UDC: **004.451.23** INFO M: str. 30-34

KONSTRUKCIJA KORISHIČKOG INTERFEJSA INTEGRACIJOM FLASH KOMPONENTI U .NET WIN APP
USER INTERFACE CONSTRUCTION INTEGRATING FLASH COMPONENTS WITH .NET WIN APP

Saša D. Lazarević, Dušan Marković, Ivan Stamenić

REZIME: U radu je prikazan pristup u rešavanju problema integracije Windows (WinForms) aplikacija sa Adobe Flash u cilju konstrukcije bogatog korisničkog interfejsa. Za implementaciju je, kao tehnološko rešenje, korišćena .NET i Flash tehnologija.
KLJUČNE REČI: Korisnički interfejs, Adobe Flash, Action Script, Flash external API, C#, ActiveX kontrole, COM, .NET, Proxy, RCW, WPF
ABSTRACT: New approach in solving Windows application and Adobe Flash integration, for rich user interface construction. Tehnologies used for implementation are .NET and Flash.
KEY WORDS: User interface, Adobe Flash, Action Script, Flash external API, C#, ActiveX controls, COM, .NET, Proxy, RCW, WPF

1. UVODNA RAZMATRANJA

U savremenoj, računarskoj eri, lakoća korišćenja i vizuelni identitet aplikacije predstavlja imperativ. Zahtevni i sofisticirani korisnici korisnički interfejs smatraju podjednako bitnim, kao i samu funkcionalnost aplikacije. Platforma .NET pruža mnogo mogućnosti u pogledu razvoja softvera i integracije sa rešenjima drugih proizvođača. Kada govorimo o razvoju Windows aplikacija *(WinForms)* ne može, a da se ne primeti da izgled formi bitno zaostaje za sličnim rešenjima drugih proizvođača. Sa druge strane, .NET tehnologija nam omogućava da u okviru formi ubacujemo kontrole koje nisu .NET. To otvara mogućnost razvoja Windows aplikacija koje imaju vizuelna rešenja nekih drugih proizvađača. Jednu od najatraktivnijih mogućnosti, za kreiranje bogatog korisničkog interfejsa, nudi Adobe Flash, koji predstavlja alat za razvoj multimedijalnih sadržaja.

2. PROBLEMI U OSTVARIVANJU INTEGRACIJE

Osnovni problem u integraciji .NET-a i Adobe Flash-a se javlja u komunikaciji između ove dve tehnologije, u cilju njihovog povezivanja radi ostvarivanja funkcionalnosti jedinstvenog sistema.

Kôd koji se izvšava u okviru Microsoft .NET *Common Language Runtime* (CLR) naziva se upravljiv kôd *(managed code)*. Taj kôd ima pristup svim servisima koje CLR obezbeđuje, kao što su međujezična integracija *(cross - language integration)*, podrška za sigurnost i nove verzije *(security and versioning support)* i skupljač smeća *(garbage collection)*. Kôd koji se ne izvršava u okviru CLR-a se naziva neupravljiv kôd *(unmanaged code)*. Sve COM komponente su po definiciji neupravljiv kôd.

Problemi koji se javljaju prilikom integracije upravljivog i neupravljivog kôda u aplikaciji su u tome što neupravljiv kôd nije prepoznat od strane CLR okruženja. Način da se ovaj problem reši je korišćenje *proxy*-a. Jednostavnije rečeno, *proxy* je deo softvera koji prima komande od komponente, modifikuje ih i prosleđuje drugoj komponenti. Određeni tip *proxy*-a koji se koristi prilikom poziva neupravljivog iz upravljivog kôda naziva se *Runtime - Callable Wrapper*, ili skraćeno RCW.

Integracija Adobe Flash-a i .NET programskog jezika C# se svodi na povezivanje upravljivog i neupravljivog kôda. Da bi to bilo omogućeno, Adobe Flash mora biti registrovan u okviru operativnog sistema.

