UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jurij Jelenc

NADGRADNJA PORTALA E-UPRAVA S SISTEMOM ZA UPRAVLJANJE Z VSEBINAMI

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Matjaž Kukar

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01679/2010
Datum: 01.09.2010

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

Kandidat: **JURIJ JELENC**

Naslov: **NADGRADNJA PORTALA E-UPRAVA S SISTEMOM ZA UPRAVLJANJE Z VSEBINAMI**

UPGRADING THE E-GOVERNANCE PORTAL WITH A CONTENT MANAGEMENT SYSTEM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V svojem diplomskem delu naj kandidat opiše nadgradnjo spletnega portala e-uprava s sistemom za upravljanje z vsebinami (CMS). Opiše naj motivacijo za nadgradnjo, opiše uporabljene spletne tehnologije pri izvedbi prenove in svojo vlogo v projektu in opiše razvoj enostavne razširitve nadgrajenega spletnega portala. Prikaže naj glavne prednosti, ki jih nadgrajeni sistem omogoča – tako za vzdrževanje in ažuriranje vsebin, kot za končne uporabnike – državljane Republike Slovenije, tako v kvalitativnem kot v kvantitativnem smislu.

Mentor:
doc. dr. Matjaž Kukar

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Franc Solina

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

diplomskega dela

Spodaj podpisani Jurij Jelenc,

z vpisno številko 63020063,

sem avtor diplomskega dela z naslovom:

NADGRADNJA PORTALA E-UPRAVA S SISTEMOM ZA UPRAVLJANJE Z VSEBINAMI.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Matjaža Kukarja,

- 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 17. 09. 2010 Podpis avtorja: _____

Zahvala

Zahvaljujem se mentorju doc. dr. Matjažu Kukarju za pomoč in nasvete pri izdelavi diplomskega dela.

Zahvalo sem dolžan tudi sodelavcem iz podjetja SRC, d. o. o., skupini e-uprava, v okviru katere sem opravil večji del praktičnega dela. Za koristne pogovore in nasvete se želim zahvaliti Andreju Komanu.

Zahvaljujem se tudi Mojci za prevod povzetka v angleščino in Simoni za lektoriranje diplomskega dela.

Na koncu bi se želel zahvaliti še svojim najbližjim, ki so me spodbujali in mi nesebično stali ob strani na moji študijski poti.

To delo posvečam svoji družini in Jelki.

Jelki za neizmerno ljubezen in podporo.

Kazalo

Kazalo slik

Seznam uporabljenih kratic

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
CMS	Content Management System
CSS	Cascading Style Sheets
DMS	Document Management System
DOM	Document Object Model
ECMS	Enterprise Content Management System
EJB	Enterprise JavaBeans
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IoC	Inversion of Control
JDBC	Java Database Connectivity
JSF	JavaServer Faces
JSP	JavaServer Pages
JSTL	JavaServer Pages Standard Tag Library
JVM	Java Virtual Machine
MVC	Model View Controller
POJO	Plain Old Java Object
RDBMS	Relational Database Management System
SQL	Structured Query Language
SSL	Secure Sockets Layer
UML	Unified Modeling Language
URL	Uniform Resource Locator
WCMS	Web Content Management System
WSDL	Web Services Description Language
WYSIWYG	What You See Is What You Get
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Povzetek

V obdobju vzdrževanja Državnega portala Republike Slovenije e-uprava so se priprava, koordinacija in potrjevanje nameščanja popravkov v testna in produkcijska okolja izkazali za zamudno in obsežno delo. Glede na izkušnje pri vzdrževanju se je pojavila potreba po nadgradnji aplikacije s sistemom, ki bi omogočil programsko podprto spreminjanje vsebine.

Diplomsko delo obravnava nadgradnjo portala e-uprava s pomočjo sistema za upravljanje z vsebinami SRCSI-CMS. Na začetku so predstavljeni sistemi za upravljanje z vsebinami in njihove lastnosti. Sledijo predstavitev portala e-uprava, opis tehnologij, ki so bile uporabljene pri njegovem razvoju, opis sistema SRCSI-CMS in njegovega delovanja ter predstavitev novih tehnologij, ki jih sistem vpelje pri uporabi. Nato je po korakih prikazana nadgradnja portala z opisanim sistemom. Predstavljena so tudi pri tem uporabljena orodja. Na koncu je opisana še implementacija razširitve sistema SRCSI-CMS.

Naloga nadgradnje portala, ki jo je v letih 2009 in 2010 izvedlo podjetje SRC, d. o. o., je bila vzpostavitev sistema za upravljanje z vsebinami (CMS). Namen nadgradnje je bil vzdrževalcem omogočiti lažje spreminjanje vsebine portala neposredno v sami aplikaciji in tako zagotoviti večjo aktualnost vsebine, pri tem pa ohraniti obstoječi prikaz za končne uporabnike. Po nadgradnji se je vzdrževanje portala zelo poenostavilo, objava popravkov je hitrejša, število sodelujočih pri njihovi pripravi pa se je močno zmanjšalo. Enostavna razširljivost sistema je v diplomskem delu prikazana z implementacijo administracije videovsebin v sistemu SRCSI-CMS.

Ključne besede: nadgradnja portala, e-uprava, sistemi za upravljanje spletnih vsebin, CMS, WCMS, Java, spletne tehnologije, urejevalnik TinyMCE

Abstract

During maintenance of the State Portal of the Republic of Slovenia e-uprava, the preparation, coordination and confirmation of patches in test and production environments proved to be time-consuming and extensive work. According to the maintenance experiences there has emerged the need to upgrade application with a system that would enable software-based content modification.

The diploma thesis deals with upgrading the e-uprava portal using the SRCSI-CMS content management system. At the beginning content management systems and their properties are introduced. It is followed by the presentation of e-uprava portal, a description of technologies used during its development, a description of SRCSI-CMS system and its operation, and presentation of new technologies introduced during its usage. Next, there is a step by step review of the portal upgrade with the described system. Used tools are also presented. Finally, the implementation of the extension in SRCSI-CMS system is described.

The task of portal upgrade, which was performed in 2009 and 2010 by the company SRC, d. o. o., was to set up a content management system (CMS). The purpose of upgrading was to enable the maintainers to modify the content of the portal directly in the application and to provide up-to-date content, while preserving the existing display to the final users. After the upgrade the maintenance of the portal has been greatly simplified, content modification is faster and the number of participants in its preparation has been substantially reduced. In this diploma thesis the ease of extending the system is shown through the implementation of administration of video content in SRCSI-CMS system.

Keywords: portal upgrade, e-uprava, web content management systems, CMS, WCMS, Java, web technologies, TinyMCE editor

1 Uvod

V obdobju vzdrževanja portala e-uprava (http://e-uprava.gov.si/e-uprava/) smo se soočili z raznimi situacijami pri pripravi novih vsebin, spreminjanju oziroma posodabljanju že obstoječih in tudi pri odpravi skritih napak. Priprava, koordinacija in potrjevanje nameščanja popravkov v testna in produkcijska okolja so se izkazali za zamudno in obsežno delo. Potrebna sta nadzor in sistematizacija postopka, ki lahko zaradi vpliva človeške napake postaneta problematična. Največkrat pa se je potreba za popravek pojavila na omejenem naboru vsebin portala, ki je glede na obseg celotnega portala statistično gledano razmeroma majhen, saj se potreba po spremembi njegovih določenih delov zaradi obveščanja uporabnikov v časovnem obdobju ponavlja.

Glede na izkušnje pri vzdrževanju se je pojavila potreba po nadgradnji aplikacije s sistemom, ki bi omogočil programsko podprto spreminjanje vsebine in poenostavil vzdrževanje portala. Naloga nadgradnje portala je bila vzpostavitev sistema za upravljanje z vsebinami. Nadgradnja je bila izvedena kot integracija sistema CMS v že obstoječo aplikacijo, s čimer smo se izognili potrebi po implementaciji že obstoječih funkcionalnosti v novem sistemu. V procesu nadgradnje sem sodeloval pri vseh fazah izvedbe in nadzoroval potek dela. Tako sem s svojim poznavanjem portala e-uprava pripomogel k ustreznemu planiranju korakov nadgradnje in njihovi ustrezni izvedbi. Samostojno sem izvedel zahtevnejše korake nadgradnje (postavitev ogrodja portala na novem sistemu, dodajanje funkcionalnosti CMS) ter tudi vse potrebne prilagoditve in nadgradnje sistema CMS. V okviru nadgradnje je bilo treba določiti vsebino, ki se periodično oziroma pogosto spreminja. Za to vsebino mora sistem programsko omogočiti urejanje, potrjevanje in objavljanje.

Nameni diplomskega dela:

- predstavitev portala e-uprava,
- analiza načina vzdrževanja portala e-uprava,
- predstavitev sistema SRCSI-CMS,
- opis nadgradnje portala e-uprava s sistemom SRCSI-CMS,
- preučitev možnosti razširitve nadgrajenega portala,
- implementacija konkretne razširitve.

Diplomsko delo vsebuje sedem poglavji. V uvodnem poglavju so predstavljeni problemsko vprašanje in nameni diplomskega dela. Naslednje poglavje pojasnjuje, kaj so sistemi CMS, katere so njihove funkcionalnosti in kaj omogočajo. V tretjem poglavju je predstavljen portal e-uprava, in sicer njegov namen, administracijske aplikacije in podportali ter tehnologije, ki so bile uporabljene pri razvoju portala. Opisan je tudi obstoječi način priprave popravka pri vzdrževanju portala. Četrto poglavje vsebuje predstavitev sistema SRCSI-CMS, njegovih komponent, lastnosti in načina delovanja. Navedene so tudi nove tehnologije, ki jih sistem vpelje ob uporabi. V naslednjem poglavju je po korakih prikazana nadgradnja portala s sistemom SRCSI-CMS. Predstavljeni so tudi pri tem uporabljena orodja in nov način objave popravkov. Šesto poglavje vsebuje opis implementacije razširitve sistema SRCSI-CMS s funkcionalnostjo administracije videovsebin od inicializacije do delovanja in prikaza razširljivosti vtičnika za urejevalnik TinyMCE. V zadnjem poglavju so podane ugotovitve o opravljenem delu in rezultati ter smernice za nadaljnji razvoj sistema. Dodatek vsebuje izvorno kodo realizirane razširitve iz šestega poglavja.

2 Sistemi za upravljanje z vsebinami

2.1 Kaj je CMS

Kratica CMS predstavlja sistem za upravljanje z vsebinami (angleško Content Management System), ki omogoča programsko podprto upravljanje in nadzor nad vsebinami. Vsebine pa so lahko spletne strani, dokumenti, multimedijske vsebine itd. Glede na tip vsebine se sistemi CMS delijo v več skupin. Diplomsko delo je osredotočeno na sisteme za upravljanje spletnih vsebin (angleško Web Content Management System), ki so predstavljeni s kratico WCMS.

Glede na vsebino, ki jo upravljamo, obstajajo različni tipi sistemov CMS:

- sistem za celovito upravljanje vsebin (ECMS - Enterprise CMS),
- sistem za upravljanje spletnih vsebin (WCMS - Web CMS),
- sistem za upravljanje datotek (DMS - Document Management System),
- sistem za upravljanje mobilnih vsebin,
- sistem za upravljanje komponentnih vsebin,
- sistem za upravljanje multimedijskih vsebin.

Diplomsko delo obravnava nadgradnjo spletnega portala s pomočjo sistema za upravljanje z vsebinami, zato je osredotočeno zgolj na sisteme WCMS za upravljanje spletnih vsebin. Zaradi poenostavitve in ker sistemi WCMS pripadajo večji skupini sistemov CMS v diplomskem delu za sisteme WCMS uporabljam kratico CMS, s tem pa je dejansko mišljen sistem WCMS.

Web Content Management System (slovensko sistem za upravljanje spletnih vsebin) je sistem, ki omogoča urejanje in vzdrževanje vsebine spletnih strani brez znanja programiranja v HTML. Urednik spletne strani tako lahko samostojno spreminja besedila, slike in druge elemente spletne strani brez pomoči osebe ali podjetja, ki je stran izdelalo.
Osveževanje spletne strani s sistemom CMS je zelo preprosto, podjetja in posamezniki pa želijo redno ažurirane strani, zato je CMS vedno bolj priljubljen. Z njim lahko dodajamo nove vsebine ali osvežujemo stare, v večini sistemov CMS pa je omogočeno tudi nalaganje slik in drugih večpredstavnostnih vsebin ter vključevanje dodatkov, vtičnikov in razširitev.

Sistemi CMS so implementirani kot spletne aplikacije za pripravo in urejanje spletnih vsebin. Uporabljajo se za upravljanje in nadzor nad veliko količino dinamične spletne vsebine (dokumenti HTML in vsebujoče slike). Sistem CMS olajša pripravo, nadzor in urejanje vsebine ter omogoča funkcije za nujna spletna vzdrževalna dela. [1]

Programska oprema vsebuje orodja, ki omogočajo enostavno pripravo in urejanje vsebine tudi uporabnikom, ki nimajo znanja o programskih ali označevalnih jezikih. Za razliko od klasične administracije spletnih strani uporabniki za administracijo s sistemi CMS ne potrebujejo tehničnega usposabljanja, saj so ti sistemi prvotno namenjeni osebam brez tehničnega znanja.

Večina sistemov CMS za hranjenje vsebine, metapodatkov in ostalih elementov sistema uporablja bazo. Pogosto je vsebina hranjena v formatu XML za enostavnejšo ponovno uporabo in fleksibilnejši prikaz, vendar pa se uporabljajo tudi druge oblike. Predstavitveni sloj

shranjeno vsebino prikazuje na osnovi predpripravljenih prikaznih vzorcev, ki omogočajo ločitev vsebine od oblike. Pogosto so prikazni vzorci v obliki datotek XSLT.

Večina sistemov CMS uporablja strežniško predpomnjenje vsebine za izboljšanje zmogljivosti prikaza. Ta tehnika deluje najbolje, če se vsebina ne spreminja pogosto, zahtevki za prikaz predpomnjene vsebine pa so s strani uporabnikov pogosti. Administracijo najpogosteje izvajamo s spletnimi vmesniki, do katerih administratorji dostopajo kar s spletnim brskalnikom.

2.2 Kaj nudi dober sistem CMS

Sistemi CMS omogočajo enostavno urejanje vsebine spletnih strani, pri čemer znanje programiranja ali izkušnje z izgradnjo spletnih strani niso potrebne. Administracija, ki temelji na spletnih brskalnikih, mora biti enostavna.

Sistemi predpomnjenja omogočajo urejanje vsebine, ki ni vidna navadnim uporabnikom portala, dokler se vsebina ne objavi. Predpomnjenje zmanjša količino potrebnih dostopov do baze, s čimer privarčujemo pri sistemskih sredstvih in hkrati pospešimo prikaz vsebine na portalu.

Pomembna lastnost sistemov CMS je prenos odgovornosti za vsebino na strokovnjake, ki vsebino pripravljajo, s tem pa se razbremeni že tako prezasedene administratorje spletnih strani (informacijske strokovnjake).

Sistemi CMS brez posredovanja strokovnjakov IT omogočajo:

- objavo novih verzij spletnih vsebin,
- objavo novic,
- objavo kompleksne vsebine glede na predpripravljene prikazne vzorce.

3 Predstavitev portala e-uprava

3.1 Kaj je e-uprava

Državni portal Republike Slovenije e-uprava (http://e-uprava.gov.si/e-uprava/), katerega skrbnik je Ministrstvo za javno upravo, je vstopna točka do različnih informacij o državni in javni upravi. Namen portala je, poleg posredovanja informacij o upravi, tudi približati upravne storitve uporabnikom preko svetovnega spleta in tako poleg klasičnih komunikacijskih poti, po katerih lahko državljani in poslovni subjekti dostopajo do storitev javne uprave, ponuditi dodatno, elektronsko pot za opravljanje storitev. Želja portala je biti prijazna in učinkovita javna uprava, usmerjena k uporabnikom.

Kot zanimivost lahko omenimo, da je e-uprava v letu 2007 dosegla zavidanja vreden uspeh na evropski ravni, saj je Slovenija po meritvah Evropske komisije o razvitosti elektronskih storitev v državah članicah EU osvojila odlično drugo mesto. Napredek s 7. mesta, kamor se je uvrstila v preteklem letu, pomeni veliko priznanje Sloveniji pri prizadevanjih za učinkovito javno upravo. Več informacij najdemo na spletni strani Evropske komisije (http://ec.europa.eu/information_society/eeurope/i2010/docs/benchmarking/egov_benchmark_2007.pdf).

Glavna lastnost portala je usmerjenost k ciljnim skupinam uporabnikov. Rešitev je narejena v obliki podportalov za državljane in pravne osebe ter podportala, namenjenega informacijam s področja javne uprave in zaposlenim v javni upravi.

Poleg informacij, ki so urejene po vsebinskih sklopih, portal posamezni ciljni skupini nudi pregled nad storitvami javne uprave preko tako imenovanih življenjskih dogodkov, kjer so podatki in storitve urejeni vsebinsko, kot jih vidi in dojema posamezni državljan. Storitve so razdeljene po področjih tako, da spremljajo vse pomembnejše dogodke v življenju posameznika od rojstva, šolanja in zaposlitve do upokojitve.

Portal e-uprava z novičarskim, anketnim in sporočilnim sistemom obiskovalcem nudi dvosmerno komunikacijo. K boljši uporabniški izkušnji pripomore tudi možnost poosebljanja portala, vsak uporabnik si namreč lahko določi vsebine, ki so mu najzanimivejše. Te se prikažejo ob vsaki ponovni prijavi uporabnika na portal. Uporabniki se lahko naročijo tudi na obveščanje o novicah in novostih po elektronski pošti. Portal je na voljo v slovenskem, angleškem, italijanskem in madžarskem jeziku. [2]

3.1.1 Administracijske aplikacije

Nemoteno delovanje portala omogoča množica zalednih aplikacij, ki so namenjene uredniku. Z njimi je mogoče objavljati dinamično vsebino, spreminjati strukturo določenih delov spletišča, objavljati novice in vključevati ankete.

Uredniku portala so na voljo zaledne aplikacije, s pomočjo katerih upravlja z dinamično vsebino in portal ločuje na več vsebinsko bogatih, nepovezanih delov. Taka razdelitev portala uredniku omogoča boljši pregled, saj so posamezni deli portala manjši in zato zlahka

obvladljivi. Osveževanje in dopolnjevanje vsebin spletnega mesta je preprosto, saj aplikacije od urednika ne zahtevajo posebnega tehnološkega znanja.

V zalednih uredniških aplikacijah so urednikom na voljo naslednji moduli:

- administracija življenjskih dogodkov in elektronskih storitev,
- generator obrazcev za elektronske storitve,
- novičarski sistem,
- anketni sistem,
- administracija pogosto zastavljenih vprašanj in odgovorov.

Vloge lahko organi javne uprave ustvarjajo sami, saj imajo na voljo administracijsko aplikacijo Administracija življenjskih dogodkov. Zelo pomembno je omeniti tudi modul te administracijske aplikacije, imenovan Generator obrazcev, ki omogoča ustvarjanje novih obrazcev na zelo preprost, predvsem pa intuitiven način. Delo s temi aplikacijami poteka tako, da organ javne uprave s pomočjo administracijske aplikacije oziroma modula Generator obrazcev ustvari nov obrazec in ga objavi na portalu e-storitve.

3.1.2 E-storitve

Portal E-storitve (elektronske storitve javne uprave) je ločena spletna aplikacija, ki predstavlja enega izmed številnih med seboj povezanih spletnih portalov v sklopu e-uprave. Portal je izjemno uporaben, saj uporabnikom na enem mestu ponuja obrazce ali prave elektronske vloge, ki jih organi javne uprave in javnega sektorja ponujajo uporabnikom portala e-uprava. S portalom E-storitve torej lahko uporabniki poiščejo in oddajajo obrazce pristojnemu organu, ne da bi morali hoditi na upravno enoto.

Na portalu je mogoče oddajati vloge, ki jih je bilo pred nastankom tega portala možno oddajati samo osebno pri referentih javne uprave v obliki papirnih obrazcev. Vloge lahko organi javne uprave ustvarjajo sami s pomočjo že omenjenih administracijskih aplikacij. Uporabnik portala e-uprava najprej najde ustrezen življenjski dogodek, na katerem so že povezave do storitev na portalu ESJU. Tam poišče želeni obrazec in ga izpolni. Portal omogoča tudi elektronski podpis obrazca in plačilo preko sistema e-plačila, če organ, ki je obrazec ustvaril in objavil, to zahteva. Nato uporabnik vlogo odda in postopek je zanj zaključen. Vloga se nato s pomočjo zalednih sistemov po vnaprej določenih dostavnih poteh pošlje izbranemu organu.

Uporabniki, ki oddajajo vloge preko portala in jih tako pošljejo ciljnim ustanovam, lahko izberejo možnost, da prejmejo vročene dokumente na elektronski način. Rešitev je integrirana z različnimi sistemi. Tako je izvedena integracija s Centralnim registrom prebivalstva, Registrom prostorskih enot, Registrom vozil in listin, Registrom zavarovanj, s Pošto Slovenije, z modulom za izvedbo spletnega plačevanja in z zalednim sistemom SPIS.
V okviru ogrodja portala je bila implementirana tudi rešitev E-podaljšanje, ki uporabnikom omogoča spletno podaljšanje veljavnosti prometnega dovoljenja za vozila z opravljenim tehničnim pregledom in za vozila, ki tehničnega pregleda ne potrebujejo. [3]

Kot lahko vidimo na primeru portala E-storitve, je portal e-uprava poleg že omenjenega posredovanja informacij državljanom mišljen tudi kot vstopna točka za veliko število drugih spletnih portalov v sklopu e-uprave, ki uporabnikom nudijo različne informacije in razne

spletne storitve. Zaradi tega je portal e-uprava osrednji portal z informacijami za uporabnike, ki se nanašajo na vse portale v sklopu e-uprava.

Informacija je za uporabnika kakovostna le v primeru, če zadošča določenim pogojem, med katerimi je tudi dosegljivost. Dosegljivost informacije pomeni, da je informacija uporabnikom dostopna takrat, ko je potrebujejo. Primer kritične informacije je na primer objava opozorila za uporabnike, da bo zaradi specifičnih nadgraditev samega portala ali sistemov, s katerimi se portal povezuje, moteno delovanje nekaterih funkcionalnosti portala e-uprava ali kakšnega izmed drugih portalov. Takšno obvestilo na prvi strani portala uporabnikom omogoča pravočasno ukrepanje glede na vrsto situacije. Prava informacija na pravem mestu in ob pravem času pa je za uporabnike portala seveda izjemno pomembna.

3.2 Tehnologije portala e-uprava

Portal je zgrajen na osnovi odprte kode, s pomočjo ogrodij Struts (razdelek 3.2.2) in Tiles (razdelek 3.2.3), ki so prilagojena in razširjena tako, da urednikom omogočajo:

- internacionalizacijo,
- avtorizacijo,
- sledenje uporabnikom in
- ustvarjanje dinamične vsebine.

Programski jezik, ki ga v našem podjetju uporabljamo za razvoj spletnih aplikacij, je v največji meri Java (razdelek 3.2.1).

Pri razvoju portala e-uprava smo uporabili odprtokodno ogrodje Struts za definicijo delovnega toka aplikacije in prav tako odprtokodno ogrodje Tiles za postavitev strukture spletne strani, saj imamo v podjetju večletne izkušnje s temi ogrodji in odločitev o tem, katero tehnologijo uporabiti za ta namen, ni bila težka.
Za hitrejši in poenostavljen razvoj spletnih aplikacij v programskem jeziku Java smo uporabili tehnologijo JSP in označevalne knjižnice JSTL (razdelek 3.2.7). Seveda pa si sodobnih spletnih aplikacij ne moremo predstavljati brez skriptnega jezika JavaScript (razdelek 3.2.4) in z njim povezane tehnologije Ajax (razdelek 3.2.5). Povezava aplikacije z bazo je implementirana s pomočjo programskega vmesnika JDBC (razdelek 3.2.6).

Aplikacije so v razvojnem, testnem in produkcijskem okolju nameščene na aplikacijskem strežniku Oracle IAS (razdelek 3.2.8). Za podatkovno bazo pa uporabljamo podatkovno bazo Oracle (razdelek 3.2.9), na kateri so bazne procedure napisane s PL/SQL, kar pomeni, da v aplikaciji ne pišemo stavkov SQL.

3.2.1 Java EE

Java je objektno usmerjen, prenosljiv programski jezik, ki ga je v podjetju Sun Microsystems razvil James Gosling s sodelavci. Je tudi razmeroma preprost in varen jezik, kar je pomembna lastnost za razvoj spletnih aplikacij, zato ga v našem podjetju izberemo najpogosteje.

Java EE (Java Platform, Enterprise Edition) je široko uporabljena platforma za programiranje v Java programskem jeziku, namenjena razvoju strežniških komponent. Od standarde platforme (Java Standard Edition Platform, Java SE) se razlikuje po dodatnih knjižnicah, ki

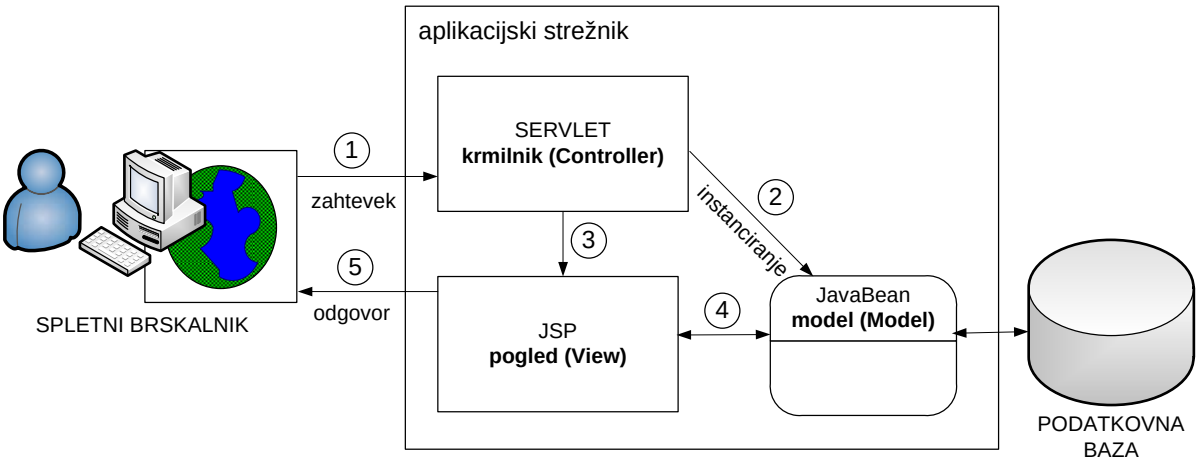
omogočajo funkcionalnosti za nameščanje večslojne, porazdeljene programske opreme, ki dovoljuje napake in omogoča njihovo obravnavo. Programska oprema največkrat temelji na modularnih komponentah, ki tečejo na aplikacijskem strežniku. [4]

3.2.2 Struts

Apache Struts je odprtokodno ogrodje za spletne aplikacije, ki je namenjeno razvoju Java EE spletnih aplikacij. Začetnik njegovega razvoja je Craig McClanahan. Ta ga je leta 2000 doniral fundaciji Apache Software Foundation.

Jedro ogrodja Struts je fleksibilna nadzorna plast, ki temelji na standardnih tehnologijah, kot so Java Servlets, JavaBeans, ResourceBundles in XML, vključuje pa tudi razne knjižnice Jakarta Commons. Struts za arhitekturo aplikacij spodbuja uporabo pristopa Model 2, ki je različica klasične arhitekture MVC (angleško Model View Controller). Slika 1 prikazuje shemo arhitekture Model 2.

