UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Anže Ciuha

Centraliziran multimedijski informacijski sistem

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Luka Šajn

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01849/2012
Datum: 13.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANŽE CIUHA**

Naslov: **CENTRALIZIRAN MULTIMEDIJSKI INFORMACIJSKI SISTEM**
CENTRALIZED MULTIMEDIA INFORMATION SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kandidat naj razišče in razvije rešitev za centralizirano urejanje in predvajanje multimedisjkih vsebin, namenjeno predvajanju na javnih panojih in zaslonih.
Rešitev naj bo sestavljena iz dveh glavnih sklopov, in sicer iz centralnega urejevalnika in dislociranega predvajalnika vsebin. Centraliziran urejevalnik naj bo narejen kot internetna aplikacija, predvajalnik pa kot Windows aplikacija. Sinhronizacija naj poteka preko spletne storitve in protokola HTTP ali FTP za vsebine.

Mentor:
doc.dr. Luka Šajn

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Anže Ciuha,

z vpisno številko 63000151,

sem avtor diplomskega dela z naslovom:

Centraliziran multimedijski informacijski sistem

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luke Šajna,

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 15. oktobra 2012 Podpis avtorja:

ZAHVALA

Zahvaljujem se mentorju doc. dr. Luki Šajnu za vse nasvete, usmeritve in spodbudo. Prav tako bi se rad zahvalil prijatelju Matjažu Mirtiču za ideje in nasvete pri razvoju rešitve.

Kazalo

Povzetek

Diplomsko delo opisuje razvoj rešitve za urejanje in predvajanje multimedijskih vsebin, namenjene predvsem predvajanju na javnih panojih in zaslonih. Rešitev je poimenovana Centraliziran multimedijski informacijski sistem oziroma na kratko CMIS.

Rešitev je sestavljena iz dveh ključnih sklopov, in sicer iz spletne aplikacije, ki skrbi za urejanje vsebin, in iz multimedijskega predvajalnika za predvajanje vsebin na dislociranih enotah.

Spletna aplikacija je razdeljena na tri glavne sklope, in sicer na urejevalnik multimedijskih vsebin, urejevalnik kanalov in urejevalnik vtičnikov.

Večpredstavnostni predvajalnik je predvajalnik multimedijskih vsebin s sposobnostjo sinhroniziranja vsebin iz spletnega strežnika. Uporabnik vsaki vsebini določi tudi parametre, ki jih predvajalnik razume in ustrezno predvaja.

Spletna aplikacija je napisana v programskem jeziku PHP, uporablja podatkovno bazo MySQL, predvajalnik pa je napisan v programskem okolju Delphi.

Ključne besede:

spletna aplikacija, spletni urejevalnik multimedijskih vsebin, multimedijski predvajalnik, predstavitev, oglas

Abstract

The diploma describes the development of the solution for editing and playing multimedia content, especially for playing the content on public boards and screens. The solution is called Centralized multimedia information system or CMIS.

The solution consists of two key components - web application for editing the content and multimedia player for presenting the content on dislocated units.

Web application is divided into three main components: multimedia content editor, channel editor and port editor.

Multimedia player is a player for multimedia content which synchronizes content from web server. The user defines the content parameters, which are understood and played by the player.

Web application is written in PHP programming language and uses MySQL database, multimedia player is written in Delphi programming environment.

Key words:

web application, online multimedia content editor, multimedia player, presentation, advertisement

1 Uvod

1.1 Ideja

Cena televizorjev in računalnikov strmo pada, poleg tega pa se na trgu že dobi televizorje z dolgo diagonalo za razmeroma ugodno ceno. Osnovni modeli računalnikov, ki v večini primerov zadostijo sistemskim zahtevam predvajalnika, so tudi cenovno zelo dostopni, dobijo se že v izredno majhnih ohišjih, z izredno majhno porabo energije. Poleg vsega naštetega je v današnjem času potreba po predstavitvi izdelka ali storitve v poslovnem svetu še bolj pomembna, zato se vedno več organizacij odloči, da v svojih poslovnih prostorih, vitrinah, oziroma povsod, kjer se zadržujejo množice, postavi zaslon za prikazovanje multimedijskih vsebin.

Večina organizacij uporablja zelo primitivne rešitve za predvajanje vsebin, kot so razne Powerpoint predstavitve, predvajalniki slik, predvajalniki videov, ki jih preko prenosnega medija prenesejo na računalnik. Torej rešitve, ki so v osnovi namenjene za nek drug namen in zato predstavljajo zelo togo rešitev problema.

1.2 Namen

Namen je izdelati rešitev za predvajanje in centralizirano urejanje vsebin ter omogočiti organizacijam, da s čim manj truda na različnih lokacijah objavljajo svoje vsebine in jim omogočiti čim večjo mero fleksibilnosti.

1.3 Cilji

Končni cilj je seveda delujoča in stabilna rešitev, ki bi bila čim bolj enostavna za uporabnika, omogočala bi tudi napredne možnosti, ki bi aplikacijo naredile še bolj uporabno.

1.4 Struktura diplomske naloge

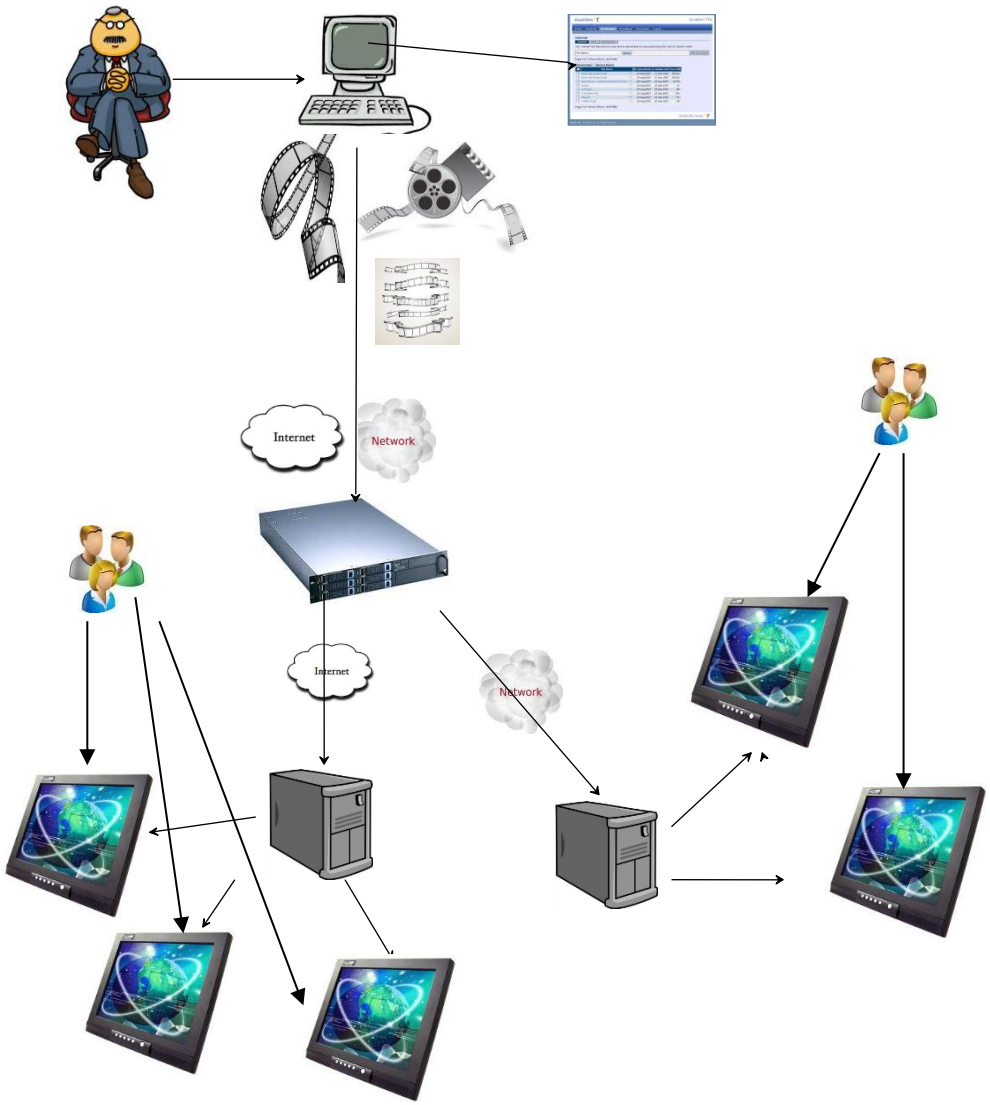
Diplomska naloga je sestavljena iz dveh glavnih sklopov, in sicer iz spletne aplikacije in predvajalnika. V prvem delu bomo opisali spletno aplikacijo, v drugem pa se bomo posvetili predvajalniku in za tem opisali še sinhronizacijo podatkov. Za nemoteno delovanje celotnega sistema moramo zagotoviti avtomatsko nadgradnjo dislociranih predvajalnikov, za kar sta potrebna še dodaten servis, ki skrbi za zadnjo verzijo predvajalnika in manjši program, ki zna zamenjati predvajalnik.

2 Predstavitev delovanja

Uporabnik preko internetnega uporabniškega vmesnika uredi multimedijske vsebine. Vsebine se tako prenesejo na strežnik preko interneta ali lokalne mreže in se tam obdelajo. Nato se klienti oz. predvajalniki teh vsebin povežejo s strežnikom, kjer dobijo seznam vsebin in jih prenesejo na lokalni računalnik, ki jih predvaja. Vsakih x sekund prevajalnik preveri, ali se je seznam spremenil, in po potrebi zbriše vsebine, ki so odveč oziroma prenese še neprenesene.

Tak način nam omogoča, da predvajalnik lahko deluje v tako imenovanem "offline" načinu, ko dostop do strežnika ni mogoč.

Slika (Slika 1) nam prikazuje delovanje celotnega sistema.

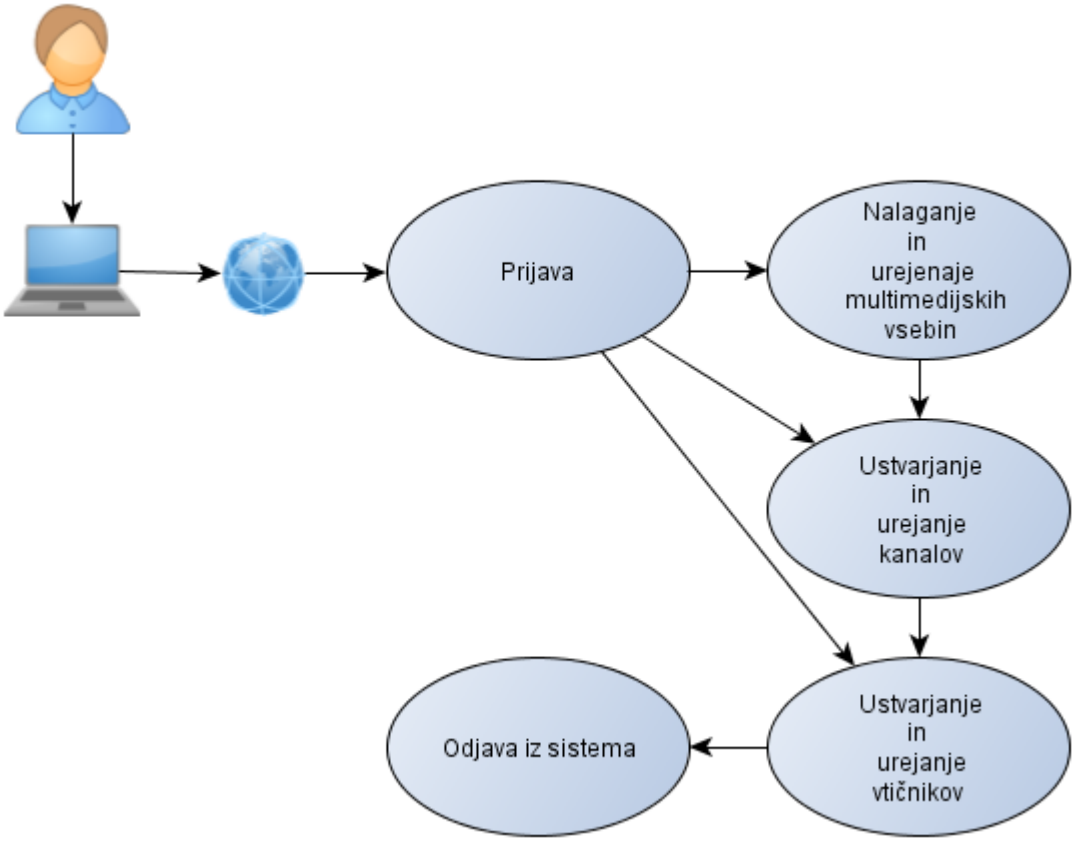


Slika 1. Prikaz celotnega sistema

2.1 Administrativni del oziroma uporabniški vmesnik

Uporabnik se prijavi v sistem, kjer vpiše pravilno uporabniško ime in geslo. Sistem pogleda, kateri grupi pripada uporabnik, in mu prikaže administrativni del oziroma uporabniški vmesnik (Slika 3) z glavnimi možnostmi urejanja multimedijskih vsebin, urejanja kanalov in urejanja vtičnikov.

Uporabnik ima nadzor samo nad vsebinami, ki pripadajo grupi, katere član je. Uporabnik lahko izbrane multimedijske vsebine naloži s svojega računalnika preko internetnega vmesnika. Ko ima uporabnik oziroma, bolj natančno, njegova grupa na strežnik naložene že vse želene vsebine, lahko uporabnik kreira svoj kanal in ga poljubno opremi z multimedijskimi vsebinami, ki so mu na voljo. Vsako postavko lahko opremi z različnimi parametri, kot so podnapisi, prehodi in podobno. Uporabnik ima sedaj enega ali več kanalov in lahko uredi vtičnike. Na vsak vtičnik se veže natanko en kanal, predstavlja pa vez med kanalom in predvajalnikom. To je tudi osnovni in celovit postopek internetnega dela, ki ga mora uporabnik izvesti za delovanje sistema oziroma da lahko prevajalnik začne predvajati vsebine. Uporabnik se lahko odjavi iz sistema. Postopek je prikazan na sliki (Slika 2).



Slika 2. Diagram delovanja sistema

Uporabniški vmesnik je razdeljen na štiri sklope.

Lokacija	Tip vsebine	Opis
Zgoraj	Statična	Desno zgoraj je administrativna navigacija, spodaj levo pa navigacija po glavnih skopih sistema. Vsakemu sklopu pripada tudi svoja barva.
Leva sredina	Dinamična	Pomembne informacije uporabniku in njegovi grupi in zadnji dogodki v grupi.
Desna sredina	Dinamična	Urejevalniki za vsak sklop posebej. Predstavlja najpomembnejši del, kjer se urejajo vsebine. V zgornjem delu oddelka je prikazana ikona v barvi sklopa.
Spodaj	Dinamična	Menjava jezika (trenutni jezik ni prikazan).