Adobe Flash, predstavlja alat za razvoj multimedijalnih sadržaja. Da bi smo ostvarili željenu funkcionalnost u Adobe Flash aplikacijama, često je potrebno komunicirati sa kodom izvan Adobe Flash Player-a. U tom slučaju moramo pozivati *(invoke) JavaScript* ili drugi programski jezik sa kojim se povezujemo, da bi željenja komunikacija bila ostvarena. Sa druge strane Flash nije samo jedan od najboljih alata za Web vizuelizaciju, već može biti vrlo koristan i u razvoju *runtime* aplikacija, kroz interaktivni korisnički interfejs, animacije i video.

Zaključno sa verzijom 7 Macromedia Flash Player-a, komunikacija na relaciji *ActionScript - JavaScript* se odvijala putem *getURL* ili *fscommand()* metode, dok je *JavaScript* bio u mogućnosti da sa Flash Player-om komunicira putem *SetVariable()*, *callFrame()* i *callLabel()* metoda.

Klasa ***ExternalInterface***, koja je predstavljenja u verziji 8 Flash Player-a, omogućava ravnopravnu komunikaciju sa drugim programskim jezicima. U ovom radu biće obrađena komunikacija sa .NET programskim jezikom C#.

3. POSTOJEĆA TEHNOLOŠKA REŠENJA INTEGRACIJE

ActionScript je skripting jezik baziran na *ECMAScript*. On se primarno koristio za razvoj Web sajtova i softvera (*software*), korišćenjem Adobe Flash Player platforme u formi SWF fajlova sadržanih (*embed*) u Web stranama, ali se takođe koristi u nekim aplikacijama baza podataka (*Alpha Five*). Originalno je razvijen u kompaniji Macromedia, ali ovaj jezik je danas u vlasništvu kompanije Adobe, koja je preuzela Macromedia-u 2005 godine. *ActionScript* je inicijalno dizajniran

za kontrolisanje jednostavnih 2D vektorskih animacija napravljenih u Adobe Flash-u (nekadašnjem Macromedia Flash-u). U kasnijim verzijama je dodata funkcionalnost za kreiranje Web - orijentisanih igara i bogatih Internet aplikacija sa strimingom multimedije (audio i video).

Komunikacija izmedju Flash *ActionScript-a* i aplikacije koja sadrži *Shockwave Flash ActiveX* kontrolu, koristi specifičan XML format za kodiranje poziva funkcijama i prosleđivanje vrednosti. Postoje dva dela u okviru XML formata koji se koristi od strane *external* API-ja. Jedan format se koristi za reprezentovanje poziva funkcija. Drugi format se koristi za reprezentovanje individualnih vrednosti; ovaj format se koristi za reprezentovanje parametara u funkcijama kao i za povratne vrednosti funkcija. XML format za poziv funkcija se koristi za pozive ka i od strane *ActionScript*-a. Prilikom poziva eksterne funkcije iz *ActionScript* kôda, Flash Player prosleđuje XML ka kontejneru, a za poziv iz kontejnera, Flash Player očekuje od aplikacije da prosledi XML string u tom formatu. Sledeći XML odlomak pokazuje primer XML formatiranog poziva funkcije:

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

Naziv korenog čvora (*root node*) je *invoke*. Ima dva atributa: *name* odredjuje naziv funkcije koja se poziva, a *returntype* je uvek XML. Ukoliko funkcija očekuje parametre, *invoke* čvor ima dete čvor (*child node*) koji se naziva *arguments* i čija će deca čvorovi biti vrednosti parametra formatirani korišćenjem formata za svaku pojedinačnu vrednost *(individual value format)* koji je objašnjen u nastavku.

Tabela 1. – *ActionScript klase i XML formati*

ActionScript class/value	C # class/value	Format
null	null	<null/>
Boolean true	bool true	<true/>
Boolean false	bool false	<false/>
String	string	<string>string value</string>
Number, int, uint	single, double, int, uint	<number>27.5</number> <number>-12</number>
Array (elements can be mixed types)	A collection that allows mixed-type elements, such as ArrayList or object[]	<array> <property id="0"> <number>27.5</number> </property> <property id="1"> <string>Hello there!</string> </property> ... </array>
Object	A dictionary with string keys and object values, such as a HashTable with string keys	<object> <property id="name"> <string>John Doe</string> </property> <property id="age"> <string>33</string> </property> ... </object>
Other built-in or custom classes		<null/> or <object></object>

Individualne vrednosti, uključujući parametre i povratne vrednosti funkcija, koriste shemu za formatiranje koja uključuje informaciju o tipu podatka i vrednost podatka. Tabela 1. pokazuje listu *ActionScript* klasa i XML formata koji se koriste za kodiranje vrednosti tog tipa podatka.