Struts določa lastno krmilno komponento (angleško Controller) ter se integrira z ostalimi tehnologijami pri elementih modela (angleško Model) in pogleda (angleško View). Za element modela lahko uporablja standarde tehnologije dostopa do baze, kot sta JDBC in EJB (angleško Enterprise JavaBean), pa tudi druge sisteme, na primer Hibernate, iBATIS ali objektno-relacijski most (angleško Object Relational Bridge). Za element pogleda se Struts dobro poveže s tehnologijami, kot je JSP v navezi z JSTL in JSF, prav tako se lahko uporabi tehnologije, kot sta Velocity Templates in XSLT, pa tudi druge sisteme za prikaz. [5]



Slika 1: Arhitektura Model 2

Spletne aplikacije uporabljajo namestitveni opisnik (angleško deployment descriptor), imenovan »web.xml«, za inicializacijo virov, kot so servleti ali označevalne knjižnice. Podobno pa tudi Struts uporablja svojo nastavitveno datoteko »struts-config.xml« za inicializacijo svojih virov. Ti viri vsebujejo elemente, kot so ActionForm za pridobivanje vnosa uporabnika, ActionMapping za usmerjanje vnosa do strežniških razredov Java, imenovanih Action, in še ActionForward, ki služi za usmerjanje (navigacijo) na strani za prikaz. Jedro ogrodja Struts pa predstavlja razred ActionServlet, ki sprejema zahteve in jih predaja ustreznim objektom aplikacije.

Ogrodje Struts ima mnoge pozitivne lastnosti, ki jih prinese njegova uporaba. Naj naštejem samo nekatere: zanesljivost, majhnost, brezplačna uporaba, preglednost aplikacije, neodvisnost pogleda in fleksibilnost. Zaradi omenjenih pozitivnih lastnosti je bila uporaba ogrodja Struts pri izgradnji naših aplikacij samoumevna. Prav tako pa imamo s tem ogrodjem že večletne izkušnje, kar samo povečuje enostavnost in krajša čas, potreben za izgradnjo novih aplikacij.

Ogrodje Struts omogoča pripravo razširljivega razvojnega okolja za aplikacije, ki temelji na standardih in znanih razvojnih vzorcih.
Druga razširitev so Tiles, ki pomagajo pri izgradnji strani za manjše fragmente.

3.2.3 Tiles

Apache Tiles je ogrodje za gradnjo strani s pomočjo prikaznih vzorcev, ki poenostavi razvoj spletnih aplikacij z uporabniškimi vmesniki. Ogrodje Tiles omogoča definicijo fragmenta strani, ki ga je mogoče vstaviti v celostno stran med izvajanjem. Te fragmente oziroma tile je mogoče uporabiti kot enostavne vključitve v stran, s čimer zmanjšamo podvajanje pogostih elementov strani. Lahko pa jih tudi vključimo v druge fragmente z namenom razvoja ponovno uporabljivih prikaznih vzorcev. Ti prikazni vzorci vpeljejo razvoj usklajenega prikaza za celotno aplikacijo. Ogrodje Tiles je vse bolj priljubljeno kot komponenta priljubljenega ogrodja Struts. Prav tako se ogrodje Tiles uporablja tudi izven ogrodja Struts za raznimi drugimi ogrodji, kot sta Struts2 in Shale. [6]

3.2.4 JavaScript

JavaScript je objektno usmerjen skriptni jezik. V raznih aplikacijah ga uporabljamo za programski dostop do obravnavanih objektov na strani klienta. Primarno pa je uporabljen kot JavaScript na strani klienta v implementaciji znotraj spletnih brskalnikov, saj omogoča naprednejše uporabniške vmesnike in dinamične spletne strani. JavaScript je karakteriziran kot dinamičen jezik, šibko vezan, temelječ na prototipih s funkcijami. Nanj je vplivalo veliko jezikov, načrtovan je bil po zgledu Jave, a neodvisno od nje. Z Javo si deli številne lastnosti in strukture, vendar pa bi naj bil enostavnejši za uporabo za neprogramerje. [7]

Program, napisan v jeziku JavaScript, vgradimo ali pa vključimo v HTML z namenom, da opravlja naloge, ki niso mogoče s samo statično stranjo. Te naloge so na primer odpiranje novih oken, preverjanje pravilnosti vnešenih podatkov, enostavni izračuni in podobno. Na žalost različni spletni brskalniki izpostavijo različne objekte za uporabo. Za podporo vseh brskalnikov je zato treba napisati več različic funkcij.

Jezik JavaScript pa uporabljamo tudi kot osnovo za tehnologijo Ajax. Ta je opisana v naslednjem poglavju.

3.2.5 Ajax

Ajax je kratica za asinhroni JavaScript in XML (angleško Asynchronous JavaScript and XML). Pomeni skupino med seboj povezanih tehnik za spletni razvoj, ki jih uporabljamo za ustvarjanje interaktivnih spletnih aplikacij. Z uporabo tehnologije Ajax spletne aplikacije pridobivajo podatke s strežnika asinhrono v ozadju, s čimer se ne spreminja prikaz ali ovira

obnašanje trenutno aktivne strani v brskalniku. Uporaba tehnik Ajax je vodila k naraščanju interaktivnosti in dinamičnosti spletnih strani. [8]

Ajax ni tehnologija sama zase, ampak predstavlja skupino tehnologij. Za predstavitev informacije uporablja kombinacijo tehnologij HTML ali XHTML in CSS. Skriptni jezik JavaScript uporabljamo za dostop do elementov v modelu DOM ter s tem za dinamičen prikaz podatkov in interakcijo uporabnika s prikazano informacijo. Objekt XMLHttpRequest pa skupaj z jezikom JavaScript omogoča asinhrono izmenjavo podatkov med brskalnikom in strežnikom, pri čemer se za delo s podatki uporabljata XML in XSLT. Vendar pa jeziki JavaScript ter XML in XSLT niso več obveza za aplikacije Ajax, saj je nadaljnji razvoj omogočil uporabo alternativnih tehnologij.

Običajne spletne aplikacije morajo za vsako spremembo na spletni strani poslati zahtevo HTTP, na katero strežnik kot odgovor pošlje celotno spletno stran, zato se osveži celoten prikaz v brskalniku. Takšno obnavljanje strani je za uporabnika moteče in počasno.
Aplikacije Ajax pa so prirejene za generiranje poizvedb samo za tiste podatke, ki jih dejansko potrebujejo. Te poizvedbe se na strežnik pošljejo asinhrono, tako lahko med čakanjem aplikacije na odgovor uporabnik nemoteno uporablja spletno stran. Ko strežnik pošlje odgovor na poizvedbo, se izvede vnaprej določena funkcija, ki poskrbi za ustrezen prikaz podatkov na strani, s čimer se izognemo osveževanju celotne strani.

Prednosti uporabe Ajax-a:
- možnost razvoja hitrejših in uporabniku prijaznejših spletnih aplikacij,
- zmanjšanje količine podatkov, potrebnih za prenos med strežnikom in odjemalcem,
- zmanjšanje obremenitve strežnika, saj veliko obdelave izvede odjemalec.

Slabosti uporabe Ajax-a:
- pogosto težji razvoj dinamičnih aplikacij Ajax v primerjavi s statičnimi stranmi,
- strani, dinamično ustvarjene z zahtevki Ajax, niso dosegljive s pomočjo zgodovine iskanja v brskalniku,
- težavnost ali sploh nezmožnost shranjevanja neposredne povezave do stanja strani, ki se je dinamično ustvarila z zahtevki Ajax,
- v konkretnih situacijah možnost motečega dinamičnega osveževanja strani, zlasti v primeru počasne spletne povezave,
- nezmožnost spletnih pajkov pri indeksiranju dinamično pridobljene vsebine, saj večinoma ne izvajajo kode JavaScript,
- težave pri prikazu dinamične vsebine v primeru, da brskalniki ne podpirajo jezika JavaScript ali objekta XMLHttpRequest,
- večje število zahtevkov Ajax, ki dostopajo do baze ali drugih sistemov, lahko povzroči zmanjšanje odzivnosti sistema ali celo poveča potrebno po strojni zmogljivosti.

Glede na vse naštete prednosti in slabosti se pri uporabi tehnologije Ajax svetuje premišljenost, saj lahko v določenih primerih prekomerna dinamičnost aplikacije prinese več slabosti. V aplikacijah e-uprave tehnologijo Ajax uporabljamo na mestih, kjer lahko uporabniku prihranimo prekomerno nepotrebno čakanje za prikaz posameznih informacij ali prepogosto prehajanje med stranmi. Primer so informacije, prikazane v drevesni strukturi: uporabniku na izbiro končnega elementa v drevesu s pomočjo klica Ajax prikažemo podatke elementa. Tako se uporabniku ne spremeni drevesni prikaz, v katerem je že morda izbral druge elemente, s čimer je brskanje po drevesu enostavno in razumljivejše. Ker pa je portal predvsem informativne narave, pri večini prikazih informacij raje uporabljamo običajni prikaz

strani z zahtevkom GET, saj tako uporabniku omogočimo hranjenje povezave in vračanje na stran z iskano informacijo.

3.2.6 JDBC

JDBC (angleško The Java Database Connectivity) API je industrijski standard za bazno neodvisno povezljivost med programskim jezikom Java in širokim naborom baz – tako baz SQL kot drugih tabelaričnih virov, kot so preglednice ali ploske zbirke podatkov. JDBC API predpisuje vmesnik API za nivo klicev za dostop do baze na osnovi jezika SQL.
Tehnologija JDBC omogoča uporabo programskega jezika Java za izrabo možnosti »Napiši enkrat, zaženi kjerkoli« (angleško »Write Once, Run Anywhere«) za aplikacije, ki zahtevajo dostop do poslovnih podatkov. Z gonilniki, ki podpirajo tehnologijo JDBC, se je možno povezati na vse poslovne vire podatkov celo v heterogenih okoljih. [9]

V naših aplikacijah se s pomočjo tehnologije JDBC povezujemo do podatkovne baze Oracle in kličemo bazne procedure, napisane v jeziku PL/SQL. To pomeni, da v aplikaciji ne pišemo stavkov SQL, ampak kličemo že vnaprej pripravljene procedure, s čimer se povečata varnost in enostavnost aplikacije.

3.2.7 JSP in JSTL

JavaServer Pages (JSP) je standardna tehnologija predstavitvene plasti za platformo J2EE, ki razvijalcem programske opreme omogoča pripravo dinamično generiranih spletnih strani na osnovi HTML, XML ali drugih dokumentnih tipov. Strani JSP so predstavitveno naravnani servleti, ki omogočajo prepletanje kode, napisane v jeziku Java, in statične spletne vsebine. Tehnologija JSP tako omogoča hiter razvoj spletnih aplikacij, ki so strežniško in platformno neodvisne.

Arhitekturno gledano je tehnologija JSP visokonivojska abstrakcija Java servletov. Strani JSP se na strežniku naložijo in izvajajo v posebni strukturi, ki se na strežnik namesti v strežniškem paketu Java za spletno aplikacijo Java EE (angleško Java EE Web Application). Takšna stran JSP se prevede in izvede na strežniku ter kot odgovor na zahtevek vrne HTML ali XML dokument. Prevedene strani in uporabljene knjižnice Java uporabljajo vmesno kodo Java (bytecode), zato se morajo izvesti v navideznem stroju Java (JVM), ki je integriran v operacijski sistem, s čimer pa omogočamo abstraktno platformno neodvisno okolje. [10]

Sintaksa JSP nudi dodatne označevalne elemente v obliki XML, imenovane akcije JSP, ki omogočajo uporabo vgrajenih funkcionalnosti. Tehnologija dodatno omogoča ustvarjanje označevalnih knjižnic JSP, ki služijo kot razširitev standardnim elementom HTML in XML.

Knjižnica JSTL (JavaServer Pages Standard Tag Library) razširja specifikacijo JSP z zbirko označevalnih elementov JSP za splošno namenske funkcionalnosti, ki jih pogosto uporabljamo v spletnih aplikacijah. JSTL je bil razvit v okviru procesa Java Community Process (JCP) z namenom poenostavitve razvoja spletnih aplikacij.

JSTL v enostavnih označevalnih elementih podpira pogoste strukturne operacije, kot so iteracija in pogojni elementi, elementi za manipulacijo z dokumenti XML, elementi za internacionalizacijo in SQL. S tem je razvijalcem omogočeno, da se osredotočijo na razvoj funkcionalnosti, ki so specifične za njihovo aplikacijo, pri tem pa jim ni treba na novo

razvijati osnovne operacije. JSTL prav tako zagotavlja ogrodje za integracijo že obstoječih lastnih označevalnih elementov z elementi JSTL. [11]

JSTL omogoča učinkovito dodajanje logike znotraj strani JSP brez uporabe neposredno vstavljene kode Java. Uporaba standardiziranega nabora označevalnih elementov namesto preskakovanja v kodo Java in iz nje vodi h kodi, ki je lažja za vzdrževanje in omogoča ločitev razvoja aplikativne kode ter uporabniškega vmesnika.

3.2.8 Aplikacijski strežnik Orcale IAS

Aplikacijski strežnik Orcale IAS zagotavlja enotno rešitev za vmesni nivo programske infrastrukture in predstavlja vsestransko rešitev za razvoj, integracijo in vpeljavo aplikacij, portalov in spletnih storitev. Orcale IAS zajema vsebnike za J2EE (Oracle Containers for J2EE), predpomnilnik (Oracle Web Cache), strežnik HTTP (Oracle HTTP Server), tehnologije za načrtovanje in izgradnjo aplikacij (Oracle Forms), poročila (Oracle Reports), integrirano poslovno informacijsko okolje za razvoj in administracijo portalskih rešitev (Oracle Portal) ter orodje za poizvedbe, poročanje in analizo (Oracle Discoverer). Vsebuje tudi rešitve za integrirano varnost, upravljanje in tehnologije za integracijo. Strežnik Orcale IAS je član družine izdelkov Oracle Fusion Middleware, kar omogoča večjo agilnost, boljše odločanje ter zmanjšanje stroškov in tveganja v primerjavi z drugimi rešitvami. [12]

Zadnja verzija strežnika Orcale IAS je 10gR3. V prihodnje pa bo Orcale IAS nadomestil strežnik WebLogic Server podjetja BEA Systems, ki ga je prevzel Oracle. WebLogic Server naj bi integriral ključne funkcionalnosti strežnika Orcale IAS.

3.2.9 Podatkovna baza Oracle DB

Podatkovna baza Oracle Database, pogosto imenovana tudi Oracle RDBMS (relational database management system) ali krajše kar Oracle, je sistem za upravljanje z relacijskimi bazami podatkov podjetja Oracle. Od leta 2009 je Oracle najpogosteje uporabljen sistem za upravljanje z bazami. [13]

Oracle Database verzije 11g prinaša rekordno zmogljivost in razširljivost v strežnikih z operacijskimi sistemi Windows, Linux in UNIX. Zagotavlja hitro povrnitev naložbe, saj uporabnikom omogoča prehod z uporabe enega strežnika na mrežno računalništvo brez spreminjanja ene same vrstice programske kode.

3.3 Pregled obstoječega sistema priprave in objave popravkov

Pri postavitvi in vzdrževanju portala sledimo načelu RTP (razvoj, test, produkcija), kar v praksi pomeni, da vsako novo storitev razvijemo in testiramo v razvojnem okolju. Storitev nato prenesemo v testno okolje. Tu jo preverimo še enkrat, šele nato jo prenesemo tudi v produkcijsko okolje. S takim pristopom se možnosti za napake ali pomanjkljivosti izrazito zmanjšajo.

Trenutno naročnik (lastnik portala) naroči izvedbo zahtevka, v katerem navede mesto in način spremembe dela spletnih strani. Pri posameznem naročilu so nato potrebna še morebitna dodatna usklajevanja s pomočjo odzivov na osnovni zahtevek oziroma dodatna vprašanja in podvprašanja, ki pomagajo vzdrževalni ekipi, ki bo izvedla implementacijo popravka, pri ustrezni izvedbi zahtevka.

Ko je popravek implementiran, glavni izvajalec popravka izvede postopek priprave popravka za pošiljanje naročniku. Ta postopek je sestavljen iz naslednjih korakov:

1. priprava arhivirane datoteke popravka, ki vsebuje vse spremenjene ali novododane datoteke;
2. dokumentiranje popravka v skupni nadzorni dokument popravkov, v katerega se vpišejo zaporedna oznaka, datum izvedbe, glavni izvajalec in namembnost popravka, prisotnost sprememb baze zaradi popravka, opis vsebine popravka ter celotni seznam datotek iz popravka;
3. namestitev popravka v razvojno okolje;
4. testiranje popravka v razvojnem okolju s strani testne ekipe;
5. shranitev popravka v repozitorij popravkov skupaj z nadzornim dokumentom popravkov;
6. obveščanje predstavnika izvajalca o pripravljenem popravku za pošiljanje naročniku.

Ko glavni izvajalec popravka zaključi zgornje korake, je za ustrezno nadaljevanje postopka vodja vzdrževalne ekipe oziroma predstavnik izvajalca v stiku z naročnikom. Vodja pripravi odziv na zahtevek, v katerem zapiše vsebino popravka, opozorila na morebitne potrebne spremembe v nastavitvah aplikacije, morebitne spremembe na bazi ali kakšna druga opozorila. Poleg tega v odzivu navede postopek nameščanja popravka glede na ciljna okolja namestitve in vrsto popravka. Glede na vrsto popravka definira tudi osebo, odgovorno za potrjevanje popravka. Odzivu na koncu pripne tudi fizično arhivirano datoteka popravka.

V celoti gledano je v tok priprave popravka vključena skupina ljudi, in sicer:
- naročnik zahtevka za popravek oziroma predstavnik naročnika za usklajevanje,
- vodja vzdrževalne ekipe oziroma predstavnik izvajalca popravka, ki koordinira usklajevanje, izvedbo, pošiljanje in nazadnje ustrezno pomoč naročniku pri namestitvi in testiranju popravka,
- glavni izvajalec popravka, ki izvede postopek priprave popravka za pošiljanje naročniku,
- ostali člani vzdrževalne ekipe, ki so potrebni pri implementaciji popravka (razvojni inženir - programer, bazni inženir, oblikovalni inženir).

Po pošiljanju popravka naročniku pa mora na strani naročnika za namestitve aplikacije zadolženi sistemski administrator poskrbeti za nameščanje popravka v ustrezno okolje in obvestiti osebo, zadolženo za potrjevanje popravka. Če popravek ni ustrezen, mora izvajalec ponoviti celoten cikel in ponovno poslati popravek naročniku. Ko je ustreznost popravka potrjena v testnem okolju, se izvede končno nameščanje v produkcijsko okolje.

4 Predstavitev sistema SRCSI-CMS

Pri nadgradnji portala e-uprava s sistemom CMS smo se najprej soočili z odločitvijo izbire med že obstoječimi sistemi in razvojem lastnega sistema. Na tržišču sicer obstaja ogromno komercialnih ter tudi odprtokodnih produktov in polproduktov, vendar nobeden izmed njih ni ustrezal specifičnosti portala e-uprava. Kot je bilo predstavljeno že v prejšnjem poglavju, je portal e-uprava živ sistem z dinamičnimi vsebinami, ki jih ni mogoče upravljati s klasičnimi sistemi CMS. Hkrati se portal integrira z ostalimi sistemi javne uprave in kompleksno obdeluje podatke za končni prikaz uporabnikom. Takšne dinamične vsebine enostavno ni mogoče urejati s sistemom CMS. Pri razvoju sistema CMS se je tako pojavila potreba po urejanju le manjših elementov znotraj poljubne dinamične strani. In kot bomo videli v nadaljevanju predstavitve, je ravno urejanje le dela strani oziroma fragmenta na že obstoječih dinamičnih straneh bistvena prednost sistema SRCSI-CMS.

Sistem, ki ga obravnavam v naslednjem poglavju, se že uporablja tudi v drugih aplikacijah, ki jih je razvijalo naše podjetje. Zelo znana je na primer aplikacija e-VEM (http://evem.gov.si/evem/). S sistemom e-VEM za gospodarske družbe smo junija 2009 prejeli prestižno mednarodno priznanje OZN za odličnost na področju javne uprave v kategoriji »Izboljšanje zagotavljanja storitev v javnem sektorju« med vsemi prijavljenimi državami Severne Amerike in Evrope. Gre za najvišje priznanje slovenski javni upravi do sedaj, ki ni ozko vezano samo na elektronske storitve, temveč na rešitve s področja storitev javne uprave v celoti.

Sistem SRCSI-CMS je knjižnica, namenjena razvoju spletnih aplikacij na osnovi tehnologij JavaEE, Struts, Tiles, Spring, JSP, JSTL, JavaScript in drugih. SRCSI-CMS vsebuje implementacijo osnovnih komponent spletnih aplikacij, na podlagi katerih je mogoče v zelo kratkem času postaviti novo spletno aplikacijo. Edino, za kar je treba poskrbeti, so postavitev baznega dela, razvoj poslovne logike in prilagoditev ali razvoj predstavitvenega sloja aplikacije. V sistemu SRCSI-CMS pa je že poskrbljeno za:

- osnovne akcije z glavnimi mehanizmi za izvajanje na osnovi razreda Struts Action,
- sistem za predpomnjenje,
- sistem za lokalizacijo,
- sistem raznih filtrov (avtorizacija, predpomnjenje),
- sistem za uporabo virov sporočil (angleško message resources),
- funkcionalnosti CMS (fragmenti, strani, dokumenti, novice).

Ker je za našo nadgradnjo portala e-uprava ključnega pomena upravljanje z vsebinami, se v nadaljevanju najprej osredotočim na opis funkcionalnosti CMS, nato pa sledijo še opisi nekaterih ostalih elementov in funkcionalnosti sistema SRCSI-CMS.

4.1 Opis funkcionalnosti CMS

Sledi opis posameznih elementov sistema SRCSI-CMS. Za prikaz delovanja funkcionalnosti CMS bom uporabil kar zaslonske slike iz vključitve sistema SRCSI-CMS v portal e-uprava, ki ga predstavljam v naslednjem poglavju. Zaslonske slike iz nadgradnje portala e-uprava podajam zaradi nazornosti prikaza delovanja in uporabe sistema SRCSI-CMS.

4.1.1 Fragmenti

Kot sem omenil že v uvodu poglavja, fragmente uporabljamo za urejanje vsebine na manjšem delu strani, ki lahko ostalo vsebino pridobi poljubno dinamično in popolnoma neodvisno od sistema CMS. Tako lahko na strani določimo le del, ki ga bo urejal urednik CMS in mu pri tem onemogočimo dostop do dinamične vsebine, ki skrbi za prikaz preostalega dela strani. Za urejanje vsebine s fragmenti so primerne strani, ki prikazujejo dinamične rezultate izvajanja akcij uporabnika portala, pri tem pa ima urednik CMS nadzor nad manjšim od dinamičnega dela neodvisnim delom strani.

Fragmenti so osnovni elementi sistema CMS, ki jih lahko v aplikacijo vključujemo samostojno z dodajanjem vključitvene kode na stran JSP ali pa posredno preko strani CMS, ki v svojem prikaznem vzorcu vsebuje enega ali več fragmentov. Dodajanje samostojno na stran mora izvesti razvijalec, dodajanje strani CMS pa lahko izvede urednik CMS.

Pri določanju fragmenta je odločilnega pomena ključ, s katerim aplikacija identificira vsebino fragmenta, ki jo pripravimo pri urejanju vsebine s sistemom CMS in shranimo na bazo za prikaz ali naknadno popravljanje. Pri pripravi fragmenta z neposrednim dodajanjem na stran mora ključ določiti razvijalec pri dodajanju vključitvene kode na JSP, pri vključitvi fragmentov v strani CMS pa se ključ določi samodejno glede na ključ strani CMS, ki ga izbere urednik CMS.

Primer vključitvene kode za dodajanje fragmenta:

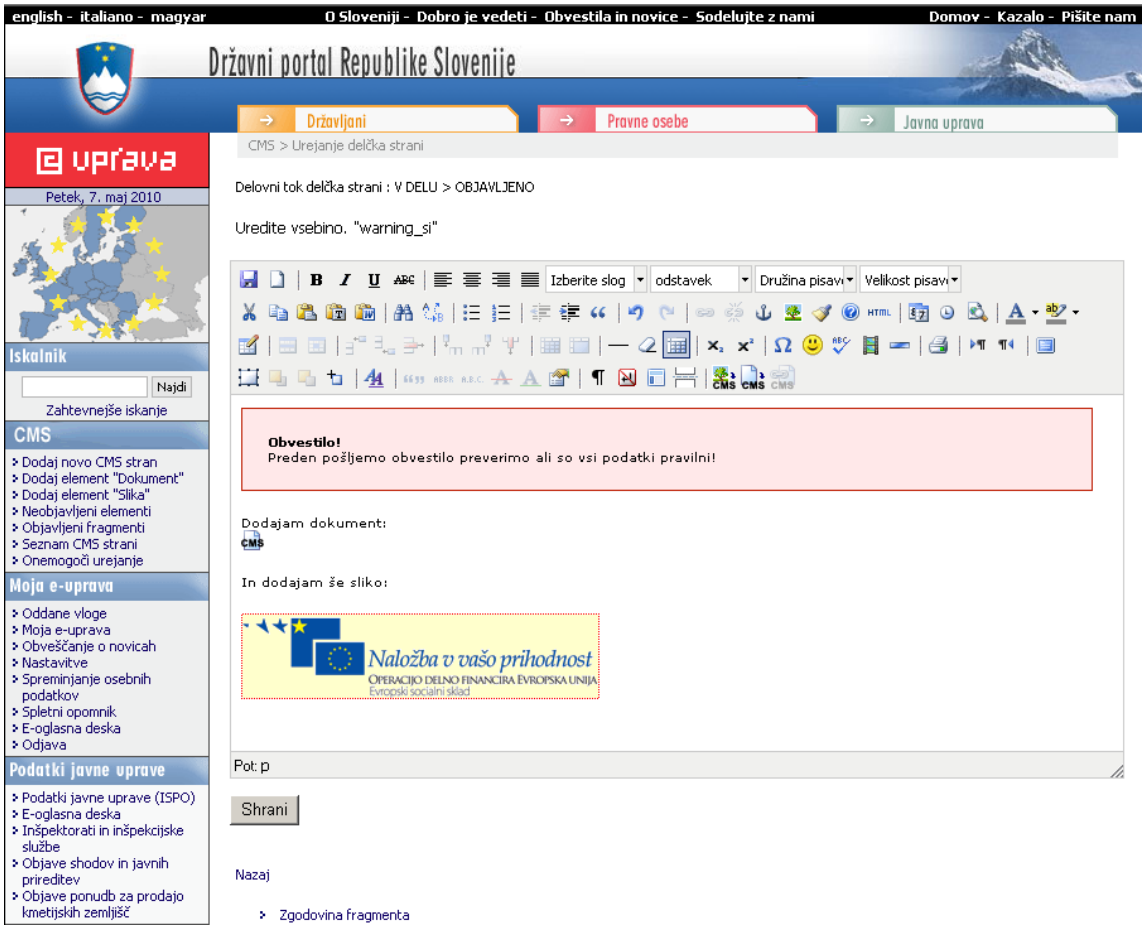
```
<jsp:include page="/cms/fragment/show.euprava?srcsi_cms_fragment_key=test"/>
```

Sistem SRCSI-CMS urednikom z ustrezno vlogo omogoča kakršno koli spreminjanje vsebine fragmenta, pripravo fragmenta za objavo, potrjevanje in objavljanje. Možen je vpogled v zgodovino sprememb fragmenta s podatki o avtorju spremembe. Dodatne akcije v sistemu SRCSI-CMS omogočajo prikaz seznama vseh neobjavljenih in objavljenih fragmentov. Na sliki 2 vidimo primer prikaza seznama vseh objavljenih fragmentov.

Status	Ključ	Datum	Opis	Komentar	Povezava
OBJAVLJENO	message_it	16.11.2009	message_it		it/portal.euprava
OBJAVLJENO	footer_it	20.11.2009	footer_it		it/portal.euprava
OBJAVLJENO	message_hu	16.11.2009	message_hu		hu/portal.euprava
OBJAVLJENO	footer_hu	25.11.2009	footer_hu		hu/portal.euprava
OBJAVLJENO	ju_message_hu	20.11.2009	ju_message_hu		hu/javni.euprava
OBJAVLJENO	ebusiness	10.05.2010	ebusiness		portal5tran.euprava?pageid=48
OBJAVLJENO	edemocracy	23.11.2009	edemocracy		potrebujemInfo.euprava
OBJAVLJENO	banners_d	26.11.2009	banners_d		drzavljani.euprava
OBJAVLJENO	banners_ju	26.11.2009	banners_ju		javni.euprava
OBJAVLJENO	about_us_hu1	26.11.2009	about_us_hu1		cms/page/about_us_hu
OBJAVLJENO	d_message_it	16.11.2009	d_message_it		it/drzavljani.euprava
OBJAVLJENO	po_message_it	16.11.2009	po_message_it		it/poslovni.euprava

Slika 2: Prikaz seznama vseh objavljenih fragmentov v sistemu SRCSI-CMS

Za pripravo fragmenta je treba urediti vsebino delčka strani. Fragment v sistemu SRCSI-CMS urejamo s pomočjo WYSIWYG urejevalnika TinyMCE. Kot je vidno na sliki 3, je priprava vsebine v tem urejevalniku enostavna kot pisanje vsebine v standardnih urejevalnikih besedil. Na voljo so bogati meniji, namenjeni oblikovanju vsebine, vstavljanju tabel, slik in dokumentov ter druge funkcije.



