst.text = "Y: "+ animtxt.my_txt.textHeight;
} else {
    _parent.tekst.text = "";
}

// ActionScript - eff11: frame 24
if(_parent.menjajIspis == _parent.trenutanIspis) {
    gotoAndPlay('animstart');
} else {
    _parent.trenutanIspis=_parent.menjajIspis;
    gotoAndPlay('animstart');
}
```

2.4. ActionScript – screen eff13: frame 1, frame 2, frame 13

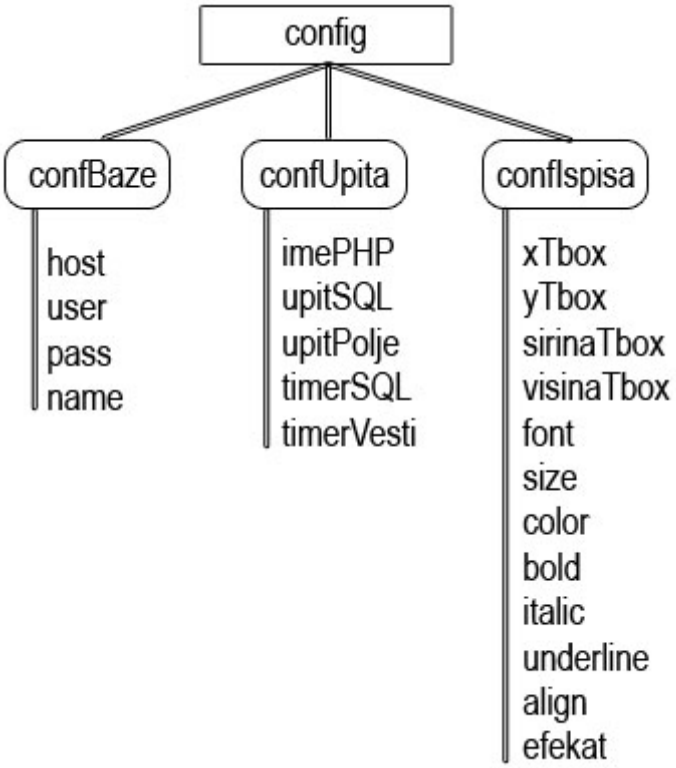
```
// ActionScript - eff11: frame 1
stop();

// ActionScript - eff11: frame 2
animtxt.my_txt._width = _parent.nMyTx;
animtxt.my_txt._height = _parent.nMyTy;
animtxt.my_txt.text = _parent.trenutanIspis;
animtxt.my_txt.setTextFormat(_parent.my_fmt);
animtxt.my_txt.wordWrap=true;
if (animtxt.my_txt.textHeight>_parent.nMyTy) {
    _parent.tekst.text = "Y: "+ animtxt.my_txt.textHeight;
} else {
    _parent.tekst.text = "";
}

// ActionScript - eff11: frame 13
if(_parent.menjajIspis == _parent.trenutanIspis) {
    gotoAndPlay('animstart');
} else
{
    _parent.trenutanIspis=_parent.menjajIspis;
    gotoAndPlay('animstart');
}
```

3. XML

Konfiguracija ima tri dela i to konfiguraciju baze, konfiguraciju upita, konfiguraciju ispisa.



Slika 33 – Grafički prikaz XML konfiguracije

U konfiguraciji baze unosi se ime hosta baze, ime korisnika koji pristupa bazi, šifra korisnika baze i ime baze u kojoj se nalaze podaci za pretragu.

Konfiguracija upita nad bazom sadrži ime php skripte kojoj se prosleđuju parametri, SQL upit, ime kolone koja sadrži odgovor, kao i dve numeričke vrednosti u sekundama koje predstavljaju tajmer za proveru novih vesti u bazi i tajmer za promenu trenutno aktivne vesti pri ispisu. Tajmer provere novih vesti na serveru ne može biti manji od 10 sekundi.

Konfiguracija ispisa sadrži podatke o tekst polja i njegovog položaja u baneru, vrstu fonta, veličinu fonta, boju, stilove bold, italic, podvučeno i poravnanje, kao i da li se koristi efekat. Tekst polje je definisano sa četiri vrednosti, po dve kordinate početne gornje leve i krajnje donje desne dimenzije polja. Prvom koordinatom je određeno i mesto pojavljivanja u vidljivoj pozornici flash banera. Format fonta određen je imenom pisma koje, ako je dostupno biva korišćeno, a ako nije koristi se osnovna definisana vrednost formata fonta Flasha.

Boja je predstavljena kombinacijom osnovnih boja crvena, zelena i plava i to zapisom 0xNNNNNN, gde su parovi NN reprezentacija decimalnih vrednosti od 0 do 255 heksadecimalnom zapisu. N je zato numerička vrednosti od 0 do 9 i slovna od A do F. Stilovi *bold, italic, underline* su predstavljeni numeričkim vrednostima 0 ili 1, tačnije isključeno ili uključeno. Poravnanje teksta u baneru definisano je unosom službenih reči ActionScripta *left*, *right*, *center*.

Parametar efekat sadrži 0 kada je ispis bez efekata, a 11,12 ili 13 za dodatne efekte.

3.1. XML kod – config.xml

