UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Uroš Fikon

Spletna podpora trženju bančnih storitev s kombinacijo odprtokodnih tehnologij

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: dr. Igor Rožanc

Ljubljana, 2011

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: 00090/2011
Datum: 01.04.2011

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

Kandidat: **UROŠ FIKON**

Naslov: **SPLETNA PODPORA TRŽENJU BANČNIH STORITEV S KOMBINACIJO ODPRTOKODNIH TEHNOLOGIJ**

A WEB BANKING SOLUTION USING A COMBINATION OF OPEN SOURCE TECHNOLOGIES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj spletne rešitve za podporo trženju bančnih storitev v večji slovenski banki. Poudarek naj bo na uporabi kombinacije učinkovitih odprtokodnih rešitev, ki združujejo različne javanske tehnologije z dokumentnim sistemom. Jedro rešitve naj bo v učinkovitem upravljaju predlog, ki omogočajo avtomatsko generiranje enotnih dokumentov. Pri prikazu predstavite predvsem tehnično plat kombiniranja različnih tehnologij.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

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: 00090/2011
Datum: 01.04.2011

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

Kandidat: **UROŠ FIKON**

Naslov: **SPLETNA PODPORA TRŽENJU BANČNIH STORITEV S KOMBINACIJO ODPRTOKODNIH TEHNOLOGIJ**

A WEB BANKING SOLUTION USING A COMBINATION OF OPEN SOURCE TECHNOLOGIES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi predstavite razvoj spletne rešitve za podporo trženju bančnih storitev v večji slovenski banki. Poudarek naj bo na uporabi kombinacije učinkovitih odprtokodnih rešitev, ki združujejo različne javanske tehnologije z dokumentnim sistemom. Jedro rešitve naj bo v učinkovitem upravljaju predlog, ki omogočajo avtomatsko generiranje enotnih dokumentov. Pri prikazu predstavite predvsem tehnično plat kombiniranja različnih tehnologij.

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

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

diplomskega dela

Spodaj podpisani/-a **Uroš Fikon**,

z vpisno številko **63010199**,

sem avtor diplomskega dela z naslovom:

Spletna podpora trženju bančnih storitev s kombinacijo odprtokodnih tehnologij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom (naziv, ime in priimek)

  viš. pred. dr. Igor Rožanc

- 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 _____ Podpis avtorja: _____

Zahvala

Zahvaljujem se staršem, ki sta mi študij omogočila in bila pri tem nadvse potrpežljiva.

Zahvaljujem se mentorju, dr. Igorju Rožancu, za nasvete in pomoč pri izdelavi diplomske naloge.

Posebna zahvala gre tudi vsem sodelavcem in prijateljem, še posebej Loreni, za podporo in spodbudo pri zaključevanju študija.

Kjer je volja je tudi moč.

Valterju v spomin.

Kazalo

Kazalo slik

Seznam kratic in simbolov

AJAX	angl. Asynchronous Javascript And Xml – asinhroni JavaScript in XML
API	angl. Application Programming Interface – aplikacijski programski vmesnik
CIFS	angl. Common Internet File System – protokol za dostop do datotek in storitev na oddaljenih računalnikih
CML	angl. Content Manipulation Language – psevdo-jezik za delo z Alfresco repozitorijem
COM	angl. Component Object Model – komponentni objektni model
DMS	angl. Document Management System – sistem za upravljanje z dokumenti
EAR	angl. Enterprise ARchive – distribucija aplikacije v Java EE okolju
(E)CMS	angl. (Enterprise) Content Management System – sistem za upravljanje z vsebinami
EJB	angl. Enterprise Java Bean – strežniška javanska komponenta
HTTP	angl. HyperText Transfer Protocol – protokol za izmenjavo večpredstavnostnih vsebin
JAR	angl. Java ARchive – javanska knjižnica, ki združuje vsebinsko povezane razrede
Java EE	angl. Java Enterprise Edition – javanska platforma namenjena delovanju v zahtevnejših informacijskih sistemih
(J)BPM	angl. (Java) Business Process Management – sistem za upravljanje s poslovnimi procesi