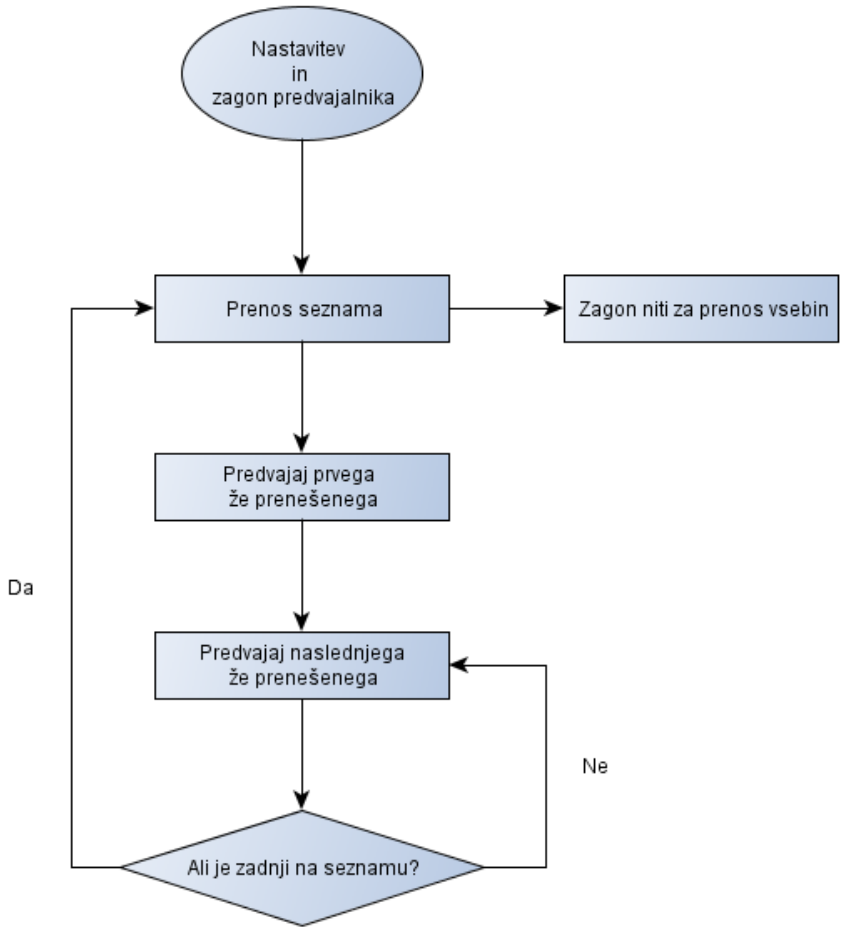
2.1.1 Grupe

Vsaka grupa vsebuje svojo množico medijev, kanalov in vmesnikov, torej vsak element te množice pripada natanko eni grupi. Tako zagotovimo zasebnost vsebin vsaki grupi posebej. Grupa ima enega ali več uporabnikov, ki urejajo vsebine grupe, kateri pripadajo.
Na sliki (Slika 3) je v levem sklopu prikazan trenutni uporabnik in kateri grupi pripada.

Slika 3. Uporabniški vmesnik

2.2 Predvajalnik

Administrator oziroma vzdrževalec predvajalnika namesti predvajalnik in v nastavitveni datoteki nastavi številko vtičnika, uporabniško ime in geslo ter opcijsko še kakšne druge parametre. Zažene program. Program preko spletne storitve dobi seznam za predvajanje s parametri, kjer je navedena tudi pot do datoteke. Predvajalnik začne v drugi niti na lokalni medij pretakati datoteke oziroma medije, istočasno pa začne predvajati že lokalno prenesene (Slika 4).



Slika 4. Diagram poteka predvajalnika

3 Predstavitev razvojnega okolja

3.1 Internetni del

Html

HTML (Hyper Text Markup Language) je označevalni jezik za izdelavo spletnih strani in spletnih aplikacij. Predstavlja osnovo spletnega dokumenta. Določa prikaz dokumenta v spletnem brskalniku in določa strukturo ter semantičen pomen delov dokumenta [6]. Zapisan je v obliki HTML elementov, ki so sestavljeni iz značk, zapisani v špičastih oklepajih. Značke so ponavadi zapisane v parih in predstavljajo začetek in konec bloka. Celotno strukturo blokov pa si lahko predstavljamo kot drevesno strukturo.

Primer:

```
<html>
  <body>
    <h1>Heading</h1>
    <p>Paragraph</p>
  </body>
</html>
```

PHP

PHP (Hypertext Preprocessor) je odprtokodni skriptni jezik, namenjen za strežniško uporabo in še bolj pogosto za izdelavo dinamičnih spletnih strani. Napisal ga je dansko kanadski programer Rasmus Lerdorf leta 1994. PHP teče na spletnem strežniku, kjer jemlje izvorno PHP kodo in generira spletno stran kot izhod [3, 10].

Primer:

```
<?php
 echo "Hello world";
?>
```

CSS

CSS (Cascading Style Sheets) so predloge, predstavljene v obliki preprostega slogovnega besedila, ki skrbijo za predstavitev spletnih strani in z njimi definiramo stil HTML elementov v smislu pravil, kako jih prikažemo na spletni strani [8].

Primer:

```
body
 {
 background-color:#d0e4fe;
 }
h1
 {
 color:orange;
 text-align:center;
 }
```

Smarty

Smarty je sistem za predloge, napisan v programskem jeziku PHP. Namenjen je predvsem, da loči obliko spletne strani od kode [7].

Primer:

```
<html>
<body>
<h1>{block name=title}Heading{/block}</h1>
<p>{block name=title}Paragraf{/block}</</p>
</body>
</html>
```

JavaScript

JavaScript je objektni skriptni programski jezik, ki ga je razvil Netscape za pomoč spletnim programerjem pri interaktivnih spletnih straneh. Razvit je neodvisno od Jave, vendar si z njo deli številne lastnosti. JavaScript lahko sodeluje s HTML kodo in razširi njene zmožnosti, predvsem na interaktivnem delu. Program se izvede na klientu, torej v internetnem brskalniku [9].

Primer:

```
<script type="text/javascript">
function displayDate()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
```

jQuery

Je najbolj popularna JavaScript knjižnica v tem času.

Primer Ajax klica [1, 3, 9]:

```
$.ajax({
  type: "POST",
  url: "example.php",
  data: "name=John&location=Boston",
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

Za vizualno interakcijo, efekte, teme in podobno pa skrbi dodatek **jQueryUI** [11].

MySQL

Je sistem za upravljanje s podatkovnimi bazami. Je odprtokodna implementacija relacijske podatkovne baze. Za delo s podatki uporablja jezik SQL. Strežnik lahko namestimo kot porazdeljen strežnik. Napisan je v programskih jezikih C in C++. [10].

Spletna storitev

Spletna storitev (Webservice) je programska oprema, ki omogoča komunikacijo med dvema elektronskima napravama preko omrežja. Za spletno storitev obstaja vmesnik WSDL, opisan v formatu, ki ga lahko naprave procesirajo. WSDL bazira na XMLju. WSDL podaja opis, kako se storitev pokliče, katere parametre zahteva in kakšno strukturo storitev vrača [17].

Programsko ogrodje FFMPEG

Je programsko ogrodje za snemanje, konverzijo in predvajanje tako zvoka kot video vsebin. Vključuje vodilno knjižnico libavcodec za obdelavo zvoka in videa. Sestavljeno je iz več pripomočkov:

- ffmpeg: orodje za obdelavo s pomočjo parametrov,
- ffserver: multimedijski strežnik za predvajanje v živo,
- ffplay: multimedisjki predvajalnik,
- ffprobe: enostavni analizator multimedijskih vsebin.

Program se poganja preko konzole s pomočjo parametrov. Omogoča mnogo različnih obdelav videa, kot so sprememba formata, velikosti, zajem določene slike in tako dalje [14].

Knjižnica GD

Odprtokodna knjižnica za dinamično kreiranje slik. Napisana je v programskem jeziku C, obstajajo pa tudi pripomočki za uporabo knjižnice v drugih programskih jezikih, med njimi tudi Perl in PHP. Ponavadi se knjižnica uporablja za kreiranje grafov, grafik, majhnih sličic in drugih slikovnih operacij. Čeprav knjižnica ni omejena samo na internetni del, se uporablja predvsem pri internetnem programiranju [16].

Plupload

Je orodje za prenos datotek na strežnik. Za prenos datototek potrebuje podporo HTML5 ali enega od vtičnikov Gears, Silverlight, Flash ali BrowserPlus. Orodje omogoča nekaj uporabnih lastnosti, na primer za napredek prenosa in za prenos datoteke po delih. Rešitev je sestavljena iz dveh delov, in sicer iz klientovega dela, ki razreže datoteko na več delov, in iz strežniškega dela, ki poskrbi, da datoteko ponovno sestavi [18].

3.2 Klient

Delphi

Razvojno okolje za razvoj aplikacij, ki je primarno namenjeno pisanju Windows aplikacijam, v novejši verzijah pa so podprte tudi druge platforme [2].

ImageEn

Knjižnica za delo z video posnetki in slikami. Omogoča razne efekte, prehode in druge uporabne zadeve [15].

kbmMemTable

Zelo zmogljiva knjižnica za delo s podatki na lokalni ravni. Omogoča skriptni jezik SQL, filtre in podobno [12].

3.3 Razlogi izbire tehnologij, komponent, modulov

Pri vsakem projektu je izbira tehnologij pomembna in težka odločitev, saj gre za kompleksno kombinacijo narave projekta in seveda znanj, ki jih premorejo razvijalci. Od te izbire so odvisni tudi uspešnost projekta, njegova kvaliteta in čas izdelave. Za navajanje na drugo okolje je potreben čas, da se razvijalci spoznajo z novim okoljem in da osvojijo širok spekter zmožnosti, saj se le tako lahko kvalitetno posvetijo dejanskim problemom.

Pri izbiri tehnologije za internetni del izbiramo med programskim jezikom ASP.net in PHP. Vsaka od teh tehnologij ima svoje prednosti, a ker v osnovi ne gre za obsežno aplikacijo, je bil izbran bolj fleksibilen programski jezik PHP, s katerim sem imel več izkušenj.

Za klienta oziroma predvajalnik sem se odločil za Delphi, ker to razvojno okolje najbolje poznam in pri iskanju primernih komponent nisem imeli nikakršnih problemov, saj je bilo za ključen del (predvajanja video vsebin) mnogo uporabnih komponent. Poleg vsega se nadejamo, da bo projekt FireMonkey, ki je po novem del razvojnega okolja, postal dodelan do te mere, da se bo dalo predvajati videe in da bo poleg trenutno podprtih operacijskih sistemov deloval tudi na platformi Android. Če bi se to zgodilo, bi bil prehod dokaj enostaven, povečala bi se uporabna vrednost in seveda bi se zmanjšala tudi cena samega predvajalnika.

Pri razvoju spletne aplikacije seveda nista dovolj samo PHP in podatkovna baza temveč je potrebno uporabiti tudi knjižnice JavaScript in sistem za predloge, s katerim ločimo programsko kodo od kode HTML. Izbral sem knjižnico jQuery, ker je pokrila vse zahteve. Podobno velja za sistem za rokovanje s predlogami Smarty. Poleg tega imata oba omenjena sistema široko bazo uporabnikov in primerov, kar je seveda zelo dobrodošlo v primeru težav. Drugih možnosti, ki jih imata jQuery in Smarty, je še ogromno in tudi razvoj omenjenih orodij poteka hitro, kar daje našemu projektu prostor za nove ideje in možnost razvoja v prihodnje.

Neznanka je bila tudi, kako v programskem jeziku PHP iz video posnetka dobiti kakšno uporabno sličico, ki prikazuje, kaj se dogaja na video posnetku, saj le tako lahko dobimo dobro predstavo o videu.
Izbira zunanjega programa ffmpeg se je izkazala za zelo uporabno pri obdelavi video vsebin. Program smo uporabili tudi pri pretvarjanju video vsebine v standardni format, znan predvajalniku.

Pri klientu smo veliko časa posvetili raziskovanju primerne komponente za predvajanje video vsebin. Daleč najboljša komponenta je bila ImageEn, ki se je izkazala, da ne samo da zna delati z video vsebinami, zna tudi s slikami, omogoča razne filtre in prehode, kar je bilo kot naročeno za projekt. Tako je sama knjižnica porajala nove ideje in zadala nove smernice. Nekaj težav je bilo z različnimi formati video vsebin, saj jih nekatere komponente niso podpirale, vendar smo težavo rešili s pretvorbo, omenjeno v prejšnjem poglavju.

Po mnogih testih je bil izbran glavni nabor orodij za izgradnjo projekta, ki nam omogoča sestavo želene rešitve.

4 Centralni del – administracija

4.1 Organizacija datotek

Datoteke so logično razvrščene v mape.

Mapa	Opis
(korenski imenik)	Vsebuje datoteko index.php.
classes	Vsebuje php datoteke s skupnimi razredi in funkcijami. V podmapah so zunanje knjižnice.
css	Datoteke CSS.
images	Slike, potrebne za internetni vmesnik.
js	Javascript datoteke in v podmapah knjižnice JS.
pageparts	Vsebuje php datoteke, ki so namenjene delom stranem. Večina teh kod uporablja zunanje predloge html (shranjene v templates).
services	Spletne storitve za potrebe klienta.
templates	Predloge strani in delov strani (Smarty predloge).
upgrade	Datoteke za nadgradnjo predvajalnika.

4.2 Organizacija strani

Stran je sestavljena iz štirih glavnih delov, vsi moduli pa se prikazujejo v desni sredini (Slika 3). Module je potrebno definirati v bazi, in sicer obstaja za to posebna tabela, kjer vsakemu modulu določimo naslednje parametre:

- ime modula,
- naslov modula,
- ali vsebuje predlogo,
- katero dodatno datotek css naj vključi za ta modul,
- katero datoteko JavaScript naj vključi za ta modul.

Sistem nato avtomatsko pravilno vključi potrebne datoteke v kodo HTML.

Na glavni strani smo definirali tudi klic JavaScript funkcije OnLoad, ki se požene na vsakem modulu, če je seveda za modul deklarirana funkcija. Prav tako so v glavi definirane tudi druge funkcije (npr za osvežitev aktualnih podatkov na levi strani, ki so na voljo modulom). Te funkcije skrbijo, da lahko moduli komunicirajo z ostalimi deli strani.

V glavi vsake datoteke PHP se vključi koda PHP, ki je potrebna vsem datotekam PHP, da preveri, ali je uporabnik prijavljen, in vključi vse potrebne razrede za delo.
V primeru, da ugotovi, da uporabnik ni prijavljen, uporabnika preusmerimo na začetno stran.

4.3 Uporabljene splošne tehnologije

4.3.1 Ajax klici, jQuery in jQueryUI

Knjižnici jQuery in jQuery smo uporabili za vse klice Ajax, povleci in spusti metodo, galerijo slik in vizualne efekte.

Klice Ajax smo uporabili za nalaganje vsebin brez ponovnega nalaganja strani in osveževanje aktualnih informacij na levi strani.

Ajax je skupina medsebojno povezanih spletnih razvojnih tehnik. S klicem Ajax lahko spletna aplikacija s strežnikom izmenjuje podatke asinhrono, brez ponovnega nalaganja strani. To v praksi pomeni, da lahko osvežimo katerikoli element HTML ne da bi ponovno naložili stran.

Primer osnovnega klica Ajax – koda HTML in JavaScript:

```
<html>
<head>
<script type="text/javascript">
function PreveriStevilo(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {//za brskalnike IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {//za brskalnike IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  //ali je stran naložena do konca in ali je rezultat uspešno
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","jestevilo.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>
Vpisite stevilo: <input type="text" onkeyup="PreveriStevilo(this.value)"
size="20" />
<p>Odgovor: <span id="txtHint"></span></p>
</body>
</html>
```

Asinhrono klicana PHP, ki vrne samo tekst glede na GET parameter q (jestevilo.php):

```
<?php
    $q=$_GET["q"];
    if (!is_numeric($q)) echo 'Ni stevilo';
    elseif ($q > 10) echo 'Število je večje od 10';
    else echo 'Število je manjše ali enako 10';
?>
```

Na primeru vidimo, kako se v programskem jeziku JavaScript kreira objekt xmlhttp, ki naloži zahtevano stran (jestevilo.php). Objekt xmlhttp sproži dogodek onreadystatechange (sproži se, ko se stanje objekta spremeni - torej tudi ob končanem prenosu), kjer mu priredimo metodo, v kateri preverimo, ali je prenos strani končan in ali je uspešen status. Takrat rezultat priredimo html elementu txtHint.

Seveda pa obstajajo tudi knjižnice, ki naredijo to in še več, tako da smo na strani uporabili klic Ajax iz knjižnice jQuery. Primer takega klica, ki je ekvivalenten zgornjemu klicu:

```
<script src="jquery.js"></script>
<script type="text/javascript">
function PreveriStevilo(str)
{
jQuery.get("jestevilo.php",
             {q: str},
             function(result){
         $("#txtHint").html(result);
           }
           );
}
</script>
```

4.3.2 Smarty

Z orodjem Smarty smo ločili kodo od oblike. V programski kodi PHP priredimo spremenljivke, polja oziroma objekte, ki jih nato orodje Smarty pretvori v html vsebino. Tak pristop je predvsem primeren zaradi lažjega in bolj strukturiranega pisanja aplikacije in nenazadnje tudi zato, da se oblika strani lažje spremeni.

Primer s poljem (PHP koda):

```
<?php
$arr = array(1000, 1001, 1002);
$smarty->assign('myArray', $arr);
?>
```

Smarty predloga:

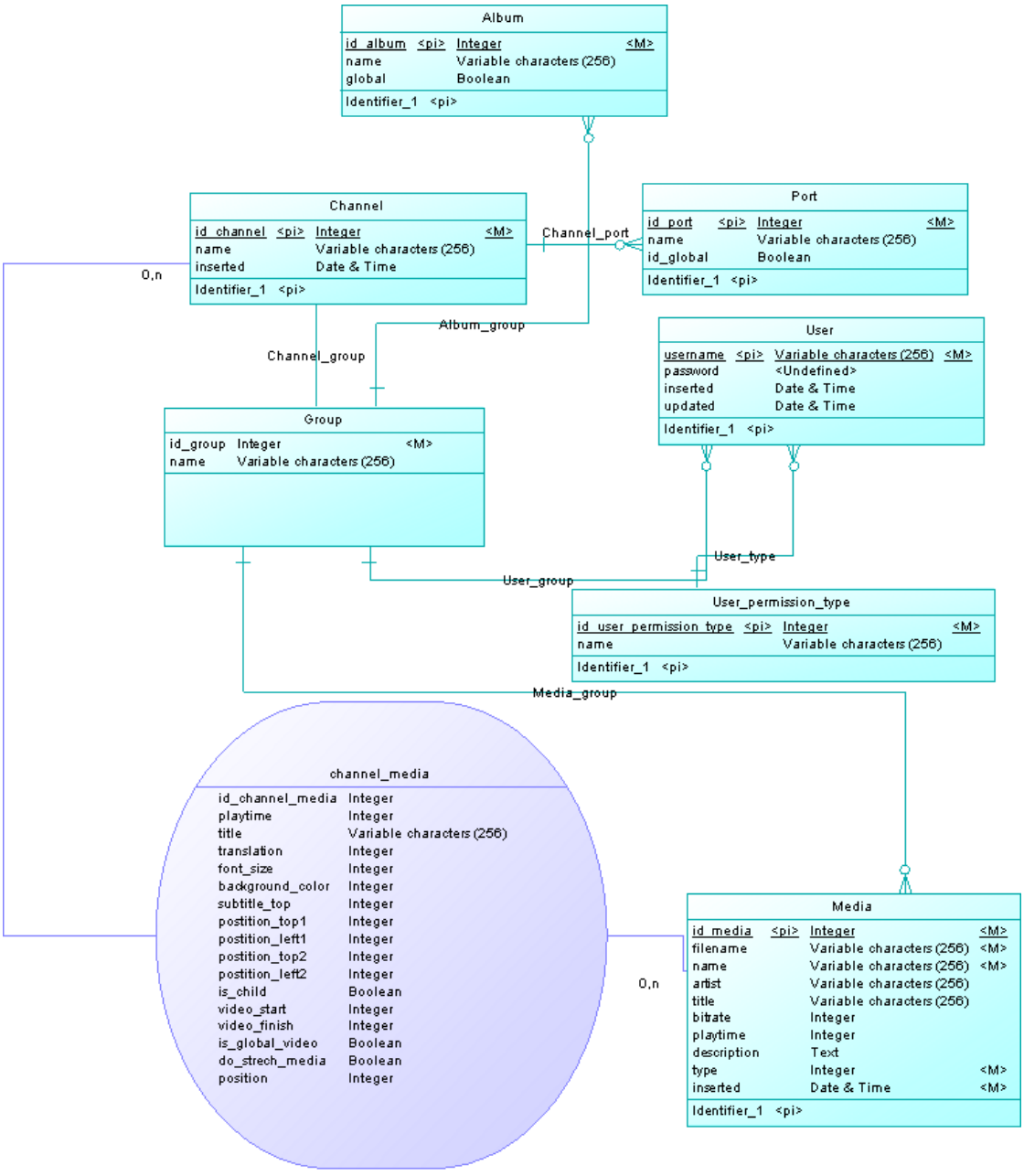
```
<ul>
{foreach from=$myArray item=foo}
    <li>{$foo}</li>
{/foreach}
</ul>
```

Rezultat:

```
<ul>
    <li>1000</li>
    <li>1001</li>
    <li>1002</li>
</ul>
```

4.4 Podatkovna struktura

4.4.1 Logični model



Slika 5. Logični model

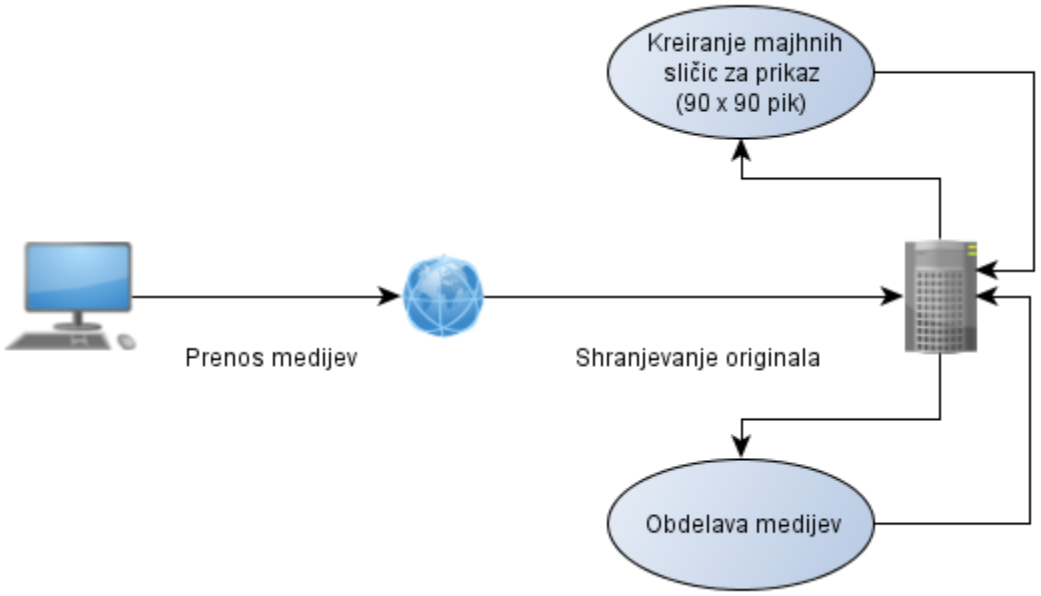
4.5 Opis modulov

4.5.1 Mediji

Mediji predstavljajo zbirko multimedijskih vsebin, ki jih uporabniku naložijo preko obrazca na strežnik. Zbirka je urejena v albumih. Vsak medij se avtomatsko poimenuje po imenu

datoteke, lahko pa mu ime tudi spremenimo. Prav tako dobi vsak medij tudi majhno sličico zaradi boljšega pregleda nad mediji, ki smo jih prenesli v sistem. Pri sliki je to kvadraten izrez slike in nato pomanjšava na 90 x 90 pik. Pri filmu je zajem slike na določenem mestu (nastavljivo), nato pa je postopek tak kot pri obdelavi slike/fotografije. Pri določenih formatih medijev se ne da dobiti dobro predstavljive majhne sličice, zato take medije predstavlja univerzalna sličica, ki prikazuje tip medija.

Prenos poteka preko obrazca, kjer se datoteka prenese na strežnik, nato se kreirajo majhne sličice za prikaz, zatem pa se vsak medij tudi obdela, da je primeren za prikaz na predvajalniku.



Slika 6. Prenos in obdelava multimedijskih vsebin

4.5.1.1 Pretakanje medijev na strežnik - upload

Tipično pošiljanje datotek na strežnik poteka preko internetnega obrazca z metodo POST. Tako pošiljanje je primerno za manjše datoteke, saj ima strežnik omejitev velikosti poslane datoteke in tudi časovno omejitev enega zahtevka. Hitro torej naletimo na omejitev, saj poleg tega, da je pošiljanje večjih datotek dokaj nepraktično, tudi nimamo nadzora nad tem, koliko je preneseno v času prenosa. Tudi če imamo možnost spreminjati te parametre, ostaja prenos še vedno nepraktičen za večje količine podatkov.

Primer takega obrazca:

```
<form enctype="multipart/form-data" action="uploader.php" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Prenesi datoteko: <input name="uploadedfile" type="file" /><br />
<input type="submit" value="Izberi datoteke" />
</form>
```

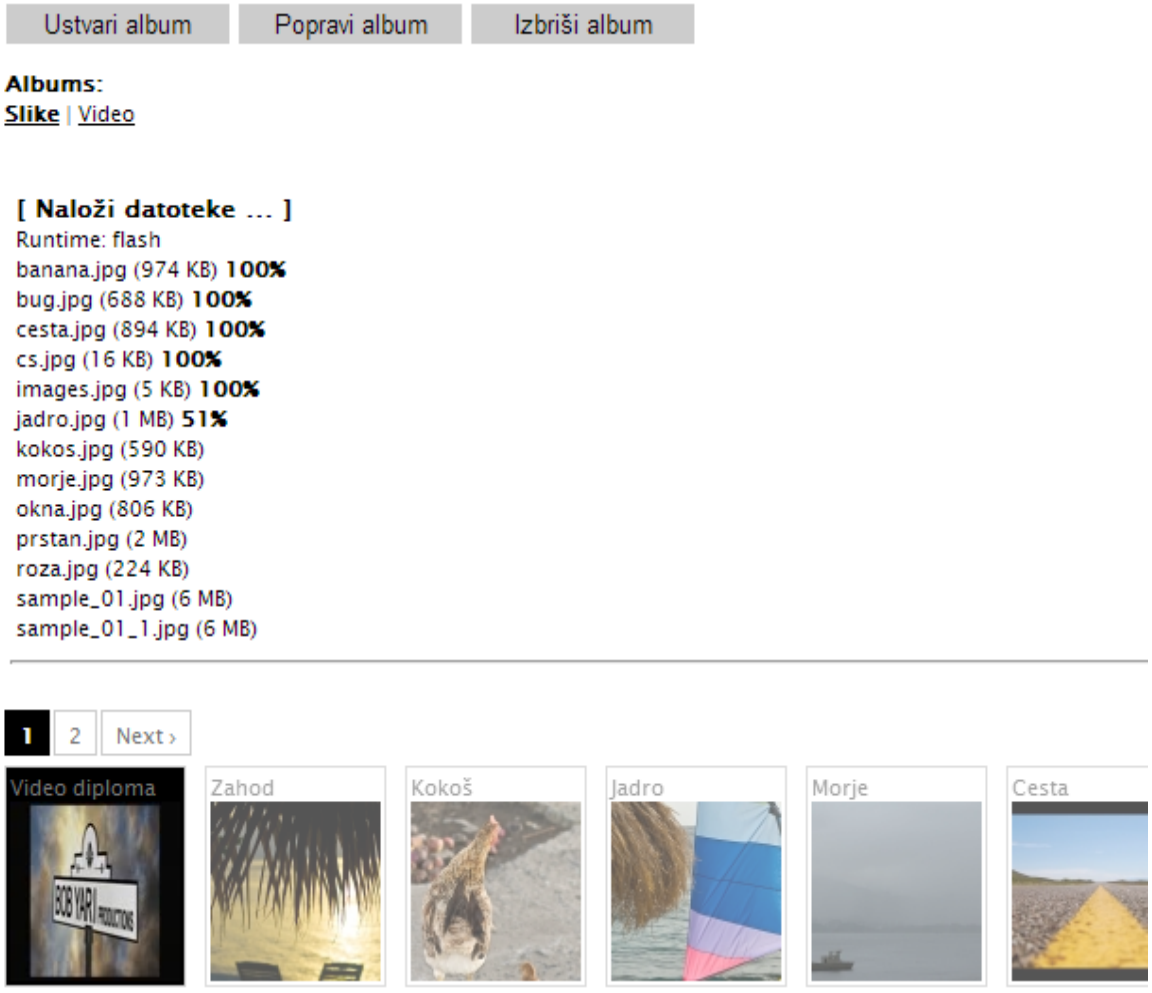
Rešitev je rezanje datoteke na manjše dele, ki jih ločeno pošiljamo na strežnik po zgornji metodi, strežnik pa jih ponovno sestavi v celoto. Seveda bi se dalo vso stvar razviti, vendar obstaja zelo priročno orodje Plupload, ki naredi ravno to – zna delati z različnimi vtičniki, integriranimi v brskalnik, poleg tega pa zna prikazati količino že prenesenega. Orodje Plupload ima torej vse, kar potrebujemo za udoben prenos velikih količin podatkov preko spletnega obrazca na strežnik.

Primer uporabe:

```
<script type="text/javascript">
$(function() {
    $("#uploader").pluploadQueue({
        // Splošne nastavitve
        runtimes : 'gears,flash,silverlight,browserplus,html5',
        url : 'upload.php',
        max_file_size : '10mb',
        chunk_size : '1mb',
        unique_names : true,
        // Sprememba velikosti slike
        resize : {width : 320, height : 240, quality : 90},
        // določitev tipov datotek
        filters : [
            {title : "Image files", extensions : "jpg,gif,png"},
            {title : "Zip files", extensions : "zip"}
        ],
        //Flash nastavitve
        flash_swf_url : '/plupload/js/plupload.flash.swf',
        //Silverlight nastavitve
        silverlight_xap_url : '/plupload/js/plupload.silverlight.xap'
    });