```
<config>
    <confBaze>
        <!-->HOST sajt sa MYSQL bazom - default: localhost<-->
        <!-->USER korisnik koji pristupa bazi - dogovoriti sa adminom MYSQL<-->
        <!-->PASS password za pristup bazi - dogovoriti sa adminom MYSQL<-->
        <!-->NAME ime tabele sa podacima za ispis - default: pmfstepen<-->
        <host>localhost</host>
        <user>korisnik</user>
        <pass>sifra</pass>
        <name>pmfstepen</name>
    </confBaze>

    <confUpita>
        <!-->IMEPHP ime php skripte koja treba da konektuje bazu- default: phpbaza.php<-->
        <!-->UPITSQL SQL upit koji vraca podatke iz baze
            default: SELECT * FROM `vesti` ORDER BY `vesti`.`vest_datum` DESC LIMIT 0 , 30<-->
        <!-->UPITPOLJE koju kolonu iz upita proslediti iz mySQL baze - default: vest_opis<-->
        <!-->TIMERSQL tajmer ponovnog slanja upita ka mySQL bazi u sekundama
            default: 30, vrednost ispod 10 sekundi flashom postaje 10.<-->
        <!-->TIMERVESTI tajmer promene ispisa za vise podataka u sekundama - default: 2<-->
        <imePHP>phpbaza.php</imePHP>
        <upitSQL>SELECT * FROM `vesti` ORDER BY `vesti`.`vest_datum` DESC LIMIT 0 ,
            30</upitSQL>
        <upitPolje>vest_opis</upitPolje>
        <timerSQL>30</timerSQL>
        <timerVesti>4</timerVesti>
    </confUpita>