JMS	angl. Java Message Service – del Java EE arhitekture, ki skrbi za (asinhrono) komunikacijo med komponentami
JNI	angl. Java Native Interface – vmesnik za integracijo javanskih aplikacij z aplikacijami drugih platform (npr. C/C++)
JPA	angl. Java Persistence API – javanski aplikacijski programski vmesnik za delo z relacijskimi podatki
JSF	angl. JavaServer Faces – tehnologija za izdelavo spletnih uporabniških vmesnikov
MIME	angl. Multipurpose Internet Mail Extension – standard za označevanje formata datotek
NTLM	angl. Windows NT Lan Manager – protokol za omrežno avtentikacijo v Windows NT okolju
ORM	angl. Object Relational Mapping – tehnika za preslikavo podatkov med relacijskim in objektnim modelom
POST	Eden od načinov posredovanja HTTP zahteve
PDF/A	angl. Portable Document Format for Archive – PDF datoteka primerna za dolgoročno hrambo
SFSB	angl. StateFul Session Bean – EJB komponenta vezana na točno določenega odjemalca, ki v svojem življenjskem ciklu ohranja stanje
SLSB	angl. StateLess Session Bean – EJB komponenta ki ni vezana na točno določenega odjemalca
SMB	angl. Server Message Block – Microsoftov protokol za dostop do datotek na oddaljenih računalnikih
SMTP	angl. Simple Mail Transfer Protocol – protokol za izmenjavo elektronske pošte

SOAP	angl. Simple Object Access Protocol – protokol na katerem temelji tehnologija spletnih storitev
UDDI	angl. Universal Description, Discovery and Integration – register v katerem so shranjene informacije o spletnih storitvah
WS	angl. Web Service – spletna storitev
WSDL	angl. Web Service Definition Language – jezik za opisovanje spletnih storitev
(X)HTML	angl. (eXtended) HyperText Markup Language – razširjeni označevalni jezik
XML	angl. eXtensible Markup Language – razširljivi označevalni jezik
XSL(T)	angl. eXtensible Stylesheet Language (Transformations) – jezik s katerim določimo obliko prikaza podatkov v XML datotekah

Povzetek

Diplomska naloga opisuje izvedbo procesa priprave ponudb v eni od slovenskih bank. Namen dela je prikazati različna odprtokodna orodja, ki smo jih pri razvoju uporabili. Poudarek je predvsem na enostavni integraciji samostojnih modulov preko spletnih storitev oziroma s pomočjo odprtokodnih knjižnic.

Najprej bomo spoznali pomembnejša orodja in tehnologije, ki smo jih uporabili pri izdelavi rešitve. Pri tem se bomo osredotočili na tista orodja in komponente, ki slonijo na programskem jeziku Java: aplikacijski strežnik JBoss, ogrodje Seam, knjižnica za izdelavo spletnih vmesnikov Richfaces ter sistem za upravljanje dokumentov Alfresco. Poleg naštetih orodij sta v tem delu naloge opisani tudi danes zelo popularni tehnologiji pri razvoju spletnih aplikacij: AJAX in spletne storitve.

Sledi prikaz procesa priprave ponudbe z vidika končnega uporabnika, kjer so opisane ključne točke, ki smo jih razvili. Izpostavljeni so tudi elementi, ki so pomembni za doseganje boljše uporabniške izkušnje.

Osrednji del diplomske naloge zajema podrobnejši opis razvitih komponent. Pri tem so izpostavljena zanimivejša področja, ki so hkrati predstavljala največje izzive pri razvoju, recimo integracija aplikacije s sistemom za upravljanje dokumentov Alfresco ter kreiranje PDF datoteke s pomočjo aplikacije Microsoft Word na oddaljenem računalniku.

Na koncu so podane ugotovitve do katerih smo prišli med razvojem, ter smernice, ki bi jih upoštevali pri morebitni nadgradnji aplikacije.

Ključne besede: odprta koda, spletne storitve, integracija, Java EE, JBoss, Seam, RichFaces, Alfresco

Abstract

The thesis describes the implementation of the process for the preparation of electronic documents in a Slovenian bank. The main goal is to describe different possibilities of integration between independent modules by using open source libraries or web services.