Slika 3: Urejanje fragmenta v sistemu SRCSI-CMS

Ko je vsebina fragmenta pripravljena in potrjena za objavo, je vidna na portalu tudi navadnim uporabnikom. Vsebina je po videzu identična kot v urejevalniku za urejanje fragmenta. Vsebine, kot so slike in dokumenti, so ustrezno glede na nastavitve prikazane na strani in na voljo uporabnikom za prenos na računalnik. Primer prikaza vsebine fragmenta po objavi na portalu (slika 4) vsebuje prikaz obvestila za uporabnike na vstopni strani skupaj z dodanima dokumentom CMS in sliko CMS.

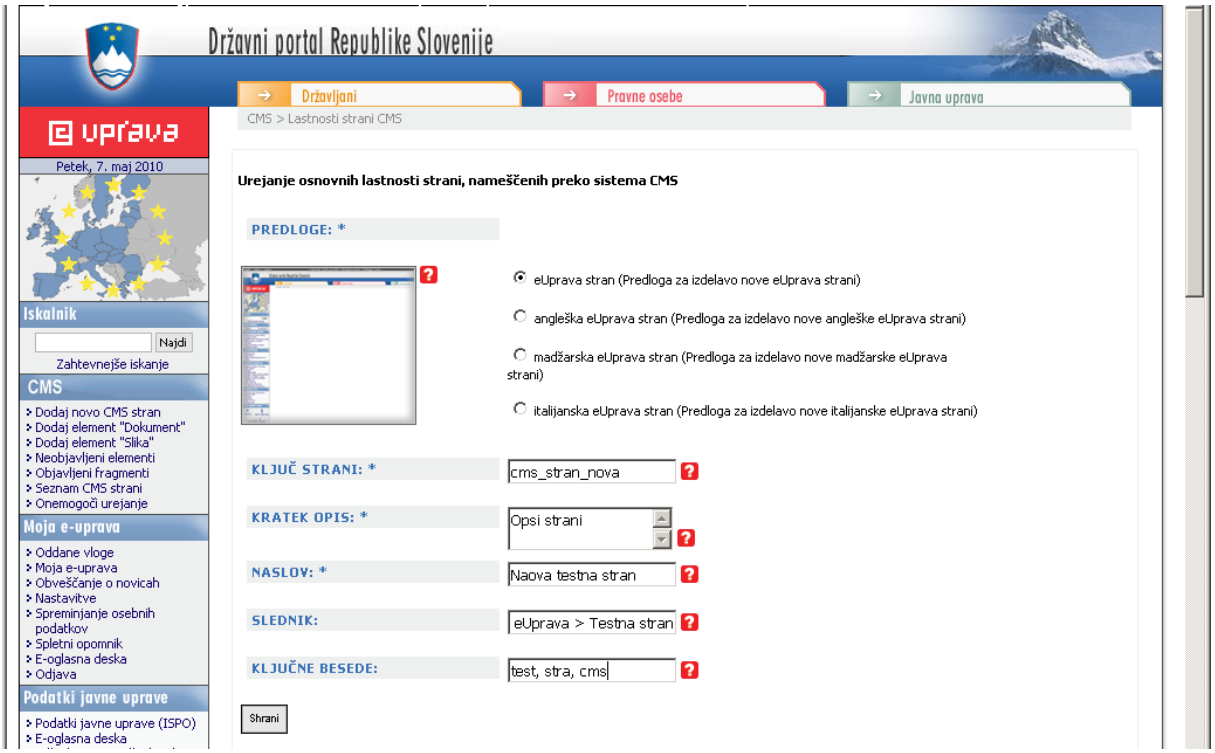
Slika 4: Prikaz vsebine fragmenta uporabniku na portalu (izven sistema CMS)

4.1.2 Strani CMS

Za razliko od fragmentov, pri katerih s sistemom CMS urejamo le del strani, pa gre pri straneh CMS za klasično urejanje celotne vsebine strani, kar je dejansko podobno načinu urejanja, kot ga omogočajo drugi sistemi CMS. Na straneh CMS mora urednik CMS poskrbeti za celotno vsebino strani razen za postavitve strani, ki že vključujejo osnovne elemente, kot so meni, glava ali noga.

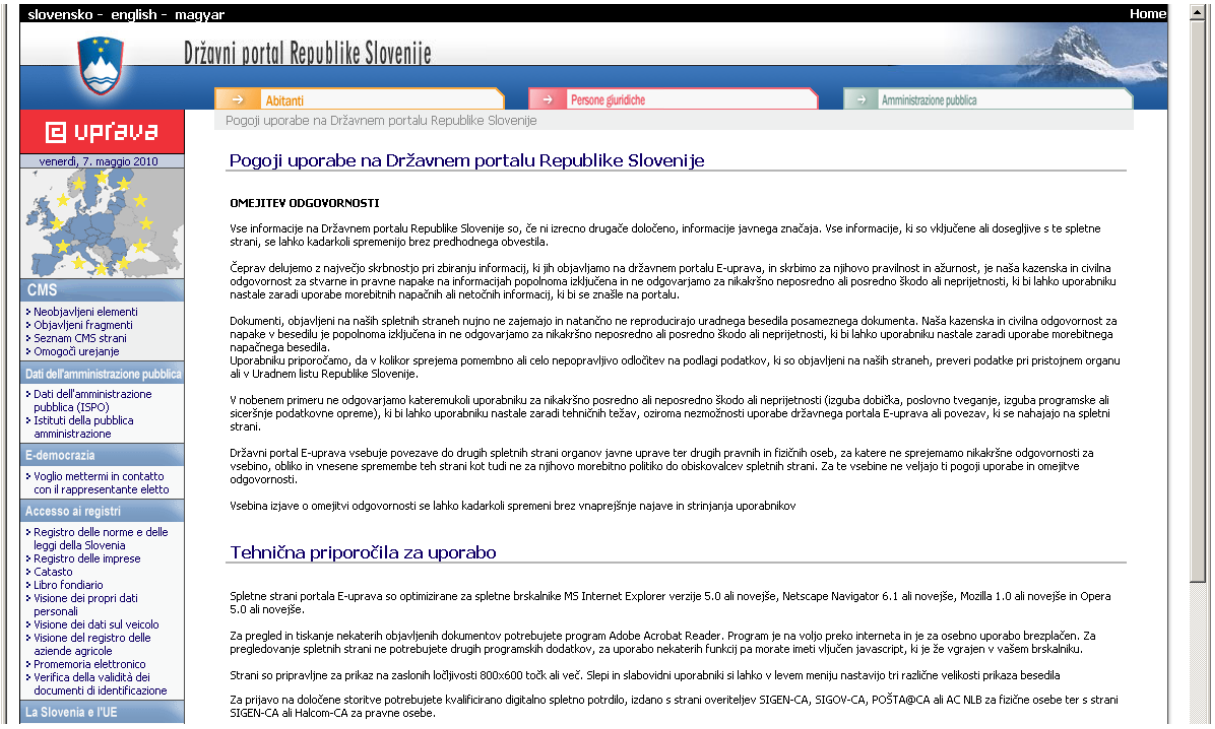
Stran CMS je zgrajena na podlagi predpripravljenega prikaznega vzorca, ki vsebuje enega ali več fragmentov. Prikazne vzorce pripravi razvijalec na podlagi obstoječega videza portala in želene postavitve elementov na strani. Vsaka stran ima svoje metapodatke, kot so naziv, slednik in ključne besede. Po dodajanju nove strani CMS mora urednik CMS pripraviti vsebino za posamezne elemente na strani in nato stran objaviti.

Za pripravo strani CMS mora urednik najprej določiti njene osnovne podatke. Najprej mora izbrati želen prikazni vzorec za določitev videza in postavitve strani. Nato mora vnesti ključ strani, na podlagi katerega je definirana povezava URL. Ta ključ je tudi osnova za ključe vseh fragmentov na tej strani. Sledijo kratek opis strani za interne namene, naslov strani, ki se uporabi za prikaz v spletnem brskalniku, slednik strani, ki se glede na izbrano postavitev prikaže uporabniku, in ključne besede, ki se izpišejo na spletno stran kot metapodatki, namenjeni spletnim iskalnikom. Na sliki 5 lahko vidimo vmesnik za urejanje osnovnih lastnosti strani, nameščene s sistemom CMS.



Slika 5: Urejanje osnovnih lastnosti strani, nameščene s sistemom CMS

Strani CMS imajo še lasten status, ki služi določanju vidnosti strani. Tako je stran, ki je v statusu »neaktivna«, spletnim uporabnikom nevidna, še vedno pa jo vidijo uredniki CMS. Ko pa urednik stran aktivira, lahko do nje dostopajo vsi spletni uporabniki. Slika 6 prikazuje aktivno stran CMS, kot jo vidijo uporabniki na portalu.

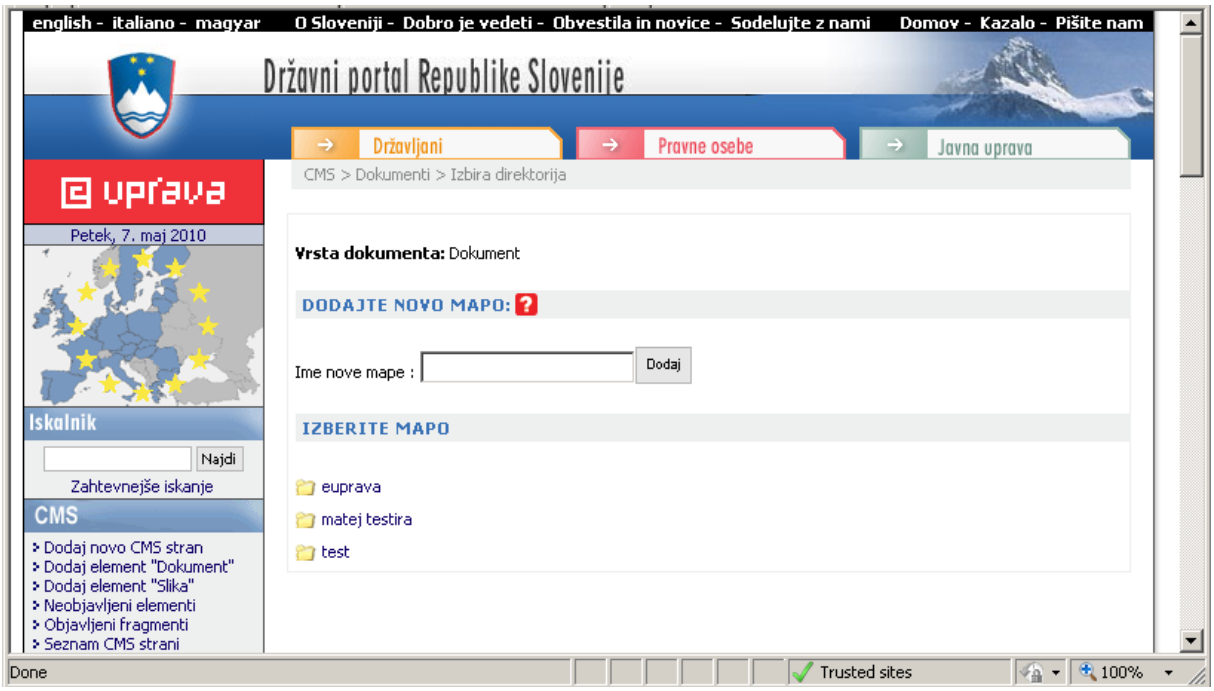


Slika 6: Prikaz vsebine strani CMS na portalu (izven sistema CMS)

Pri urejanju vsebine strani CMS imajo uredniki podobne možnosti kot pri urejanju fragmentov, saj je stran dejansko sestavljena iz enega ali več fragmentov. Za vsak fragment lahko urednik poljubno spreminja vsebino, izvaja akcije potrjevanja in objavljanja glede na delovni tok ter pregleduje zgodovino sprememb fragmenta s podatki o avtorju spremembe. Sistem SRCSI-CMS omogoča še dodatne akcije za prikaz seznama vseh neobjavljenih in objavljenih strani CMS.

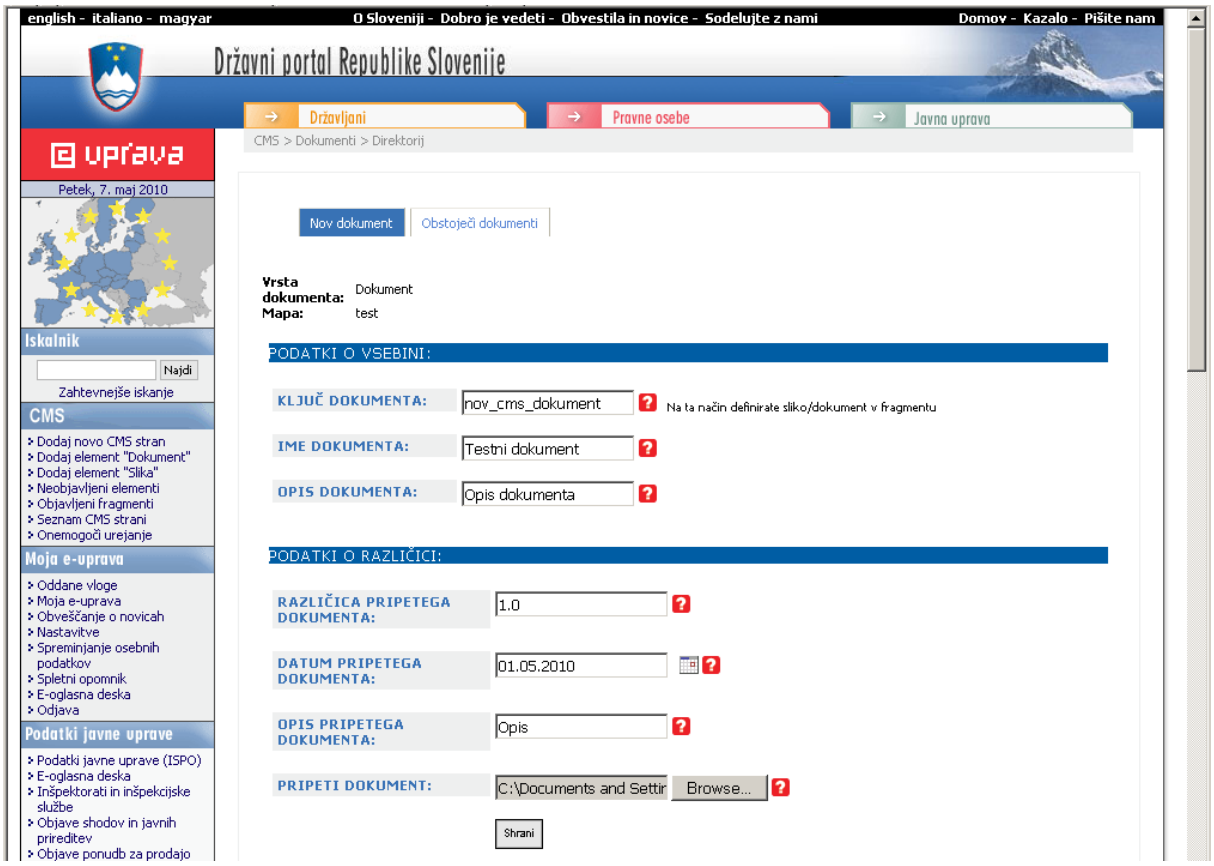
4.1.3 Dokumenti CMS

Dokumente CMS skupaj z metapodatki in fizično datoteko hranimo v bazi. Dokumente v sistem CMS dodajamo s pomočjo vmesnika. Ta v prvem koraku zahteva izbiro fiktivnih map, ki so namenjene razporejanju datotek in jih lahko tudi poljubno dodajamo. Slika 7 prikazuje primer izbire mape za dodajanje dokumenta CMS.



Slika 7: Izbira fiktivne mape, v kateri bo shranjen dokument CMS

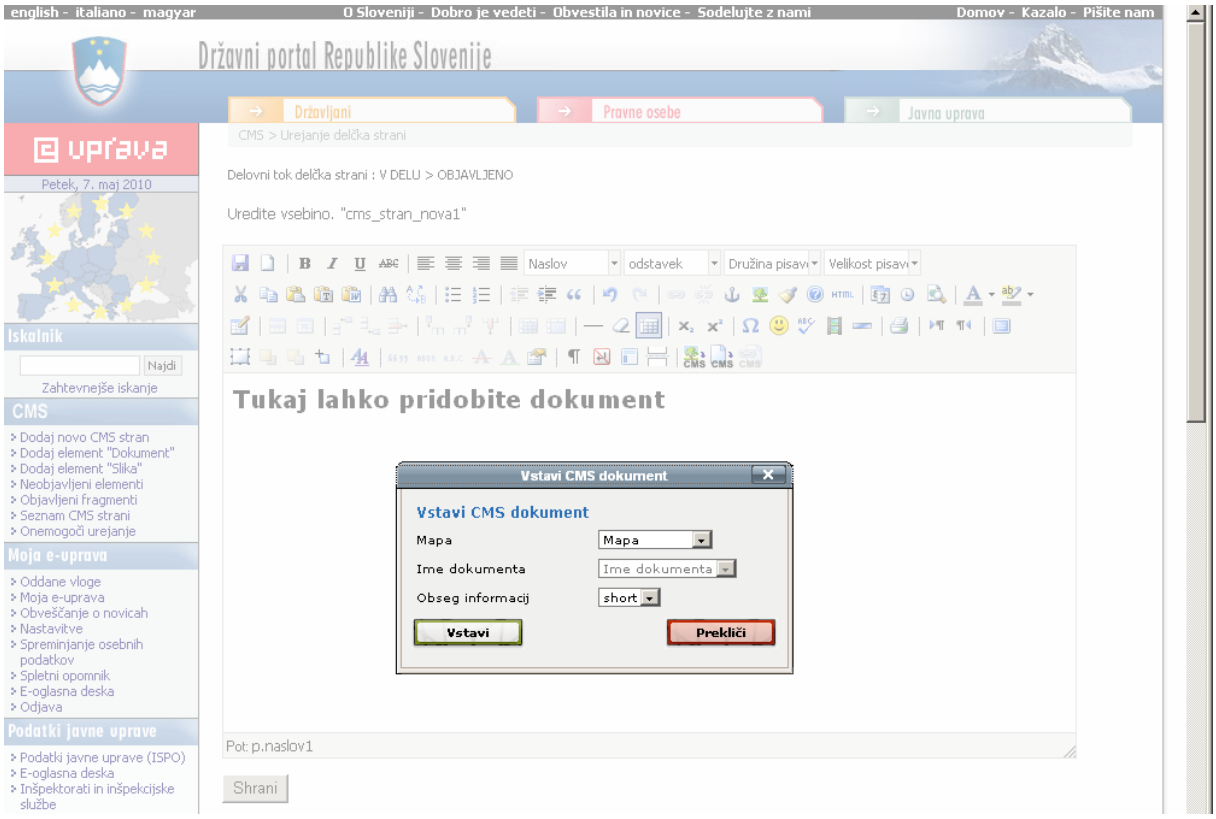
V drugem koraku pa vnesemo metapodatke in pripnemo samo fizično datoteko. Pri tem je treba določiti tudi ključ, s katerim je dokument dosegljiv za prenos s spleta oziroma se identificira za dodajanje v vsebine sistema CMS. Metapodatki o dokumentu, ki jih je poleg ključa možno vnesti, so: ime, opis, verzija in datum dokumenta. Na sliki 8 vidimo vmesnik za vnos metapodatkov o dokumentu in pripenjanje fizične datoteke. Tudi za dokumente sta na voljo funkcija dodajanja nove verzije dokumenta in pregled zgodovine objavljenih verzij.



Slika 8: Vnos metapodatkov in pripenjanje fizične datoteke za dokument CMS

Uporaba dokumentov CMS v aplikaciji je možna na dva načina. Osnovni in najlažji način je uporaba namensko razvitih vtičnikov v urejevalniku TinyMCE. V koraku urejanja vsebine fragmenta lahko v uporabniškem vmesniku izberemo ikono »Vstavi CMS-dokument«, ki aktivira vtičnik za dodajanje dokumenta. Prikaže se pojavno okno (pop-up), kjer iz spustnega menija najprej izberemo mapo, v katero smo dokument dodali. Glede na izbiro mape se osveži še spustni meni s ključi dokumentov, med katerimi lahko izberemo iskani ključ dokumenta. Nazadnje še določimo tip prikaza dokumenta oziroma obseg prikazanih informacij, ki je lahko vrste »short« z nazivom, povezavo in velikostjo dokumenta ali pa vrste »long«, ki poleg omenjenih podatkov izpiše še vse ostale podatke o dokumentu, kot so oznaka, opis, verzija in datum dokumenta. Pri tem velja omeniti, da sistem SRCSI-CMS omogoča zelo enostavno razširitev tipov prikaza z dodatnimi tipi, ki jih definiramo v bazi, in za njih pripravimo poljubno logiko za prikaz v aplikaciji.

Slika 9 prikazuje uporabo namenskega vtičnika za dodajanje dokumenta CMS v fragment, slika 10 pa primer prikaza dokumenta CMS uporabnikom portala glede na različen obseg prikazanih informacij o dokumentu.



Slika 9: Uporaba namenskega vtičnika za dodajanje dokumenta CMS v fragment

Slika 10: Prikaz dokumenta CMS uporabnikom portala glede na različen obseg prikazanih informacij o dokumentu

Drugi način vključitve dokumentov CMS v aplikacijo, ki je bolj namenjen razvijalcem, pa je neposredno dodajanje vključitvene kode na spletno stran, kar zahteva nekaj znanja za pravilno definicijo kode.

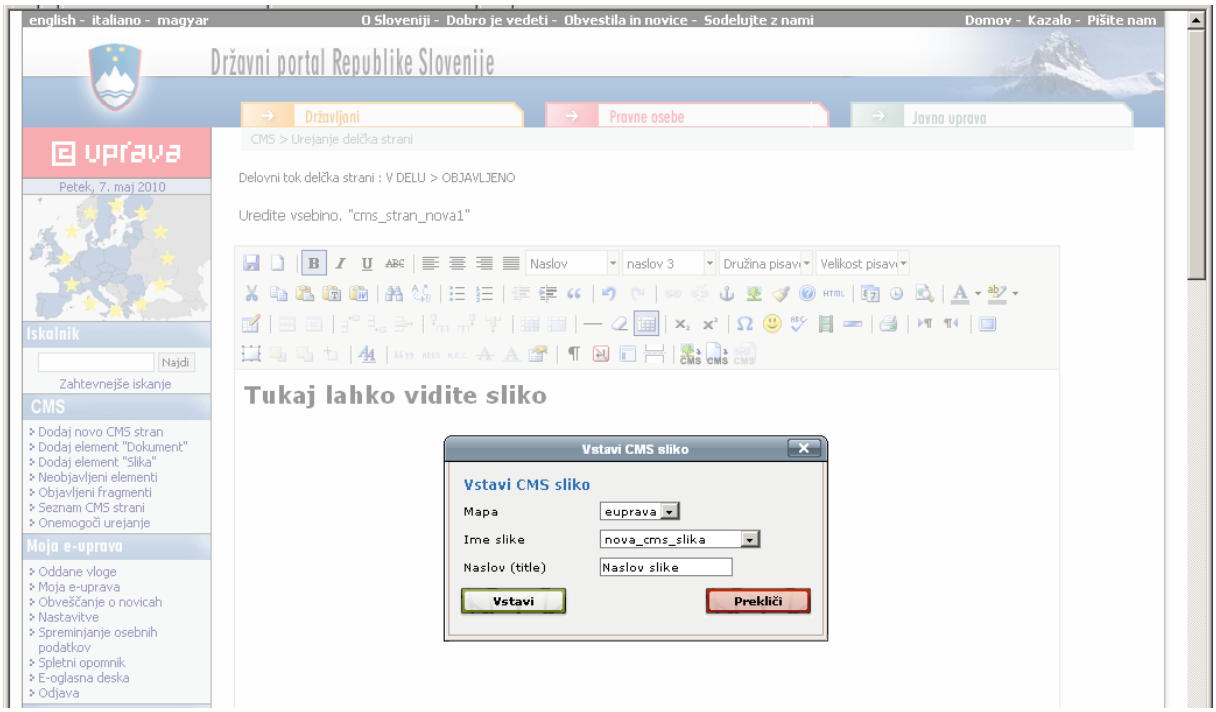
Primer vključitvene kode:

```
<jsp:include page="/cms/doc/info.euprava" flush="true">
  <jsp:param name="srcsi_cms_doc_key" value="test_dokument"/>
  <jsp:param name="srcsi_cms_doc_view" value="long"/>
</jsp:include>
```

4.1.4 Slike CMS

Podobno kot velja za dokumente, velja tudi za slike. Tudi te fizično hranimo v bazi in imajo enake metapodatke kot dokumenti. Za dodajanje je prav tako potrebna izbira fiktivne mape za razporejanje ter v drugem koraku vnos ključa, metapodatkov in fizične datoteke. Vmesnik za vnos metapodatkov o sliki in za pripenjanje fizične datoteke je podoben vmesniku za dodajanje dokumenta CMS (slika 8).

Slike CMS v aplikacijo lahko prav tako kot pri dokumentih vključimo na dva načina. Za dodajanje slike je v koraku urejanja vsebine fragmenta v uporabniškem vmesniku na voljo ločena ikona »Vstavi CMS-sliko«, ki aktivira vtičnik za dodajanje slike. V pojavnem oknu izberemo mapo in ključ dokumenta. Za razliko od dokumentov CMS pa tretji podatek ni tip prikaza, ampak naslov slike, saj vključitev slike v vsebino CMS pomeni neposreden prikaz slike. Na sliki 11 je prikazana uporaba namenskega vtičnika za dodajanje slike CMS v fragment.



Slika 11: Uporaba namenskega vtičnika za dodajanje slike CMS v fragment

Drugi način vključitve, namenjen predvsem razvijalcem, je dodajanje vključitvene kode na spletno stran.

Primer vključitvene kode:

```
<img src="cms/download/image/test_slika"/>
```

4.1.5 Novice CMS

V sistemu SRCSI-CMS je razvita funkcionalnost, ki urednikom CMS omogoča dodajanje novic za portal. Dodajanje izvedemo z vmesnikom, v katerem izberemo tip oziroma domeno novice in vpišemo podatke o novici. Ti podatki so: naslov, datum, povzetek in oznake ter celotna vsebina novice. Uredniki CMS imajo za nadzor nad objavljanjem na voljo celoten delovni tok potrjevanja in objavljanja z ustreznim obveščanjem vseh vpletenih oseb.

Novice so uporabnikom dosegljive v prikaznih fragmentih seznamov novic, ki jih na portal vstavi razvijalec s posebno kodo, pri čemer določi število novic za prikaz in želeno domeno. V seznamu so prikazani kratki opisi in datum objave novice. Za vsako novico v seznamu je dodana povezava do strani, na kateri lahko uporabnik na ločenem prikazu prebere celotno vsebino novice.

Primer vključitvene kode za novice:

```
<jsp:include page="/cms/news/list.euprava">
  <jsp:param name="domain" value="drzavljani"/>
  <jsp:param name="num" value="3"/>
</jsp:include>
```

4.2 Delovni tok sistema CMS

V sistemu SRCSI-CMS je razvit močan sistem delovnega toka za pripravo, potrjevanje in objavljanje posameznih vsebin. Ta sistem definira možna stanja vsebin, njihove prehode in dovoljene uporabniške vloge, ki lahko izvedejo nek prehod oziroma spremembo stanja vsebine.

Osnovni elementi za delovni tok, ki jih definira sistem SRCSI-CMS, so:

- delovni tokovi, ki določajo možne prehode v izvajanju,
- akcije, ki jih je možno izvesti v delovnem toku,
- uporabniške vloge, ki lahko sodelujejo v delovnem toku,
- statusi vsebine, ki določajo območje njene vidnosti.