    <confIspisa>
        <!-->XTBOX leva gornja kordinata po X text filda  - default: 0<-->
        <!-->YTBOX leva gornja kordinata po Y text filda  - default: 0<-->
        <!-->SIRINATBOX sirina po X text filda  - default: 350<-->
        <!-->VISINATBOX visina po Y text filda  - default: 100<-->
        <!-->FONT format fonta - default: Arial<-->
        <!-->SIZE velicina fonta - default: 24<-->
        <!-->COLOR boja fonta - default: 0xFF0000 crvena<-->
        <!-->BOLD podebljana slova 1 ili 0 - default: 1<-->
        <!-->ITALIC zakosena slova 1 ili 0 - default: 0<-->
        <!-->UNDERLINE podvucena slova 1 ili 0 - default: 0<-->
        <!-->ALIGN podvucena slova left, center, right - default: center<-->
        <!-->EFEKAT umesto klasicnog ispisa koristi se nesto od 11, 12, 13 - default: 0<-->
        <xTbox>0</xTbox>
        <yTbox>0</yTbox>
        <sirinaTbox>350</sirinaTbox>
        <visinaTbox>100</visinaTbox>
        <font>Arial</font>
        <size>24</size>
        <color>0x326EAA</color>
        <bold>1</bold>
        <italic>0</italic>
        <underline>0</underline>
        <align>center</align>
        <efekat>0</efekat>
    </confIspisa>
</config>
```

4. PHP

PHP skriptu phpbaza.php započećemo prikupljanjem promenljivih koje je Flash baner poslao metodom **POST**. Funkcija **$_POST[]** zadužena je za ovaj posao prosleđivanja konfiguracije php promenljivama. Zatim se otvara konekcija funkcijom **mysql_connect()** prema zadatoj bazi i smešta u promenljivu *$connection*. Funkcijama **mysql_select()** i **mysql_query()** selektovali bi potrebnu bazu i izvršili upit sa potrebnim upitom. Rezultat će biti smešten u promenljivu *$result*.

Zatim bi **while** petljom pristupili odvajanju potrebne kolone funkcijom **mysql_fetch()** iz dobijenog rezultata upita. U istoj petlji bi konfigurisali slanje potrebnih podataka Flash baneru komandom **echo**. Po završetku bi zatvorili otvorenu konekciju.

4.1. PHP kod - phpbaza.php

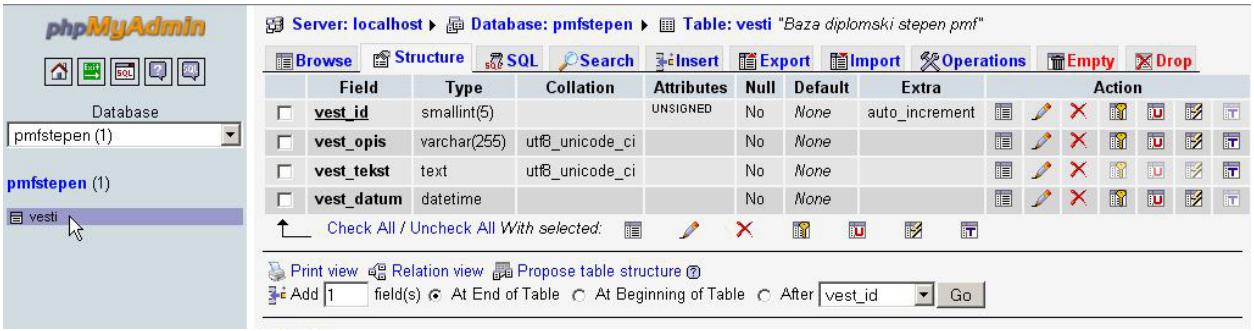
```
<?php
	$db_host=$_POST['dbhost'];
	$db_user=$_POST['dbuser'];
	$db_password=$_POST['dbpass'];
	$db_name=$_POST['dbname'];
	$db_upit=$_POST['upit'];
	$db_polje=$_POST['polje'];

	$connection=@mysql_connect($db_host, $db_user, $db_password) or die ("error connection");
	mysql_select_db($db_name, $connection);