First, we introduce the most important tools and technologies used in our project. We focus on java based tools such as the JBoss application server, the Seam framework, the RichFaces tag library and the open source document management system Alfresco. Additionally, this section describes two popular technologies used in web application development nowadays: AJAX and web services.

Next, we describe the development process from the end-user's point of view exposing the main parts of the application we developed. We present some best practices we adopted to achieve a better user experience.

The main part of the thesis consists in the detailed explanation of the most interesting components developed to support the business process. Examples of those are the integration with the Alfresco document management system and the generation of PDF files using Microsoft Word on a remote machine.

In conclusion we provide important findings about the solution we implemented and guidelines we would eventually take in consideration when redesigning the application.

Key words: open source, web services, integration, Java EE, JBoss, Seam, RichFaces, Alfresco

1 Uvod

1.1 Problem in izhodišča

V obstoječi aplikaciji za podporo prodaji v eni od slovenskih bank smo želeli implementirati rešitev, ki bi referentom olajšala proces priprave ponudb pri trženju bančnih storitev. Ugotovljeno je bilo namreč, da uporabniki s tem porabijo preveč časa.

Poleg časovnega vidika, smo velik pomen namenili tudi uporabniški izkušnji. Ciljna skupina je namreč zajemala širok spekter uporabnikov ki so se razlikovali tako po letih, kot po računalniški izobrazbi.

Istočasno smo iskali tudi enostavno rešitev, ki bi omogočala tako dolgoročno hranjenje izdelanih elektronskih dokumentov kot dostop do teh iz ostalih informacijskih sistemov v banki, npr. spletne banke.

1.2 Pričakovani cilji

Cilji, ki smo jih pri vzpostavitvi rešitve želeli doseči, so zajemali tri vidike:

1. Uporabniški:
    - enostaven, odziven in uporabniško prijazen vmesnik,
    - uporaba znanih tehnologij in orodij za čim hitrejše uvajanje zaposlenih,
    - možnost ročnega dopolnjevanja izdelanih dokumentov in
    - možnost iskanja po poslanih dokumentih.
2. Organizacijski:
    - poenoten proces ne glede na izhodni kanal (elektronska pošta, spletna banka, poštna pošiljka,...),
    - konsolidacija izgleda predlog ob prenovi celostne grafične podobe in
    - lažje vzdrževanje in upravljanje s predlogami.
3. Sistemsko/razvojni:
    - centralno vodenje predlog (nahajajo se na enotnem viru, tj. v mapi v skupni rabi),
    - centralno upravljanje s pravicami dostopa do predlog (branje/urejanje),
    - uporaba že razvitih funkcionalnosti iz drugih sistemov ter

- uporaba standardnih (če je le mogoče odprtokodnih) orodij pri novem razvoju.

V diplomski nalogi se bomo osredotočili le na proces priprave ponudb ter na aktivnosti, ki so z njim povezane.

2 Uporabljena orodja in tehnologije

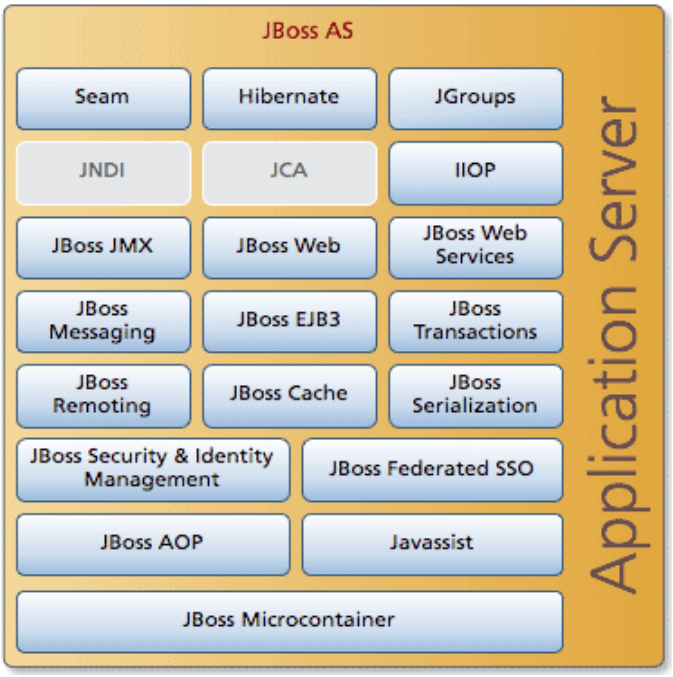
V tem poglavju so predstavljena orodja in tehnologije, ki smo jih uporabili pri izdelavi posameznih delov aplikacije. Pri razvoju zahtevnejših delov kode smo poskušali najti morebitne uveljavljene odprtokodne rešitve, ki bi nam delo olajšale.