Sa druge strane, klasa ***ExternalInterface*** se takođe naziva *External* API i predstavlja novi podsistem koji omogućava laku komunikaciju izmedju *ActionScript-a* i Flash Player kontejnera na HTML strani koja sadrzi *JavaScript* ili sa desktop aplikacijama koje sadrze Flash Player. Klasa *External* API zamenjuje stariju funkciju za interoperabilnost *fscommand()*, svojom robusnijom funkcijonalnošću. Nova svojstva *External* API-ja su:

- Mogu se koristiti sve funkcije drugih programskih jezika, ne samo one koje se mogu koristiti sa *fscommand()* funkcijom.
- Može se proslediti neograničen broj argumenata, sa bilo kojim imenom, više ne postoji limit na prosleđivanje komande i argumenata.
- Mogu se proslediti različiti tipovi podataka (*Boolean*, *Number*, *String*, *Object*), više ne postoji limit na parametre tipa *String*.
- Mogu se primiti vrednosti poziva, i da se ta vrednost odmah vrati *ActionScript*-u, kao povratna vrednost poziva koji je napravljen.

ExternalInterface klasa je dostupna u *ActionScript-u* 1.0 i Flash Player-u 8 i izlaže sledeća svojstva i metode koje su predstavljene u Tabeli 2. i 3.

Tabela 2. – *Svojstva klase ExternalInterface*
Svojstva

Modifikator	Svojstvo	Opis
static	available:Boolean [read-only]	Utvrđuje da li Player u konterjneru nudi eksterni interfejs.

Tabela 3. – *Metode klase ExternalInterface*
Metode

Modifikator	Svojstvo	Opis
static	addCallback:(methodName:String, instance:Object, method:Function) : Boolean	Registruje metodu ActionScript-a kao pozivajuću, tako da može primiti poziv od strane kontejnera.
static	call:(methodName:String, [parameter1:Object]) : Object	Poziva funkciju koju izlaze Flash Player kontejner, prosleđujući 0 ili više argumenata.

3. REALIZACIJA INTEGRACIJE

Početak integracije predstavlja dodavanje specijalne *ocx* kontrole Windows formi. Da bi se dodala takva kontrola u Windows formu, mora se prvo dodati referenca na istu.

Dodavanje se vrši selekcijom *shockwaveflashobject*-a u okviru "COM components" prozora. Nakon toga, kontrola se može koristiti kao i sve ostale .NET kontrole. Važno

je napomenuti da tom prilikom Visual Studio automatski generiše omotač (*wraper*) pomoću koga je omogućena komunikacija između kontejnera (Windows aplikacija) i COM kontrole (Flash.ocx). Najvažnije metode koje se mogu pozvati za referenciranu kontrolu su one koje se odnose na učitavanje flash filma (*movie*) sa extenzijom .swf, kao i njegovim manipulisanjem (*play, stop, rewind*).

Nakon što je kontrola ubačena u formu ona treba da preuzme ulogu korisničkog interfejsa. Korisnički interfejs treba da primi zahteve od korisnika i prosledi ih na obradu, kao i da prikaže rezultate obrade korisniku. Da bi Flash kontrola preuzela ulogu korisničkog interfejsa, mora da prima zahteve iz C# koda i vraća obrađene podatke.

Flash kontrola koja je ubačena u formu može da izaziva događaje. Forma se "pretplatila" na događaj i nakon dešavanja događaja, poziva se metoda:

```
void axShockwaveFlash1_FlashCall(object sender,
AxShockwaveFlashObjects._IShockwaveFlashEvents_
FlashCallEvent e)
```

sender – objekat koji je izazvao događaj

e – argument događaja koji sadrži XML poruku koja je poslata od strane Flash kontrole.