	$result=mysql_query($db_upit, $connection) or die(mysql_error());
	$i=0;
	while ($row = mysql_fetch_object($result))
		{
		echo "&$i=".$row->$db_polje."&";
		$i++;
		}
	mysql_close($connection);
?>
```

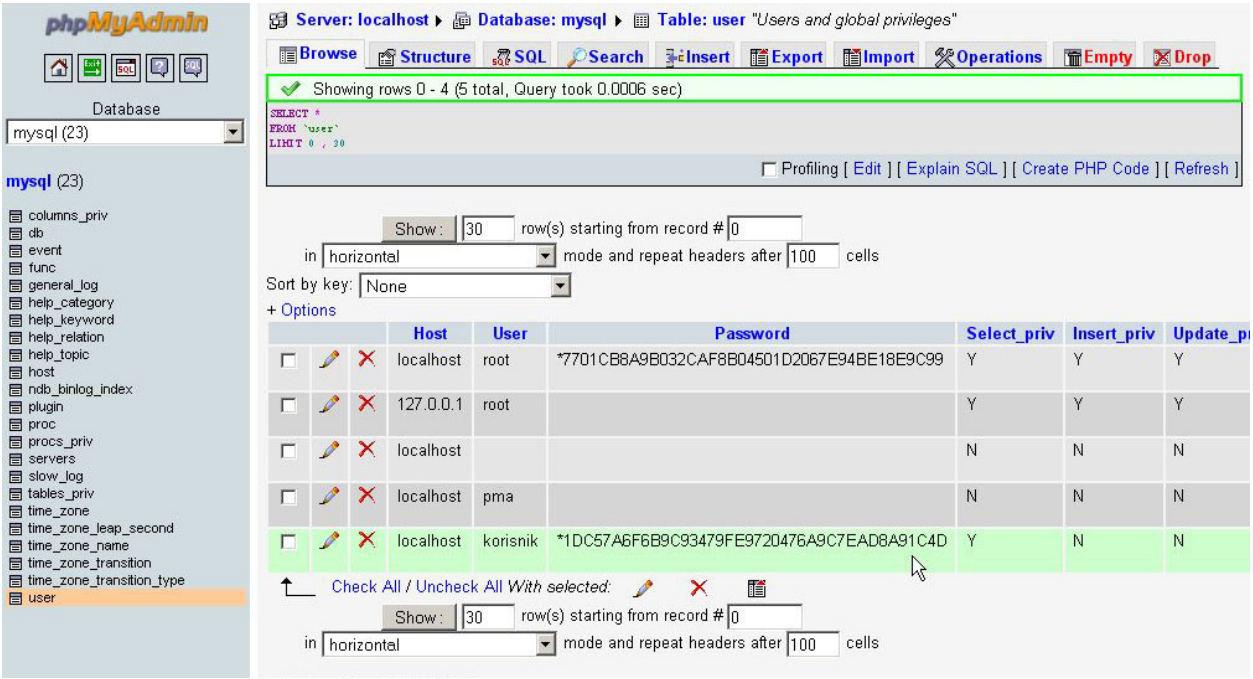
5. MYSQL

Ovaj deo je samo primer baze koja je iskorišćena za proveru rada veze Flasha i PHP-MySQL. Baza kao i njen eksport napravljen je u phpMyAdmin, a testiranje iste je izvršeno na na localhost uz pomoć XAMPlite-a.



Slika 34 – phpMyAdmin prikaz tabele vesti u bazi pmfstepen

Za potrebe testiranja na localhostu dodat je korisnik mysql sa imenom *korisnik* i sa šifrom *sifra* i sa atributima samo za čitanje.



Slika 35 – phpMyAdmin prikaz korsnika mysql baze u tabeli user sa pravima samo za čitanje

Ovo je urađeno zbog mogućnosti direktonog pristupa xml fajlu, i mogućnosti da se vidi šifra korisnika root.

5.1. phpMyAdmin - SQL opis