Izpostavili bomo predvsem tista orodja, ki so širšemu občinstvu manj znana, a hkrati zelo zmogljiva.

2.1 Aplikacijski strežnik JBoss

Aplikacijski strežnik JBoss [2] je zmogljiv odprtokodni aplikacijski strežnik, ki implementira platformo Java EE (Enterprise Edition). Nudi podporo različnim tehnologijam, kot so:

- EJB 3.0
- JMS
- Quartz [22]
- spletne storitve
- ...



Slika 1. Prikaz tehnologij, ki jih podpira aplikacijski strežnik JBoss

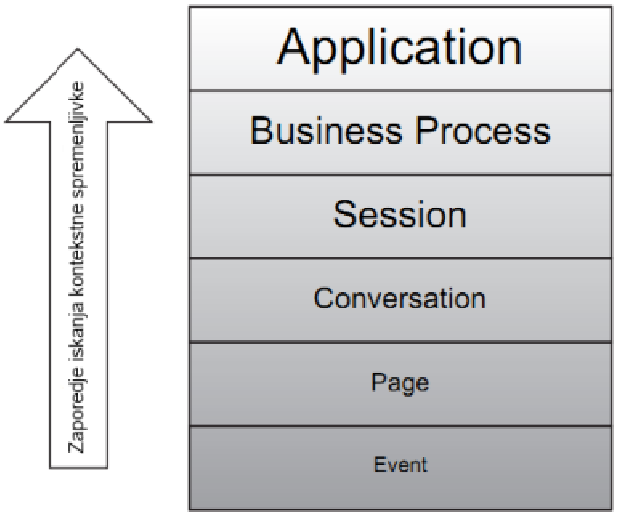
Poleg tega podpira tudi delovanje v gruči in s tem pripomore k visoki dosegljivosti aplikacij, kar je danes še posebej pomembno. Je tudi med vodilnimi oz. najbolj uporabljenimi aplikacijskimi strežniki v Java EE okolju. JBoss je na voljo v dveh paketih: odprtokodnem (Community Edition) in plačljivem (Enterprise Application Platform), katerega izdaja in zanj skrbi družba Red Hat.

2.2 Ogrodje JBoss Seam

Seam [1, 21] je odprtokodno spletno ogrodje (angl. web framework), ki povezuje standardne tehnologije v okviru platforme Java EE, kot so EJB 3.0, JPA, BPM s široko paleto nestandardnih komponent, ter jih tako združuje v poenoten razvojni model.

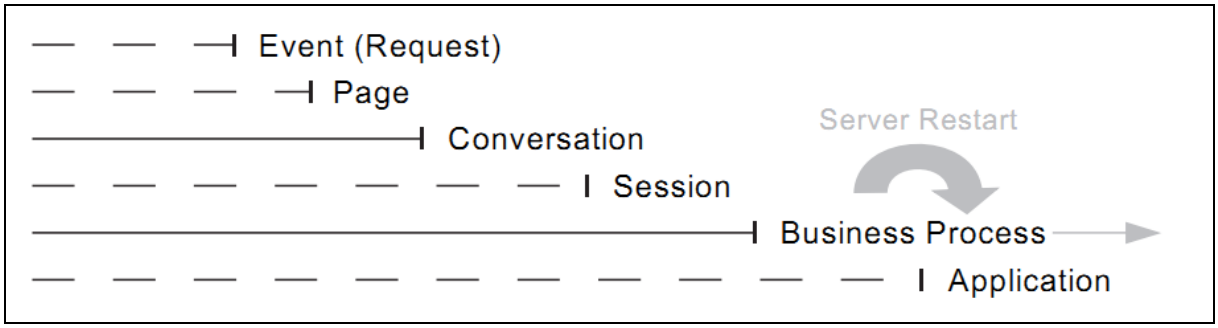
Razvijalcem omogoča hitrejši razvoj kompleksnih spletnih aplikacij z uporabo različnih funkcionalnosti, ter bogatim naborom komponent za uporabniški vmesnik. Z uporabo Seam-ovega izraznega jezika (angl. expression language) lahko s temi komponentami upravljamo neposredno v predstavitvenem sloju.