XML poruka koja je poslata od strane Flash kontrole se dobija konvertovanjem argumenta "e" u XML dokument tip.

```
XmlDocument dokument = new XmlDocument();
dokument.LoadXml(e.request);
```

Sada je samo potrebno naći "*invoke*" čvor i na osnovu vrednosti njegovog atributa *name* utvrditi koju metodu ili događaj Flash kontrola želi da pozove ili izazove. Poželjno je korišćenje *Swich* naredbe ili u složenijim slučajevima posebne klase koja je zadužena za obradu zahteva od strane Flash kontrole.

```
XmlNodeList lista = dokument.
GetElementsByTagName("invoke");

switch (lista[0].Attributes["name"].Value.
ToString())
{
case "punidrvo":
ZatrazeniPodsistemi(this, new EventArgs());  break;
…
}
```

U ovom slučaju, flash kontrola je izazvala događaj *ZatrazeniPodsistemi*. Kontroler interfejsa obrađuje događaj i iz baze uzima tražene podatke i dostavlja ih formi pozivajući metodu *DajSvePSiPro()*. Metoda obrađuje dobijene podatke shodno potrebama aplikacije i smešta ih u listu koju je potrebno proslediti Flash kontroli radi prikazivanja.

```
public void DajSvePSiPro(DataTable tabelaPS,
DataTable tabelaPro)
{
 List<object> listaPodsistema = new List<object>();
 foreach (DataRow red in tabelaPS.Rows)
  {
     string flag = "0";
     listaPodsistema.Add(red["Naziv"].ToString());
     listaPodsistema.Add(red["Id"].ToString() +
red.Table.TableName);
     foreach (DataRow red2 in tabelaPro.Rows)
     {
         if (red2["IdPodsistema"].ToString() ==
red["Id"].ToString())
          {
              flag = "1";
          }
      }
      listaPodsistema.Add(flag);
   }
}
```

Podaci za slanje, u ovom slučaju *listaPodsistema*, moraju biti formatirani u skladu sa XML formatom Flash API-ja. Poželjno je pravljenje generičke metode ili klase koja će obavljati zadatak konvertovanja.

Sledi primer metode koja, na osnovu naziva metode i liste objekata generiše XML poruku za poziv *ActionScript* metode. Ta metoda kao parametar prima niz.

```
public string ApiRequestGeneratorNiz(List<object>
lista, string NazivFunkcije)
{
 string request = null;
 int i = 0;
 request = "<invoke name=" + '"' + NazivFunkcije
+ '"' + "         returntype=\"xml\"><arguments><ar
ray>";

foreach (object o in lista)
{
   request += "<property id='" + i + "'><string>"
+ o.ToString() + "</string></property>";
    i++;
 }
          request = request + "</array></argu-
ments></invoke>";

    return request;
 }
```

Metoda *ApiRequestGeneratorNiz()* kao parametre prima listu objekata i naziv *ActionScript* funkcije. Na osnovu toga, generiše XML koji će služiti pozivu bilo koje *ActionScript* funkcije koja kao parametar prima niz. *ActionScript* funkcija *loadtree()*, prima niz kao parametar i prikazuje ga u obliku hijerarhijskog stabla. Funkcija *loadtree()* se poziva na sledeći način:

```
axShockwaveFlash1.CallFunction(ApiRequestGenerat
orNiz(listaPodsistema,"loadtree"));
```

Metoda *CallFunction()*, Flash kontrole, kao parametar prima podatak tipa string, koji sadrži XML zahtev Forme.

ApiRequestGeneratorNiz (listaPodsistema, "loadtree") metoda je kao parametre primila listu (*listaPodsistema*) i string naziv metode ("*loadtree*"), i na osnovu toga napravila ispravan XML zahtev za izvršenje *ActionScript* funkcije *loadtree()*.