```
-- phpMyAdmin SQL Dump
-- version 3.1.1
-- Host: localhost
-- Generation Time: Apr 04, 2009 at 05:21 PM
-- Server version: 5.1.30
-- PHP Version: 5.2.8
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
--
-- Database: `pmfstepen`
--
-- --------------------------------------------------------
--
-- Table structure for table `vesti`
--
CREATE TABLE IF NOT EXISTS `vesti` (
  `vest_id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `vest_opis` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `vest_tekst` text COLLATE utf8_unicode_ci NOT NULL,
  `vest_datum` datetime NOT NULL,
  PRIMARY KEY (`vest_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci COMMENT='Baza diplomski stepen pmf'
AUTO_INCREMENT=4 ;
--
-- Dumping data for table `vesti`
--
INSERT INTO `vesti` (`vest_id`, `vest_opis`, `vest_tekst`, `vest_datum`) VALUES
(1, 'Analiza III septembar 2009', 'Polozili su:\r\n1. Ime 1\r\n2. Ime 2\r\n3. Ime 3', '2009-02-24 22:13:03'),
(2, 'Rezultati Racunarstvo III septembar 2009', 'Polozili su:\r\n1. Ime 1\r\n2. Ime 2\r\n3. Ime 3', '2009-02-24 2:13:38'), (3,
'Racunarstvo 2 usmeni', 'Odrzace se sutra 17.05.2009.', '2009-03-12 22:13:38');
```

6. HTML

Za dobijanje najprostijeg HTML-a koji bi sadržao baner korišćen je automatski čarobnjak Flash-a opisan u teorijskom delu 8.1. Deo koji je čarobnjak generisao proširen je za FlashVars promenljive koje sadrže vrednosti *myURL* i *myCFG* i to na dva načina. Prvi definisanjem parametra i njegove vrednosti:

<param name="FlashVars" value="myURL=http://localhost/pmf&myCFG=config.xml" />

drugi definisanjem stringa:

FlashVars="myURL=http://localhost/pmf&myCFG=config.xml"

Ovo je potrebno zbog razlike u pozivanju HTML objekata u Internet Exploreru i Mozilla Firefoxu. U nastavku je dat primer HTML koda za pozivanje banera.

6.1. HTML kod za startovanje banera

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        <title>Proba</title>
    </head>

    <body bgcolor="#000000">
        <!--url's used in the movie-->
        <!--text used in the movie-->
        <!-- saved from url=(0013)about:internet -->
        <object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
                codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab
                #version=8,0,0,0" width="350" height="100" id="baner" align="middle">
            <param name="allowScriptAccess" value="sameDomain" />
            <param name="movie" value="baner.swf" />
            <param name="quality" value="high" />
            <param name="scale" value="noscale" />
            <param name="salign" value="lt" />
            <param name="bgcolor" value="#ffffff" />
            <param name="FlashVars" value="myURL=http://localhost/pmf&myCFG=config.xml" />
            <embed src="baner.swf"
                FlashVars="myURL=http://localhost/pmf&myCFG=config.xml"
                quality="high"
                scale="noscale"
                salign="lt"
                bgcolor="#ffffff"
                width="350"
                height="100"
                name="baner"
                align="middle"
                allowScriptAccess="sameDomain"
                type="application/x-shockwave-flash"
                pluginspage="http://www.macromedia.com/go/getflashplayer"
            />
        </object>

    </body>
</html>
```

7. INSTALACIJA BANERA

Da bi se priloženi baner sa cd-a aktivirao i ugradio u HTML kod, potrebno je preduzeti sledeće korake:

1. Na server prekopirati u isti folder fajlove: *index.htm*, *baner.swf*, *config.xml*, i *phpbaza.php*.
2. Fajl *index.htm* sadrži kod za prikaz osnovne html strane sa ubačenim Flash objektom. Deo sa **<object>** tagom odnosi se na integrisani Flash i kao takav se mora kopirati u drugi html fajl u slučaju integracije u tuđ HMTL. Sve ispravke u tagu moraju se uraditi dva puta jer se jedan deo odnosi na prikaz u Internet Exploreru, a drugi u Mozila pretraživaču.
   a) Ukoliko se promeni ime *baner.swf* u neko drugo potrebno je ispraviti html kod.
   b) Treba uneti vrednosti širine i visine Flash stage-a (**width** i **height**) koji će biti vidljiv prilikom prikaza *baner.swf*. Podrazumevana vrednost je 350 piksela širine i 100 piksela visine.
   c) Parametar **scale** treba da ostane na *noscale* jer se ostale vrednosti se odnose na automatsko skaliranje Flash-a, pa korisnik nema programski uticaj na veličinu.
   d) Parametar **FlashVars** prenosi u flash dva parametra *myURL* i *myCFG*. Prvi je ime hosta ili podfoldera na hostu u kome se baner nalazi. Drugi je ime konfiguracionog xml fajla. Podrazumevana vrednost **FlashVars** promenljive je *"myURL=http://localhost/pmf&myCFG=config.xml"*. Ovde obavezno treba promeniti ime *myURL* parametra u *http://imehosta* ili *http://imehosta/imefoldera*. Ukoliko se ime xml fajla promeni potrebno je promeniti i ovu vrednost.
3. Fajl *config.xml* sadrži tri oblasti podešavanja i to konfigurisanje baze, konfigurisanje upita na bazu i konfiguraciju ispisa. Sva osnovna podešavanja sa osnovnim vrednostima navedena su u komentarima u samom xml fajlu.
   a) Ukoliko se promeni ime skripte koju flash poziva ovde treba izvršiti ispravku imena u delu za konfigurisanje upita u tagu **imePHP**.
   b) U delu za konfigurisanje ispisa **xTbox** i **sirinaTbox** su parametri horizontalne veličine, sa tim da se prvi odnosi na x kordinatu gore levo početka iscrtavanja tekst boksa u flash pozornici, a drugi na njegovu širinu. Zbir ove dve vrednosti mora biti manji od vrednosti navedene u pozivu flasha iz html-a (**width)** vrednost. Ukoliko je veće Flash automatski smanji ove navedene vrednosti tako da tekst boks stane u vrednost navedenu u html tagu. Na isti način se ponašaju i vertikalne vrednosti **yTbox** i **visinaTbox** u odnosu na **height** vrednost iz html-a.