Glede na osnovne elemente je delovni tok sistema CMS definiran z naslednjimi pravili:

- za vsako uporabniško vlogo so določene možne izvedbe prehodov iz enega v drug status vsebine s točno določeno akcijo;

- vsak delovni tok sestoji iz posameznih korakov; nekateri delovni tokovi imajo določene korake, ki niso obvezni in jih lahko preskočimo;
- za vsak korak delovnega toka je določen možen prehod iz enega v drug status vsebine s točno določeno akcijo.

Celoten delovni tok je s sodelujočimi elementi in pravili definiran s konfiguracijo v bazi, ki se ob zagonu aplikacije naloži v objekte Java za uporabo med izvajanjem administracije CMS. Za posamezno registrirano aplikacijo sistema SRCSI-CMS v bazi glede na želje določimo delovne tokove, ki bodo v uporabi pri urejanju vsebine. Tako imamo na voljo enostavni način definicije delovnih tokov po želji posameznih uporabnikov sistema oziroma naročnika.

Primer izvajanja delovnega toka podajamo na konkretni situaciji iz vključitve sistema SRCSI-CMS v portal e-uprava. Ko aplikacija prikaže neko vsebino CMS, se poleg vsebine uredniku prikaže tudi seznam možnih akcij delovnega toka. Ta seznam aplikacija prikaže glede na uporabniško vlogo urednika in glede na status vsebine. Tako se na primer za fragment, ki je v statusu »v potrjevanju« glede na običajen delovni tok uporabniku, ki ima rolo »potrjevalec«, pri vsebini pojavijo akcije »pošlji v dopolnjevanje«, »zavrni« in »objavi«. Če potrjevalec izbere akcijo »pošlji v dopolnjevanje«, se mu pojavi še dodatno polje za vnos komentarja, v katerega lahko vpiše navodilo za dopolnjevanje vsebine, namenjeno njenemu urejevalcu. Če izvede akcijo »zavrni«, se prav tako pojavi okno za komentar ob zavrnitvi, zavrnjena verzija se iz administracije briše in na bazi prenese v ločeno tabelo zavrnjenih vsebin. V aplikaciji ostane aktualna samo prejšnja verzija vsebine. Če pa potrjevalec izvede akcijo »objavi«, vsebina preide v status »objavljeno« in tako postane dosegljiva za vpogled vsem spletnim uporabnikom strani.

4.3 Predpomnjenje

Predpomnjenje je implementirano s pomočjo sistema ehcache, iz katerega izpeljemo več specifičnih filtrov za predpomnjenje. Hkrati ob spremembi vsebin z urejanjem s sistemom CMS razveljavimo vsebino v predpomnilniku.

4.3.1 Ehcache

Kot upravljalec s predpomnilnikom v aplikaciji uporabimo razred CacheManager iz sistema ehcache. Za konfiguracijo le-tega uporabljamo nastavitveno datoteko »ehcache-default.xml«. V njej je zapisana konfiguracija posameznih predpomnilnikov za posamezne elemente sistema CMS. Konfigurirani so:

- predpomnilnik za strani »PagesCache«,
- predpomnilnik za fragmente »FragmentsCache«,
- predpomnilnik za dokumente »DocumentsCache«.

Predpomnilnik je konfiguriran tako, da se njegova vsebina hrani v datotečnem sistemu.

4.3.2 Filtri za predpomnjenje

Sistem ehcache za uporabo ponuja različne filtre za izvedbo predpomnjenja. Za predpomnjenje celotne strani se priporoča uporaba filtra SimplePageCachingFilter ali pa priprava lastnega filtra, ki razširja razred CachingFilter. Za predpomnjenje samo dela strani pa je priporočena uporaba filtra SimplePageFragmentCachingFilter ali pa lastna implementacija filtra z razširjanjem razreda PageFragmentCachingFilter.

V sistemu SRCSI-CMS so filtri implementirani na osnovi razredov CachingFilter in PageFragmentCachingFilter:

CachingFilter:
- DocumentCachingFilter filter za dokumente CMS (uporablja se predpomnilnik »DocumentsCache«),
- CmsPageCachingFilter filter za strani CMS (uporablja se predpomnilnik »PagesCache«).

PageFragmentCachingFilter:
- CmsFragmentCachingFilter filter za fragmente sistema CMS (uporablja se predpomnilnik »FragmentsCache«).

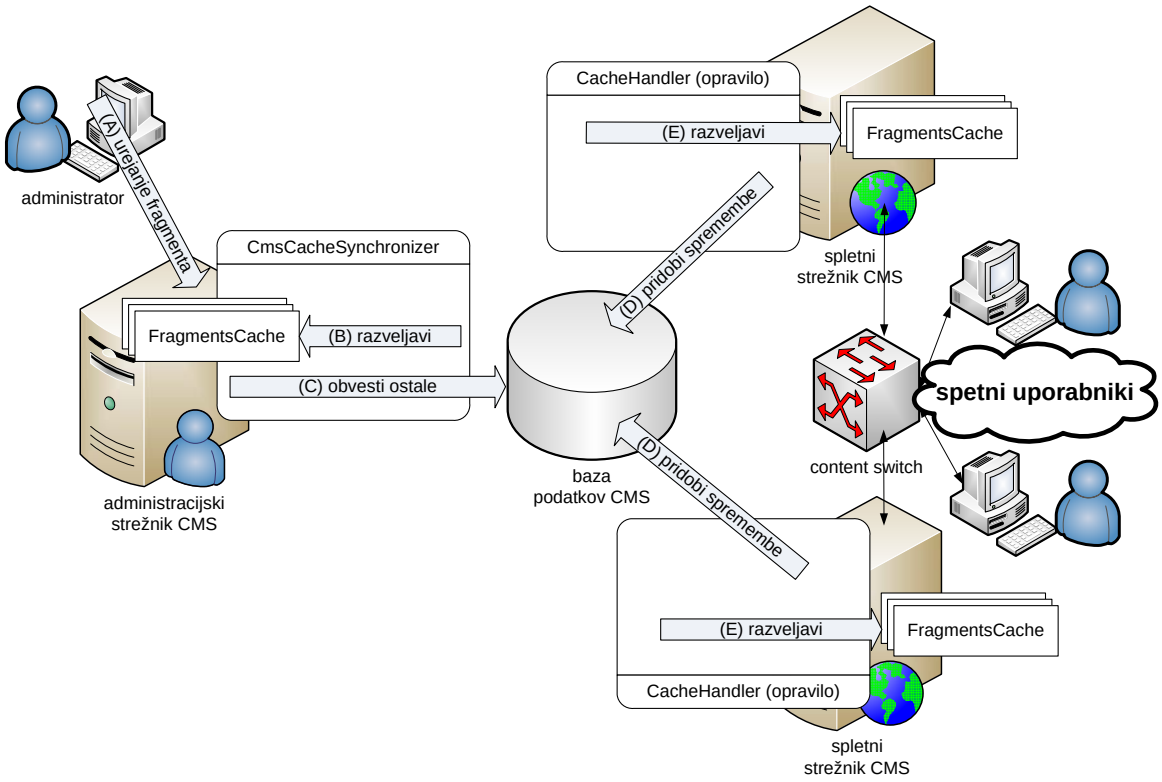
4.3.3 Sinhronizacija predpomnilnikov

V primeru več strežnikov je potrebna sinhronizacija predpomnilnikov na vseh strežnikih. Zato je treba ob spremembi vsebine na administracijskem strežniku tudi razveljaviti vsebino po ostalih ne-administracijskih strežnikih, ki so namenjeni izključno prikazovanju. Za ta namen je v sistemu SRCSI-CMS razvit sistem za sinhronizacijo razveljavljene vsebine.

Razred CmsCacheSynchronizer skrbi za razveljavljanje in sinhronizacijo razveljavljene vsebine med strežniki. Najprej si poglejmo primere razveljavljanja vsebine v predpomnilniku na administracijskem strežniku za konkretne situacije. Ko se spremeni vsebina fragmenta, se izvede brisanje fragmenta iz predpomnilnika »FragmentsCache« in brisanje vseh strani CMS, ki ta fragment vsebujejo. Ob spremembi metapodatkov ali fizične datoteke dokumenta CMS se izvede brisanje dokumenta iz predpomnilnika »DocumentsCache« ter brisanje fragmentov in strani, ki vsebujejo referenco na ta dokument. Ob spremembi osnovnih podatkov o strani CMS ali spremembi njenega statusa se izvede brisanje strani iz predpomnilnika »PagesCache« glede na ključ strani.

Sledita opis postopka urejanja vsebine fragmenta na administracijskem strežniku in opis postopka sinhronizacije vsebine med strežniki. Slika 12 grafično prikazuje postopek sinhronizacije. Najprej administrator uredi vsebino na administracijskem strežniku (A). Ko razred CmsCacheSynchronizer razveljavi vsebino na administracijskem strežniku (B), v naslednjem koraku poskrbi še za sinhronizacijo razveljavljene vsebine med strežniki (C). Za obveščanje ostalih strežnikov se na bazo shranijo metapodatki o spremenjeni vsebini, s pomočjo katerih lahko drugi strežniki brišejo spremenjene vsebine iz svojih predpomnilnikov. Ti strežniki, ki niso administracijski, za razveljavljanje vsebine uporabijo razred CacheHandler. Ta pridobi metapodatke o razveljavljenih vsebinah iz baze (D) in razveljavi vsebine na konkretnem strežniku (E). Razred za razveljavitev vsebine glede na podatke iz baze je definiran kot opravilo (Job), ki ga proži sprožilec (Trigger). Takšna implementacija je

bila izvedena s pomočjo sistema za razporejanje opravil Quartz. Na ta način se spremenjena vsebina ustrezno posodobi v vseh predpomnilnikih, tako v administracijskem, v katerem se vsebina dejansko ureja, kot tudi na ostalih prezentacijskih strežnikih, ki so namenjeni izključno prikazovanju vsebine.



Slika 12: Postopek sinhronizacije vsebine fragmenta med strežniki CMS

4.4 Lokalizacija

Sistem SRCSI-CMS ima programsko podprto lokalizacijo, zato razvijalcem omogoča napredno konfiguracijo lokalizacije za vsako posamezno komponento sistema posebej. Tako lahko poljubno lokaliziramo vse vsebine sistema, tako navadno vsebino kot tudi vse vsebine sistema CMS (fragmente, strani, dokumente, slike, novice).

V sistemu SRCSI-CMS so na voljo trije različni nivoji lokalizacije:

- brez lokalizacije,
- lokalizacija,
- lokalizacija z vračanjem.

Nivo brez lokalizacije pomeni, da se jezik uporabnika pri prikazu vsebine ne upošteva. Na nivoju lokalizacije se za prikaz uporabi jezik uporabnika. Če izbrani jezik za neko vsebino ni podprt, se uporabniku izpiše opozorilo. Pri tretji možnosti nivoja lokalizacije z vračanjem pa se v primeru, da za izbrani jezik vsebina ne obstaja, uporabniku prikaže vsebina za osnovni

oziroma privzeti jezik sistema, s čimer se izognemo opozorilom zaradi manjkajoče jezikovne vsebine.

Konfiguracija lokalizacije

Lokalizacijo konfiguriramo s pomočjo nastavitvenega Java objekta za lokalizacijo, ki ga definiramo v konfiguraciji aplikacije. Nastavimo lahko naslednje lastnosti:

- aktivnost lokalizacije (»usesLocalization«) – glavna nastavitev, s katero določimo, ali aplikacija uporablja lokalizacijo vsebine ali ne;
- privzeti jezik (»defaultLanguage«) – privzeti jezik aplikacije, ki se uporablja pri lokalizaciji z vračanjem;
- podprti jeziki (»availableLanguages«) – nabor dovoljenih jezikov za lokalizacijo aplikacije.

Primer konfiguracije lokalizacije:

```
 <bean id="localizationConfig"
class="srcsi.diploma.localization.DefaultLocalizationConfig">
   <property name="usesLocalization" value="true"/>
   <property name="defaultLanguage" value="sl"/>
   <property name="availableLanguages">
     <set>
       <value>en</value>
       <value>it</value>
       <value>hu</value>
     </set>
   </property>
 </bean>
```

Lokalizacija navadne vsebine oziroma splošnih akcij aplikacije

Kot navadno vsebino sistema pojmujemo vso vsebino, ki ni pripravljena s komponentami sistema CMS. Sem spadajo vse strani, katerih prikaz se generira z ogrodjem Struts, vsebina pa se nahaja v datotekah JSP ali pa je pridobljena iz baze kot poizvedba glede na parametre in ne kot vsebina CMS. Za te strani lahko kot odgovor na zahtevek prikažemo različno lokalizirano vsebino glede na izbrani jezik uporabnika in konfiguracijo lokalizacije. Navadno vsebino za posamezno stran lokaliziramo z različnimi jezikovnimi prikazi odgovora na zahtevek v konfiguraciji akcije.

V ta namen je v sistemu SRCSI-CMS razširjen razred ActionMapping iz ogrodja Struts. V razširjenem razredu je uporabljeno prekrivanje metode findForward, ki preveri nastavljen nivo lokalizacije za akcijo in glede na jezik uporabnika najde ustrezno navigacijo za prikaz odgovora. Tako pri iskanju ustreznega prikaza za navigacijo uporabimo ključ, ki ga glede na nivo lokalizacije ustrezno dopolnimo še z oznako za jezik. Nivo lokalizacije za navadno vsebino določimo z lastnostjo »forwardLocalizationValue« v definiciji akcije.

Primer lokalizacije za navadno vsebino:

```
<action path="/vstop" type="srcsi.diploma.actions.VstopnaAction">
  <set-property property="forwardLocalizationValue"
       value="LOCALE_FALLBACK"/>
  <forward name="success" path="/WEB-INF/vstopna.jsp"/>
  <forward name="success_en" path="/WEB-INF/vstopna_en.jsp"/>
</action>
```

V zgornjem primeru imamo za akcijo »/vstop« definirani dve navigaciji za privzeti in angleški jezik. Nivo lokalizacije pa je nastavljen na lokalizacijo z vračanjem (LOCALE_FALLBACK). Če uporabnik izbere slovenski jezik, se mu bo kot rezultat prikazala stran »vstopna.jsp«. Pri izbiri angleškega jezika pa bo uporabniku prikazana stran »vstopna_en.jsp«.

Lokalizacija fragmentov

Za prikaz fragmentov uporabljamo prikazno akcijo »/cms/fragment/show«. Nivo lokalizacije za fragmente izberemo podobno kot pri lokalizaciji navadne vsebine s pomočjo lastnosti v definiciji akcije, le da se pri fragmentih imenuje »fragmentLocalizationValue«. Lokalizacija vsebine fragmenta pa je implementirana v sami prikazni akciji za fragmente, kjer se za prikaz pravilnega fragmenta ključ »key« za fragment izračuna na podlagi izbranega nivoja lokalizacije in jezika uporabnika. Tako lahko z lokaliziranim klicem akcije za prikaz fragmenta dosežemo, da uporabnik vidi lokalizirano vsebino.

Primer lokalizacije za fragment:

```
<jsp:include
page="/cms/fragment/showLocaleFallback.euprava?srcsi_cms_fragment_ke
y=test" />
```

V zgornjem primeru vidimo, da je za prikaz fragmenta uporabljena akcija »/cms/fragment/showLocaleFallback«, ki ima v definiciji akcije nastavljen nivo lokalizacije z vračanjem (LOCALE_FALLBACK). Tako se bo v primeru, da uporabnik izbere slovenski jezik, prikazala slovenska vsebina fragmenta. V primeru izbire angleškega jezika pa bo uporabniku prikazana angleška vsebina fragmenta, če le-ta obstaja. Če angleška vsebina za fragment ne obstaja, se bo zaradi nivoja lokalizacije z vračanjem uporabniku prikazala vsebina v privzetem, v tem primeru slovenskem jeziku.

Lokalizacija novic

Nivo lokalizacije za novice izberemo podobno kot pri lokalizaciji fragmentov s pomočjo lastnosti v definiciji akcije, le da se pri novicah imenuje »newsLocalizationValue«. Lokalizacija prikaza seznama novic pa je implementirana v sami akciji, kjer je za prikaz pomemben ključ domene, ki ji novice pripadajo. Ključ domene se izračuna na podlagi izbranega nivoja lokalizacije in jezika uporabnika. Pomembno je poudariti, da same vsebine novice ne lokaliziramo, ampak jo napišemo v posameznem jeziku in nato dodamo na ustrezno jezikovno domeno. Tako uporabniku z lokaliziranim klicem akcije prikažemo seznam novic za domeno v njegovem jeziku, ta pa mu omogoča dostop do vsebine novice.

Primer lokalizacije seznama novic:

```
<jsp:include page="/cms/news/listLocaleFallback.euprava">
  <jsp:param name="domain" value="drzavljani"/>
  <jsp:param name="num" value="3"/>
</jsp:include>
```

V zgornjem primeru vidimo, da je za prikaz seznama novic uporabljena akcija »/cms/news/listLocaleFallback«, ki ima v definiciji akcije nastavljen nivo lokalizacije z vračanjem (LOCALE_FALLBACK). Tako se bo v primeru, da uporabnik izbere slovenski jezik, prikazal seznam treh novic za domeno »drzavljani« v slovenskem jeziku. V primeru izbire angleškega jezika bo uporabniku prikazan seznam novic za izbrano domeno v angleškem jeziku. Če takšna domena v sistemu ne obstaja, pa bodo prikazane novice v privzetem jeziku.

Lokalizacija dokumentov
Podobno kot za fragmente velja tudi za dokumente sistema CMS. Nivo lokalizacije za dokumente izberemo s pomočjo lastnosti v definiciji akcije, ki se imenuje »docLocalizationValue«. Pri dokumentih fizične datoteke ne lokaliziramo oziramo ne obstaja več fizičnih datotek za isti ključ. Lokalizirani so zgolj metapodatki dokumenta. Prikaz metapodatkov za dokument se izvede glede na ključ dokumenta, ki se izračuna na podlagi izbranega nivoja lokalizacije in jezika uporabnika.

Primer lokalizacije za metapodatke dokumenta:

```
<jsp:include page="/cms/doc/infoLocaleFallback.euprava"
      flush="true">
  <jsp:param name="srcsi_cms_doc_key" value="test_dokument"/>
  <jsp:param name="srcsi_cms_doc_view" value="long"/>
</jsp:include>
```

V zgornjem primeru vidimo, da je za prikaz metapodatkov dokumenta uporabljena akcija »/cms/doc/infoLocaleFallback«, ki ima v Struts definiciji nastavljen nivo lokalizacije z vračanjem (LOCALE_FALLBACK). Tako se bodo v primeru, da ima uporabnik izbran slovenski jezik, prikazali metapodatki za dokument s ključem »test_dokument« v slovenskem jeziku. V primeru izbire angleškega jezika bodo uporabniku prikazani metapodatki v angleškem jeziku, če pa ti niso navedeni, se bodo prikazali metapodatki v privzetem jeziku.

Lokalizacija slik
Lokalizacija slik je glede metapodatkov identična lokalizaciji dokumentov. Pri slikah se pojavi še dodatna možnost lokalizacije samega prikaza s parametrom »level« pri klicu servleta za prikaz slike. S parametrom lahko podamo nivo lokalizacije, ki se uporabi pri izračunu ključa za sliko.

Primer lokalizacije za sliko:

```
<img src="cms/download/image/test_slika?level=LOCALE_FALLBACK"/>
```

V zgornjem primeru vidimo, da je za prikaz slike uporabljen nivo lokalizacije z vračanjem (LOCALE_FALLBACK). Tako se bodo v primeru, da uporabnik izbere privzeti jezik aplikacije, prikazala slika s ključem »test_slika«. V primeru izbire angleškega jezika bo uporabniku prikazana slika za ključ »test_slika_en«, če pa slika pod tem ključem ne obstaja, bo zaradi lokalizacije z vračanjem prikazana slika s ključem »test_slika«.

Lokalizacija strani CMS
Stran CMS je dejansko en prikazni vzorec, ki ga uporabimo za sestavljanje strani z več elementi. Za lokalizacijo strani CMS mora poskrbeti razvijalec tako, da za posamezne sestavne elemente strani CMS določi ustrezne nivoje lokalizacije. Tako mora določiti nivoje lokalizacije za slike, dokumente in fragmente sistema CMS. Poseben poudarek je pri lokalizaciji strani CMS namenjen tekstovnim poljem, saj je za njih treba priskrbeti ustrezne tekstovne prevode s pomočjo uporabe virov sporočil za vsak jezik. Običajno uporabimo označevalno knjižnico, imenovano Struts Bean Tags. Ti prevodi se nato z uporabljenimi označevalnimi elementi prikažejo na strani glede na jezik, ki je nastavljen v seji.

Primer uporabe virov sporočil z označevalno knjižnico Struts Bean Tags:

```
<bean:message key="msg.drzavni_portal_rs"/>
```

Pri lokalizaciji strani CMS je programsko poskrbljeno za večjezični vnos metapodatkov, tako lahko urednik sistema CMS vnese podatke, kot so naziv, slednik in ključne besede za vsako jezikovno različico posebej. Za lokalizacijo same vsebine strani pa mora poskrbeti za vnos ustreznih prevodov za vse elemente strani in za vse podprte jezike. Prevode za elemente strani vnese po sistemu lokalizacije, kot je bila opisana v predhodnih odstavkih tega podpoglavja.

4.5 Podpora nameščenosti na več strežnikih

Vedno več vzdrževalcev in lastnikov spletnih strani se zaveda pomembnosti izenačevanja obremenitve (angleško load balancing) za spletne strani. Izenačevanje obremenitve je pomembno s stališča optimizacije in izboljšanja prometa v omrežju, saj za strežbo spletne strani omogoča uporabo več strežnikov, med katerimi se promet porazdeli. Pri postavitvi spletne aplikacije na več strežnikih se je treba zavedati določenih težav, ki nastanejo s takšno postavitvijo. Nekatere težave odpravijo že namenski programi ali strojna oprema, ki jih uporabljamo za izvedbo izenačevanja obremenitve. Za nekatere težave pa mora poskrbeti aplikacija sama.

V sistemu SRCSI-CMS je izenačevanje obremenitve predvideno že zaradi ločene namestitve ene instance aplikacije na poseben strežnik, ki je namenjen administraciji CMS. Hkrati pa sistem SRCSI-CMS skrbi tudi za ustrezno medsebojno obveščanje med strežniki.

Sistem SRCSI-CSM predvideva naslednjo konfiguracijo postavitev aplikacije na več strežnikih:

1. administracijski strežnik (CMS–admin), ki je namenjen urednikom CMS,
2. navadni strežnik (non- CMS–admin) za strežbo spletnih zahtevkov s sistemom za izenačevanje obremenitve, ki je namenjen navadnim uporabnikom portala.

Za ustrezno delovanje nameščenosti aplikacije na več strežnikih so bile v sistemu SRCSI-CMS razvite razne rešitve. Nekatere med njimi sem že opisoval v predhodnih poglavjih:

- predpomnjenje s sinhronizacijo razveljavljene vsebine med strežniki: metapodatki za sinhronizacijo razveljavljene vsebine se pripravijo ciljno za vsak strežnik posebej;
- konfiguracija za posamezen strežnik: nastavitev posamezne namestitve aplikacije kot administracijski ali pa neadministracijski strežnik; vklop ali izklop predpomnjenja.

4.6 Varnost

V sistemu SRCSI-CMS je dobro poskrbljeno za varnost pri upravljanju z vsebinami s funkcionalnostmi CMS. Pri izvajanju akcij CMS se v sistemu najprej preveri, ali se akcija izvaja na administracijskem strežniku. Kajti sistem predvideva postavitev administracijski/navadni strežnik, zato se akcije CMS lahko izvajajo samo na administracijskem strežniku. Nadalje sistem preverja še avtorizacijo uporabnika tako, da preveri, če ima uporabnik ustrezno uporabniško vlogo, ki je zahtevana za izvajanje posamezne akcije CMS.

Pri vključitvi sistema SRCSI-CMS v portal e-uprava smo pri varnosti naredili še en korak več, saj se za prijavo v administracijski del CMS zahteva prijava s certifikatom. S tem je poleg avtorizacije uporabniške vloge izvedena tudi avtentikacija uporabnika s pomočjo certifikata, ki dokazuje, da je uporabnik res oseba, za katero se predstavlja. Po prijavi uporabnika v administracijo CMS s certifikatom se vzpostavi varna povezava SSL po protokolu HTTPS, ki skrbi za šifriranje podatkov pri komunikaciji med urednikom CMS in strežnikom.

4.7 Tehnologije sistema SRCSI-CMS

Poleg že opisanih tehnologij portala e-uprava (razdelek 3.2) smo ob vključitvi sistema SRCSI-CMS uporabili še nekaj novih tehnologij, ki jih predstavljam v nadaljevanju.

4.7.1 Spring

Spring Framework je odprtokodno aplikacijsko ogrodje za platformo Java. Ogrodje je bilo prvič izdano pod licenco Apache 2.0 v juniju 2003. Osrednje funkcionalnosti ogrodja Spring lahko uporabljamo v kateri koli aplikaciji Java. Obstajajo pa razširitve za gradnjo spletnih aplikacij na osnovi platforme Java EE. Čeprav ogrodje Spring ne vsiljuje nobenega specifičnega programskega modela, je v skupnosti Java postalo zelo priljubljeno kot alternativa, zamenjava ali celo kot dodatek modela Enterprise JavaBean (EJB).

Osnovni princip jedra (modula Core) ogrodja Spring je inverzija nadzora (angleško Inversion of Control ali IoC). Zabojnik za inverzijo nadzora pa predstavlja osrednji del ogrodja Spring, ki omogoča enoten način konfiguriranja in upravljanja objektov Java z uporabo povratnih klicev. Zabojnik je odgovoren za upravljanje življenjskega cikla objektov: ustvarjanje objektov, klic inicializacijskih metod in konfiguriranje objektov s povezovanjem med njimi. [14]

Tehnologija ogrodja Spring omogoča ločevanje logike krmilnika od poslovnih objektov. Tako lahko vse programske dele aplikacije preprosto povežemo skupaj z datoteko XML. Pri tem je pomembno le, da je vsak programski del na najnižjem nivoju napisan na preprost način, v obliki zrn. V datoteki XML definiramo tudi lastnosti posameznih zrn. Ko zabojnik IoC opremi objekte z referencami objektov, od katerih so odvisni, uporablja privzeto vrivanje odvisnosti (angleško Dependency Injection) z uporabo lastnosti razreda oz. metod »set«. Ker se lastnosti objektov vstavijo šele po njihovi tvorbi, temu pravimo inverzija nadzora. Inverzija pomeni, da aplikacija ne nadzira svoje strukture, temveč zanjo skrbi ogrodje IoC.

V sistemu SRCSI-CMS smo uporabili sistem inverzija nadzora za celotno konfiguriranje aplikacije.

4.7.2 Ehcache

Ehcache je široko uporabljen odprtokoden porazdeljen javanski predpomnilnik za splošno uporabo znotraj Java EE in drugih strežnikov. Omogoča hranjenje v delovni pomnilnik ali na trdi disk, repliciranje s pomočjo kopiranja in razveljavitev, poslušalce, predpomnilniško nalaganje, razširitve predpomnilnika, obravnavanje napak predpomnjenja, servlet filter za stisnjeno predpomnjenje in še mnogo več. Ehcache je dosegljiv pod licenco Apache open source ter se aktivno razvija, vzdržuje in zagotavlja podporo. [15]

Ehcache je odprtokoden predpomnilnik na osnovi standardov. Uporabljamo ga za izboljšavo zmogljivosti, zmanjševanje obremenitev baze in poenostavljeno upravljanje. Je robusten z vsemi potrebnimi funkcionalnostmi, zaradi česar je postal najpogosteje uporabljen predpomnilnik na osnovi Jave. [16]

Sledi predstavitev osnovnih filtrov za predpomnjenje v sistemu ehcache.