Poziv C# metode iz Adobe Flash-a. Da bi se iz C# koda mogli kontrolisati Flash pozivi, forma se mora „pretplatiti" na događaj koji flash kontrola izaziva:

```
this.axShockwaveFlash1.FlashCall += new
AxShockwaveFlashObjects._IShockwaveFlashEvents_
FlashCallEventHandler(this.axShockwaveFlash1_
FlashCall);
```

Na taj način će se uvek kada flash kontrola pošalje poziv C# kodu, izvršiti metoda:

```
   void axShockwaveFlash1_FlashCall(object sender,
AxShockwaveFlashObjects._IShockwaveFlashEvents_
FlashCallEvent e)
```

Argument *e* sadrži stringovni zahtev (*request*), u okviru koga je smešten formiran XML poslat od strane Flash kôda. Radi lakšeg kôdiranja, poželjno je ovaj string konvertovati u XML tip podatka.

```
   XmlDocument dokument = new XmlDocument();
   dokument.LoadXml(e.request);
   XmlNodeList lista = dokument.
GetElementsByTagName("invoke");
```

Nakon toga se pristupanjem atributu *name* određuje naziv metode koja je pozvana, a i korišćenjem naredbe *switch* i poziva željena metoda za ispravno prosleđen zahtev:

```
   switch (lista[0].Attributes["name"].Value.
ToString())
   {
      case "punidrvo": PuniDrvo(); break;
   ……..
   }
```

U Flash aplikaciji, nakon implementacije grafičkog dela aplikacije (ekranske forme), pristupa se kodiranju komponenti i kontrola koje su dodate na formu. U *ActionScript*-u, pre svega, moramo dodati referencu na klasu *ExternalInterface*:

```
   import flash.external.ExternalInterface;
```

U konkretnom primeru, obrađuje se događaj klika na komponentu dugme, čije je ime instance *sistem* (Slika 1.):

```
sistem.onRelease = function():Void {
     ExternalInterface.call("punidrvo", "");
}}
```

Funkcija, koja je instanci *sistem*, dugmeta dodata za događaj po puštanju (*onRelease*), poziva eksternu metodu funkcijom *call()*, koja prima dva parametra tipa stringa. Prvi parametar predstavlja naziv C# metode koja se poziva, a drugi (prazan string), se koristi za prosleđivanje argumenata pozivajućoj metodi.

U drugom delu se dodaje funkcija koja će biti spremna za pozivanje iz eksternog koda, nakon što tražena operacija bude izvršena, kako bi se dobijeni rezultati prikazali na ekranskoj formi. Klasi *ExternalInterface* se u metodi *addCallback()* prosleđuju tri parametra, od kojih je prvi tipa string i predstavlja naziv metode koja je izložena eksternom kodu. Drugi parametar, koji je u konkretnom primeru *null*, je argument koji se prosleđuje pozivajućoj metodi, a treći parametar je naziv *ActionScript* funkcije koja će biti pozvana:

```
   ExternalInterface.addCallback("loadtree", null,
loadtree);
```

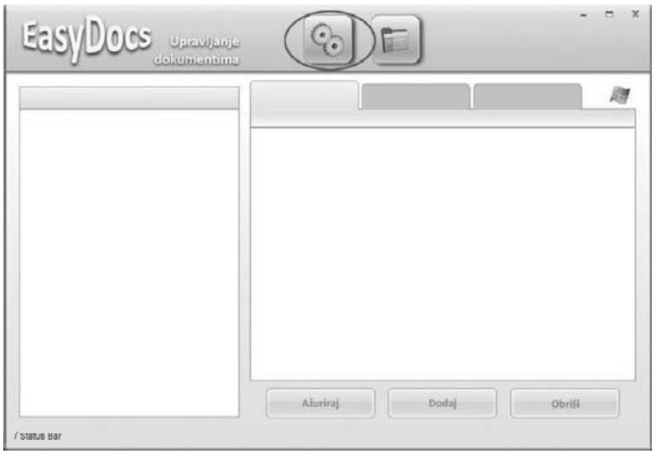
ActionScript funkcija *loadtree()* ima za cilj da formatira niz, dobijen izvršavanjem C# metode, koja je bila pozvana *call()* funkcijom, i da napuni *treeview* komponentu na ekranskoj formi.

```
   function loadtree(nizdrvo:Array):Void{
        myTree.removeAll();
        var i:Number=0;
        for(i;i<nizdrvo.length;)
        {
                if (nizdrvo[i+2] == 1)
                        myTree.setIsBranch(myTree.
addTreeNode(label=nizdrvo[i], data=nizdrvo[i+1]), true);
                else
                        myTree.setIsBranch(myTree.
addTreeNode(label=nizdrvo[i], data=nizdrvo[i+1]),
false);
                i+=3;
        }

   // Set movieclip as icon for 2nd node.
   myTree.setIcon(myTree.getTreeNodeAt(0), 'imageIcon');
                        ...
   }
```