    $('form').submit(function(e) {
        var uploader = $('#uploader').pluploadQueue();
        if (uploader.files.length > 0) {
           uploader.bind('StateChanged', function() {
                if (uploader.files.length === (uploader.total.uploaded +
          uploader.total.failed)) {
                    $('form')[0].submit();
                }
            });
            uploader.start();
        } else {
            alert('Izberite vsaj eno datoteko.');
        }
        return false;
    });
});
</script>
```

Za nas pomembne nastavitve parametrov:

Parameter	Opis
runtimes	Po prioriteti, z vejico ločeni vtičniki, ki jih internetni brskalnik lahko uporabi.
url	Pot do strežniške php datoteke, kjer se datoteka iz razrezanih delov sestavi nazaj v prvotno datoteko.
max_file_size	Omejimo maksimalno velikost datoteke.
chunk_size	Kako veliki deli se pošiljajo preko ene zahteve. Velikost mora biti manjša, kot je na strežniku dovoljena maksimalna velikost POST forme oziroma velikost datoteke, poslane s POST obrazcem. Ob vsakem delu, ki se pošlje na strežnik, se osveži tudi prikazovalnik poteka prenosa, tako da v primeru, da želimo ažurnost prikaza prenosa podatkov, moramo nastaviti malo vrednost parametra, glede na pričakovano hitrost in željo po ažurnem osveževanju.
filters	Določimo, katere tipe datotek sprejemamo.



Slika 7. Primer prenosa več vsebin hkrati

4.5.1.2 Obdelava slik

Uporabniki imajo navadno slike v različnih velikostih in formatih, zato je potrebno za pravilno delovanje sistema slikam spremeniti velikost, da so primerne za prikaz na zaslonu. Poleg tega pa potrebujemo tudi majhne sličice za vizualni prikaz multimedijske vsebine.

Kreiranje majhnih sličic poteka na sledeč način:
Naredimo izrez kvadrata v sredini slike, ki zajema 50 % daljšega roba slike, nato pa sliko pomanjšamo na 90 x 90 pik. To je nekakšen kompromis med zajemom informacij na sliki in razpoznavnostjo motiva. Poleg tega ohranja razmerje motiva na sliki oziroma slika ni raztegnjena po širini ali višini. Vse majhne sličice pretvorimo v format JPEG.

Slika 8. Primer originala in izrezane in pomanjšane slike

Za obdelavo slik skrbi razred ImageHelper, ki poskrbi za pretvorbo med različnimi slikovnimi formati, za izreze in povečavo. Obdelano sliko na koncu lahko shranimo na disk.

Primer uporabe razreda ImageHelper za kreiranje majhne sličice in prikaz rezultata:

```
$ih = new ImageHelper;
$ih->setImage($FullFileNameNew);
$ih->createThumb();
$ih->saveImage($FullFileNameNewThumb);
```

Izvleček glavnih metod in njihove vsebine iz razreda ImageHelper:

```
class ImageHelper{
...
    function createThumb()
    {
      $thumbSize = THUMB_SIZE; //90 x 90
      $this->thumb = imagecreatetruecolor($thumbSize, $thumbSize);
      imagecopyresampled($this->thumb, $this->myImage, 0, 0,$this->x,  $this->y,
$thumbSize, $thumbSize, $this->cropWidth, $this->cropHeight);
    }
    function setImage($image)
    {
       $this->imgSrc = $image;
       list($width, $height) = getimagesize($this->imgSrc);
       $size=getimagesize($this->imgSrc);
        switch($size["mime"]){
            case "image/jpeg":$this->myImage = imagecreatefromjpeg($this->imgSrc); //jpeg
          break;
            case "image/gif":$this->myImage = imagecreatefromgif($this->imgSrc); //gif
          break;
            case "image/png":$this->myImage = imagecreatefrompng($this->imgSrc); //png
          break;
            case "image/bmp":$this->myImage = ImageCreateFromBMP($this->imgSrc); //png
          break;
        }
       if($width > $height) $biggestSide = $width; //najdi daljšo stranico
       else $biggestSide = $height;
       $cropPercent = .5; //odreži pol od daljšega roba
       $this->cropWidth   = $biggestSide*$cropPercent;
       $this->cropHeight  = $biggestSide*$cropPercent;
       //dobi zgornjo levo koordinato
       $this->x = ($width-$this->cropWidth)/2;
       $this->y = ($height-$this->cropHeight)/2;
    }
    function saveImage($image)
    {
        imagejpeg($this->thumb, $image, 100);
    }
...
}
```

4.5.1.3 Obdelava video vsebin

Video vsebine predstavljajo večji problem kot slikovne vsebine, saj obstaja bistveno več uveljavljenih formatov. Na začetku naletimo na problem, saj naš video predvajalnik določenih formatov ne zna prikazovati oziroma bi na računalniku, kjer je zagnan predvajalnik, morali namestiti kar nekaj dekoderjev. Nenazadnje bi se lahko zgodilo, da bi pri kakšnem videu prišlo do napake nepravilnega dekoderja ali bi s časom prišel kakšen nov format, kar bi pomenilo, da bi morali vse predvajalnike nadgraditi, kar pa seveda predstavlja obsežno nadgradnjo na vseh operacijskih sistemih predvajalnika. Najbolj zanesljiva rešitev je, da že na strežniku spremenimo format, za katerega smo sigurni, da ga predvajalnik zna predvajati. S pravilno izbiro lahko celo zmanjšamo velikost datoteke, ki je pri video vsebinah lahko relativno velika, še posebno takrat, ko govorimo o prenosu preko interneta. V primeru, da se pojavi kakšen nov format, težavo rešimo na strežniku in se tako izognemo sistemskim nadgradnjam na sistemu klienta.

Pri video vsebinah je tako spreminjanje formata časovno zahtevna operacija, ki ni tako hitra kot pri slikah, ko lahko čas obdelave praktično zanemarimo. Torej postopek, kot smo ga uporabili pri slikah, ko naredimo vse potrebne operacije po končanem prenosu, ne pride v poštev.

Naslednji problem, na katerega smo naleteli, je, da želimo iz videa narediti majhno sličico, ki predstavlja vsebino.

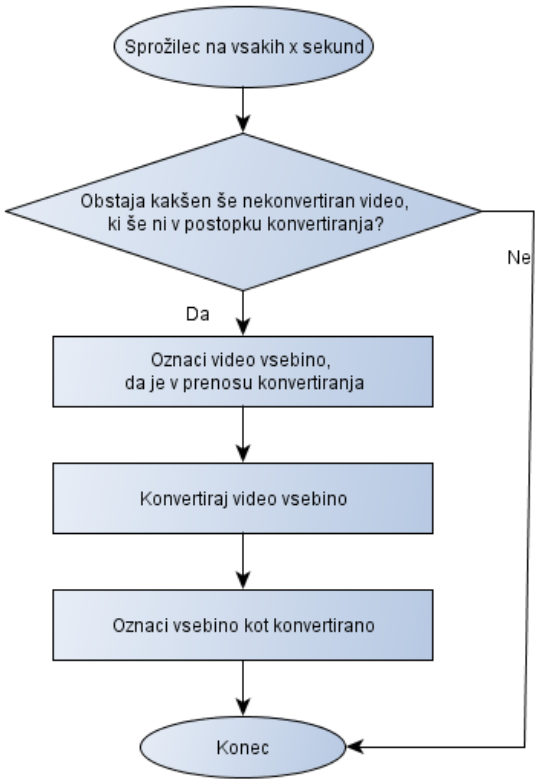
Za oba problema obstaja izvrsten program za konvertiranje video vsebin med različnimi formati in tudi zajem določene slike v video vsebini. Program se imenuje ffmepg in omogoča tudi konzolno klicanje s pomočjo parametrov.

Po testiranjih je zajem določene slike iz video vsebine z omenjenim programom dokaj hitra operacija in jo lahko vključimo v del obdelave po končanem prenosu, tako da lahko uporabnik takoj vidi, kaj je preneseno na strežnik, in seveda tudi uredi določene parametre za lažjo nadaljnjo prepoznavo medija.

Primer klica za zajem slike iz video vsebine:

```
ffmpeg -f mjpeg -vframes 1 -ss 20 -s 90x90
```

Parameter	Opis
-f mjpeg	izhodni format naj bo mjpeg
-vframes 1	število sličic
-ss 20	zamik naj bo 20s
-s 90x90	velikost 90x90 pik



Slika 9. Diagram poteka servisa za konvertiranje video vsebin v prevajalniku znan format

Ostane še konvertiranje v format, za katerega smo prepričani, da ga bo predvajalnik znal predvajati. Rešitev je zunanji program ali servis, ki poišče vse še nekonvertirane video vsebine in jih počasi v ozadju konvertira (Slika 9). Pri tem moramo v našem uporabniškem vmesniku označiti, da vsebina še ni na voljo. Prav tako mora spletna storitev, ki skrbi za sinhronizacijo seznama medijev za predvajanje, "vedeti", da takih video vsebin še ne vključi na seznam. Ob končanem konvertiranju vsebine pa označimo, da je vsebina na voljo.

Rešitev z ločenim servisom nam omogoča tudi izkoriščanje večih jeder procesorja oziroma procesorjev, lahko pa teče tudi na drugem računalniku ali več njih. Servis je nekakšen manjši program, ki prevzame nalogo konvertiranja določene video vsebine in to lokalno tudi zabeleži tako, da morebitnim drugim servisom sporoči, da se video že konvertira. Drugi servisi tako preskočijo to video vsebino in poiščejo novo, še nekonvertirano.

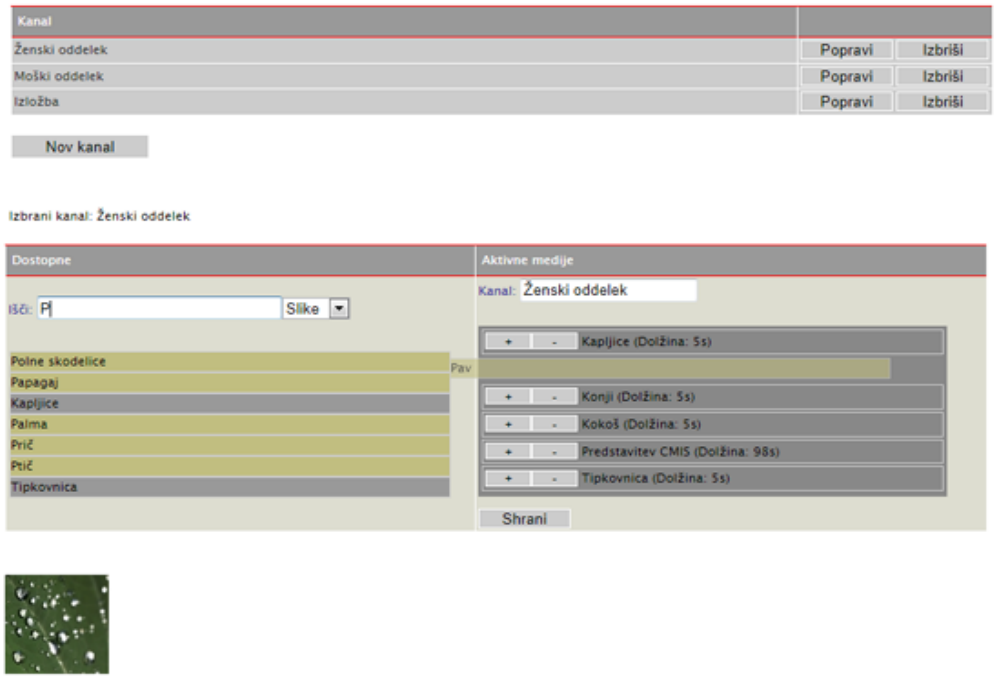
Konvertiranje poteka s pomočjo zunanjega programa ffmpeg. Za izhodni format video zapisa smo izbrali mpeg4, za zvočni zapisa pa ac3. Uporabili smo sledeče parametre:

```
ffmpeg -vcodec mpeg4 -acodec ac3 -ar 48000  -ab 192k -sameq
```

Parameter	Opis
-vcodec mpeg4	uporabi mpeg4 video kodiranje za izhodni video zapis
-sameq	uporabi kvaliteto video zapisa izvirnika
-acodec ac3	uporabi ac3 zvočno kodiranje za izhodni zvočni zapis
-ar 48000	zvočno vzorčenje 48000Hz
-ab 192k	zvočni podatki 192kbit/s

4.5.2 Kanali

Kanal predstavlja urejen seznam medijev. Vsakemu elementu oziroma seznamu določimo tudi parametre. S parametri določimo, kako se bo medij predvajal na klientu oziroma predvajalniku. Parametri se razlikujejo glede na tip medija. Na sliki (Slika 10) je prikazan modul kanal.



Slika 10. Urejanje kanala s povleci in spusti metodo

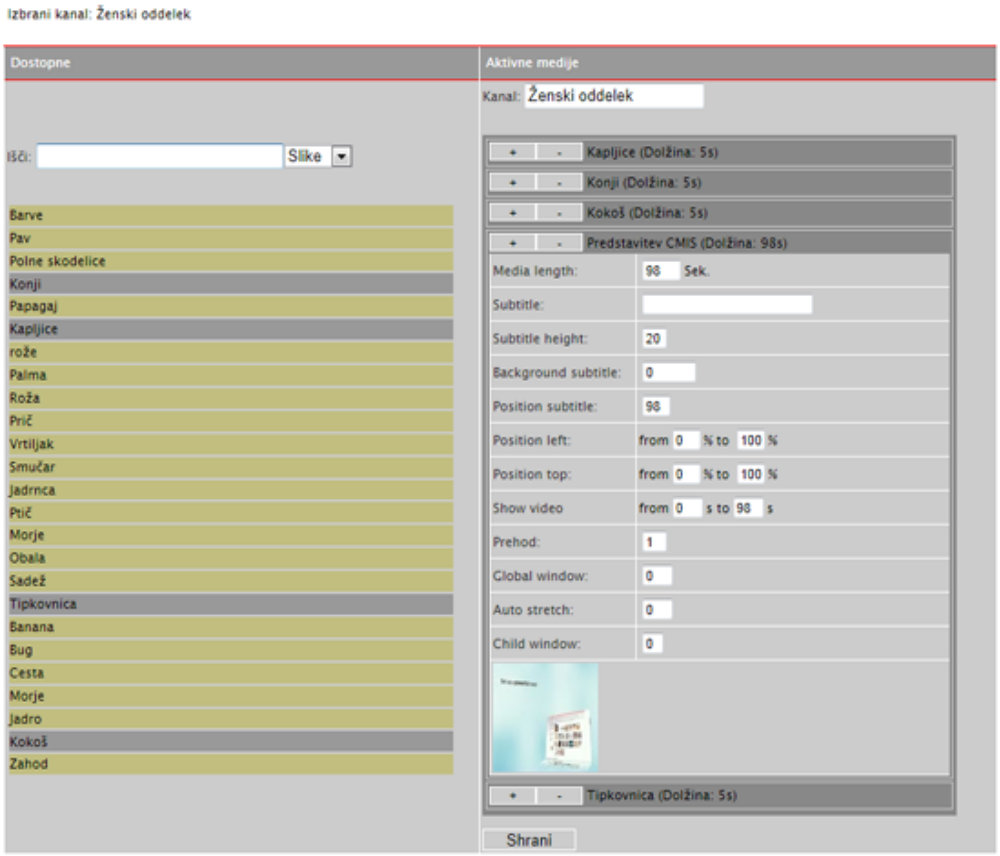
V zgornjem delu sta ikona v barvi kanala in opis.

V srednjem delu je prikazan seznam kanalov, ki pripadajo tej grupi. Kanal lahko uredimo, ga zbrišemo ali pa dodamo novega. Pri popravljanju in vstavljanju kanala se nam z Ajax klicem prikaže spodnji seznam, ki je razdeljen na dve polovici. Na levi strani so po albumih urejene multimedijske vsebine. V primeru daljšega seznama lahko vsebine tudi opisno iščemo. Iskalnik išče katerokoli izpeljanko iskane besede v imenu ali opisu medija v albumu. Ko se z miško postavimo na medij, se nam v spodnjem delu prikaže majhna sličica, ki predstavlja medij. Zaradi varčevanja s prostorom slike ne prikazujemo v samem seznamu.

Desni seznam prikazuje medije kanala z informacijo o času predvajanja. Vsebuje tudi dve ikoni za prikaz parametrov (Slika 11) in dodatnih informacij ter za skrivanje le-teh za bolj kompakten pregled. Mediji z rumenim ozadjem nam prikazujejo še neuporabljene medije v tem kanalu, sivo ozadje pa označuje, da je medij vsaj enkrat že uporabljen v trenutnem kanalu.

Prenos iz dostopnih (levo) v aktivne (desno) medije je narejen z metodo povleci in spusti ("drag and drop"). Metoda je prikazana na sliki (Slika 10) pri prenosu medija z imenom "Pav". Barva ozadja zapisa se deduje z leve strani, tako da lahko vidimo, katere medije smo dodali od zadnje potrditve. Z metodo povleci in spusti se ureja tudi vrstni red medijev.

Z gumbom Shrani shranimo vse spremembe v bazo in v tem trenutku so na voljo predvajalniku, če je seveda kakšen predvajalnik povezan s tem kanalom.



Slika 11. Urejanje parametrov kanala

Možni parametri:

Parameter	Privzeta vrednost
Dolžina	Dedovano od medija
Podnapis	
Višina podnapisa v pikah	20
Barva ozadja podnapisa	Črna
Pozicija podnapisa v % zaslona	98
Pozicija medija od leve strani(od, do) v procentih	0 - 100
Pozicija medija od zgoraj (od, do) v procentih	0 - 100
Prikaži video (od - do) v sekundah	0 - dedovan od medija
Prehod	0 (brez prehoda)
Raztegni medij	Ne

4.5.2.1 Povleci in spusti

V tem poglavju je predvsem zanimiva rešitev povleci in spusti. Rešitev je narejena s pomočjo jQuery UI knjižnice s tako imenovano interakcijo Sortable in CSS.

Interakcija Sortable je rešitev, ki nam omogoča, da v seznamu HTML, torej v seznamu z značko <ul>, urejamo elemente po principu povleci in spusti. Ta seznam lahko tudi povežemo z drugim seznamom in tako prenašamo elemente iz enega seznama v drugega. V našem primeru imamo levi seznam, ki je seznam možnih elementov, in desni seznam, ki prikazuje aktivne elemente.
Primer – HTML:

```
<style>
  #sortable1, #sortable2 { list-style-type: none; margin: 0;
    padding: 0 0 2.5em; float: left; margin-right: 10px; }
  #sortable1 li, #sortable2 li { margin: 0 5px 5px 5px;
    padding: 5px; font-size: 1.2em; width: 120px; }
</style>
 <ul id="sortableLeft" class="connectedSortable">
    <li class="ui-state-default">Element levi 1</li>
    <li class="ui-state-default">Element levi 2</li>
    <li class="ui-state-default">Element levi 3</li>

</ul>
<ul id="sortableRight" class="connectedSortable">
    <li class="ui-state-highlight">Element desni 1</li>
    <li class="ui-state-highlight">Element desni 2</li>
</ul>
```

HTML koda prikazuje dva seznama. Imata skupen class selektor connectedSortable in vsak svoj id selektor[1] - sortableLeft oziroma sortableRight.

[1] V html jeziku imam ID selektor in class selektor, ki imata podobne lastnosti. V jeziku CSS jih ločimo z znakovno predpono lojtra (#) oziroma piko (.). Prvi se uporablja, ko imamo samo en element takega tipa, drugi pa, ko imamo več istih tipov na strani.

Javascript:

```
<script>
$(function() {
    $( "#sortableLeft, #sortableRight" ).sortable({
        connectWith: ".connectedSortable"
    }).disableSelection();
});
</script>
```

Knjižnico JQuery in interakcijo sortable uporabimo na zgoraj opisan način v programski kodi JavaScript. V našem primeru smo vključili oba seznama. Seveda morata imeti isti class selektor.

DisableSelection je sicer nedokumentiran pripomoček JQueryUI knjižnice, ki nam onemogoči označevanje besedila. Tako se izognemo dvoumnosti, saj je velika podobnost med povleci in spusti ter označevanjem besedila. V tem primeru se ne da označiti besedila za npr. kopiranje.

Imamo dva seznama, med katerima lahko elemente premikamo z metodo povleci in spusti, vendar pa potrebujemo še dodatne možnosti:

- med aktivne medije (desni seznam) lahko povlečemo več istih medijev iz dostopnega seznama (levi seznam),
- omogočiti hočemo iskanje po levem seznamu,
- omogočiti moramo shranjevanje desnega seznama,
- v desnem seznamu moramo imeti možnost urejanja parametrov.

Seznam aktivnih medijev mora biti dinamičen in se mora osvežiti ob vsakem premiku. Implementiramo klic Ajax, ki dobi seznam medijev, ki so na voljo in ki ustrezajo izbranemu albumu ter izbranemu pogoju za iskanje. Klic Ajax izvedemo pri vsakem premiku povleci in spusti in vsakič znova osvežimo seznam dostopnih medijev. Izkaže se, da za dogodek premika obstaja v knjižnici jQuery sprožilec, kateremu določimo osveževanje seznama z dostopnimi mediji, s čimer zagotovimo, da je vedno na voljo vsa vsebina v seznamu dostopnih medijev.

Primer uporabe metode sortable s pripadajočo metodo za osveževanje seznama:

```
<script>
$(function() {
    $( "#contentLeft, #contentRight" ).sortable({
        connectWith: ".connectedSortable",
        opacity: 0.6,
        cursor: 'move',
        update :function () {
                RefreshAvailableChannels(current_channel);
            }
    }).disableSelection();
});
</script>
```

Določimo še transparentnost objekta, ki ga premikamo, za bolj atraktiven izgled.

Za shranjevanje desnega seznama objamemo desni seznam HTML v POST formo in opremimo vsak element seznama še s skritim html poljem z unikatno vrednostjo medija. Kot ime mu podamo seznam z imenom media za kasnejše pridobivanje informacij o vrstnem redu.

```
<input type="hidden" name="media[]" value="'. $line["id_media"]  .'" />
```

Rešiti je potrebno še zadnjo točko, s katero želimo doseči, da se elementi na levi strani razlikujejo od desnih. To dosežemo s pomočjo CSS in JavaScript. Ideja je, da levi in desni seznam sestavimo na enak način, pri tem pa vse elemente, ki se ne smejo videti, skrijemo, a jim še vedno nastavimo privzete vrednosti. V CSS nastavimo drugačne lastnosti levim in desnim elementov. Pri sklicevanju na sekcije v jeziku html izhajamo iz starša[2], ki predstavlja levi oziroma desni seznam.

Seznam ima naslednjo strukturo:

```
<ul id="contentLeft" class="connectedSortable">
    <li class="contentLeftUsedli">' ali <li class="contentLeftNotUsedli">
        <table width="100%" border="0">
        <tr>
        <td>
        <a class="additionalbuttons">
            <input type="submit" class="button_large" value="+"
                onclick="AdditionalShowHide('.$r.', 1); return false;" />
            <input type="submit" class="button_large" value="-"
                onclick="AdditionalShowHide('.$r.', 0); return false;" />
        </a>
        <input class="button" type="hidden" name="media[]"
               value="$line['id_media']" />'
        <a class="additional" >
        <table width="100%" border="0" id="table_inner_chanel">...
            <td>Dolina</td>
            <td><a id="ShowHideMediaLength'.$r.'" class="input_length"><input
                   type="text"
                name="playtime[]" value=" $line['playtime']"/>'sekund'
                </a>
            </td>
            ...
            + ostale vnosna polja parametrov
        </td>
        </tr>
        </table>
    </li>
<ul>
```

[2] Strukturo HTML lahko razumemo kot drevesno strukturo. V jeziku CSS pa se lahko sklicujemo na točno določen element oziroma nabor elementov, začenši pri nekem poddrevesu.

Da na levi strani elementi niso vidni, vsem elementom HTML v CSS nastavimo lastnost:

```
#contentLeft li a.input_title, #contentLeft li a.input_length,
#contentLeft li a.input_translation, #contentLeft li a.additional,
#contentLeft li a.additionalbuttons, #contentLeft li a.input_font_size,
#contentLeft li a.input_background_color, #contentLeft li a.input_subtitle_top,
#contentLeft li a.input_position_top1, #contentLeft li a.input_position_left1,
#contentLeft li a.input_position_top2, #contentLeft li a.input_position_left2,
#contentLeft li a.input_is_child, #contentLeft li a.input_video_start,
#contentLeft li a.input_video_finish, #contentLeft li a.input_is_global_window,
#contentLeft li a.input_do_strech_media
{
    display:none;
}
```

CSS je podoben za desno stran, vendar tam spustimo additionalbuttons, torej gumbe, s katerimi lahko dostopamo do parametrov. Na dogodek klika gumbom nastavimo funkcijo JavaScript, s katero preostalim elementom določimo, da naj bodo vidni ali ne. Uporabljena je metoda jQuery za skrivanje in prikaz HTML elementa, in sicer hide in show. Prvi parameter nam pove čas prehoda za bolj atraktiven prehod iz skritega v prikazan pogled.

```
<script>
function AdditionalShowHide(element_id, direction)
{
    if (direction == 1)
    {
        $("#ShowHideMediaLength"+element_id).show(300,'linear');
        + drugi elementi
    }
    else
    {
        $("#ShowHideMediaLength"+element_id).hide(300,'linear');
        + drugi elementi
    }
}
</script>
```

4.5.3 Vtičnik

Vtičniki v logičnem smislu predstavljajo povezavo med kanalom in predvajalnikom. Zaradi praktičnih razlogov se predvajalnik vključi na vtičnik, ta pa je povezan z določenim kanalom. Sicer bi se lahko predvajalnik povezoval direktno na kanal, vendar se je izkazalo, da imamo bistveno večjo centralno podporo, če vmes definiramo še vtičnik. Vsak vtičnik pa je lahko povezan natanko z enim predvajalnikom. Zakaj?

- Kadarkoli lahko na vtičniku spremenimo kanal in nam za spremembo ni potrebno posegati po predvajalniku.
- Nadgradnja vtičnikov je možna s časovniki, ki bi spreminjali kanale glede na določeno obdobje. V določenih urah bi lahko predvajali drug kanal.
- V primeru, da kupcu zaračunavamo uporabnino, je to ena od možnosti.

4.6 Uporabniki in grupe uporabnikov

Uporabniki se delijo na tri nivoje. Vsak uporabnik pripada eni vnaprej definirani grupi uporabnikov[3]. V splošnem je vzdrževalec sistema v grupi uporabnikov z vsemi pravicami. Sledi vzdrževalec grupe, ki skrbi za vse uporabnike v svoji grupi. Uporabniki seveda nimajo pravice do urejanja uporabnikov, razen svojih podatkov, lahko pa urejajo vsebine.

Grupa uporabnikov	Opis
Uporabniki	Urejajo lahko medije in kanale
Administratorji	Vse, kar ima grupa Uporabniki, + urejanje vtičnikov + urejanje uporabnikov v isti grupi
Globalni administratorji	Vse, kar ima grupa Administratorji, + urejanje uporabnikov v katerikoli grupi

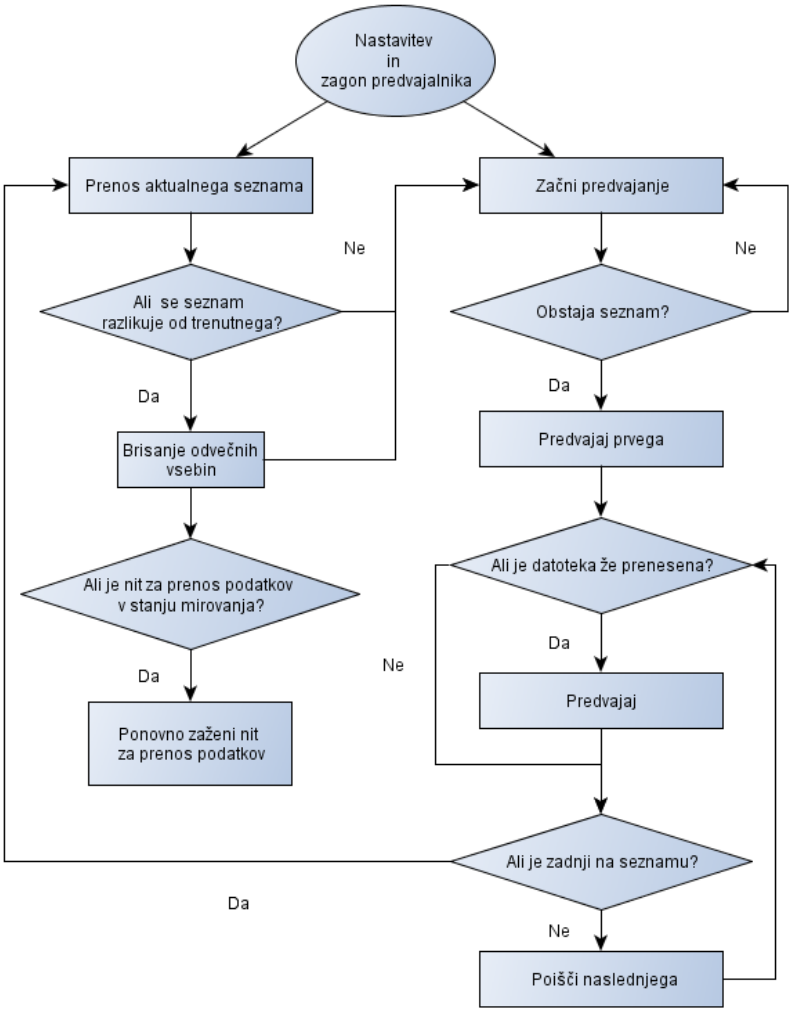
[3] Gre za grupo, ki predstavlja nivo uporabnika, in ne za grupo, kot je opisana v tem delu.

5 Predvajalnik

Predvajalnik oziroma klient je v celoti napisan v programskem okolju Delphi. Ker v tem trenutku obstaja prevajalnik Delphi samo za Windows okolje, je prevajalnik omejen na to platformo. Za lažje delo smo uporabili že narejene knjižnice za delo s podatki in večpredstavnostnimi vsebinami.

Uporabljene zunanje knjižnice in komponente:

- komponenta[4] kbmMemTable od Components4Developers,
- zbirka knjižnic za delo s slikami in videi ImageEn.



Slika 12. Diagram poteka predvajalnika

[4] Ime komponenta uporabljamo za vse razrede, ki so nasledniki razreda TComponent. Skupna lastnost komponentam je, da jih lahko postavimo na obrazec, kjer lahko vizualno urejamo lastnosti in dogodke, prevajalnik pa poskrbi za kreiranje in inicializacijo komponent ob kreiranju obrazca.

5.1 Uporabljene knjižnice in pristopi

5.1.1 kbmMemTable

Komponenta za lokalno delo s podatki. Predstavimo jo lahko kot lokalno shranjeno tabelo, ki se obnaša kot set podatkov, pridobljenih iz tabele iz baze.

Prirejena je za lokalno delo, vsi podatki pa so shranjeni v pomnilniku. Omogoča tudi shranjevanje in nalaganje vsebin v oziroma iz datoteke. Format datoteke določimo poljubno, definirani pa so že določeni formati.

Komponento smo izbrali za delo s seznami za predvajanje, ima pa naslednje uporabne lastnosti:

- Je hitra, podatki so v pomnilniku ne potrebuje nobene podatkovne baze za delovanje.
- Komponento lahko povežemo eno na drugo. Tako preko večih delov programa neodvisno dostopamo do podatkov in jih urejamo. Npr. v niti, ki skrbi za prenos vsebin, lahko spremenimo status datoteke iz neprenesene v preneseno.
- Omogoča shranjevanje in ponovno nalaganje seznama. V primeru, da se program zapre in ponovno odpre v času, ko je brez povezave do strežnika, se naloži zadnji shranjeni seznam.
- Omogoča filtriranje in sortiranje podatkov.

5.1.2 ImageEn

Knjižnica za delo s slikami in videi. Knjižnica vsebuje ogromno zbirko komponent za obdelavo, analizo in prikaz tako slik kot tudi video vsebin.

Za projekt pomembne funkcionalnosti knjižnice:

- prikaz slik in video vsebin,
- sprotna sprememba velikosti vsebine,
- lepi prehodi med dvema vsebinama,
- omogoča sloje – primerni za sliko v sliki in napise.

Primer prikaza slike in prehoda:

```
ImageEnView1.io.LoadFromFile('slika.jpg'); //naložimo sliko
ImageEnView1.Show; //prikažemo sliko na zaslonu z TImageEnView komponento
  //ali
ImageEnView1.RunTransition(TIETransitionType(4) , 300);
  //naredimo prehod z indeksom 4 v času 300ms
```

5.1.3 Niti

V programskem okolju Delphi se nit implementira kot razred, izpeljan iz razreda TThread. Nit ima dve stanji, in sicer je lahko v mirovanju ali pa v izvajanju. V izvajanju se pokliče metoda Execute, po končani metodi pa se nit sprosti. Naša nit nikoli ne zapusti metode Execute, saj imamo v njej neskončno zanko. Nit, ki skrbi za prenos podatkov, se spravi v mirovanje v trenutku, ko zaključi s prenosom vseh potrebnih datotek.

Osnovna deklaracija niti:

```
TDownloadThread = class(TThread)
protected
  procedure Execute; override;
public
  constructor Create();
  destructor Destroy; override;
end;

//implementacija
constructor TDownloadThread.Create();
begin
  inherited Create(True); //kreiramo nit v mirovanju
end;
destructor TDownloadThread.Destroy;
begin
  inherited;
end;
procedure TDownloadThread.Execute;
begin
  while True do
  begin
    while not Suspended do
    begin
     // koda niti
    end;
  end;
end;
```

Problem nastane, ko želimo popravljati podatke, ki so v drugi niti. Nekako moramo zagotoviti, da popravljamo podatke samo mi. V programskem jeziku Delphi imamo dva razširjena pristopa, in sicer nitni razred Synchronize in semaforje (t.i. CriticalSection). Poznamo pa tudi drug pristop za reševanje sinhronizacijskih problemov, in sicer preko Windows Messages, kot je opisano v [13].

5.1.3.1 Synchronize

Metodo uporabljamo pri dostopanju do uporabniških objektov na formi. V samem razredu niti določimo metodo, ki skrbi za dostop do objektov. Metodo pokličemo s pomočjo vgrajene metode Synchronize.

Primer:

```
procedure TMyThread.DoProgress;
begin
    FProgressBar.Position := FProgressBar.Position + 1;
end;

procedure TMyThread.Execute;
begin
//...
 Synchronize(DoProgress) ;
//...
end;
```

5.1.3.2 CriticalSection

Implementiramo semafor, ki se obnaša, kot bi imeli en sam proces (Slika 13). Definiramo spremenljivko tipa TCriticalSection, ki mora biti vidna vsem, ki dostopajo do določenega objekta, za katerega nočemo hkratnih dostopov. Bloku kode nastavimo začetek in konec, vmes pa je tako imenovana kritična sekcija. Hkrati se ne moreta izvajati dva bloka kode, ki pripadata isti sekciji, in tako zagotovimo varno dostopanje do objektov. Seveda morajo biti te sekcije kar se da kratke oziroma časovno nezahtevne, saj mora imeti nit prost vstop v sekcijo, sicer mora čakati.

Slika 13. CriticalSection

Primer:

```
//deklariramo spremenljivko
var CriticalSection: TCriticalSection;
//inicializiramo kritično sekcijo
CriticalSection := TCriticalSection.Create;
//vstopimo v kritično sekcijo
procedure TMyThread.Execute;
begin
    //...
    CriticalSection.Enter;
    //koda
    //izstopimo iz kritične sekcije
    CriticalSection.Leave;
    //...
end;
```

5.1.3.3 Windows Messages

V okolju Windows je nekako najbolj uveljavljena metoda za komunikacijo med programi Windows Messages. V splošnem je sporočilo poslano oknu programa. To sporočilo se shrani v Message Queue, torej v neki vrsti v pomnilniku, sporočila pa se nato pošljejo ciljnemu programu oziroma obrazcu programa.

V našem primeru iz ene niti pošljemo v glavno nit sporočilo z ustreznimi parametri. V glavni niti definiramo funkcijo, ki procesira ta sporočila in jih obdeluje. Torej v splošnem definiramo v glavni niti vse funkcije, katere spreminjajo skupne objekte.

Definiramo konstanto, ki predstavlja našo številko sporočila:

```
const
  TH_MESSAGE = WM_USER + 1;
```

Nato definiramo proceduro, ki servisira ta sporočila:

```
procedure MessageFromThread( var Message : TMessage ); message
TH_MESSAGE;

procedure TMainForm.MessageFromThread(var Message: TMessage);
var mailinfo: PInfoFromThread;
begin
  case Message.WParam of
  TH_SEND: begin
   mailinfo := PInfoFromThread(Message.LParam);
      if fMWServerAdm1.edMailSendLogEnable.Checked then
        fMWServerAdm1.edMailSenderLog.Lines.Add(mailinfo^.sMessage);
    end;
  end;
end;
```

Podamo poljubne podatke in pošljemo z metodo PostMessage.

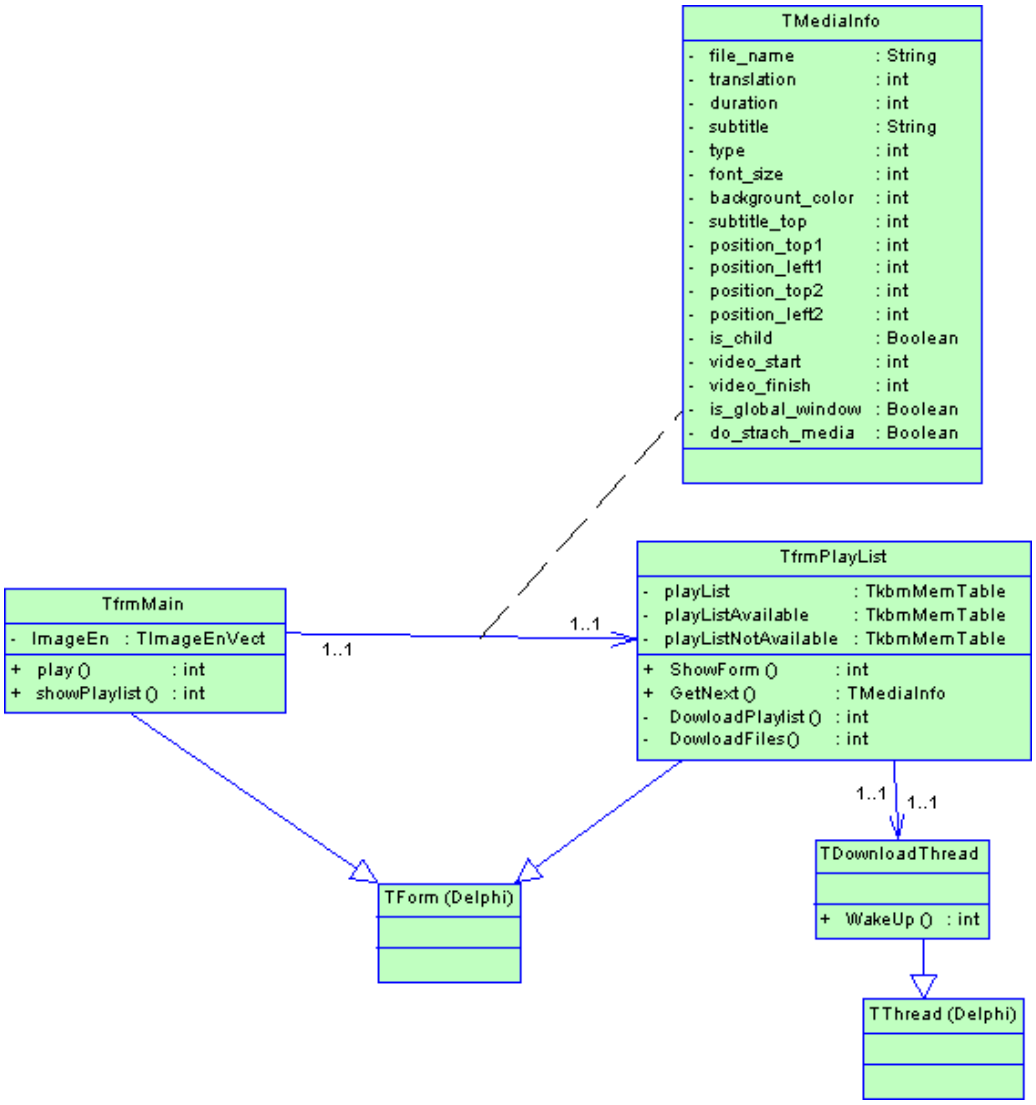
```
var
PInfoFromThread = ^TThreadMsgRecord;

TThreadMsgRecord = record
poljubniparameter1:  integer;
poljubniparameter2: string[255];
end;

threadmsg: PInfoFromThread;
//..
begin
//..
  threadmsg^.poljubniparameter1 := 0;
  threadmsg^.poljubniparameter2 := 'Tekst';
  PostMessage(frmSitiServerAdm.Handle, TH_MESSAGE, TH_SEND, integer(threadmsg));
//..
end;
```

5.2 Opis

Aplikacija je sestavljena iz glavne niti, ki skrbi za prenos seznama za predvajanje in za samo predvajanje. Ker pa želimo karseda hitro začeti predvajanje vsebin, bomo začeli predvajati vsebine še preden so vse prenesene. Torej potrebujemo v seznamu še informacijo, ali je vsebina na voljo, poleg tega pa moramo implementirati novo nit, ki skrbi za prenos vsebin. Nit za prenos vsebin kreiramo ob zagonu aplikacije.

5.3 Organizacija

Pri inicializaciji aplikacije se kreirata dva obrazca[5] oziroma razreda, podedovana iz razreda TForm, in sicer glavni TfrmMain in TfrmPlayList. Glavni obrazec je prikazan tudi na zaslonu in skrbi za prikazovanje vsebin, medtem ko se obrazec frmPlayList pokaže na zahtevo, servisira pa glavni razred z informacijami o tem, kaj mora predvajati. Obrazca sta definirana v Delphi enoti[6] uMain in uPlayList. Diagram (Slika 14) prikazuje celotno strukturo predvajalnika.



Slika 14. Razredni diagram predvajalnika

[5] Izraz obrazec uporabljamo za vse razrede, ki so podedovani iz razreda TForm. Takemu razredu pripada vizualni obrazec, na katerem imamo lahko definirane komponente.

[6] Enota oziroma angleško unit. V programskem jeziku Delphi je to datoteka s končnico pas. V tej enoti so definirani razredi, funkcije, spremenljivke, tipi itd.

5.4 Implementacija

5.4.1 Seznam za predvajanje

Seznam se osvežuje pri vsakem zagonu in vsakič, ko se predvajanje seznama zaključi. Za sezname skrbi komponenta mtPlayList, ki je tipa TkbmMemTable ter hrani elemente tipa TMediaInfo. Seznam dobi preko spletne storitve implementirane na strežniku. Ob vsaki spremembi seznama se seznam shrani tudi v lokalno datoteko za primer, če se aplikacija načrtovano ali nenačrtovano zapre. Ko se aplikacija ponovno zažene, se seznam ponovno naloži in seveda tudi osveži, če je mogoče (če obstaja povezava do strežnika). V kolikor ni povezave do strežnika, se predvaja shranjeni seznam.

Na komponento mtPlayList sta povezani še dve komponenti istega tipa, ki imata nastavljene filtre, da prikazujeta podatke o vsebinah, ki so že na voljo, in tiste, ki niso. Ta dva seznama se uporabljata za pridobivanje vrstnega reda vsebin, ki so na voljo, in seznama vsebin, ki jih moramo še prenesti. Omenjena povezava komponent je izredno priročna, saj delamo nad enimi podatki.

5.4.2 Sinhronizacija datotek

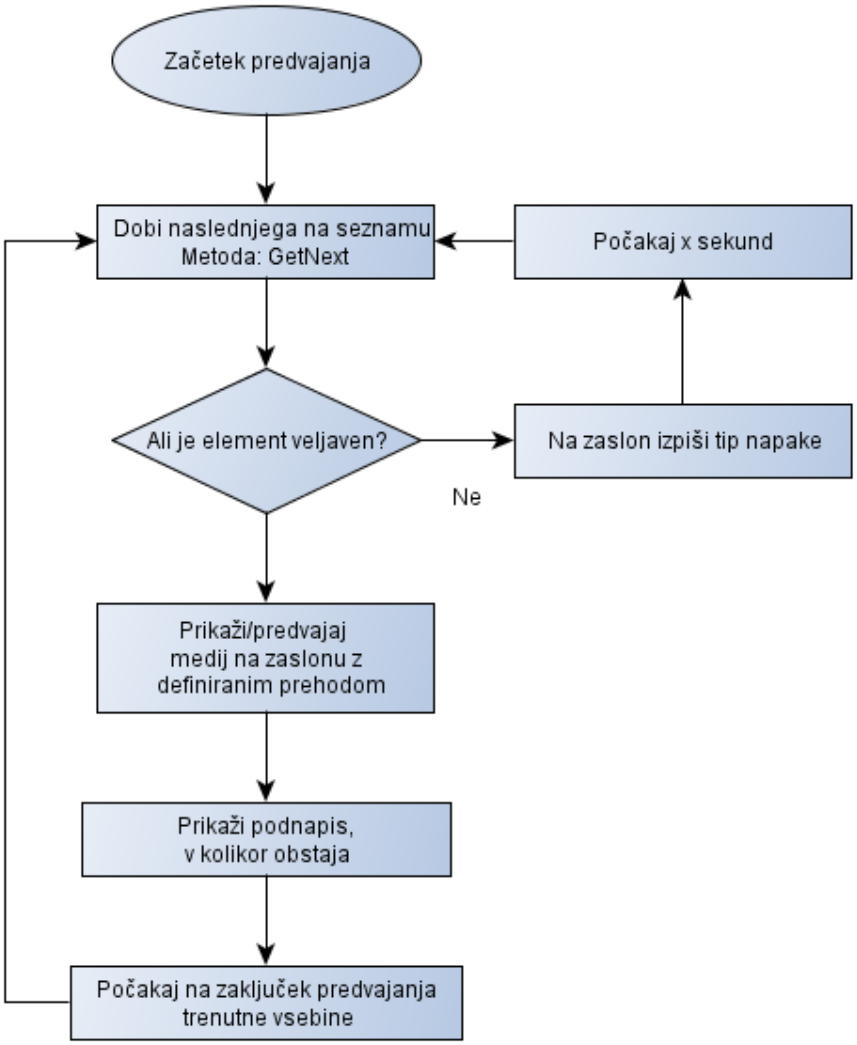
Za sinhronizacijo datotek skrbi posebna nit, ki jo kreiramo iz glavne niti. Metoda Execute vsebuje neskončno zanko, torej se nit nikoli ne zaključi. Nit pregleda vse še neprenesene datoteke, nato pa gre v mirovanje. V primeru spremembe seznama nit iz glavne niti ponovno zbudimo. Postopek skrbi za prenos datotek in posodabljanje statusa na seznamu predvajanja. Dostop do podatkov seznama je v drugi niti in za dostop do teh podatkov je potrebno uporabiti neko metodo za varen dostop do podatkov med nitmi. Uporabili smo semaforje, torej metodo CriticalSection (opisano v poglavju 0). V dodatku B je prikazana implementacija niti na sinhronizacijo datotek z uporabo semaforjev.

5.4.3 Prikaz vsebin

Predvajanje multimedijskih vsebin se začne takoj ob zagonu programa. Kliče se metoda seznama GetNext, ki vrne informacije o mediju. Metoda vrača samo že prenesene medije. V kolikor pride do napake, se na zaslonu izpiše tip napake. Potek je prikazan na sliki (Slika 15)

Tipi napak:

- Seznam je prazen.
- Ni dostopa do strežnika (v kolikor je seznam prazen in ni dostopa do strežnika).
- Sinhronizacija medijev (ko še noben medij ni prenesen, seznam pa obstaja).



Slika 15. Diagram poteka predvajanja vsebin

V kolikor gre za veljavno vsebino, se predvajalnik odzove glede na tip datoteke.

5.4.3.1 Slikovni prikaz

Za prikaz vsebin smo uporabili sloje, torej ne rišemo direktno na osnovno površino. To nam omogoča, da lahko poljubno postavljamo vsebine na želena mesta in morebiti eno čez drugo, kot bomo videli kasneje v poglavju 5.4.3.3.

Ker se slika ne prilagaja zaslonu, določimo tudi ozadje. Uporabnik ima možnost izbire barve ozadja za vsak medij posebej. Če imamo nastavljen parameter za prilagajanje zaslonu, sliko ustrezno povečamo ali pomanjšamo sorazmerno na robova, da ohranimo pravilno razmerje med višino in širino, nato pa kličemo metodo za izris, v našem primeru s prehodom. Na koncu še poženemo časovnik, ki se sproži po nastavljenem času. Časovnik ob sprožitvi kliče metodo ShowNext, ki ponovi postopek z novim medijem.

Programska koda:

```
ImageEnView.LayersAdd;
//naloimo datoteko
ImageEnView.io.LoadFromFile(MediaInfo.FileName);
//nastavimo ozadje
ImageEnView.Background := MediaInfo.Background_color;
//spremenimo velikost
ImageEnProc.Resample(frmMain.Width,frmMain.Height, rfFastLinear,
                     not MediaInfo.do_strech_media);
//postavimo na sloj
FitPictureOnLayer(nLayer, mediainfo.IsPicture,
                    MediaInfo.do_strech_media);
ImageEnView.PrepareTransition;
//naredimo prehod
ImageEnView.RunTransition(MediaInfo.Transition, 2000);
//pretvorimo cas prikaza v sekunde
tNextMedia.Interval := MediaInfo.Duration*1000;
//poenemo casovnik
tNextMedia.Enabled  := True;
```

5.4.3.2 Video prikaz

Video prikaz poteka na podoben način kot prikaz slike, vendar se pri video prikazu ignorira čas prikaza, upoštevata pa se začetek in konec predvajanja. Za začetek predvajanja enostavno povemo ImageEn komponenti, kje naj začne predvajati video vsebino. Malenkost drugačna pa je implementacija konca predvajanja video vsebine, če je nastavljeno, da se vsebina ne predvaja do konca. Pri sliki smo pognali časovnik, tukaj pa bomo kreirali proceduro, ki jo bomo dodelili dogodku komponente ImageEn. Dogodek se imenuje ShowNewFrame, proži pa se ob vsakem prikazu sličice, kar pomeni na najmanjši možen časovni interval videa. V tem dogodku preverimo trenutno zaporedno številko sličice in jo primerjamo z našo izračunano. Računanje zaporedne številke sličice v video vsebini je dokaj lahko opravilo, če imamo informacijo o tem, koliko sličic na sekundo vsebuje naš video zapis. Izkaže se, da nam komponente ImageEn omogočajo pridobivanje informacij o video parametrih, med katerimi je tudi število sličic na sekundo.

```
var media:TIEMediaReader;
//…
media := tTIEMediaReader.create(mediainfo.filename);
try
  //izracunamo pri kateri zaporedni sli諨i v video
  //zapisu moramo prenehati s predvajanjem
  cframesdurationfinish :=
    Trunc((mediainfo.video_finish - mediainfo.video_start) *
            media.framerate);
finally
  media.free;
end;
```

5.4.3.3 Podnapisi

Podnapise želimo prikazovati na zaslonu ob prikazovanju medija. Podnapis je lahko daljši kot je širina zaslona, zato smo se odločili, da podnapis potuje po zaslonu z desne proti levi. V kolikor podnapis preide čez celoten zaslon, se začne postopek prikazovanja ponovno. Primer prikaza podnapisa je prikazan na sliki (Slika 16).

Slika 16. Primer predvajalnika pri prikazovanju slike in podnapisa

Za podnapise naredimo svoj sloj, ki je nad slojem, ki prikazuje video vsebino ali sliko. Za premikanje podnapisa uporabimo časovnik.

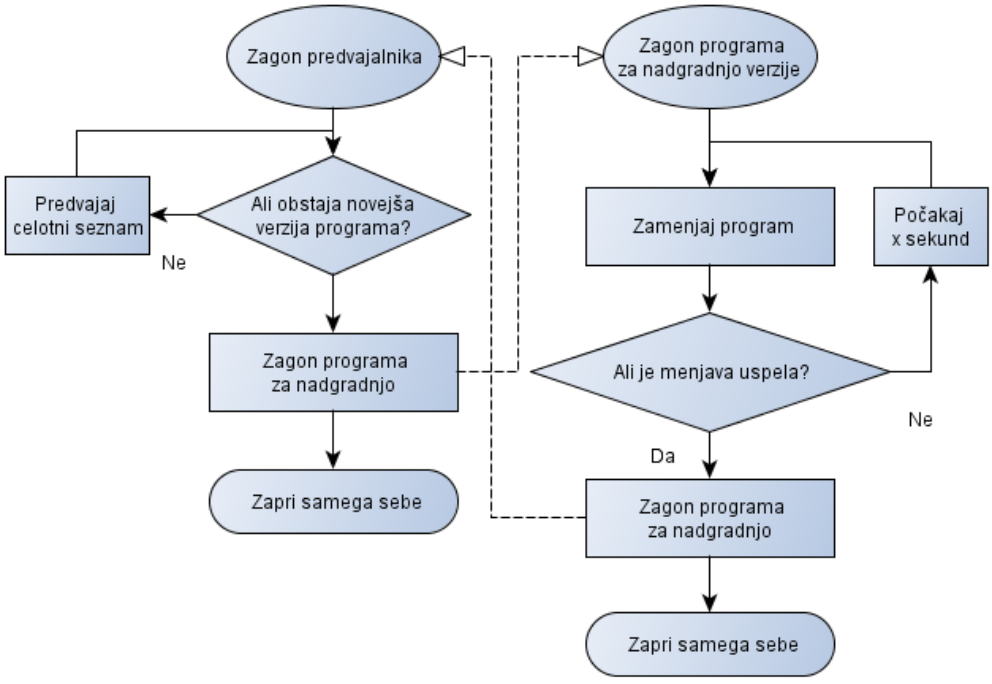
6 Pomožne aplikacije in servisi

V uvodu sem omenil, da je cilj izdelati kompleten celovit in zanesljiv sistem. Administracijski del je nameščen na strežniku, kjer imamo popoln nadzor in dostop, tako da nadgradnje ne predstavljajo problema. Težava je pri predvajalnikih, kjer nimamo dostopa do sistema, poleg tega pa je lahko število predvajalnikov veliko. Ročna nadgradnja ne pride v poštev, zato smo razvili sistem za avtomatsko nadgradnjo.

V drugem delu tega poglavja bomo pogledali, kako je narejeno pridobivanje informacij o programu in podatkov o najnovejši verziji na strežniku.

6.1 Nadgradnja predvajalnika

Nadgradnja predvajalnika je sestavljena iz servisnega programa, ki se vključi med Windows servise, in programa, ki zamenja najnovejšo verzijo s trenutno. Servis skrbi, da je v določeni mapi vedno zadnja verzija programa. V praksi je to navadno kar podmapa predvajalnika z imenom Nadgradnja. Predvajalnik ob zagonu in po vsakem zaključku predvajanja seznama preveri, ali je v tej mapi na voljo novejša verzija. V kolikor ugotovi da obstaja, požene pomožni program CmisUpgradeVersion, ki posodobi verzijo, in ponovno zažene novo verzijo programa. Tako zagotovimo, da imajo vsi klienti zadnjo verzijo. Na sliki (Slika 17) je prikazano, kako predvajalnik in program sodelujeta ter potek delovanja.



Slika 17. Diagram poteka nadgradnje predvajalnika

Preverjanje verzij programa poteka preko standardnih podatkov izvedljive datoteke. Številčenje mora potekati po standardu. Torej verzija vsebuje štiri številke ločene s piko, in sicer »major.minor.build.revision«, kjer številka major predstavlja največjo vrednost, revision pa najmanjšo. Primer: 5.1.17.1018. Tak način nam omogoča, da ugotovimo, katera verzija je novejša.

Primer pridobivanja podatkov o verziji iz izvršilne datoteke:

```
var
  VerInfoSize: DWORD;
  VerInfo: Pointer;
  VerValueSize: DWORD;
  VerValue: PVSFixedFileInfo;
  Dummy: DWORD;
//...
try
VerInfoSize := GetFileVersionInfoSize(PChar(sFile), Dummy);
GetMem(VerInfo, VerInfoSize);
GetFileVersionInfo(PChar(sFile), 0, VerInfoSize, VerInfo);
VerQueryValue(VerInfo, '\', Pointer(VerValue), VerValueSize);
with VerValue^ do
begin
  Result.Major := dwFileVersionMS shr 16;
  Result.Minor := dwFileVersionMS and $FFFF;
  Result.Release := dwFileVersionLS shr 16;
  Result.Build := dwFileVersionLS and $FFFF;
end;
FreeMem(VerInfo, VerInfoSize);
except
 //napaka
end;
```

Zgornja koda je del razreda, ki ga uporabljamo tako v predvajalniku, kot tudi servisu in programu, ki skrbi za zamenjavo verzij. Razred se imenuje TFile, hrani pa vse informacije o trenutnih produkcijskih datotekah in datotekah za nadgradnjo. Omogoča tudi primerjavo in zamenjavo verzij ter zagon programov. Skratka, je kompleten razred za nadgradnjo.

Predvajalnik uporablja razred za preverjanje verzije za nadgradnjo. Servis uporablja razred za preverjanje verzije na strežniku in preverjanje lokalne verzije za nadgradnjo. Program za zamenjavo pa uporablja razred za preverjanje produkcijske verzije z verzijo za nadgradnjo, za zamenjavo verzij ter za sam zagon predvajalnika.

6.2 Spletne storitve (Web service)

Univerzalna komunikacija preko mreže ali interneta so spletne storitve. Na strežniku smo kreirali spletno storitev za pridobivanje seznama za predvajanje z vsemi potrebnimi parametri. V seznamu je tudi pot do datoteke, ki jo mora kasneje predvajalnik naložiti preko protokola HTTP. Spletno storitev smo napisali v programskem jeziku PHP. Definirati moramo funkcije in njihove spremenljivke ter rezultat in parametre rezultata.

Klicana funkcija je v našem primeru getMedia s tremi parametri: številka vtičnika, uporabniško ime in geslo. Definicija odgovora je getMediaResponse, ki vsebuje celoten seznam medijev s pripadajočimi parametri.

Nato moramo v programskem jeziku PHP definirati tudi servis. Najprej moramo pripraviti podatke, ki so konsistentni s podatki WSDL. To naredimo v programskem jeziku tako, da za vhodne in izhodne parametre naredimo razreda s pripadajočimi spremenljivkami:

```
class Media {
  public $id_media; // int
  public $location; // string
  public $file; // string
  public $playtime; // int
  public $translation; // int
  public $title; // string
  public $font_size; // int
  public $type; // int

  public $background_color; // int
  public $subtitle_top; // int
  public $position_top1; // int
  public $position_left1; // int
  public $position_top2; // int
  public $position_left2; // int
  public $is_child;  // boolean
  public $video_start; // int
  public $video_finish; // int
  public $is_global_window; // boolean
  public $do_strech_media;  // boolean
}

class getMedia {
  public $id_port; // string
  public $username; // string
  public $password; // string
}
```

Naredimo še razred CmisWebService:

```
class CmisWebService
{
  function getMediaList($params)
    {
     //iz baze naloži vse podatke, ki ustrezajo pogojem
     //naredi polje
    $positions = array();
      while (//pojdi po vseh vrsticah*/)
      {
        $position= new Media();
       //preveri če je video vsebina že konvertirana
       $positions[] = $position; //sestavi polje
      }
    return $positions; //vrni polje vseh medijev
    }
  function getMedia($params)
    {
        return $this->getMediaList($params);
    }
}
```

Nato naredimo preslikavo med razredom PHP in razredom WSDL:

```
$classmap = array('Media' => 'Media',
                  'getMedia' => 'getMedia',
                  );
```

Kreiramo objekt in mu določimo zgornjo direktivo ter razred CmisWebService. Poženemo servis:

```
$server = new SoapServer('CmisWebService.wsdl',array('classmap'=>$classmap));
$server->setClass("CmisWebService");
$server->handle();
```

S tem je zaključen del na strežniku, to spletno storitev pa moramo uporabiti tudi na klientu. Izkaže se, da lahko z razvojnim okoljem Delphi ta servis enostavno uvozimo. Z uvozom se generirajo ustrezni razredi z vsemi želenimi parametri, ki so definirani v WSDL.
Nato uporabimo že vgrajeno komponento tipa THTTPRIO ter ji nastavimo ustrezne parametre in servis. S klicem definirane spletne metode getMedia pridobimo podatke na klienta:

```
HTTPRIO := HTTPRIO.Create(self);
HTTPRIO.WSDLLocation := uIni.URLWebService;
HTTPRIO.Service := 'CmisService';
HTTPRIO.Port := 'CmisServiceHttpSoap12Endpoint';
_CmisServicePortType := (HTTPRIO as CmisServicePortType);
try
    arr :=  _CmisServicePortType.getMedia(inttostr(idPort), '1', '1');
    for i:= 0 to length(arr) - 1 do
begin //naredimo seznam
    //...
    //podaki so na voljo
    kbmPlayListDuration.AsString := arr[i].playtime;
    kbmPlayListTransition.AsInteger := arr[i].translation;
    kbmPlayListSubTitle.AsString := arr[i].title;
    //...
end;
```

V dodatku A je priložena datoteka WSDL za sinhronizacijo seznama.

7 Zaključek

V diplomskem delu je opisana rešitev centraliziranega multimedijskega sistema za urejanje vsebin, ki dislociranim klientom preko spletnih storitev servisira multimedijske vsebine. Tako nam omogoča centralni nadzor nad predvajalniki.

Za razvoj administrativnega dela se je izkazalo, da je internetna aplikacija izredno dobra rešitev, saj nam omogoča dostop od koderkoli, kjer imamo dostop do interneta. Tudi izbira programskega jezika PHP v kombinaciji s podatkovno bazo MySQL se je izkazala za dovolj fleksibilno in hitro rešitev. Razvoj spletnih tehnologij je v zadnjem času šel hitro naprej in nam postregel z raznimi knjižnicami, kot so Javascript knjižnica jQuery in podobne. Te knjižnice nam omogočajo precej napredne in atraktivne metode, ki jih nismo bili vajeni v internetnih aplikacijah. Omenili bi še ogromno skupnost razvijalcev, ki nam omogoča hitro podporo in veliko uporabnih rešitev, katere lahko poljubno kombiniramo in dopolnjujemo.

Omenjena aplikacija na trgu še ni zaživela, vendar je nekaj uporabnikov, ki sistem testirajo in dajejo pripombe. Seveda je še veliko možnosti za izboljšave in nadgradnje, toda namen te naloge sta bili postavitev in implementacija celotne rešitve z zdravim jedrom.

Naštel bi nekaj možnih smernic za nadgradnje, katere imamo namen narediti v prihodnje:

- Podprli bi mnogo drugih multimedijskih formatov, kot so PowerPoint, Word, PDF, YouTube predvajanje, prikaz internetnih strani in podobno. Za omenjene nadgradnje bi morali malenkost spremeniti centralni del in omogočiti predvajanje v prevajalniku, torej bi morali izbrati dodatne knjižnice in jih implementirati.
- Omogočili bi povezovanje grup za deljenje vsebin med grupami.
- Nadgradili bi statistiko v centralnem delu in jo primerno prikazovali v uporabniškem vmesniku v obliki tabel in grafov.
- S prejšnjima dvema nadgradnjama bi lahko omogočili tudi zaračunavanje med uporabniki sistema.

Z vsemi omenjenimi razširitvami bi prišla centralizacija sistema še posebej do izraza.
Pri razvoju predvajalnika bi bilo potrebno podpreti že omenjene druge formate medijev, predvajalnik pa napisati tudi za druge platforme. Velik potencial predstavljajo tablice, ki bi jih lahko montirali v izložbe, trgovine in podobno. Možnosti je ogromno. Omembe vredno je vsekakor tudi okolje FireMonkey, ki je, ravno tako kot Delphi, del razvojnih orodij Embarcadero RAD Studio. Pred kratkim je bila predstavljena druga različica FireMonkey, imenovana FM^2, ki ima nekaj pomembnih izboljšav, med njimi za nas najbolj pomemben je prikaz video vsebin. Orodje omogoča prevajanje aplikacij za različne platforme, med katerimi je tudi iOS in v prihodnje tudi Android, ki sta trenutno najbolj razširjena v svetu tabličnih računalnikov, poleg tega pa cenovno ugodna.

Seznam slik

Literatura in viri

1. B. Bibeault, Y. Katz, *jQuery in Action, Second Edition,* Stamford: Manning Publications Co., 2010.
2. M. Cantu, Mastering Delphi 7, Sybex, Inc., 2003
3. J. Lengstorf, Pro PHP and jQuery, New York: Springer Science+Business Media, LLC., 2010.
4. R. Mansfield, *CSS Web Design For Dummies*, Indianapolis: Wiley Publishing, Inc., 2005.
5. E. Naramore, J. Gerner, Y. Le Scouarnec, J. Stolz, M. K. Glass, *Beginning PHP5, Apache, and MySQL® Web Development,* Indianapolis: Wiley Publishing, Inc., 2004.
6. B. Pfaffenberger, S. M. Schafer, C. White, B. Karow, *HTML, XHTML, and CSS Bible, 3rd Edition*, Indianapolis: Wiley Publishing, Inc., 2004.
7. J. Prado Maia, H. Hayder, L. Gheorghe, *Smarty, PHP Template Programming and Applications*, Birmingham: Packt Publishing Ltd., 2006.
8. D. Sawyer McFarland, *CSS: The Missing Manual, Second Edition*, Sebastopol: O'Reilly Media, Inc., 2009.
9. D. Sawyer McFarland, *JavaScript & jQuery: The Missing Manual, Second Edition*. Sebastopol: O'Reilly Media, Inc., 2011
10. J. Valade, *PHP & MySQL® Everyday Apps For Dummies®*, Indianapolis: Wiley Publishing, Inc., 2005.
11. D. Wellman, *jQuery UI 1.8, The User Interface Library for jQuery*, irmingham: Packt Publishing Ltd., 2011.
12. Components4Developers. Dostopno na: http://www.components4programmers.com/
13. Delphi threading by example. Dostopno na: http://edn.embarcadero.com/article/22411
14. FFmpeg. Dotopno na: http://www.ffmpeg.org/ffmpeg.html
15. ImageEn. Dostopno na: http://www.imageen.com/
16. PHP: GD. Dotopno na: http://php.net/manual/en/book.image.php
17. PHP: Web Services - Manual. Dostopno na: http://php.net/manual/en/refs.webservice.php
18. Plupload. Dostopno na: http://www.plupload.com/documentation.php

Priloge

Dodatek A. WSDL datoteka za sinhronizacijo seznama

```
<?xml version=«1.0« encoding=«UTF-8«?>
<wsdl:definitions                                    xmlns:wsdl=«http://schemas.xmlsoap.org/wsdl/«
xmlns:mime=«http://schemas.xmlsoap.org/wsdl/mime/«     xmlns:http=«http://schemas.xmlsoap.org/wsdl/http/«
xmlns:soap12=«http://schemas.xmlsoap.org/wsdl/soap12/«          xmlns:ns1=«http://org.apache.axis2/xsd«
xmlns:wsaw=«http://www.w3.org/2006/05/addressing/wsdl«
xmlns:ax21=«http://dataobjects.Cmis.vector.at/xsd«                    xmlns:ns=«http://Cmis.vector.at«
xmlns:xs=«http://www.w3.org/2001/XMLSchema«          xmlns:soap=«http://schemas.xmlsoap.org/wsdl/soap/«
targetNamespace=«http://Cmis.vector.at«>
    <wsdl:documentation>
        Cmis/CDP Interface
    </wsdl:documentation>
    <wsdl:types>
        <xs:schema attributeFormDefault=«qualified« elementFormDefault=«qualified«
                targetNamespace=«http://dataobjects.Cmis.vector.at/xsd«>
            <xs:complexType name=«Media«>
                <xs:sequence>
                    <xs:element minOccurs=«0« name=«id_media« nillable=«true«
                            type=«xs:int«></xs:element>
                    <xs:element minOccurs=«0« name=«location« nillable=«true« type=«xs:string«/>
                    <xs:element minOccurs=«0« name=«file« nillable=«true« type=«xs:string«/>
                    <xs:element minOccurs=«0« name=«type« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«playtime« nillable=«true« type=«xs:string«/>
                    <xs:element minOccurs=«0« name=«translation« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«title« nillable=«true« type=«xs:string«/>
                    <xs:element minOccurs=«0« name=«font_size« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«background_color« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«subtitle_top« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«position_top1« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«position_left1« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«position_top2« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«position_left2« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«is_child« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«video_start« nillable=«true« type=«xs:float«/>
                    <xs:element minOccurs=«0« name=«video_finish« nillable=«true« type=«xs:float«/>
                    <xs:element minOccurs=«0« name=«is_global_window« nillable=«true« type=«xs:int«/>
                    <xs:element minOccurs=«0« name=«do_strech_media« nillable=«true« type=«xs:int«/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema xmlns:ax22=«http://dataobjects.Cmis.vector.at/xsd«
                attributeFormDefault=«qualified« elementFormDefault=«qualified«
                targetNamespace=«http://Cmis.vector.at«>
            <xs:import namespace=«http://dataobjects.Cmis.vector.at/xsd«/>
            <xs:element name=«getMedia«>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element minOccurs=«0« name=«id_port« nillable=«true« type=«xs:string«/>
                        <xs:element minOccurs=«0« name=«username« nillable=«true« type=«xs:string«/>
                        <xs:element minOccurs=«0« name=«password« nillable=«true« type=«xs:string«/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name=«getMediaResponse«>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element       maxOccurs=«unbounded«       minOccurs=«0«       name=«return«
                                nillable=«true« type=«ax22:Media«/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name=«getMediaRequest«>
        <wsdl:part name=«parameters« element=«ns:getMedia«/>
    </wsdl:message>
    <wsdl:message name=«getMediaResponse«>
        <wsdl:part name=«parameters« element=«ns:getMediaResponse«/>
    </wsdl:message>
    <wsdl:portType name=«CmisServicePortType«>
        <wsdl:operation name=«getMedia«>
            <wsdl:input message=«ns:getMediaRequest« wsaw:Action=«urn:getMedia«/>
            <wsdl:output message=«ns:getMediaResponse« wsaw:Action=«urn:getMediaResponse«/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name=«CmisServiceSoap11Binding« type=«ns:CmisServicePortType«>
        <soap:binding transport=«http://schemas.xmlsoap.org/soap/http« style=«document«/>
        <wsdl:operation name=«getMedia«>
            <soap:operation soapAction=«urn:getMedia« style=«document«/>
            <wsdl:input>
```

```
                <soap:body use=«literal«/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use=«literal«/>
            </wsdl:output>
        </wsdl:operation>
     </wsdl:binding>
    <wsdl:binding name=«CmisServiceSoap12Binding« type=«ns:CmisServicePortType«>
        <soap12:binding transport=«http://schemas.xmlsoap.org/soap/http« style=«document«/>
        <wsdl:operation name=«getMedia«>
            <soap12:operation soapAction=«urn:getMedia« style=«document«/>
            <wsdl:input>
                <soap12:body use=«literal«/>
            </wsdl:input>
            <wsdl:output>
                <soap12:body use=«literal«/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name=«CmisServiceHttpBinding« type=«ns:CmisServicePortType«>
        <http:binding verb=«POST«/>
        <wsdl:operation name=«getMedia«>
            <http:operation location=«CmisService/getMedia«/>
            <wsdl:input>
                <mime:content type=«text/xml« part=«getMedia«/>
            </wsdl:input>
            <wsdl:output>
                <mime:content type=«text/xml« part=«getMedia«/>
            </wsdl:output>
        </wsdl:operation>
        </wsdl:binding>
    <wsdl:service name=«CmisService«>
        <wsdl:port name=«CmisServiceHttpSoap11Endpoint« binding=«ns:CmisServiceSoap11Binding«>
            <soap:address location=«http://acserver/cmis/services/CmisWebService.php«/>
        </wsdl:port>
        <wsdl:port name=«CmisServiceHttpSoap12Endpoint« binding=«ns:CmisServiceSoap12Binding«>
            <soap12:address location=«http://acserver/cmis/services/CmisWebService.php«/>
        </wsdl:port>
        <wsdl:port name=«CmisServiceHttpEndpoint« binding=«ns:CmisServiceHttpBinding«>
            <http:address location=«http://acserver/cmis/services/CmisWebService.php«/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Dodatek B. Implementacija niti za sinhronizacijo datotek

```
TDownloadThread = class(TThread)
protected
  procedure Execute; override;
public
  constructor Create();
  destructor Destroy; override;
end;

//implementacija
procedure TDownloadThread.Execute;
var
  internetFile,
  localFileName: string;
  doSuspend: boolean;
begin
  while True do
  begin
    while not Suspended do
    begin
      doSuspend := false;
      try
        LockList.Enter;
        try
          FormPlaylist.mtEmpty.Refresh;
          if FormPlaylist.mtEmpty.IsEmpty then
            doSuspend := true
          else
          begin //dobi prvega neprenesenega
            internetFile  :=
                FormPlaylist.mtEmptyFileLocation.AsString + '/' +
                FormPlaylist.mtEmptyFileName.AsString;
            localFileName := MediaLocation + '\' +
                FormPlaylist.mtEmptyFileName.AsString;
          end;
        finally
           LockList.Leave;
        end;
        while (not doSuspend) do
        begin
          if not FileExists(localFileName) then
          begin
            if GetInetFile(internetFile, localFileName + '.tmp') then
            //dobi datoteko
            begin
              RenameFile(localFileName + '.tmp', localFileName);
              LockList.Enter;
              try
                FormPlaylist.mtEmpty.Edit;
                FormPlaylist.mtEmptyState.AsInteger := 1;
                //označi kot preneseno
                FormPlaylist.mtEmpty.Post;
              finally
                LockList.Leave;
              end;
            end;
          end
          else
          begin
           LockList.Enter;
           try
             FormPlaylist.mtEmpty.Edit;
             FormPlaylist.mtEmptyState.AsInteger := 1;
             //spremeni status, če smo našli lokalno datoteko
```

```
            FormPlaylist.mtEmpty.Post;
          finally
            LockList.Leave;
          end;
        end;
        LockList.Enter;
        try
          FormPlaylist.mtEmpty.Refresh;
        if FormPlaylist.mtEmpty.IsEmpty then
          doSuspend := true
        else
        begin
          internetFile  := FormPlaylist.mtEmptyFileLocation.AsString
           + '/' + FormPlaylist.mtEmptyFileName.AsString;
          localFileName := MediaLocation + '\'
                         +  FormPlaylist.mtEmptyFileName.AsString;
        end;
        finally
           LockList.Leave;
        end;
      end;
    finally
      if doSuspend then
         Suspend;
    end;
  end;
end;
```