CachingFilter
CachingFilter je skladen s Servlet 2.3. Sicer obstajajo tudi predpomnilniki, ki jih uporabljamo kot označevalne knjižnice JSP, npr. OSCache, vendar pa so takšni predpomnilniki odvisni od tehnologije, ki izvaja prikaz. Nasprotje so predpomnilniki z uporabo filtrov. Veriga filtrov se izvede vsakič, ko je vključen RequestDispatcher. To je za vsak »jsp:include« in vsak Servlet. Programsko pa lahko dodajamo lastne filtre. Za vso vsebino, pripravljeno z JSP, Velocity, FreeMarker, String Template, XSLT, Servlet izhodom ali čimer koli drugim, lahko izvajamo predpomnjenje s filtrom CachingFilter. Tako lahko pri programiranju mislimo samo na to, kaj želimo prikazati, pri tem pa nam ni treba skrbeti za predpomnjenje.
Strani se hranijo na osnovi metode `calculateKey` za izračun ključa za predpomnjenje. Filter za predpomnjenje ima abstraktno definirano metodo `calculateKey`, zato omogoča implementacijo glede na željo uporabnika. Pri tem lahko poskrbimo za optimalno implementacijo, saj lahko pri računanju ključa v parametrih zahtevka prezremo parametre, ki ne vplivajo na prikaz v odgovoru.

PageFragmentCachingFilter
Če želimo v predpomnilniku hraniti samo del strani (fragment), lahko uporabimo PageFragmentCachingFilter. Takšni primeri se zgodijo, ko se na primer nek naslov spreminja zelo poredko, delček informacije na strani pa se spreminja pogosto. Za primer lahko vzamemo tudi portal z veliko komponentami in z različnimi stalnimi deli strani. Filter je primeren tudi pri uporabi funkcionalnosti repliciranja predpomnilnika, pri čemer želimo ponovno pridobiti le del strani, ki je bila razveljavljena. PageFragmentCachingFilter ne izvaja stiskanja podatkov, saj se morajo predpomnjeni deli strani sestaviti še z drugimi deli.

Uporabo predpomnilnika ecache v sistemu SRCSI-CMS sem podrobneje predstavil že v podpoglavju 4.3.

4.7.3 Časovnik Quartz za razporejanje opravil

Quartz je celosten odprtokodni sistem za razporejanje opravil. Lahko ga integriramo ali uporabljamo z vsemi aplikacijami Java EE ali Java SE – od najmanjših samostojnih aplikacij do največjih sistemov za elektronsko trgovanje. Quartz lahko uporabljamo za enostavno ali pa zelo kompleksno razporejanje opravil za izvajanje na desetine, stotine ali celo desettisoče različnih opravil. Pri tem lahko opravila izvajajo naloge, ki so definirane kot standardne komponente Java, zato lahko preko njih izvajamo praktično vse, za kar jih programiramo. [17]

Časovnik Quartz lahko izvaja opravila, ki se morajo izvesti v točno določenem trenutku. Prav tako ga lahko uporabimo za ponavljajoča vzdrževalna opravila, ki jih mora opraviti sistem. Primer uporabe razporejanja opravil s časovnikom Quartz:

- vodenje procesa delovnega toka: ob novem naročilu se razporedi opravilo, ki se izvede po 2 urah. Opravilo preveri status naročila, doda sprožilec za opozorilo, če sistem še ni pridobil potrdila za naročilo, in spremeni status naročila;
- sistemsko vzdrževanje: razporejanje opravila za odlaganje vsebine iz baz v datoteko XML na vsak delovni dan ob 12:30;
- priprava opozorilnega sistema za aplikacijo.

Jedro tehnologije Quartz sestoji iz dveh vmesnikov, imenovanih Job (opravilo) in Scheduler (razporejevalec), ter dveh razredov, JobDetail (podatki opravila) in Trigger (sprožilec). Razred Job predstavlja opravilo, ki ga želimo izvajati. Scheduler pa je vmesnik, s katerim izvajamo ukaze za zagon, ustavitev ali začasno prekinitev izvajanja opravila. Razred Trigger vsebuje logiko proženja in omogoča tudi lastno implementacijo, če jo potrebujemo. Razred JobDetail je namenjen ograditvi stanja opravila in posredovanju informacij med njegovimi zaporednimi proženji.

Zaradi enostavnosti, številnih dodatnih funkcionalnosti in enostavnosti razširitve je bila uporaba sistema Quartz za razporejanje opravil v naših aplikacijah samoumevna.

4.7.4 TinyMCE

TinyMCE je platformno neodvisen spletni urejevalnik Javascript za HTML vsebine, ki je objavljen pod odprtokodno licenco LGPL. Je urejevalnik WYSIWYG (angleško What You See Is What You Get), kar pomeni, da je vsebina med vnosom oblikovno podobna ali enaka končnemu rezultatu njenega prikaza. Uporabniški vmesnik spominja na standardne urejevalnike besedil, kot sta Microsoft Word ali Open Office Writer.

Urejevalnik TinyMCE je:

- **enostaven za integracijo**: potrebnih je samo nekaj vrstic kode;
- **prilagodljiv**: različne teme in vtičniki, prisilni atributi, določevanje neveljavnih elementov;
- **prijazen brskalnikom**: Mizilla, MSIE, FireFox, Opera, Safari, Chrome;
- **lahkoten**: PHP/.NET/JSP/Coldfusion GZip compressor naredi TinyMCE 75 % manjši in hitrejši za nalaganje.
- **Kompatibilen s tehnologijo Ajax**: enostavna uporaba Ajax-a za shranjevanje in nalaganje vsebine**;**
- **internacionalen**: podpora večjezičnosti z uporabo jezikovnih paketov;
- **odprtokoden**: brezplačen pod licenco LGPL; veliko prostovoljcev, ki dnevno testirajo urejevalnik in ga tako pomagajo izboljšati. [18]

Urejevalnik TinyMCE lahko kakršno koli tekstovno polje HTML ali druge elemente HTML pretvori v instanco urejevalnika. Lahko ga enostavno integriramo s katerim koli sistemom za upravljanje z vsebinami. [19]

Zaradi široke uporabe v sistemih CMS je bil urejevalnik TinyMCE logična izbira odprtokodnega spletnega urejevalnika WYSIWYG za sistem SRCSI-CMS. V aplikaciji smo uporabili verzijo urejevalnika z oznako 3.2.1.1.

Pri vključitvi urejevalnika TinyMCE v aplikacijo lahko z enostavno konfiguracijo prilagodimo prikaz in funkcije urejevalnika glede na želje in potrebe uporabnikov. Urejevalnik TinyMCE je tudi relativno enostavno razširljiv za dodajanje lastnih komponent za uporabo v urejevalniku. Njegova razširljivost je bila v sistemu SRCSI-CMS dobro uporabljena za implementacijo dodatnih vtičnikov (plug-in), ki omogočajo dodajanje dokumentov CMS in slik CMS v fragmente.

5 Nadgradnja portala e-uprava s sistemom SRCSI-CMS

5.1 Uporabljena razvojna orodja

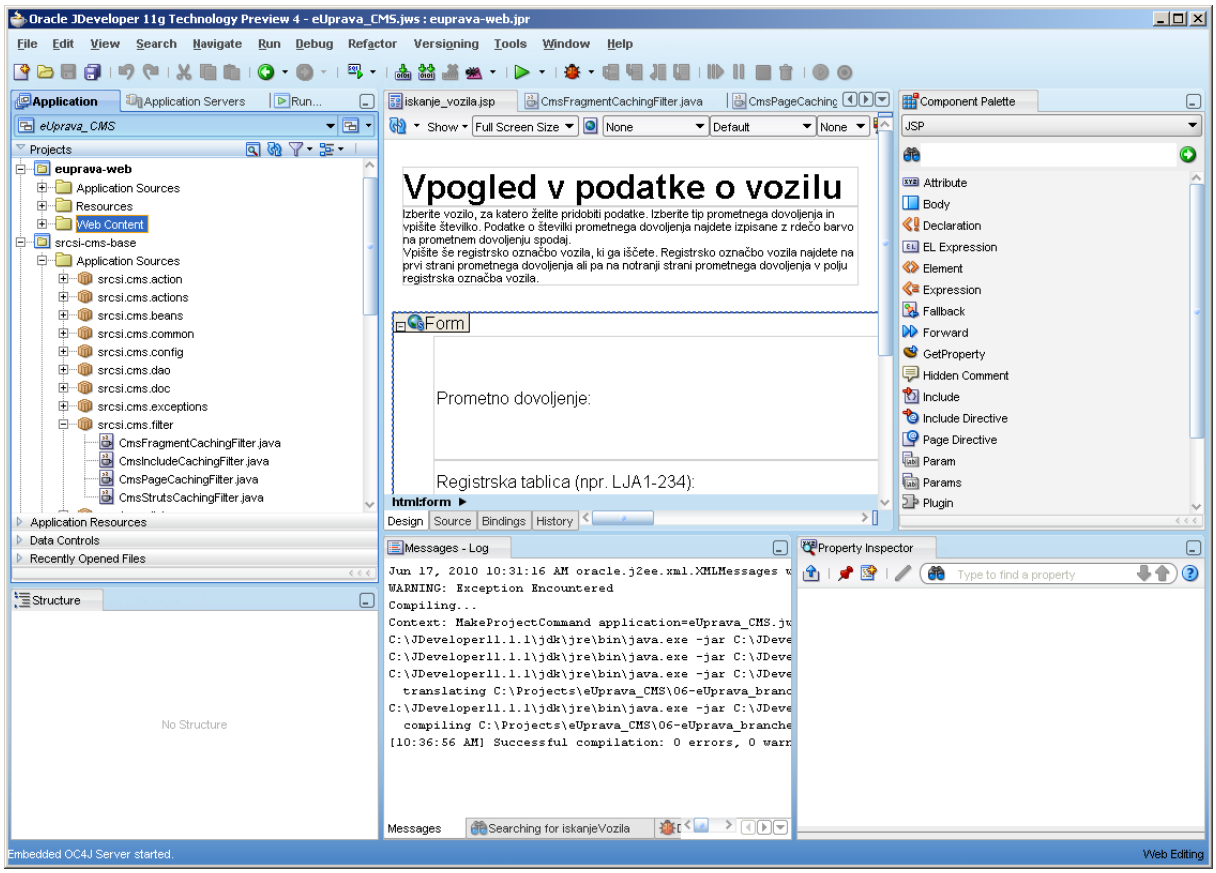
V uvodu poglavja so predstavljena orodja, ki so bila uporabljena pri nadgradnji portala. Predstavljena orodja v naši skupini v podjetju uporabljamo že dlje časa. Razvoj z njimi je enostavnejši in hitrejši, zato smo jih uporabili tudi pri nadgradnji portala e-uprava s sistemom SRCSI-CMS. Za razvoj aplikacije smo uporabili razvojno orodje JDeveloper (razdelek 5.1.1), za upravljanje s podatkovno bazo pa SQL Developer (razdelek 5.1.2) in Toad for Oracle (razdelek 5.1.3).

5.1.1 JDeveloper

JDeveloper je brezplačno integrirano razvojno okolje podjetja Oracle, ki poenostavi razvoj aplikacij in uporabniških vmesnikov. To orodje ponuja funkcionalnosti za razvoj v tehnologijah Java, XML, SQL in PL/SQL, HTML, JavaScript, BPEL in PHP. JDeveloper pokriva celoten krog razvoja od načrtovanja, razvoja, razhroščevanja, optimizacije do priprave namestitvenega paketa. Omogoča poenostavitev razvoja aplikacij z vizualnim in deklarativnim pristopom ter z naprednim okoljem za kodiranje. Jedro orodja JDeveloper izpostavlja programski vmesnik API, ki omogoča razvoj razširitev za JDeveloper. Osnovna platforma JDeveloperja je bila uporabljena kot osnova za še en produkt podjetja Oracle, imenovan SQL Developer, ki ga predstavljam v naslednjem poglavju.

Osnovne funkcionalnosti orodja JDeveloper so: urejevalnik in navigacija po kodi, samodejno preoblikovanje kode, podpora knjižnici Swing, testiranje Unit, upravljanje z verzijami, razhroščevanje, profiliranje, podpora za Ant in XML, odprtokoden programski vmesnik in razširitve, pomoč uporabnikom, podpora tehnologijam, kot so JSP, Struts, JSF, EJB, TopLink, razvoj spletnih servisov, modeliranje z jezikom UML, razvoj baze, priprava namestitvenega paketa.

Orodje JDeveloper ponuja bogat nabor funkcij za kodiranje, koristna orodja in vizualna orodja za drugačen pogled na kodo ter deklarativna pogovorna okna za poenostavitev razvoja komponent Java EE. JDeveloper vsebuje vizualni urejevalnik WYSIWYG za predstavitvene tehnologije HTML, JSP, JSF in Swing. Urejevalnik omogoča, da razvijalec vizualno spremeni prikaz in lastnosti komponent, pri tem pa orodje avtomatsko generira ustrezno kodo. Vse spremembe v kodi so v trenutku vidne tudi v vizualnem pogledu. Na voljo je med seboj podobna podpora toku strani za tehnologiji JSF in Struts. Deklarativne funkcije omogočajo generiranje javanskih zrn EJB in objektov POJO na podlagi obstoječih tabel v bazi. JDeveloper avtomatizira pripravo artefaktov Java EE. Z enostavnim klikom je namreč mogoče razred Java spremeniti v spletni servis, pri tem pa JDeveloper generira WSDL in potrebne komponente JAX-RPC. [20]



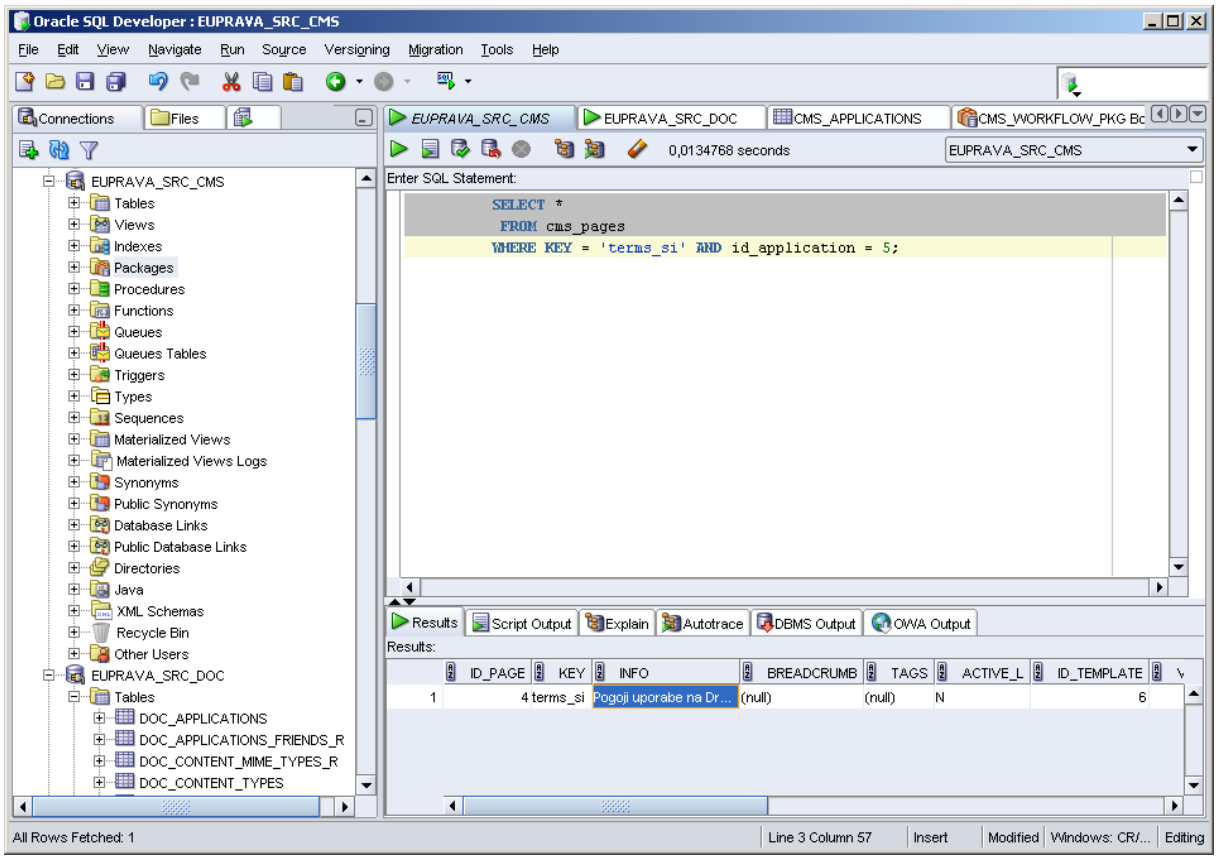
Slika 13: Razvojno orodje JDeveloper

5.1.2 SQL Developer

Oracle SQL Developer je brezplačno grafično orodje za upravljanje Oracle podatkovnih baz. Z njegovo pomočjo je mogoče brskanje po baznih objektih, izvajanje stavkov SQL in skript, urejanje in razhroščevanje stavkov PL/SQL. Omogočeno je tudi ustvarjanje pripravljenih ali lastnih poročil za spremljanje baze podatkov. [21]

Kljub temu da so starejše različice orodja SQL Developer podpirale tudi povezave za druge baze, kot so Microsoft Access, Sybase, in MySQL, je od verzije 1.5 dalje podprta samo baza Oracle. So pa pri drugih razvijalcih na voljo številni vtičniki, ki podpirajo tudi povezavo na druge baze.

SQL Developer ima tudi funkcije sodobnega integriranega razvojnega okolja, kot so avtomatični zavihki, samodejno dopolnjevanje ukazov, formatiranje kode, ujemanje oklepajev in sintaktično obarvanje za PL/SQL. [22]



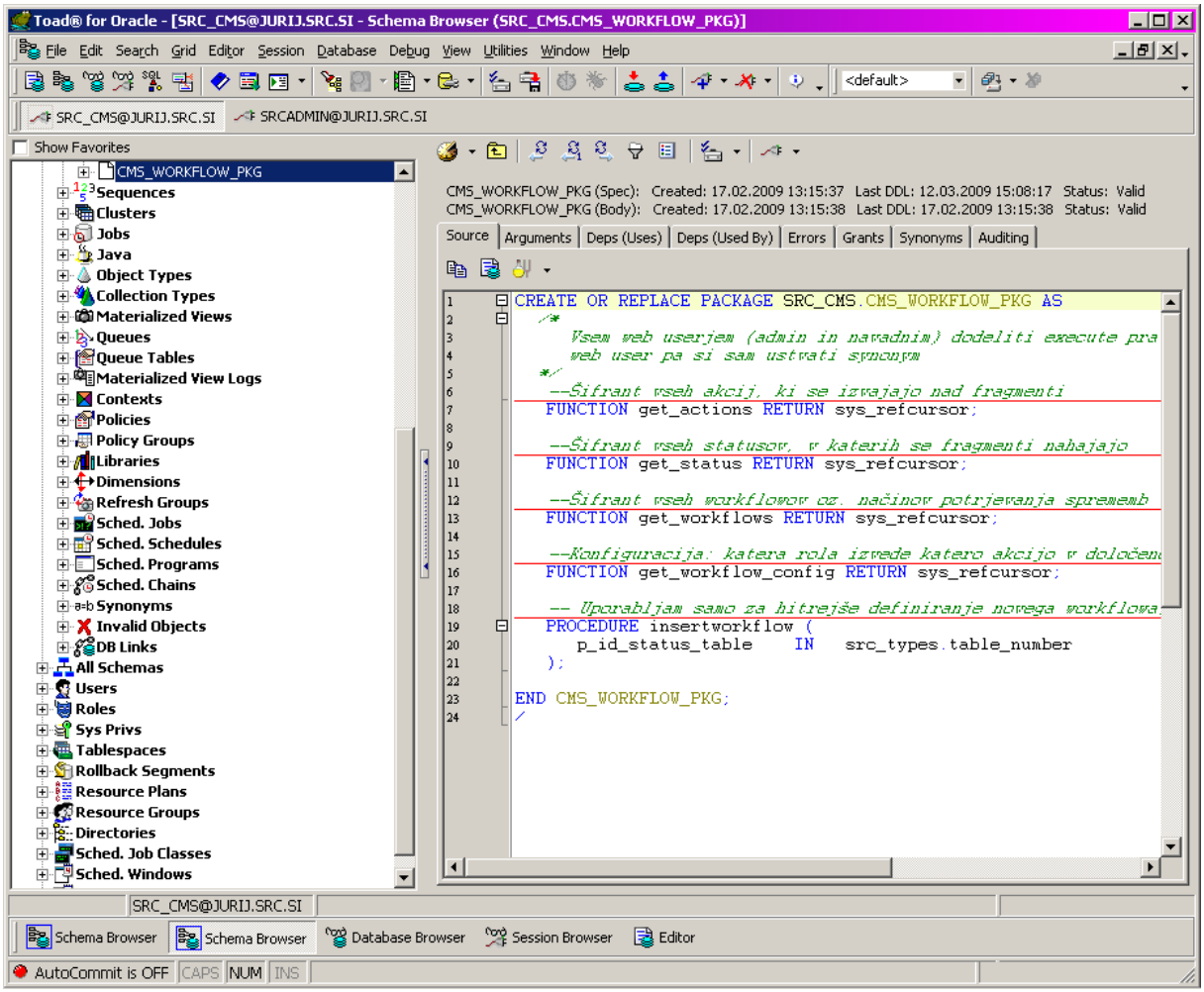
Slika 14: Razvojno orodje za podatkovne baze SQL Developer

Namestitev je enostavna, saj je potrebna le razširitev datoteke ZIP na poljubno mesto. Ker je orodje napisano v Javi, je za delovanje treba zagotoviti ustrezno javansko razvojno okolje.

5.1.3 Toad for Oracle

TOAD (Tool For Oracle Application Development) je močno orodje za hiter in enostaven razvoj podatkovne baze podjetja Quest Software. Orodje ponuja funkcije za poenostavitev administracije in povečanje produktivnosti pri delu z bazo.
Prav tako so na voljo razna poročila z možnostjo izpisa v različne formate, kot so HTML, PDF, XLS, JPG in RTF.

Orodje Toad razvijalcem omogoča gradnjo, testiranje in razhroščevanje paketov PL/SQL, procedur, prožilcev in funkcij. Omogoča kreiranje in urejanje baznih objektov, kot so tabele, pogledi, indeksi, omejitve in uporabniki. Urejevalnik SQL ponuja enostaven način pisanja ter testiranja skript in poizvedb. Močna podatkovna omrežja (angleško data grids) omogočajo pregledovanje in urejanje podatkov.



Slika 15: Razvojno orodje za podatkovne baze Toad for Oracle

5.2 Kaj smo hoteli doseči

Z nadgradnjo portala e-uprava s sistemom CMS smo želeli poenostaviti vzdrževanje portala tako, da bi omogočili hitrejšo objavo popravkov in večjo aktualnost objavljene vsebine. Z uvedbo sistema CMS bi lahko izrazito zmanjšali količino klasičnih popravkov, pri katerih morata intenzivno sodelovati tako izvajalec kot tudi naročnik. Priprava vsebine bi s tem prešla z definicije popravljanja portala s pomočjo izvajalca na definicijo upravljanja portala s pomočjo za vsebino zadolženih uporabnikov sistema CMS na strani naročnika oziroma lastnika portala.

Vemo, da je v današnji dobi hitrega načina življenja najpomembnejša prava informacija ob pravem času. To pa je mogoče doseči zgolj s sistemom, ki programsko podpira pripravo, urejanje in objavljanje informacij neposredno na portalu na hiter in enostaven način. Upravljanje z vsebinami mora biti razumljivo in pripravno za uporabo tudi tistim uporabnikom, ki nimajo naprednega znanja o spletnem oblikovanju in jim je poglavitna vsebina, ki nudi pravo informacijo na pravem mestu.

5.3 Opis postopka prehoda na SRCSI-CMS

Kot vsako drugo podjetje ali organizacija, ki je bila postavljena pred dejstvo nadgradnje spletne strani, je tudi naša skupina v podjetju SRC ugotovila, da si nadgradnje ni mogoče več predstavljati brez sistema za upravljanje z vsebinami – CMS. Večina izvajalcev nadgradnje s sistemom CMS je najprej postavljena pred dilemo uporabe že obstoječih odprtokodnih ali komercialnih sistemov in na drugi strani možnostjo razvoja lastnega sistema, ki je lahko prilagojen specifičnim potrebam uporabnikov. Ker smo se v skupini s potrebami upravljanja vsebin na spletnih straneh v preteklosti že ukvarjali pri drugih projektih, smo imeli prednost internega poznavanja potreb za sistem ter znanje in že implementirane rešitve, ki se uporabljajo na drugih projektih. Ta rešitev se imenuje SRCSI-CMS, ki se že uspešno uporablja v aplikaciji e-VEM. Nadgradnjo portala e-uprava s sistemom za upravljanje z vsebinami je bilo tako možno izvesti s prehodom na (znotraj skupine in aplikacij e-uprave) poenoteno aplikativno ogrodje za razvoj spletnih aplikacij SRCSI-CMS.

Nadgradnja portala e-uprava s sistemom SRCSI-CMS ni pomenila postavitve portala čisto na novo v sistemu CMS, ampak je bila izvedena bolj kot integracija sistema CMS v že obstoječo aplikacijo. S tem smo se izognili potrebi po implementaciji že obstoječih funkcionalnosti v novem sistemu CMS.

Izvedba nadgradnje portala e-uprava je zajemala pet sklopov. To so:
- analiza nadgradnje portala,
- postavitev ogrodja portala e-uprava na sistemu SRCSI-CMS,
- prilagajanje sistema SRCSI-CMS potrebam portala e-uprava,
- dodajanje želenih funkcionalnosti CMS na portal,
- vnos obstoječih vsebin in strani portala v sistem CMS.

5.3.1 Analiza

Pri analizi nadgradnje portala e-uprava s sistemom CMS smo se prioritetno osredotočili na lociranje in definiranje vsebine, ki bi jo glede na vrsto in pogostost spreminjanja bilo primerno in najenostavneje upravljati s sistemom CMS. Upoštevali smo podatke iz nadzornega dokumenta popravkov, ki vsebuje natančno popisane popravke skupaj s seznamom datotek aplikacije, ki so bile spremenjene v nekem popravku. Z identifikacijo pogosto spremenjenih datotek smo dejansko lahko locirali vsebine, ki se pogosteje spreminjajo in so kot take zelo primerne za urejanje s sistemom CMS.

Za potrebe definiranja vsebine za vključitev v sistem CMS smo pripravili dokument, v katerem so bili zbrani vsi za CMS primerni elementi portala e-uprava. Za vsak element smo zabeležili okvirni opis vsebine, datoteko JSP, v kateri se trenutno nahaja vsebina elementa, način, s katerim je element vključen v portal (definicija Tile, vključitev s pomočjo dinamičnega sistema za dodajanje strani) in predlagan ključ novega fragmenta ali strani. Ključ fragmenta ali strani je pri postopku nadgradnje osrednjega pomena, saj bo kot enolični identifikator uporabljen za dostop do nove vsebine sistema CMS preko novih povezav.

5.3.2 Postavitev ogrodja

Portal e-uprava je v času svojega obstoja doživel številne razširitve z novimi funkcionalnostmi, s čimer je postal obsežen sistem, ki implementira več različnih rešitev. Zato je bilo treba novo ogrodje, ki je zasnovano na sistemu SRCSI-CMS, zastaviti dovolj široko, tako da bo lahko pokrilo vse dosedanje implementirane funkcionalnosti portala.

Postavitev ogrodja portala na sistemu SRCSI-CMS je bila osnovna in začetna naloga, ki jo je bilo treba opraviti premišljeno in kakovostno, saj je dobro ogrodje temelj, na katerem bo postavljena celotna aplikacija. Kot vemo, pa so dobri temelji ključ do uspeha. Za aplikacijo portala e-uprava je to zahtevalo temeljito analizo trenutnega stanja, raziskavo vseh uporabljenih rešitev in preučitev možnosti uporabe rešitev znotraj novega ogrodja.