Funkcija kao parametar prima niz i nema povratnih vrednosti. Pošto je obrađeni deo aplikacije adminstrativni, na ekranskoj formi (Slika 2.) je moguće vršiti izvesne izmene, pa je iz tog razloga potrebno dinamički vršiti izmene postojećeg prikaza, čime se objašnjava prva linija koda, metode *loadtree()*, gde je pozvana predefinisana funkcija *removeAll()*, koja olobađa *treeview* komponentu postojećeg sadržaja. U nastavku se iterativnom narebom *for*, prolazi kroz dobijeni niz i podacima puni *treeview* komponenta. Na kraju se, u zavisnosti od tipa prikazanog podatka (stavke *treeview* komponente), svakoj stavci dodeljuje grafički identitet u vidu ikone (*icon*).

Na (Slici 2.) je prikazan rezultat izvršenja događaja *onRelease* dugmeta, čije je ime instance *sistem*, a koji obuhvata sve navedene faze.



Slika 1. – *Izgled prazne forme*

Slika 2. – *Izgled forme sa napunjenim treeview*

4. NEDOSTACI NAVEDENOG NAČINA INTEGRACIJE

Prilikom implementacije integracije Adobe Flash-a i .NET programskog jezika C#, uočeni su sledeći nedostaci koji se pre svega odnose na tehnološka ograničenja:

– COM komponente i problemi sa memorijom: COM komponente nemaju podršku .NET-a kada je u pitanju rad sa memorijom računara i shodno tome otežavaju rad, jer se posebna pažnja mora obratiti na curenje memorije *(memory leak)*.
– Na računaru mora biti instaliran Adobe Flash Player, a Flash kontrola registrovana, što može predstavljati veliki problem prilikom razvoja distribuiranih aplikacija koje rade na više računara. Jedno od rešenja je da se Flash kontrola uključi u *msi* paket i da se prilikom instalacije programa na klijentskoj mašini registruje kontrola na pravi način.
– Problemi prilikom razvoja i otkrivanja grešaka: ukoliko dođe do pojave greške, posebno u pogledu komunikacije, kôd se mora proveravati sa obe strane i iz više alata, što posebno komplikuje razvoj i otklanjanje grešaka.
– Visok rast kompleksnosti razvoja Flash korisničkog interfejsa sa povećanjem složenosti poslovnog sloja (C#).

5. ZAKLJUČNA RAZMATRANJA

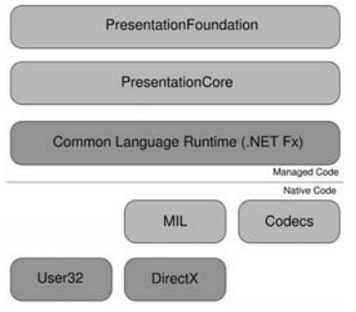
Potpuna integracija komponenti Adobe Flash-a i .NET programskog jezika C#, u cilju konstrukcije korisničkog interfejsa je složen i komplikovan proces koji zahteva daleko više vremena od razvoja standarnog korisničkog interfejsa Windows aplikacije.

U toku izrade ove aplikacije je izašla nova verzija *Visual Studia 2008* u kojoj je implementiran *Windows Presentation Foundation* (WPF), što ukazuje da je i Microsoft-ov razvojni tim uočio pomenute nedostatke koji se odnose na izgled standardnih korisničkih intefrejsa Windows aplikacija.