8. ZAVRŠNE NAPOMENE

Na kraju bih naveo neke od napomena vezanih za instalaciju i dalji razvoj, trenutne probleme i moguće revizije trenutne implementacije. Rešenje koje je nastalo u implementaciji je samo jedno od mogućih rešenja i u praksi bi trebalo proveriti koja bi kombinacija rešenja donela najbrže i najoptimalnije rešenje. Na primer PHP može da generiše XML, pa bi Flash mogao da komunicira samo sa XML fajlom. Kao daljnji pravac razvoja naveo bih pozivanje ActionScripta PHP-om, kao i vezu sa AJAX-om kao asinhronom vezom JavaScripta i XML-a.

Flash je vizuelni alat koji treba da ima vezu između veličina Flash scene i njegovog mesta u dizajnu gotove prezentacije. Ovo je i sa programerske strane pravilo problem oko promenljive veličine jer smo mogli samo da isčitavamo parametre scene koji su inače *read-only* i programski ih je nemoguće promeniti u ovoj verziji ActionScripta. Ovaj problem je prevaziđen startovanjem Flash fajla iz HTML-a sa **scale** opcijom i parametrom *noscale*, takođe i levim poravnanjem same scene, a onda se pribeglo otkrivanju i sužavanju vidljivog dela same scene koja je inače klasično 350 puta 100 piksela. Dalje revizije ActionScripta donele su promene samog jezika pa bi i ovaj problem mogao verovatno drugačije da se reši.

Pošto je zadatak bio programiranje komunikacije ActionScripta, dalji deo razvoja bi bio upućen na mogućnost pravljenja animacija teksta u kodu, umesto vizuelnog animiranja, pa zatim povezivanja. Način koji je primenjen u implementaciji samo ukazuje na mogućnosti koje bi ActionScript mogao da ponudi. Deo na koji bi trebalo obratiti pažnju je i problem lokalizacije i domaćih slova, mogućnosti importovanja domaćih fontova u Flash i veza između podataka pristiglih od PHP-a. Na isti način bitna je i lokalizacija XML fajla sa mogućnošću podešavanja odnosno konfigurisanja iz samog Flesh-a i to centriranje, izbor pisma, izbor boje.

U kontekstu predhodne priče o veličini ispisa u razvoju je uvedeno tekst polje koje pomaže proveri toka podataka u obliku statusne linije, koja je izvedena kao fiksno tekstualno polje, mada je i ona mogla da bude izvedena uz pomoć ActionScripta. Nepostojanje config.xml, phpbaza.php ili konfiguracijom drugih imena istih fajlova kao i ne definisanje neke od vrednosti *myURL* i *myCFG* u html fajlu, ispisuje poruku o grešci u statusnoj liniji banera.

Ukoliko je navedena veličina fonta za ispis velika za traženu vrednost teksta unetog iz baze u tekst boks, u gornjem levom uglu biće ispisana potrebna vertiklana vrednost u pikselima. Ovu vrednost treba uneti u html tagove i u veličinu vertikalne vrednosti tekst boksa da bi u datom fontu bio ispisan kompletan tekst.

LITERATURA:

1. Joey Lott and Robert Reinhardt: ACTIONSCRIPT BIBLE, Wiley Publishing, Inc., 2006
2. Steve Webster: FOUNDATION PHP FOR FLASH, Friendso, 2005
3. Dejan Katašić: FLASH I ACTIONSCRIPT, diplomski rad, 2002
4. Reference za XML, PHP, MySQL - *http://www.w3schools.com*
5. Priručnik PHPMySQL - *http://www.adsoglasi.com/html/php_mysql_prirucnik/php_mysql/*