Ključni koraki in implementacije v postopku postavitve ogrodja so bili:

- razširitev definicij Struts z akcijami za delo s sistemom CMS;

- priprava razširjenega razreda Java za objekt uporabnika, ki podpira tudi funkcionalnosti CMS;

- priprava razširjenih razredov Java za aplikacijski filter zahtevkov in varnostno komponento;

- konfiguracija administracijskega dela portala za upravljanje in nadzor nad delovanjem portala;

- uporaba zabojnika Spring IoC za delo z nastavitvami;

  potrebni sta bili priprava in inicializacija osnovnih konfiguracijskih datotek na osnovi ogrodja Spring. Konfiguracija je spisana na osnovi tehnologij, ki jih v svoji implementaciji uporablja in nudi za uporabo sistem SRCSI-CMS. Konfiguracijske datoteke vsebujejo objekte, ki so osnova za delovanje aplikacije. Kot primer naj navedem nekaj sklopov konfiguracije: predpomnjenje, povezovanje z bazo, viri sporočil, varnostne komponente, šifranti, lokalizacija;

- zamenjava lastnih osnovnih in enostavnih akcij z akcijami, ki jih nudi sistem SRCSI-CMS:
  - zamenjava osnovnega razreda za izvajanje akcij, ki razširja razred Action (novi razred omogoča dodatne uporabne funkcionalnosti),
  - zamenjava osnovnega razreda za akcije Ajax ,
  - zamenjava osnovnega razreda za akcije Tile,
  - zamenjava osnovnega razreda s preverjanjem varne povezave za varnostni filter,
  - zamenjava razredov z uporabnimi orodji za poenostavitev delovanja,
  - zamenjava razredov, ki implementirajo funkcionalnosti za optimalnejše delovanje aplikacije;

- sprememba baznih razredov z dosedanjih »singleton« razredov na zrna Java, ki jih upravlja Spring;

- selitev inicializacije zunanjih sistemov portala e-uprava iz obstoječega inicializacijskega servleta v konfiguracijo Spring z uporabo zabojnika IoC (CRP, RPE, MNZ servis eRisk, MRVL);

- priprava predpripravljenega prikaznega vzorca za strani CMS glede na portal e-uprava;

za pripravo vzorca osnovne strani portala e-uprava za uporabo na straneh CMS je bilo treba podrobno analizirati komponente za prikaz strani portala. Pri tem smo morali upoštevati tudi jezikovne module, za katere je prikaz nekaterih delov strani drugačen.

5.3.3 Prilagajanje sistema SRCSI-CMS

Sistem SRCSI-CMS predstavlja osnovo spletnih aplikacij s funkcionalnostmi sistemov CMS. Ker pa noben sistem ne more predvideti vseh funkcionalnosti končne implementacije in potrebne širine ogrodja aplikacije, so bile v postopku vključevanja sistema potrebne določene prilagoditve in razširitve neposredno v sistemu SRCSI-CMS. Te spremembe so bile implementirane tako, da ne vplivajo na že obstoječe aplikacije, ki uporabljajo sistem SRCSI-CMS. Prilagoditve so bile naslednje:

- lokalizacija;

zaradi specifičnosti portala in različne implementacije jezikovnih podportalov e-uprave je bila potrebna nadgradnja lokalizacije sistema SRCSI-CMS. Portal e-uprava na jezikovnih podportalih nima predvidenih jezikovnih različic vseh vsebin portala, zato je bilo treba implementirati in konfigurirati drugačen sistem delovanja portala, ki za posamezen jezikovni modul prikazuje specifično vsebino glede na jezik uporabnika. Prav tako je bilo treba privzeti sistem obravnave uporabniškega jezika iz sistema SRCSI-CMS zamenjati s specifično implementacijo, ki jo uporablja portal e-uprava;

- razvoj dodatnih komponent sistema CMS (administracija videovsebin s sistemom CMS);

opis implementacije funkcionalnosti za administracijo videovsebin v sistemu SRCSI-CMS je posebej predstavljen v poglavju 7. Namen razvoja te funkcionalnosti je bil prikaz enostavnosti razširitve in razvoja dodatnih komponent sistema SRCSI-CMS na konkretnem primeru;

- sistem izbire velikosti pisave, ki potrebuje posebno obravnavo prikaza strani;

na portalu e-uprava je implementiran sistem pomoči za lažji ogled strani, ki omogoča trinivojsko izbiro velikosti pisave. Zaradi specifične implementacije je treba pred prikazom rezultata akcije v request nastaviti pot do prikazne akcije, ki se uporabi pri izpisu logike za spreminjanje pisave. Za ustrezno delovanje je bila potrebna razširitev osnovnega razreda za izvajanje akcij v sistemu SRCSI-CMS;

- vključitev specifičnega delovanja za prikaz dinamičnih strani portala;

za prikaz strani na portalu se uporablja dinamičen sistem za obogaten prikaz strani, za kar je potrebna dodatna priprava dinamičnih podatkov pred prikazom. Podatki se berejo iz konfiguracijskih datotek, ki vsebujejo podatke o naslovu strani, metapodatke in naslovne slike, ki se pojavijo v glavi strani. Prav tako je omogočeno dinamično dodajanje strani z uporabo konfiguracije, ki določa naslov in pot do datoteke JSP, ki se vključi v prikaz strani. Zaradi tega je treba pred prikazom rezultata akcije vse dinamične elemente prebrati

iz konfiguracije in jih shraniti v request za pravilen prikaz (renderiranje) strani. Ustrezno delovanje je bilo doseženo s specifično implementacijo metode, ki jo osnovni razred za izvajanje akcij v sistemu SRCSI-CMS pokliče pred prikazom rezultata;

- funkcija prikaza objavljenih fragmentov;

za lažjo sledljivost fragmentov sistema CMS na portalu e-uprava sem implementiral funkcionalnost prikaza seznama objavljenih fragmentov aplikacije.

Če pogledamo vse naštete prilagoditve sistema SRCSI-CMS portalu e-uprava, ugotovimo, da sistem SRCSI-CMS veliko prilagoditev ni potreboval. Večino teh je bilo treba izvesti zgolj zaradi specifične implementacije rešitev na portalu, ki so bile tako implementirane samo iz zgodovinskih razlogov. Če bi se razvoja rešitev za te probleme lotili s pomočjo novega sistema, prilagoditve sistema SRCSI-CMS najverjetneje sploh ne bi potrebovali. Nizka stopnja potrebnih sprememb na sistemu dokazuje njegovo kakovost in široko zastavljenost, ki omogoča uporabo za razvoj različnih spletnih portalov.

5.3.4 Dodajanje funkcionalnosti CMS na portal

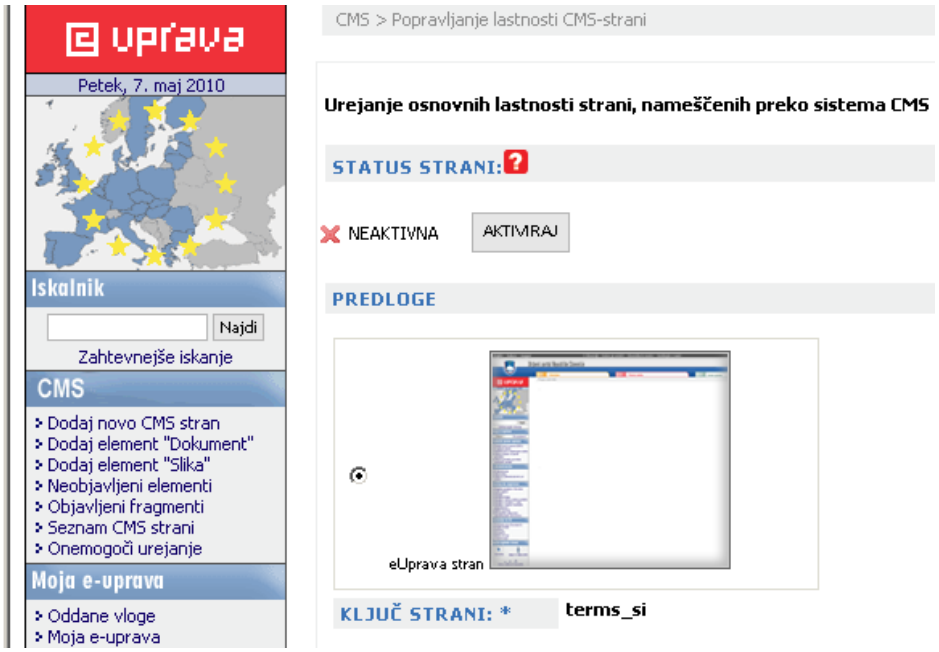
Večina sistemov CMS upravljanje z vsebinami implementira neposredno znotraj iste aplikacije, v kateri so vsebine tudi prikazane. S tem je omogočeno, da si lahko uredniki CMS lažje predstavljajo, kako bo neka vsebina prikazana in umeščena na spletni strani po objavi. Enak princip implementira tudi sistem SRCSI-CMS, ki omogoča urejanje vsebine neposredno na mestih, na katerih je ta prikazana uporabnikom portala.

Postavitvi ogrodja portala e-uprava v sistemu SRCSI-CMS je sledilo vključevanje funkcionalnosti CMS v obstoječo postavitev na predstavitvenem nivoju portala. Vključevanje se je izvedlo v treh korakih.

Prijava urednikov CMS v administracijski del
Najprej je bilo treba najti način za dostop urednikov CMS do funkcionalnosti sistema CMS. Za ustrezen nadzor nad uporabniki sistema CMS se je uporabila avtorizacija prijave na portal s certifikatom, ki je že do sedaj navadnim uporabnikom omogočal naprednejše možnosti uporabe portala e-uprava. Ta način je bil izbran zaradi enostavnosti za uporabo, prav tako pa se z njim izognemo potrebi po dodatni povezavi na portalu za prijavo urednikov CMS v administracijski del portala.

Pridobivanje vlog uporabnika iz varnostne sheme
Po prijavi urednikov CMS na portal je treba za posameznega uporabnika pridobiti njegove vloge, ki so mu po varnostni shemi dodeljene za uporabo sistema CMS. V ta namen se po prijavi urednika izvede klic bazne procedure, ki za podani certifikat urednika iz varnostne sheme CMS pridobi vse vloge. Vloge se nato shranijo na instanco objekta User, ki nosi podatke o uporabniku in se hrani v celotni seji uporabnika. Ta objekt nadalje služi sistemu SRCSI-CMS za ustrezno izvajanje vseh akcij, ki za svoje delovanje zahtevajo ustrezno uporabniško vlogo.

Aktivacija funkcionalnosti glede na varnostno shemo in umestitev v uporabniški meni
Zadnji korak pri dodajanju funkcionalnosti CMS na portal je njihova aktivacija. Za ustrezno delovanje CMS na portalu je bilo treba glede na vlogo, ki jo uporabnik sistema CMS ima, določiti funkcionalnosti CMS, ki jih portal uporabniku omogoča oziroma dovoljuje. V praksi je to dejansko pomenilo pripravo in umestitev menija s seznamom funkcionalnosti CMS v sam portal, tako da bodo na voljo urednikom CMS za uporabo. Za najprimernejše mesto je bil izbran levi meni na portalu, ki ima že sedaj funkcijo hitrega dostopa do najpomembnejših vsebin portala. Administracijski meni CMS je tako umeščen nad sklop »Moja e-uprava«, ki navadnim uporabnikom omogoča naprednejše možnosti uporabe portala e-uprava. Uredniki CMS imajo tako v levem meniju funkcionalnosti CMS neposredno pri roki, s čimer jim je olajšana administracija in omogočen hiter dostop do vseh funkcij sistema CMS. Slika 16 prikazuje meni s funkcionalnostmi CMS.



Slika 16: Meni, namenjen urednikom sistema za uporabo funkcionalnosti CMS

5.3.5 Vnos vsebin v CMS

V našem primeru je nadgradnja portala e-uprava pomenila integracijo sistema CMS v že obstoječo aplikacijo. Torej je večina vsebine na portalu že pripravljena in ta trenutek aktualna za objavo. Vsebine, ki so bile glede na analizo določene kot primerne za upravljanje s sistemom CMS, je tako že bilo možno vnesti v sistem CMS.

Glede na vrsto in namen vsebine se je le-ta v sistem CMS dodajala na dva različna načina. Prvi način je bil dodajanje fragmenta neposredno v obstoječo stran, kjer pod upravljanje CMS preide samo delček strani. Drugi način pa je bila priprava celotne strani, ki se je vnesla kot stran CMS z lastno povezavo.

Dodajanje fragmentov na predstavitveni sloj portala in dodajanje obstoječe vsebine
Analiza je na obstoječem portalu identificirala manjše dele strani, ki se zaradi postavitve in vrste vsebine spreminjajo pogosteje. Kot primer lahko navedemo vstopno stran, na kateri je ob občasnih nadgraditvah sistemov portala ali ostalih aplikacij v sklopu e-uprave treba objaviti obvestilo, ki uporabnike obvešča o motnjah v delovanju posameznih funkcionalnosti portala. Drugi primer je lahko desni del vstopne strani na portalu in podportalih e-uprave, v katerem so nanizane slikovne povezave (angleško banners) na posebej izpostavljene funkcionalnosti portala, na druge aplikacije v sklopu e-uprave ali pa na druge strani, ki so pomembne za uporabnike. Pri obeh primerih gre za spremembo vsebine samo na manjšem delu strani, pri čemer ne želimo vplivati na vsebino preostale strani.
Tukaj bi lahko brez poznavanja ostalih delov strani menili, da bi v takšnih situacijah pod upravljanje CMS lahko dodali celotno stran in pri zahtevah po spremembi vsebine enega elementa spremenili samo del strani. Vendar je takšno mišljenje zmotno, saj je zaradi vključevanja sistema CMS v že obstoječo aplikacijo marsikatera vsebina strani že pridobljena dinamično iz drugih podatkovnih virov, ki se polnijo in urejajo s pomočjo drugih administracijskih aplikacij. Glede na situacijo, v kateri se vsebine pridobivajo dinamično iz različnih virov, je urejanje vsebine CMS po delih nujno.

Priprava strani CMS in dodajanje obstoječe vsebine
Nekatere vsebine, ki so se izkazale kot pogosto spremenljive, pa so podane kot lastne strani in so kot takšne primerne za vnos v sistem CMS kot ločene strani CMS z lastno povezavo in lastnostmi. Pri dodajanju nove strani CMS je bilo treba glede na predpripravljene vzorce strani izbrati ustrezen jezik, definirati enoličen ključ strani, preko katerega bo stran dosegljiva, določiti slednik strani, podati opis in naslov strani ter ključne besede, ki določajo, kakšne podatke stran vsebuje in jih bo tako s pomočjo ključnih besed in spletnih iskalnikov lažje najti. Ker priprava nove strani CMS pomeni tudi drugačno povezavo, je treba za dostop do vsebine te strani uporabiti novo povezavo, ki jo določa enoličen ključ strani. Zaradi tega je treba preiskati obstoječe vsebine portala in morebitne pojavitve povezave na staro stran zamenjati s povezavo na novo stran CMS.

5.4 Nov način objave popravkov

Objava popravkov na portalu e-uprava se je z vključitvijo sistema SRCSI-CMS močno poenostavila. Priprava in izvedba sprememb na portalu sta možni brez posredovanja vzdrževalne ekipe. Naročnik lahko v okviru delovnega toka sistema CMS samostojno pripravi novo vsebino. Ob potrditvi za objavo vsebina pridobi status v potrjevanju. Po varnostni shemi je na strani naročnika oseba, ki je zadolžena za potrjevanje in objavo vsebine. Tako lahko v skrajnem primeru vsebino spremeni in jo objavi za končne uporabnike samo ena oseba na strani naročnika, ki je zadolžena za popravek vsebine. Posredovanje zunanjega izvajalca ni potrebno. Sprememba vsebine portala je zelo hitra in na ta način tudi ažurna. Vrednost informacije pa se tako poveča.

Neposredne spremembe za končne uporabnike po nadgradnji portala e-uprava ni. Zelo pozitiven posredni učinek je večja aktualnost vsebine oziroma informacij, kar pa je za končnega uporabnika lahko zelo pomembno. S pomočjo novega sistema SRCSI-CMS je omogočena hitra objava prave informacije ob pravem času in na pravem mestu.

6 Implementacija administracije videovsebin v sistemu SRCSI-CMS

V tem poglavju opisujem implementacijo funkcionalnosti za administracijo videovsebin v sistemu SRCSI-CMS. Namen razvoja te funkcionalnosti je prikaz enostavnosti razširitve in razvoja dodatnih komponent sistema SRCSI-CMS na konkretnem primeru.

Pod pojmom administracija videovsebine na portalu je mišljeno dodajanje vključitvene kode, ki omogoča predvajanje videa z nekega strežnika z obogateno vsebino. Takšna administracija bi uporabnikom sistema SRCSI-CMS omogočila na enostaven način dodajati razne videovsebine, ki bi obogatile uporabniško izkušnjo portala in mu dale dodano vrednost. Videovsebine so lahko primeren način za podajanje aktualnih informacij, za zanimivejšo predstavitev posamezne tematike, lahko pa so celo videovodnik za informiranje uporabnikov pri uporabi funkcionalnosti portala.

Pri razvoju sistema SRCSI-CMS smo se v podjetju že srečali s potrebo za dograditev spletnega urejevalnika TinyMCE. Takšna primera sta bila dodajanje dokumentov CMS in slik CMS na spletne strani. Ker se dokumenti in slike sistema SRCSI-CMS hranijo v bazi, je bilo treba implementirati vtičnik za TinyMCE, ki omogoča dodajanje že pripravljenih in v bazi shranjenih dokumentov ali slik. Omenjeni funkcionalnosti sem že predstavil v poglavjih 4.1.3 in 4.1.4.

Za implementacijo administracije videovsebin se je kot najboljša možnost izkazala razširitev urejevalnika TinyMCE. Razvoj vtičnika TinyMCE za administracijo videovsebin je bil razmeroma enostaven, saj je urejevalnik TinyMCE že v osnovni postavitvi zastavljen kot skupek različnih vtičnikov. Razvoj in delovanje vtičnika opisujem v naslednjih podpoglavjih.

6.1 Način razvoja vtičnika za TinyMCE

Za razvoj vtičnika v urejevalniku TinyMCE je dovolj poznavanje osnov tehnologij HTML, CSS in JavaScript. HTML uporabljamo za prikaz morebitnih pojavnih oken za uporabnika, stile CSS uporabljamo za določitev vizualne predstavitve prikazane vsebine, v skriptnem jeziku JavaScript pa je napisano jedro delovanja vtičnikov in urejevalnika TinyMCE.

Posamezne elemente razvoja je treba pripraviti v točno določeni mapi znotraj strukture urejevalnika TinyMCE. Pravila za dodajanje vsebine so dobro definirana, opis zahtevane strukture kode pa je opisan v podpoglavju 6.1.1. Za poenostavljen razvoj vtičnikov je na voljo obširen in dobro dokumentiran programski vmesnik TinyMCE API, katerega dokumentacija in navodila za uporabo so na voljo tudi na spletnih straneh. Podrobneje programski vmesnik obravnavam v podpoglavju 6.1.2.

Pri razvoju vtičnika je treba implementirati določene klice za inicializacijo vtičnika, ki omogoča ustrezno delovanje. Za uporabo vtičnika sta potrebni tudi registracija v inicializaciji TinyMCE in vključitev v nadzorno ploščo. Uspešni inicializaciji vtičnika sledi razvoj želenih funkcionalnosti. Primer inicializacije in razvoja konkretnega vtičnika predstavljam v poglavju 6.2.

Ob zaključku razvoja je za uporabo vtičnika priporočena uporaba stiskanja skriptne kode JavaScript za hitrejše delovanje in nalaganje vtičnika ter urejevalnika TinyMCE. Takšno orodje iz podane kode odstrani vse komentarje in odvečne presledke. Predlagana je uporaba orodja Javascript compressor (http://javascriptcompressor.com/), ki sem ga ob zaključku razvoja vtičnika uporabil tudi sam.

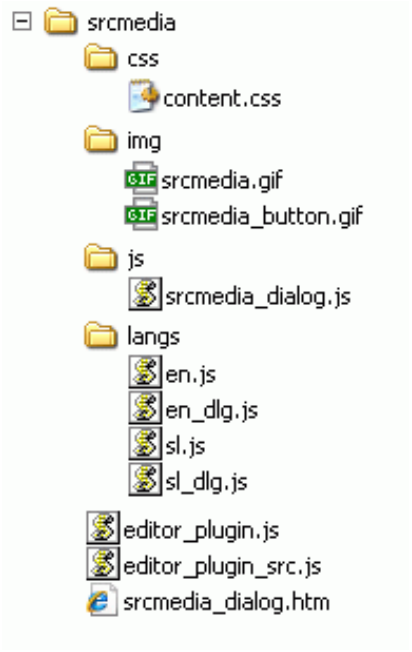
6.1.1 Struktura datotek

Posamezne elemente vtičnika (HTML, CSS, JavaScript) je treba pripraviti v točno določeni mapi vtičnikov znotraj strukture TinyMCE (mapa z imenom plugins) in znotraj nje definirati novo mapo z imenom vtičnika. Ime vtičnika mora biti enolično, da ne pride do napak pri delovanju. Prav tako je priporočljiva uporaba enoličnega imena vtičnika v imenih nastavitev in možnosti, ki so specifične za vtičnik.

Osnovna struktura datotek v enolično poimenovani mapi vtičnika je:

/css	datoteke css z oblikovanjem, specifičnim za vtičnik
/img	slikovne datoteke, specifične za vtičnik
/js	skriptna koda, specifična za delovanje vtičnika in pojavnega okna
/langs **/sl.js** **/sl_dlg.js**	skriptna koda z jezikovnimi vsebinami, specifičnimi za vtičnik Mapa za vsak podprt jezik vsebuje dve datoteki (podajamo primer za slovenski jezik, katerega datoteke imajo v imenu oznako 'sl'): - jezikovna vsebina z opisom vtičnika, ki je uporabljena v nadzorni plošči; - jezikovna vsebina, ki je uporabljena na pojavnem oknu vtičnika.
/editor_plugin.js	stisnjena skriptna koda za inicializacijo vtičnika v urejevalniku
/editor_plugin_src.js	izvorna skriptna koda za inicializacijo vtičnika v urejevalniku
/dialog.htm	datoteka HTML za pojavno okno vtičnika

Primer konkretne strukture datotek za vtičnik za administracijo videovsebin z imenom srcmedia prikazuje slika 17.



Slika 17: Struktura datotek vtičnika srcmedia

6.1.2 Programski vmesnik TinyMCE API

Aplikacijski programski vmesnik TinyMCE API omogoča poenostavljen razvoj vtičnikov za urejevalnik TinyMCE. Dokumentacija z navodili za uporabo je dosegljiva na spletni strani http://wiki.moxiecode.com/index.php/TinyMCE:API.

Razredi programskega vmesnika se delijo v štiri skupine: razredi jedra (angleško Core classes), razredi za dostop do modela DOM (angleško DOM classes), razredi uporabniškega vmesnika (angleško UI classes) in pomožni razredi s koristnimi funkcijami za lažje delo (angleško Utility classes).

V nadaljevanju predstavljam najvažnejše razrede, s katerimi sem se tudi sam srečal in jih uporabil ob razvoju vtičnika, pa tudi druge zanimive razrede.

Razredi jedra (Core classes)

- tinymce

»tinymce« je globalni imenski prostor in vsebuje vse razrede programskega vmesnika TinyMCE API in tudi nekaj osnovnih funkcionalnosti jedra. Med njimi najdemo zelo uporabne metode za delo z nizi, za izvajanje funkcij preko seznamov in druge metode.

- tinyMCEPopup

Razred tinyMCEPopup je pomožni razred za pojavna okna vtičnika, ki ponuja enostaven dostop do osnovne instance urejevalnika in vsebuje metode za delo s pojavnim oknom. Tako na primer omogoča njegovo zapiranje in spreminjanje velikosti.

- tinymce.Editor

Razred Editor vsebuje jedro logike za urejevalnik TinyMCE. Vsebuje lastnosti z instancami ostalih razredov za delo v urejevalniku. Z metodami razreda je možno registrirati številne poslušalce, ki se aktivirajo ob dogodkih v urejevalniku. Prav tako pa nam razred ponuja metode za upravljanje z vsebino urejevalnika, za upravljanje same instance urejevalnika in njegovega videza.

- tinymce.Plugin

Osnovni razred Plugin je izmišljeni razred, ki prikazuje način izgradnje vtičnika za TinyMCE. V konstruktorju za vtičnik so zahtevane metode za inicializacijo (init), nadzor (createControl) in metapodatke z informacijami o vtičniku (getInfo).

Poleg zgoraj naštetih razredov v globalnem imenskem prostoru »tinymce« najdemo še naslednje razrede: tinymce.ControlManager za upravljanje z instancami uporabniških vmesnikov, tinymce.EditorManager za upravljanje z več instancami urejevalnika, tinymce.PluginManager za nalaganje vtičnikov in njihovih jezikovnih paketov, tinymce.UndoManager za upravljanje z nivoji zgodovine sprememb, tinymce.WindowManager za nadzor nad pojavnimi okni in še nekatere druge razrede.

Razredi modela DOM (DOM classes)

- tinymce.dom.DOMUtils

DOMUtils je zelo obširen razred z uporabnimi metodami za upravljanje z modelom DOM. Omogoča pridobivanje vrednosti atributov, celotnih elementov in njihove vsebine ter tudi njihovo spreminjanje in ustvarjanje novih elementov ali atributov.

- tinymce.dom.Element

Razred Element vsebuje metode za delo z elementom modela DOM, ki so večinoma enake kot v razredu DOMUtils, hkrati pa omogoča še premikanje in spreminjanje velikosti elementa.

- tinymce.dom.Selection

Razred Selection je namenjen upravljanju z vsebino, ki jo uporabnik označi v urejevalniku TinyMCE. Prav tako pa lahko označimo poljubni element. Iz označene vsebine je možno pridobiti element ali neposredno vsebino HTML, lahko pa se vsebina tudi zamenja z drugo.

Poleg treh zgoraj naštetih pomembnejših razredov paket »tinymce.dom.« vsebuje še naslednje razrede za delo v modelu DOM: tinymce.dom.Event za upravljanje dogodkov, tinymce.dom.ScriptLoader za nalaganje skriptnih datotek, tinymce.dom.Serializer za serializacijo dreves DOM, tinymce.dom.StringWriter za izpisovanje vozlišč v nize in tinymce.dom.XMLWriter za izpisovanje vozlišč v dokumentno strukturo XML.

Razredi uporabniškega vmesnika (UI classes)

Razredov paketa »tinymce.ui.« za delo z uporabniškim vmesnikom pri razvoju lastnega vtičnika neposredno nisem potreboval. Ta paket omogoča zahtevnejše ravnanje z elementi uporabniškega vmesnika v urejevalniku. Za manj zahtevni razvoj vtičnika pa so dovolj že nekatere pripravljene metode znotraj osnovnih razredov urejevalnika, ki poenostavijo glavne korake razvoja enostavnih vtičnikov. Takšen primer je razred tinymce.ui.Button, ki je

uporabljen pri dodajanju gumba za vtičnik v nadzorno ploščo urejevalnika. Za poenostavljeno kreiranje in dodajanje gumba je že pripravljena metoda addButton v razredu tinymce.Editor, ki kreira instanco razreda tinymce.ui.Button in s pomočjo razreda tinymce.ControlManager doda gumb v nadzorno ploščo.

Pomožni razredi (Utility classes)
Razredi paketa »tinymce.util.« vsebujejo koristne funkcije za lažje delo. Kot najuporabnejšega naj omenim razred tinymce.util.JSON, ki omogoča serializacijo in deserializacijo objektov. Zanimivi pa so tudi ostali: tinymce.util.Cookie za upravljanje s piškotki, tinymce.util.URI za urejanje, serializacijo in deserializacijo naslovov URL, tinymce.util.XHR za pošiljanje zahtevkov Ajax in drugi razredi.

6.2 Razvoj vtičnika za administracijo videovsebin

V naslednjem poglavju opisujem postopek in konkretne primere, ki ponazarjajo, kako je potekal razvoj vtičnika za administracijo videovsebin. Začetne faze so zajemale inicializacijo, registracijo in vključitev vtičnika v nadzorno ploščo urejevalnika. Glavni in najzahtevnejši del pa je bil razvoj funkcionalnosti delovanja vtičnika, katerega ključne elemente predstavljam na koncu.

6.2.1 Inicializacija vtičnika

Pred začetkom razvoja je za ustrezno delovanje vtičnika treba pripraviti osnovno skriptno kodo za inicializacijo vtičnika v urejevalniku. Inicializacijska skriptna koda vtičnika se nahaja v datoteki, imenovani »editor_plugin.js« v osnovnem imeniku vtičnika. Njena vsebina mora ustrezati konstruktorju osnovnega razreda za vtičnike tinymce.Plugin, ki zahteva metode za inicializacijo (init), nadzor (createControl) in metapodatke z informacijami o vtičniku (getInfo). Ključna vsebina se nahaja v metodi init, v kateri definiramo vse osnovne elemente delovanja vtičnika.