WPF predstavlja grafički podsistem u .NET Fremework-u 3.0 koji je direktno povezan sa XAML *(Extensible Application Markup Language)* i obezbeđuje konzistentnost programskog modela u izradi aplikacija, kao i jasno razgraničenje između korisničkog interfejsa i poslovne logike. On objedinjuje sledeće servise host aplikacije: korisnički interfejs, 2D i 3D grafiku, fiksne i dinamičke dokumente, napredne topografije, vektorsku grafiku, restersku grafiku, animaciju, povezivanje sa podacima, audio i video.

WPF aplikacije mogu biti postavljene kao desktop aplikacije ili hostovane u Web pretraživaču. One omougućavaju bogate kontrole, dizajn i razvoj vizuelnih aspekata Windows aplikacija.

Na Slici 3. je prikazana WPF arhitektura, gde tamno sivi elementi predstavljaju Windows komponente, a svetlo sivi elementi predstavljaju WPF komponente.



Slika 3. – *WPF arhitektura*

Microsoft *Silverlight* je Web orijentisan podskup WPF-a koji omogućava Flash-olike Web i mobilne aplikacije sa istim programskim modelom kao i .NET aplikacije. 3D mogućnosti nisu podržane, ali *XML Paper Specification* (XPS) i vektorski orijentisana grafika jesu. Ipak treba naglasiti da postoje bitne razlike između Adobe Flash-a i *Silverlight-a* i da *Silverlight* ne može u potpunosti biti zamena za postojeću Flash tehnologiju.

Pored ostalih, u konkretnom slučaju treba naglasiti sledeće razlike između Silverlight-a i Adobe Flash-a:

– Sliverlight ne podržava povezivanje sa modelima, podacima, kao ni konektovanje na mrežne resurse zarad dobijanja podataka.
– Silverlight ne podržava biblioteke za dugmad (*buttons*), čekboksove (*checkboxes*), list boksove (*list boxes*), list vju (*list views*), gridove (*grids*).
– Ne može da procesira zvuk.
– Programiranje Soketa (*Socket*) nije moguće.
– Ne postoji podrška za Editovanje bitmapa po pikselu , kao ni bitmap filtere (*convulution, color matrix*), ili bitmap efekte (*drop shadow, blur, glow*).
– Nije ugrađena ni podrška za *upload/download* fajlova.
– Performanse *Silverlight*-a i Adobe Flash-a su približno iste. *Silverlight* koristi XAML kao deskriptivni jezik u nekompresovanom formatu, pa je veličina njegovih komponenti u proseku 10-20 puta veća u odnosu na Adobe Flash, u praktičnoj implementaciji sličnih komponenti.
– Trenutno ne postoji podrška za izvršavanje *Silverlight* objekata u Windows aplikacijama.

LITERATURA

[1] Livedocs Adobe, http://livedocs.adobe.com/flash/8/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001599.html
[2] Livedocs Adobe, http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000346.html
[3] Microsoft MSDN Library, http://msdn.microsoft.com/en-us/default.aspx
[4] Wikipedia, http://en.wikipedia.org/wiki/Windows_Presentation_Foundation
[5] Wikipedia, http://en.wikipedia.org/wiki/ActionScript
[6] Silverlight, http://silverlight.net/forums/t/3015.aspx

Dr Saša D. Lazarević, docent, Fakultet organizacionih nauka u Beogradu, Katedra za softversko inženjerstvo
Oblast interesovanja: softversko inženjerstvo, informacioni sistemi, baze podataka, sistemi za upravljanje dokumentacijom, .NET platforma

Dušan Marković, Fakultet organizacionih nauka u Beogradu, Katedra za softversko inženjerstvo
Oblast interesovanja: Softversko inženjerstvo, Razvoj .NET aplikacija, Dokumentacioni sistemi

Ivan Stamenić, Fakultet organizacionih nauka u Beogradu, Katedra za softversko inženjerstvo
Oblast interesovanja: Softversko inženjerstvo, Razvoj .NET aplikacija, Multimedija