V metodi init registriramo:
- akcijo, ki se izvede ob aktivaciji vtičnika;
- element uporabniškega vmesnika, ki bo vtičnik aktiviral (to je najpogosteje gumb);
- poslušalce, ki se aktivirajo ob dogodkih v urejevalniku in so specifični za vtičnik.

Izsek iz datoteke editor_plugin.js za vtičnik srcmedia, ki prikazuje metodo init:

```
/**
 * Initializes the plugin, this will be executed after the plugin has been created.
 */
init : function (ed, url) {
  // Register the command
  ed.addCommand('mceSrcMedia', function () {
    ed.windowManager.open( {
      file : url + '/srcmedia_dialog.htm',
      width : 350 + parseInt(ed.getLang('srcmedia.delta_width', 0)),
      height : 220 + parseInt(ed.getLang('srcmedia.delta_height', 0)),
      inline : 1
    },
    { plugin_url : url });
  });

  // Register button
  ed.addButton('srcmedia',  {
    title : 'srcmedia.desc',
    cmd : 'mceSrcMedia',
    image : url + '/img/srcmedia_button.gif'
  });

  {...}

  // Load the specified CSS file into the document.
  ed.onInit.add(function() {
    if (ed.settings.content_css !== false)
      ed.dom.loadCSS(url + "/css/content.css");
  });
},
```

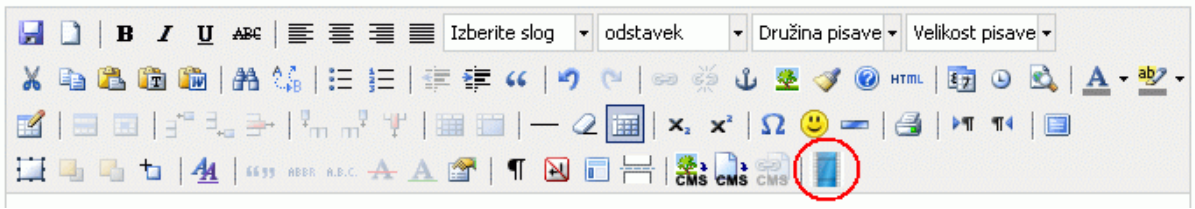
V kodi je jasno razvidno, da smo v urejevalnik za vtičnik srcmedia pod ukaz 'mceSrcMedia' registrirali akcijo odpiranja pojavnega okna z vsebino iz datoteke srcmedia_dialog.htm. Nadalje smo registrirali gumb z imenom 'srcmedia' za ukaz 'mceSrcMedia' in mu določili sliko z naslova /img/srcmedia_button.gif. Poleg ostalih poslušalcev pa smo registrirali še poslušalca, ki ob inicializaciji vtičnika v dokumentno strukturo naloži datoteko /css/content.css z oblikovanjem, specifičnim za vtičnik.

6.2.2 Registracija vtičnika in vključitev v nadzorno ploščo urejevalnika

Fazi inicializacije sledi registracija vtičnika. Za uporabo vtičnika je treba v inicializacijski kodi urejevalnika TinyMCE dodati ime vtičnika v seznam z imenom »plugins«. Pogoj za ustrezno delovanje vtičnika je uporaba naprednejše teme urejevalnika.

Drugi del pa je dodajanje gumba za vtičnik na uporabniški vmesnik oziroma v nadzorno ploščo urejevalnika. Gumb lahko poljubno dodamo na enega izmed seznamov z gumbi. Pri tem pa je pomembno, da uporabimo ime, s katerim smo gumb registrirali. Ob ponovnem

nalaganju urejevalnika pa se na ustreznem mestu v nadzorni plošči pojavi gumb, ki smo ga registrirali ob inicializaciji vtičnika v prejšnjem poglavju. Gumb je viden na sliki 18.



Slika 18: Vtičnik srcmedia, vključen v nadzorno ploščo urejevalnika

Sledi še izsek iz datoteke tiny_init.jsp, v kateri se v sistemu SRCSI-CMS inicializira urejevalnik TinyMCE:

```
<script type="text/javascript" language="javascript">
  tinyMCE.init({
    // General options
    {...}
    theme : "advanced",
    plugins : "safari,pagebreak,style,layer,table,save,advhr," +
        {...} +
        "nonbreaking,xhtmlxtras,template,srccms,srcmedia",

    // Theme options
    {...}
    theme_advanced_buttons4 : "insertlayer,moveforward," +
        {...} +
        ",|,srccms_insertImage,srccms_insertDoc,|,srcmedia",
    {...}

  });
</script>
```

V kodi izpostavimo definicijo uporabe naprednejše teme urejevalnika (theme : "advanced"), registracijo vtičnika srcmedia v vrstici »plugins« in dodani gumb z imenom »srcmedia« v seznamu »theme_advanced_buttons4«.

6.2.3 Razvoj funkcionalnosti

Ko je vtičnik uspešno inicializiran in registriran, lahko sledi nadaljnji razvoj funkcionalnosti. Kot je bilo omenjeno že v prejšnjih poglavjih, je sledila priprava ostalih datotek, potrebnih za delovanje vtičnika:

- datoteka HTML za pojavno okno vtičnika (/srcmedia_dialog.htm):

v datoteki /srcmedia_dialog.htm sem pripravil vse potrebno za prikaz vnosne forme in gumbov za izvajanje akcij vtičnika. Za ločen prikaz dveh območij za različen vnos podatkov sem uporabil v TinyMCE vgrajeno skripto za prikazovanje vsebine z zavihki (razred MCTabs v datoteki mctabs.js). V glavi sem poleg omenjene skripte za zavihke registriral še skripto tiny_mce_popup.js za uporabo priročnega razreda tinyMCEPopup ter mojemu vtičniku lastno skripto /js/srcmedia_dialog.js in datoteko /css/content.css s specifičnim oblikovanjem css;

- datoteka css z oblikovanjem, specifičnim za vtičnik (/css/content.css);

- slikovne datoteke, specifične za vtičnik (/img/…):

v aplikaciji za vtičnik uporabljamo dve sliki. Slika srcmedia.gif je namenjena označitvi videovsebine znotraj urejevalnika, ki je bila dodana z vtičnikom srcmedia. Drugo sliko srcmedia_button.gif pa, kot je že bilo omenjeno, pa uporabimo pri prikazu gumba za vtičnik;

- skriptna koda z jezikovnimi vsebinami, specifičnimi za vtičnik (/langs/…):

v mapi /langs/ se nahajajo jezikovne datoteke, ki vsebujejo mape s pari »ključ: 'vrednost'«. Ključ uporabimo pri definiciji prikaza vtičnika in znotraj skripte za delovanje v določeni obliki `{#srcmedia_dlg.key}`, ob samem delovanju pa se namesto te oblike izpiše pravilna vrednost iz ustrezne jezikovne datoteke glede na jezik uporabnika. Mapa /langs/ za vsak podprti jezik vsebuje dve datoteki: ena se uporabi v nadzorni plošči, druga s končnico »_dlg.js« pa je namenjena vsebini, ki je uporabljena na pojavnem oknu vtičnika;

- skriptna koda, specifična za delovanje vtičnika in pojavnega okna (/js/srcmedia_dialog.js):

jedro delovanja vtičnika se nahaja v skriptni datoteki /js/srcmedia_dialog.js. Tej kodi bom posvetil največ pozornosti, saj je v postopku razvoja predstavljala največji delež vloženega časa in tudi zajema največ logike delovanja vtičnika.

V skriptni datoteki /js/srcmedia_dialog.js sem razvil razred SrcMediaDialog, ki je zadolžen za celoten cikel delovanja pojavnega okna vtičnika od inicializacije ob zagonu, vmesnih pomožnih metod za procesiranje in osveževanje vsebine med vnosom, do generiranja in shranjevanja ustrezne vsebine na pravo mesto kot končni rezultat. Za vsako izmed naštetih akcij je v razredu SrcMediaDialog napisana ena ali več metod, ki so ustrezno aktivirane ob uporabniških akcijah.

Glavne metode razreda SrcMediaDialog so:

- metoda init: inicializacija ob zagonu vtičnika;

metoda init se izvede neposredno ob prikazu pojavnega okna, ki ga uporabnik odpre ob kliku na gumb za vtičnik. Ključna vloga metode init je branje vsebine, ki jo je uporabnik označil pred aktivacijo vtičnika. Če metoda v označeni vsebini prepozna element s točno določenim atributom, je to znak vtičniku, da je uporabnik označil vsebino, ki je bila nekoč ustvarjena s tem vtičnikom. Posledično se lahko izvede procesiranje vsebine in nastavljanje ustreznih

vrednosti na vnosna polja vtičnika. Tako ima uporabnik možnost urejanja obstoječega elementa in ne samo dodajanje novega;

- metodi updateGeneral in updateCode: procesiranje in osveževanje vsebine med vnosom;

ko uporabnik prehaja med dvema načinoma vnosa podatkov, metodi poskrbita za ustrezno osveževanje vsebine v vnosnih poljih vtičnika;

- metoda insert: generirane in shranjevanje ustrezne vsebine;

ob uporabnikovem zaključku z delom v pojavnem oknu vtičnika se ob kliku na gumb za shranjevanje aktivira metoda insert, ki najprej izvede generiranje vključitvene kode glede na uporabnikov vnos. Če glede na podane podatke kode ni mogoče generirati, je uporabnik pozvan k neposrednem vnosu vključitvene kode v ustrezno polje vtičnika. Po uspešni pripravi kode metoda izvede dodajanje ali pa osveževanje vsebine na izbranem mestu v urejevalniku. Kot najprimernejši način vključitve vsebine se je izkazala uporaba elementa HTML tipa SPAN. Na tem elementu sem lahko definiral podane dimenzije, ki v samem urejevalniku uporabniku prikazujejo dejansko velikost videovsebine, in ustrezne stile css, ki omogočajo označitev elementa s sliko vtičnika ter skritje celotne vsebine znotraj elementa. Vsebina elementa je namreč ravno generirana vključitvena koda videovsebine, ki pa je znotraj urejevalnika ni primerno aktivirati, saj bi samo nalaganje videovsebine motilo proces urejanja ostalih elementov v urejevalniku;

- metoda _generateEmbededCode: generiranje vključitvene kode;

metoda se uporabi v fazi osveževanja vsebine med vnosom in ob zaključku dela z vtičnikom. Pri tem se glede na vnesene parametre izvede odločitvena logika, ki poskrbi za ustrezno izgradnjo vključitvene kode za spletno stran. Ta metoda je tudi pravo mesto za razširitev vtičnika, kar predstavljam v poglavju 6.4. V metodi lahko z dodajanjem odločitvene logike omogočimo generiranje vključitvene kode za poljuben videostrežnik, za katerega podamo ustrezno pravilo za gradnjo kode glede na uporabnikove parametre;

- pomožne metode _updateEmbededCode, _parseElementAttributes, _replaceAttributeValues, _fixCodeElements, _fixAttributeElements;

pomožne metode se uporabljajo v metodah updateGeneral in updateCode in so namenjene obdelavi vnesene kode ter branju njenih elementov.

6.3 Funkcionalnosti in delovanje vtičnika za administracijo videovsebin

Delovanje vtičnika srcmedia za administracijo videovsebin temelji na dveh načinih vnosa podatkov za predvajanje videovsebine. Osnovna ideja uporabe vtičnika je bil vnos osnovnih podatkov videovsebine (vir, dimenzije), na podlagi katerih bi vtičnik generiral ustrezno vključitveno kodo za predvajanje videovsebine. Kmalu pa se je izkazalo, da je ob generiranju nemogoče predvideti vse oblike in potrebne parametre za vključitveno kodo, saj je različnih oblik toliko, kot je različnih strežnikov za predvajanje videovsebine. Zato sem se odločil za implementacijo dveh načinov vnosa podatkov:

- osnovni način: možnost vnosa povezave URL z dimenzijami videa,
- način kode: možnost neposrednega vnosa kode za videovsebine.

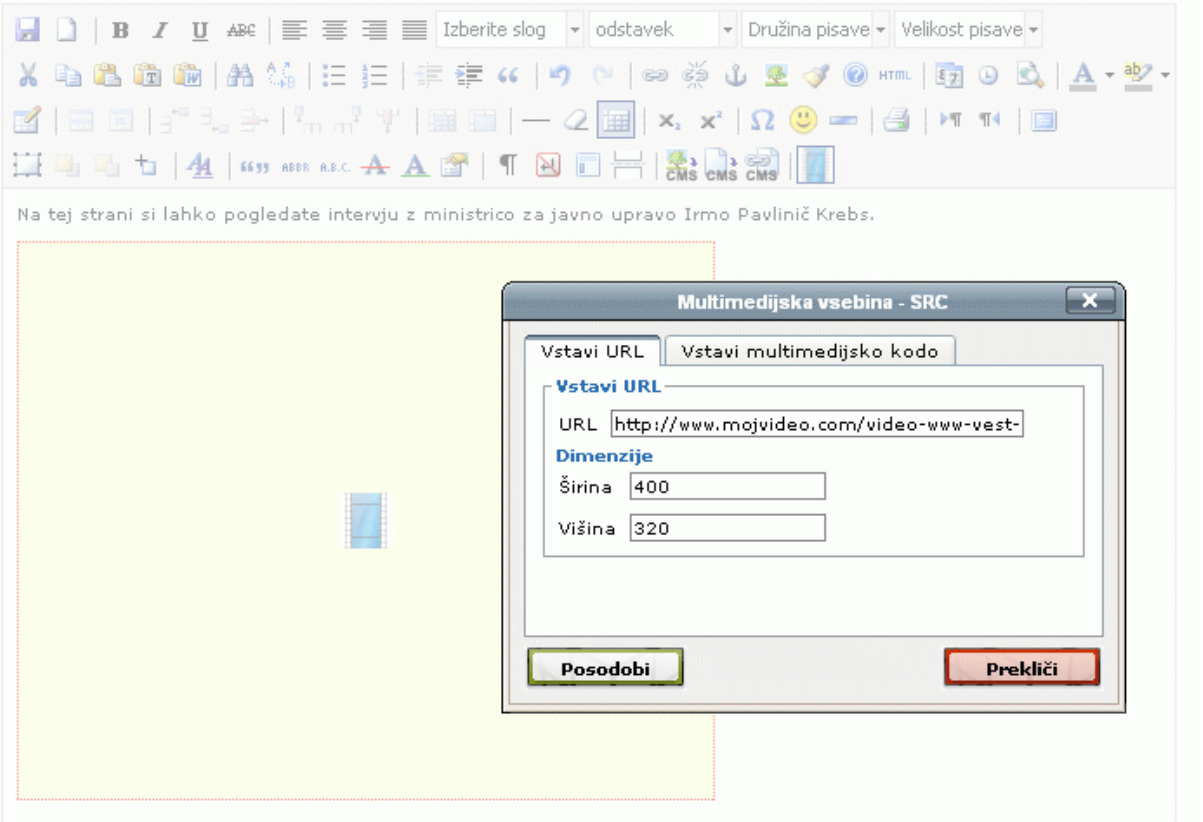
Pri osnovnem načinu je treba vnesti naslov videa, na katerem se nahaja videovsebina. Pri tem lahko kot URL navedemo kar neposredno povezavo do spletne strani, na kateri videovsebina izvorno obstaja. Vtičnik pa poskrbi, da iz povezave pridobi ustrezno oznako videa, na podlagi katere lahko generira vključitveno kodo. Pri osnovnem načinu uporabnik navede še želene dimenzije videa za širino in višino. Pri načinu za kodo pa izpolni samo eno tekstovno polje, v katerem poda celotno vključitveno kodo za video, ki jo je po navadi možno pridobiti neposredno na izvorni strani videovsebine.

Načina vnosa sta med sabo ločena s prikazom na dveh različnih območjih, med katerima uporabnik prehaja s pomočjo zavihkov v pojavnem oknu. Ob prehodih vtičnik poskrbi za osveževanje vsebine v vnosnih poljih vtičnika. Izvede se sinhronizacija med osnovnim prikazom in vneseno kodo ter tudi znotraj vnesene kode med posameznimi pojavitvami parametrov (vir, dimenzije).

V primeru nepoznanega vira vnosa povezave na osnovnem načinu je vtičnik sposoben iz predhodno vnesene kode prebrati parametre za prikaz na osnovnem načinu, tako lahko uporabnik vidi povezavo URL in dimenzije videa. Parametre je nato možno na osnovnem načinu urejati, vtičnik pa spremembe sinhronizira v podani kodi.

V urejevalniku TinyMCE je možno urejanje tudi že obstoječe vsebine, ki je bila pripravljena z vtičnikom. To izvedemo tako, da videovsebino, ki jo želimo popraviti, označimo (ustrezno se označi tudi gumb vtičnika) in ob kliku na gumb vtičnika se pojavi pojavno okno z že napolnjenimi polji iz obstoječe vsebine. V primeru obstoječe vsebine si vtičnik zapomni tudi način zadnjega vnosa vsebine – osnovni način ali pa način kode. Glede na uporabnikov vnos se vtičnik ob naslednjem prikazu pojavnega okna za podajanje vsebine postavi na ustrezno vnosno območje, v katerem je uporabnik nazadnje spreminjal vsebino. Tako lahko uporabnik hitro ugotovi in popravi želeno vsebino ali parametre.

V primeru nepoznanega vira vnosa povezave URL in hkrati odsotnosti neposredno vnesene kode uporabnika, shranjevanje vnosa videovsebine ni uspešno, zato je uporabnik z opozorilnim oknom pozvan k neposrednemu vnosu kode za videovsebino.



Slika 19: Prikaz uporabe vtičnika srcmedia v urejevalniku TinyMCE

6.4 Razširljivost

Razviti vtičnik za administracijo videovsebin je zelo uporabno razširljiv, saj omogoča enostavno dodajanje odločitvene logike, ki pripomore k enostavnejšemu urejanju vsebine portala in s tem k večji uporabnosti sistema SRCSI-CMS. Tako lahko v aplikacijo dodamo logiko, ki generira vsebino, specifično za določeni strežnik za predvajanje videovsebin. Uredniku portala pa konkretno poznavanje potrebnih parametrov za predvajanje vsebin s tega strežnika ni potrebno, saj ustrezno kodo generira vtičnik. Urednik portala v sistem CMS vnese zgolj naslov, na katerem se video nahaja, in želene dimenzije videa.

Kot primer razširitve predstavljam predvajanje vsebin s spletnega videoportala www.mojvideo.com. V primeru dodajanja videovsebine omenjenega videoportala je na izbrano spletno stran treba dodati določeno kodo, ki vsebuje parametre, potrebne za ustrezno predvajanje.

Primer kode za dodajanje videovsebine s portala www.mojvideo.com:

```
<object width="650" height="500">
<param  name="movie"  value="  http://www.mojvideo.com/video-www-
vest-si-irma-pavlinic-krebs/41e63736db92898c3af4"></param>
<param name="wmode" value="transparent"></param>
<embed src="http://www.mojvideo.com/v/41e63736db92898c3af4"
     type="application/x-shockwave-flash" wmode="transparent"
     width="650" height="500"></embed>
</object>
```

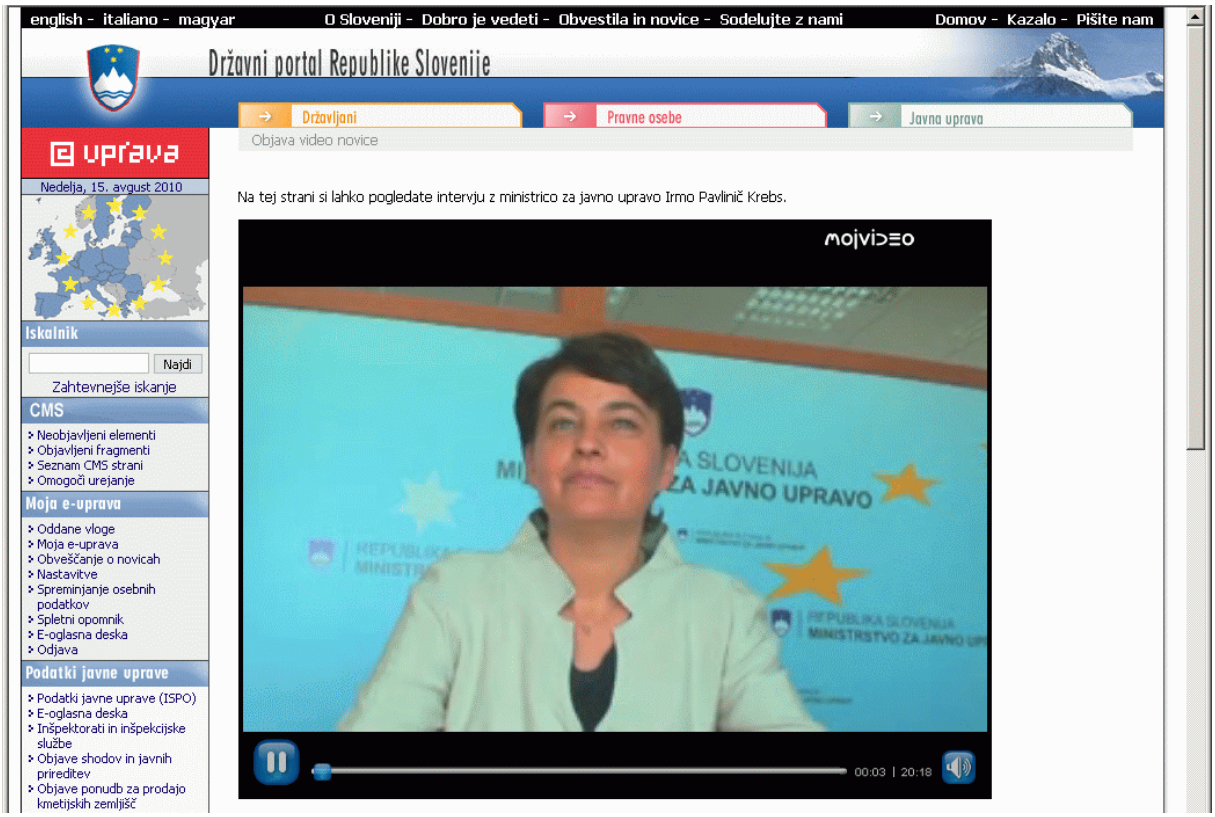
Za podano kodo v aplikacijo dodamo odločitveno logiko, ki na podlagi podatkov o viru videa (`src`) in dimenzijah (`width`, `height`) generira ostalo potrebno kodo.

Odločitvena logika, ki jo je treba dodati v aplikacijo za videovsebine s portala www.mojvideo.com:

```
//Moj video
if (src.indexOf('www.mojvideo.com') != -1) {
  src = 'http://www.mojvideo.com/v/' +
                  src.substring(src.lastIndexOf('/')+1);

  code = '<object width="'+width+'" height="'+height+'" >\n' +
    '<param name="movie" value="'+src+'" />\n' +
    '<param name="wmode" value="transparent" />\n' +
    '<embed type="application/x-shockwave-flash"' +
    ' wmode="transparent" width="'+width+'"' +
    ' height="'+height+'" src="'+src+'">' +
    '</embed>\n' +
    '</object>';
}
```

Rezultat dodane odločitvene logike je enostavno dodajanje videovsebin z omenjenega videoportala na spletno stran s pomočjo sistema SRCSI-CMS. Za dodajanje vsebine mora urednik portala zgolj podati vir (npr.: `http://www.mojvideo.com/video-www-vest-si-irma-pavlinic-krebs/41e63736db92898c3af4`) in izbrane dimenzije (npr.: 650, 500). Videovtičnik ob dodajanju vsebine poskrbi za generiranje kode, rezultat pa je ustrezno konfigurirana videovsebina za predvajanje na spletni strani (slika 20).



Slika 20: Prikaz videovsebine na spletni strani e-uprave

Kot lahko vidimo, je razširitev vtičnika za administracijo videovsebin razmeroma enostavna. Sicer zahteva poseg razvijalca, ki je potreben samo enkrat ob definiciji ustrezne kode, ob nadaljnji uporabi pa sistem kodo generira avtonomno. Najprimernejši pa je vnaprejšnji dogovor med razvijalcem in uporabnikom, ki definira vse predvidene vire, s katerih bo videovsebina uporabljena oziroma predvajana. Tako lahko razvijalec implementira odločitveno logiko, ki zajame vso potrebno generiranje kode za podane vire.

7 Zaključek

Nadgradnja portala e-uprava s sistemom SRCSI-CMS ni pomenila postavitve portala čisto na novo na sistemu CMS, ampak je bila izvedena bolj kot integracija sistema CMS v že obstoječo aplikacijo. S tem smo se izognili potrebi po implementaciji že obstoječih funkcionalnosti v novem sistemu CMS. Nadgradnja portala e-uprava je zajemala analizo nadgradnje, postavitev ogrodja na sistemu SRCSI-CMS, prilagajanje sistema SRCSI-CMS potrebam portala e-uprava, dodajanje želenih funkcionalnosti CMS na portal ter vnos obstoječih vsebin in strani portala v sistem CMS.

Po nadgradnji portala smo ugotovili, da se je njegovo vzdrževanje zelo poenostavilo. S sistemom SRCSI-CMS je upravljanje portala hitrejše in enostavnejše. Prvi odzivi vzdrževalcev portala so pozitivni. Naročnik se je hitro vpeljal v uporabo sistema SRCSI-CMS in po začetni fazi njegove uporabe že aktivno sodeluje in daje ideje, katere do sedaj statične strani bi bilo še primerno dodati v administracijo vsebine s sistemom SRCSI-CMS.

Sklepamo lahko, da je bila izvedena nadgradnja portala uspešna. Pri sami nadgradnji sistema nismo imeli večjih težav. Zaradi vključevanja sistema v že obstoječo aplikacijo so bile potrebne določene prilagoditve, ki pa za implementacijo niso bile težavne.

Dodana vrednost nadgrajenega portala je najvidnejša pri objavi popravkov portala. Če je priprava popravka pred nadgradnjo zahtevala večštevilčno ekipo na strani izvajalca, je pri novem sistemu sploh ne potrebujemo. Prav tako pa je na strani naročnika dovolj zgolj ena odgovorna oseba, ki ima možnost objave popravka v roku nekaj minut in samo z nekaj kliki.

Nadalje lahko poudarimo, da je bilo pred nadgradnjo treba izvajati večje število popravkov lahko tudi zaradi minimalnih sprememb, kot so: popravek nekaj črk ali ločil, sprememba povezave, zamenjava slike, zakritje neaktualne povezave in podobno. Število popravkov se po nadgradnji izrazito zmanjša, saj je po analizi najpogosteje spreminjane vsebine možno spreminjati s pomočjo sistema SRCSI-CMS. Potreba po pošiljanju popravkov se tako omeji zgolj na vsebine zunaj sistema SRCSI-CMS, spremembe v logiki aplikacije in ostale dograditve sistema.

Še en pozitiven vidik pa je tudi konsistenca vsebine popravka. Pri starem načinu objave popravkov se je namreč ustreznost vsebine popravka preverjala na več posameznih namestitvah aplikacije po načelu RTP (razvoj, test, produkcija). Pri tem je lahko prišlo do nekonsistentnosti nameščenih popravkov in zaradi tega do napačnega prikaza vsebine. Popravki vsebine s sistemom SRCSI-CMS pa se vršijo v enem okolju, ki omogoča delovni tok pregledovanja in potrjevanja vsebine za objavo. Tako do napačnega prikaza vsebine ne more priti.

Zaradi uporabe naprednih tehnologij je pričakovana življenjska doba portala lahko zelo dolga. V tem času tehnična nadgradnja ne bo potrebna. Vsebinske spremembe portala pa so enostavno izvedljive preko njegove administracije s pomočjo sistema SRCSI-CMS. Seveda pa je vse odvisno od zahtev naročnika. Lahko se pojavi potreba po spremembi oblike vsebine, kar pa spet ne pomeni tehnične nadgradnje, ampak zgolj menjavo sloja, namenjenega vizualizaciji nad enako vsebino. Dograditve sistema niso problematične. Široko zastavljen programski vmesnik pušča veliko prostora za nadgraditev sistema z novimi komponentami.

Enostavno razširljivost sistema sem v diplomskem delu prikazal z implementacijo administracije videovsebin v sistemu SRCSI-CMS.

Sam sem sodeloval v vseh petih fazah izvedbe nadgradnje portala e-uprava: od analize, postavitve ogrodja, prilagajanja sistema, razvoja dodatnih funkcionalnosti in vnosa vsebin v CMS. V okviru analize sem sodeloval pri planiranju potrebnih korakov in svetoval pri lociranju vsebin za sistem CMS. Sam sem tudi zastavil, koordiniral in s pomočjo sodelavca implementiral postavitev ogrodja na sistemu SRCSI-CMS. Glede na to, da sem v preteklih treh letih od junija 2006 aktivno sodeloval pri vzdrževanju in razvoju novih funkcionalnosti na portalu e-uprava, sem bil s portalom seznanjen v tolikšni meri, da sem lahko brez večjih težav določil potrebne korake za postavitev novega ogrodja. Pri tem sem s svojimi napotki in navodili pri izvajanju posameznih korakov usmerjal ter nadziral sodelavca pri izvedbi posameznih sprememb v kodi aplikacije. Hkrati pa sem samostojno izvedel kompleksnejše korake pri postavitvi ogrodja. Po posvetovanju z glavnim arhitektom sistema SRCSI-CMS sem izvedel tudi določene nadgradnje in prilagoditve sistema SRCSI-CMS, ki so bile ključne in nujne za uporabo sistema pri nadgradnji portala e-uprava. Implementiral sem dodajanje želenih funkcionalnosti CMS na portal in na koncu pomagal ter nadziral prenos obstoječih vsebin in strani portala v sistem CMS.

Med sodelovanjem in samostojnim izvajanjem večjega dela nadgradnje portala e-uprava s sistemom za upravljanje z vsebinami sem se podrobno seznanil s posameznimi funkcijami in koncepti sistemov CMS. Pri tem sem pridobil veliko bogatih izkušenj s področij uporabljenih tehnologij za predpomnjenje, razporejanja opravil, spletnega urejevalnika vsebin WYSIWYG in ogrodja Spring.

7.1 Možne izboljšave in razširitve sistema

Po nadgradnji portala e-uprava sem glede na izkušnje, pridobljene pri analizi, lahko kritično ocenil sistem SRCSI-CMS in oblikoval predloge za njegovo izboljšavo. V naslednjih podpoglavjih podajam po mojem mnenju pomembnejše funkcionalnosti, ki jih v obstoječi implementaciji pogrešam in bi bile zelo primerne za dograditev sistema SRCSI-CMS.

7.1.1 Administracija uporabnikov CMS znotraj sistema SRCSI-CMS

V SRCSI-CMS do sedaj ni bilo modula za administracijo uporabnikov. Razlog je specifična obravnava uporabnikov pri posamezni aplikaciji, ki vsaka zase definirajo različen način dodeljevanja pravic uporabnikom in njihovo administracijo znotraj drugih delov aplikacije ali pa celo s pomočjo drugih administracijskih aplikacij.

Na e-upravi administracija uporabnikov portala še ni bila narejena. Obstaja ločena aplikacija, ki je bila implementirana za pripravo in urejanje nekaterih dinamičnih delov portala eUprava, in vsebuje tudi interno administracijo uporabnikov za dodeljevanje pravic urednikovanja uporabnikom za posamezne dinamične vsebine portala.

Želeli bi si administracijo uporabnikov CMS, ki bi bila vgrajena v sam sistem CMS na portalu. Trenutno bo verjetno zaradi varnosti portala dodeljevanje pravic izvedeno neposredno v bazi.

7.1.2 Administracija anket portala

Trenutno je za dodajanje novih anket na portal eUprava potreben postopek, ki ga mora zaradi zahtevnosti izvesti vzdrževalec sistema. Razlog za to je v pomanjkljivi aplikaciji, ki ni dobro integrirana s samim portalnim delom, zato je potrebno ročno dodajanje spremenljive vsebine ankete, ki nato omogoča glasovanje uporabnikov portala. V sistemu SRCSI-CMS bi lahko implementirali komponento ankete, ki bi omogočala administracijo in prikazovanje anket na portalu. S takšno komponento bi naročnik lahko objavil ankete samostojno brez pomoči vzdrževalcev sistema.

7.1.3 Časovno odvisno objavljanje vsebin

Trenutno objavljanje vsebine v sistemu SRCSI-CMS se izvaja hipno na uporabnikov ukaz objave vsebine. Ker pa je dostikrat potrebno vsebino na portalu objaviti časovno odvisno, bi bil zelo uporaben sistem avtomatskega osveževanja prikaza posameznih strani fragmentov glede na časovni okvir. Objava vsebine CMS glede na čas bi tako dodala možnost vnaprejšnje priprave ali umika vsebine. To bi na primer prišlo zelo prav pri objavi obvestila, ki se tiče določene nadgradnje na sistemu, ki pa se bo izvajala samo v določenem časovnem obdobju, takoj po izvedbi pa bi bil smiseln umik obvestila.

7.1.4 Hranjenje elementov CMS v raznih podatkovnih shrambah

V diplomskem delu sem pri opisu sistema SRCSI-CMS kot podatkovno shrambo za elemente CMS vedno navedel podatkovno bazo. Tako se v trenutni implementaciji poleg metapodatkov v podatkovni bazi hranijo tudi fragmenti in pomnilniško zahtevnejši dokumenti CMS in slike CMS. Za večji nadzor nad pomnilnikom bi bila primerna in zelo uporabna razširitev sistema SRCSI-CMS z vmesnikom, ki bi omogočal izbiro podatkovne shrambe za posamezne elemente CMS. Tako bi namesto podatkovne baze lahko uporabili tudi na primer datotečni sistem ali druge namenske sisteme za hranjenje podatkov.

Dodatek A

Izvorna koda vtičnika srcmedia za administracijo videovsebin.

A.1 \srcmedia\editor_plugin.js

```
(function () {
  // Load plugin specific language pack
  tinymce.PluginManager.requireLangPack('srcmedia');

  tinymce.create('tinymce.plugins.SrcMediaPlugin',  {
    /**
     * Initializes the plugin, this will be executed after the plugin has
     * been created. This call is done before the editor instance has
     * finished it's initialization so use the onInit event
     * of the editor instance to intercept that event.
     *
     * @param {tinymce.Editor} ed Editor instance that the plugin is init. in.
     * @param {string} url Absolute URL to where the plugin is located.
     */
    init : function (ed, url) {
      // Register the command so that it can be invoked by using
      // tinyMCE.activeEditor.execCommand('mceSrcMedia');
      ed.addCommand('mceSrcMedia', function () {
        ed.windowManager.open( {
          file : url + '/srcmedia_dialog.htm',
          width : 350 + parseInt(ed.getLang('srcmedia.delta_width', 0)),
          height : 220 + parseInt(ed.getLang('srcmedia.delta_height', 0)),
          inline : 1
        },
        {
          plugin_url : url // Plugin absolute URL
        });
      });

      // Register button
      ed.addButton('srcmedia',  {
        title : 'srcmedia.desc',
        cmd : 'mceSrcMedia',
        image : url + '/img/srcmedia_button.gif'
      });



      ed.onPreInit.add(function() {
        ed.serializer.addValidChildRules('span[object|param|embed]');
      });

      // Add a node change handler, selects the button in the UI when a
      // specified element is selected
      ed.onNodeChange.add(function (ed, cm, n) {
        cm.setActive('srcmedia', (n.nodeName == 'SPAN') &&
            (ed.dom.getAttrib(n, 'class').indexOf('SrcMediaFile') != -1) );
      });
```

```
      // Load the specified CSS file into the document.
      ed.onInit.add(function() {
        if (ed.settings.content_css !== false)
          ed.dom.loadCSS(url + "/css/content.css");
      });
    },


    /**
     * Creates control instances based in the incomming name. This method
     * is normally not needed since the addButton method of the
     * tinymce.Editor class is a more easy way of adding buttons but
     * you sometimes need to create more complex controls like listboxes,
     * split buttons etc then this method can be used to create those.
     *
     * @param {String} n Name of the control to create.
     * @param {tinymce.ControlManager} cm Control manager to use inorder to
     *      create new control.
     * @return {tinymce.ui.Control} New control instance or null if no
     *      control was created.
     */
    createControl : function (n, cm) {
      return null;
    },


    /**
     * Returns information about the plugin as a name/value array. The
     * current keys are longname, author, authorurl, infourl and version.
     *
     * @return {Object} Name/value array cont. inform. about the plugin.
     */
    getInfo : function () {
      return  {
        longname : 'SrcMedia plugin',
        author : 'Jurij Jelenc',
        authorurl : 'http://www.src.si',
        infourl : 'mailto:jurij.jelenc@src.si',
        version : "1.0"
      };
    }
  });


  tinymce.PluginManager.add('srcmedia', tinymce.plugins.SrcMediaPlugin);
})();
```

A.2 \srcmedia\srcmedia_dialog.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <title>{#srcmedia_dlg.title}</title>
      <script type="text/javascript" src="../../tiny_mce_popup.js"></script>
      <script type="text/javascript" src="../../utils/mctabs.js"></script>
      <script type="text/javascript" src="js/srcmedia_dialog.js"></script>
  <link href="css/content.css" rel="stylesheet" type="text/css" />
</head>

<body>
<form onsubmit="SrcMediaDialog.insert();return false;" action="#">

  <div class="tabs">
    <ul>
      <li id="general_tab" class="current">
        <span>
          <a href="javascript:mcTabs.displayTab('general_tab',
            'general_panel');updateGeneral();" onmousedown="return false;">
            {#srcmedia_dlg.general}
          </a>
        </span>
      </li>
      <li id="code_tab">
        <span>
          <a href="javascript:mcTabs.displayTab('code_tab',
            'code_panel');updateCode();" onmousedown="return false;">
            {#srcmedia_dlg.code}
          </a>
        </span></li>
    </ul>
  </div>

  <div class="panel_wrapper">
    <div id="general_panel" class="panel current">
      <fieldset>
        <legend>{#srcmedia_dlg.general}</legend>
        <table border="0" cellpadding="4" cellspacing="0">
          <tr>
            <td><label for="src">{#srcmedia_dlg.file}</label></td>
            <td>
              <input type="text" id="src" name="src" value="" size="45%" />
            </td>
          </tr>
        </table>

        <legend>{#srcmedia_dlg.size}</legend>
        <table border="0" cellpadding="4" cellspacing="0">
          <tr>
            <td><label for="width">{#srcmedia_dlg.width}</label></td>
            <td><input type="text" id="width" name="width" value="" /></td>
          </tr>
          <tr>
            <td><label for="height">{#srcmedia_dlg.height}</label></td>
            <td>
```

```
            <input type="text" id="height" name="height" value="" />
          </td>
        </tr>
      </table>

    </fieldset>
  </div>

  <div id="code_panel" class="panel">
    <fieldset>
      <legend>{#srcmedia_dlg.code}</legend>
      <textarea rows="8" cols="50" id="code" name="code" ></textarea>
    </fieldset>
  </div>
</div>


<div class="mceActionPanel">
  <div style="float: left">
    <input type="button" id="insert" name="insert" value="{#insert}"
          onclick="SrcMediaDialog.insert();" />
  </div>

  <div style="float: right">
    <input type="button" id="cancel" name="cancel" value="{#cancel}"
          onclick="tinyMCEPopup.close();" />
  </div>
</div>
</form>

</body>
</html>
```

A.3 \srcmedia\css\content.css

```
span.SrcMediaFile {
  border:1px dotted #ff0000;
  background-position:center;
  background-color:#ffffcc;
  background-image:url(../img/srcmedia.gif);
  background-repeat:no-repeat;
  display: block;
  /*minimum definitions*/
  width: 10px;
  height: 10px;
}

span.SrcMediaFile * {
  display: none;
}

/* controlling of tabs settings */
.panel_wrapper div.current {
  height: 140px;
  overflow: auto
}
```

A.4 \srcmedia\js\srcmedia_dialog.js

```
tinyMCEPopup.requireLangPack();

var SrcMediaDialog = {
  CONST_SrcMediaFile : 'SrcMediaFile',
  CONST_SrcMediaCode : 'SrcMediaCode',
  src : '',
  width : '',
  height : '',
  code : '',
  SrcMediaCode : false,
  action_type : 'insert',

      init : function() {
    //Resizes the dialog to the inner size of the window.
    //This is needed since var. browsers have different border sizes on windows.
    tinyMCEPopup.resizeToInnerSize();

    var f = document.forms[0], ed = tinyMCEPopup.editor;

    var node = ed.selection.getNode();
    if (node) {

      var nodeClass = ed.dom.getAttrib(node, 'class');
      if(nodeClass&&nodeClass.indexOf(SrcMediaDialog.CONST_SrcMediaFile)!=-1) {
        SrcMediaDialog.action_type = 'update';

        if (nodeClass.indexOf(SrcMediaDialog.CONST_SrcMediaCode) != -1) {
          SrcMediaDialog.SrcMediaCode = true;

          //Switch tab to InsertEmbededCode tab
          mcTabs.displayTab('code_tab','code_panel');
        }

        //Setting value for title
        f.src.value = node.title;
        SrcMediaDialog.src = f.src.value;


        //Parsing the media dimensions
        var nodeStyle = ed.dom.getAttrib(node, 'style');
        if(nodeStyle) {

          //remove dimensions specifications for easier user display
          nodeStyle = nodeStyle.replace(/px/gi, '');
          nodeStyle = nodeStyle.replace(/%/gi, '');

          //Get data from style info
          var style = ed.dom.parseStyle(nodeStyle);

          if (style['width']) {
            f.width.value = style['width'] == '' ? 100 : style['width'];
            SrcMediaDialog.width = f.width.value;
          }
```

```
      if (style['height']) {
        f.height.value = style['height'] == '' ? 100 : style['height'];
        SrcMediaDialog.height = f.height.value;
      }
    }

    //Setting embed code
    var embcode = ed.selection.getContent({format : 'html'});
    if (embcode) {
      embcode = _fixCodeElements('object', embcode);

      var start = embcode.indexOf('<object');
      var end = embcode.indexOf('</object>');
      if(start != -1 && end != -1) {
        end = end + '</object>'.length;
        embcode = embcode.substring(start, end);
        f.code.value = embcode;
        SrcMediaDialog.code = f.code.value;
      }
    } else {
      //problem with loadig the content from selection (update type)
      // try generating code from general data
      updateCode();
      SrcMediaDialog.code = f.code.value;
    }

  } //if (nodeClass..)
} //if (node)

f.insert.value = ed.getLang( (SrcMediaDialog.action_type == 'update') ?
        'update' : 'insert');

  },

  insert : function() {
        // Insert the contents from the input into the document
var f = document.forms[0], ed = tinyMCEPopup.editor;

//The code generation and actualization
if (SrcMediaDialog.SrcMediaCode) {
  updateGeneral();
  updateCode();
} else {
  updateCode();
  updateGeneral();
}

//Check the dimensions if they are set (it must be after code gen.!)
if ( f.width.value == '' || isNaN(f.width.value) || f.height.value ==
        '' || isNaN(f.height.value) ) {
  //Force the default dimensions if they are not set
  if (f.width.value == '' || isNaN(f.width.value)) {
    f.width.value = 100;
  }
  if (f.height.value == '' || isNaN(f.height.value)) {
    f.height.value = 100;
  }

  //Update the code with new dimensions
  updateCode();
}
```

```
    var file = f.src.value;
    var fwidth = f.width.value;
    var fheight = f.height.value;

    var style = {};
    style['width'] = fwidth;
    style['height'] = fheight;
    //Serialize style info
    var nodeStyle = ed.dom.serializeStyle(style);

    var nodeClass = SrcMediaDialog.CONST_SrcMediaFile +
      (SrcMediaDialog.SrcMediaCode?' '+SrcMediaDialog.CONST_SrcMediaCode:'');

    //Restores any stored selection.
    //This can be useful since some browsers looses it's selection
    //if a control element is selected/focused inside the dialogs.
    tinyMCEPopup.restoreSelection();

    //We read the code from form, where it was possibly already generated
    var embededCode = f.code.value;
    if (embededCode) {

      // Insert the contents from the input into the document
      if (SrcMediaDialog.action_type == 'insert') {
        var insert_span;
        insert_span = ed.dom.createHTML(
          'span', {
            'class': nodeClass,
            title : file,
            'style': nodeStyle
          }, embededCode);
        ed.execCommand('mceInsertContent', false, insert_span);

      } else {
        //Updating code
        var node = ed.selection.getNode();
        if (node) {
          ed.dom.setAttrib(node, 'title', file);
          ed.dom.setAttrib(node, 'style', nodeStyle);
          ed.dom.setAttrib(node, 'class', nodeClass);
          ed.dom.setHTML(node, embededCode);

          //Repaints the ed. Sometimes the browser has graphic glitches.
          ed.execCommand('mceRepaint');
        }
      }

      tinyMCEPopup.close();

    } else {
      //Code not succesfully generated
      alert( ed.getLang('srcmedia_dlg.errorGenerateCode') );
    }

      }
};
```

```
// Updates general media data from inserted code
function updateGeneral() {
    var f = document.forms[0], ed = tinyMCEPopup.editor;

    var code = f.code.value;
    if (code && code != SrcMediaDialog.code) {

      var embedAttribs = _parseElementAttributes('embed', code);
      if(embedAttribs) {
        var file = embedAttribs['src'];
        var fwidth = embedAttribs['width'];
        var fheight = embedAttribs['height'];

        var objectAttribs = _parseElementAttributes('object', code);
        if (objectAttribs) {
          if (objectAttribs['width'] && (objectAttribs['width'] !=
                embedAttribs['width']) ) {
            if (SrcMediaDialog.width == embedAttribs['width']) {
              fwidth = objectAttribs['width'];
            }
            //if only one param. was changed, we have to update the other
            code = _updateEmbededCode(code, null, fwidth, null);
            f.code.value = code;
          }
          if (objectAttribs['height'] && (objectAttribs['height'] !=
                embedAttribs['height']) ) {
            if (SrcMediaDialog.height == embedAttribs['height']) {
              fheight = objectAttribs['height'];
            }
            //if only one param. was changed, we have to update the other
            code = _updateEmbededCode(code, null, null, fheight);
            f.code.value = code;
          }
        }

        //Setting values
        if (file) {
          f.src.value = file;
          SrcMediaDialog.src = f.src.value;
        }
        if (fwidth) {
          f.width.value = fwidth
          SrcMediaDialog.width = f.width.value;
        }
        if (fheight) {
          f.height.value = fheight;
          SrcMediaDialog.height = f.height.value;
        }
      }

      //Save code to local var
      SrcMediaDialog.code = code;

      //Save the code flag
      SrcMediaDialog.SrcMediaCode = true;
    }

}
```

```
// Updates code from inserted general media data
function updateCode() {
    var f = document.forms[0], ed = tinyMCEPopup.editor;
    var file = f.src.value;
    var fwidth = f.width.value;
    var fheight = f.height.value;

    //There have been changes made on parameters so we generate code
    if (file != SrcMediaDialog.src || fwidth != SrcMediaDialog.width ||
            fheight != SrcMediaDialog.height) {

      var embededCode = _generateEmbededCode(file, fwidth, fheight);
      if(embededCode) {
        f.code.value = embededCode;
        SrcMediaDialog.code = f.code.value;

        //Save the general flag
        SrcMediaDialog.SrcMediaCode = false;

      } else if (f.code.value) {
        //If the code was not generated yet, we try to update embeded code
       f.code.value = _updateEmbededCode(f.code.value, file, fwidth, fheight);
        SrcMediaDialog.code = f.code.value;

        //Save the code flag
        SrcMediaDialog.SrcMediaCode = true;

      }

      SrcMediaDialog.src = file;
      SrcMediaDialog.width = fwidth;
      SrcMediaDialog.height = fheight;

    } else if (!f.code.value) {
      //There was no changes, but the code was not generated yet!
      // We have to generate code now - we force the code generation

      var embededCode2 = _generateEmbededCode(file, fwidth, fheight);
      if(embededCode2) {
        f.code.value = embededCode2;
        SrcMediaDialog.code = f.code.value;
      }
    }
}

// Generates media object for insertion into page
function _generateEmbededCode (src, width, height) {
  if (!src && !width && !height)
    return null;

  var embededCode;
  //Searching for the optional media provider
  //YouTube
  if (src.indexOf('youtube') != -1 && (src.indexOf('watch?v=') != -1 ||
      src.indexOf('/v/') != -1) ) {
    if (src.indexOf('watch?v=') != -1) {
      src = 'http://www.youtube.com/v/' +
            src.substring(src.indexOf('watch?v=')+'watch?v='.length);
    } else {
      //the src parameter with /v/ in link is already OK!
    }
```

```
    embededCode = '<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-
444553540000" ' +
      'width="'+width+'" height="'+height+'"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.c
ab#version=6,0,40,0">' +
      '<param name="src" value="'+src+'" />' +
      '<embed type="application/x-shockwave-flash" ' +
      'width="'+width+'" height="'+height+'" src="'+src+'">' +
      '</embed>' +
      '</object>';
  }

  //Google video
  if (src.indexOf('video.google.') != -1 && src.indexOf('videoplay?docid=')
      != -1) {
    //Google url-s have '#' at the end
    if(src.charAt(src.length-1) == '#') {
      src = src.substring(0, src.length-1);
    }
    src = 'http://video.google.com/googleplayer.swf?docId=' +
src.substring(src.indexOf('videoplay?docid=')+'videoplay?docid='.length);

    embededCode = '<object
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.c
ab#version=6,0,40,0" ' +
      'width="'+width+'" height="'+height+'" >\n' +
      '<param name="src" value="'+src+'" />' +
      '<embed type="application/x-shockwave-flash" ' +
      'width="'+width+'" height="'+height+'" src="'+src+'">' +
      '</embed>\n' +
      '</object>';
  }

  //Moj video
  if (src.indexOf('www.mojvideo.com') != -1) {
    src = 'http://www.mojvideo.com/v/'+src.substring(src.lastIndexOf('/')+1);

    embededCode = '<object width="'+width+'" height="'+height+'" >\n' +
      '<param name="movie" value="'+src+'" />\n' +
      '<param name="wmode" value="transparent" />\n' +
      '<embed type="application/x-shockwave-flash" wmode="transparent" ' +
      'width="'+width+'" height="'+height+'" src="'+src+'">' +
      '</embed>\n' +
      '</object>';
  }
  return embededCode;
}

// Update the existing code with new parameters
function _updateEmbededCode (embededCode, src, width, height) {
  if (!embededCode || embededCode.length < 2)
    return null;

  if (src) {
    embededCode = _replaceAttributeValues('src', SrcMediaDialog.src, src,
            embededCode);
    //src property is present in other places, so we do also normal replace:
    embededCode = embededCode.replace(new
            RegExp('"'+SrcMediaDialog.src+'"','gi'),'"'+src+'"');
  }
```

```
  if (width) {
    embededCode = _replaceAttributeValues('width', SrcMediaDialog.width,
            width, embededCode);
  }
  if (height) {
    embededCode = _replaceAttributeValues('height', SrcMediaDialog.height,
            height, embededCode);
  }

  return embededCode;
}


function _parseElementAttributes (element, code) {
  if (!code || code.length < 2)
    return null;

  var eltStart = '<'+element;
  //Clean possible white space elements
  code = _fixCodeElements(element, code);

  var attribs;
  var startPos = code.indexOf(eltStart);
  if (startPos != -1) {
    var endPos = code.indexOf('>', startPos);
    attribs = _parseAttributes(code.substring(startPos + eltStart.length,
            endPos));
  }
  return attribs;
}


function _parseAttributes (attribute_string) {
  if (!attribute_string || attribute_string.length < 2)
    return null;

  //first, we fix all the quotes and whitespaces
  attribute_string = _fixAttributeElements(attribute_string);
  var attributes = new Array();
  var attributesSplit = attribute_string.split(" ");
  for (var i=0; i < attributesSplit.length; i++) {
    var attribute = attributesSplit[i];
    if(attribute && tinymce.trim(attribute) != '') {
      var pos = attribute.indexOf("=");
      var name = attribute.substring(0, pos).toLowerCase();
      //the value of attribute lies between '="' and last '"'
      attributes[name] = attribute.substring(pos+2,
            attribute.lastIndexOf('"'));
    }
  }
  return attributes;
}


function _fixCodeElements (element, code) {
  if (!code || code.length < 2)
    return null;

  // Fix the code elements of specified element
  code = code.replace(new RegExp('<[ ]*'+element,'gi'),'<'+element);
```

```
  code = code.replace(new RegExp('<[ ]*/'+element+'[
]*>','gi'),'</'+element+'>');
  return code;
}


function _replaceAttributeValues (attribute, oldvalue, newValue, code) {
  if (!code || code.length < 2)
    return null;

  // Repalce the attribute values of specified attribute
  code = code.replace(new RegExp(attribute+'[ ]*=[
]*"'+oldvalue+'"','gi'),attribute+'="'+newValue+'"');
  return code;
}

function _fixAttributeElements (code) {
  if (!code || code.length < 2)
    return null;

  // Fix the attribute elements
  code = code.replace(new RegExp("'",'gi'),'"');
  code = code.replace(new RegExp('[ ]*=[ ]*"','gi'),'="');
  return code;
}

tinyMCEPopup.onInit.add(SrcMediaDialog.init, SrcMediaDialog);
```

A.5 \srcmedia\langs\sl.js

```
tinyMCE.addI18n('sl.srcmedia',{
  desc : 'Vstavi multimedijsko vsebino SRC'
});
```

A.6 \srcmedia\langs\sl_dlg.js

```
tinyMCE.addI18n('sl.srcmedia_dlg',{
  title : 'Multimedijska vsebina - SRC',
  file:"URL",
  size:"Dimenzije",
  width:"\u0160irina",
  height:"Vi\u0161ina",
  general:'Vstavi URL',
  code:'Vstavi multimedijsko kodo',
  errorGenerateCode: 'Napaka pri generiranju multimedijske kode! Prosimo
vstavite multimedijsko kodo.'
});
```

Viri

[1] Web content management. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://en.wikipedia.org/wiki/Web_content_management_system.

[2] Portal e-Uprava. [online]. [uporabljeno 2010-05-06]. Dostopno na: http://www.src.si/resitve/javnauprava/drzavljani1.asp.

[3] E-storitve – elektronske storitve javne uprave. [online]. [uporabljeno 2010-05-06]. Dostopno na: http://www.src.si/resitve/javnauprava/drzavljani4.asp.

[4] Java Platform, Enterprise Edition. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition.

[5] Apache Struts. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://struts.apache.org.

[6] Apache Tiles. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://tiles.apache.org.

[7] JavaScript. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://en.wikipedia.org/wiki/JavaScript.

[8] Ajax programming. [online]. [uporabljeno 2010-05-11]. Dostopno na: http://en.wikipedia.org/wiki/Ajax_(programming).

[9] The Java Database Connectivity (JDBC). [online]. [uporabljeno 2010-05-04]. Dostopno na: http://java.sun.com/javase/technologies/database/index.jsp.

[10] JavaServer Pages (JSP). [online]. [uporabljeno 2010-05-04]. Dostopno na: http://java.sun.com/products/jsp.

[11] JavaServer Pages Standard Tag Library (JSTL). [online]. [uporabljeno 2010-05-04]. Dostopno na: http://java.sun.com/products/jsp/jstl.

[12] Oracle Internet Application Server. [online]. [uporabljeno 2010-06-05]. Dostopno na: http://www.oracle.com/technology/products/ias/index.html.

[13] Oracle Database. [online]. [uporabljeno 2010-06-05]. Dostopno na: http://en.wikipedia.org/wiki/Oracle_Database.

[14] Spring Framework. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://en.wikipedia.org/wiki/Spring_Framework.

[15] EHcache. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://en.wikipedia.org/wiki/EHcache.

[16] Ehcache Distributed Cache. [online]. [uporabljeno 2010-05-04]. Dostopno na: http://ehcache.org.

[17] What is Quartz. [online]. [uporabljeno 2010-05-06]. Dostopno na: http://www.quartz-scheduler.org.

[18] TinyMCE Home. [online]. [uporabljeno 2010-05-06]. Dostopno na: http://tinymce.moxiecode.com.

[19] TinyMCE. [online]. [uporabljeno 2010-05-06]. Dostopno na: http://en.wikipedia.org/wiki/TinyMCE.

[20] JDeveloper. [online]. [uporabljeno 2010-06-05]. Dostopno na: http://en.wikipedia.org/wiki/JDeveloper.

[21] Oracle SQL Developer. [online]. [uporabljeno 2010-06-05]. Dostopno na: http://www.oracle.com/technology/products/database/sql_developer/index.html.

[22] Oracle SQL Developer. [online]. [uporabljeno 2010-06-05]. Dostopno na: http://en.wikipedia.org/wiki/Oracle_SQL_Developer.