Pomembnejši del ogrodja je tudi tako imenovani *bijection* [1], ki pod enim imenom združuje dve funkcionalnosti: *injection* in *outjection*. S prvo lahko v Seam komponenti na enostaven način (z anotacijo `@In`) pridobimo vrednost kontekstne spremenljivke tj. lastnosti druge Seam komponente. Iskanje poteka po vseh kontekstih (slika 2) oz. se zaključi takoj, ko Seam najde iskano spremenljivko – če iskane spremenljivke ne najde, vrne vrednost `null`. Na podoben način lahko z anotacijo `@Out` poljubno lastnost Seam komponente pretvorimo v kontekstno spremenljivko, in jo tako damo na voljo ostalim komponentam.



Slika 2. Zaporedje iskanja spremenljivke v Seam kontekstih

Enostavnost uporabe so pri Seam-u dosegli z zmanjšanjem števila nastavitvenih XML datotek ter z njihovo zamenjavo z Java anotacijami. Poleg tega se Seam ponaša z razširjenim Servlet modelom. Glavna ideja je razširitev standardnih kontekstov v okviru spletne aplikacije (zahteva, seja, aplikacija) z vpeljavo novih, ki predstavljajo realnejše kontekste, ki se pojavijo v življenjskem ciklu aplikacije (npr. stran, pogovor) [1]. Slika 3 prikazuje podprte kontekste in njihov življenjski cikel.



Slika 3. Konteksti spletne aplikacije podprti v ogrodju Seam

2.3 Ogrodje JavaServer Faces

JSF [18] je ogrodje za razvoj spletnih uporabniških vmesnikov v programskem jeziku Java. Bistvena prednost ogrodja je v tem, da lahko vrednosti gradnikov uporabniškega vmesnika povežemo neposredno z zaledno komponento. S tem si prihranimo veliko časa, ki bi ga sicer porabili z obdelovanjem podatkov posredovanih ob POST zahtevi.

2.3.1 Knjižnica RichFaces

RichFaces [20] je knjižnica, ki nadgrajuje osnovni JSF model z obsežnim naborom zahtevnejših komponent in jim nudi podporo za AJAX funkcionalnosti (tj. proženje delnih predložitev form). Z njeno pomočjo je razvoj bogatih spletnih aplikacij (angl. rich web applications) enostavnejši, saj so komponente izdelane tako, da jih je zelo enostavno prilagajati. Večino nastavitev (npr. jezik prikaza ali sam izgled) lahko namreč nadzorujemo s parametri ob inicializaciji.

Istočasno pa knjižnica RichFaces vsebuje še vrsto zahtevnejših elementov, katerih izdelava bi razvijalcu vzela kar precej časa.

Med pomembnejšimi komponentami lahko izpostavimo:

- `dataTable` (slika 4): komponenta za tabelarično prikazovanje podatkov z možnostjo sortiranja, filtriranja, paginacije...
- `calendar`: standardni gradnik za izbiro datuma, z obsežnim naborom nastavitev
- `tree`: gradnik za prikazovanje drevesnih struktur

key	summary	assignee	fixVersion	reporter	priority	status	resolution	created	updated
RF-6338	PanelMenu does not work in 3.3.1.SNAPSHOT	Alex Kolonitsky	03.03.2001	Alexander Dubovsky	Critical	Open	UNRESOLVED	23/Feb/09 06:17 AM	24/Feb/09 11:59 AM
RF-5960	FileUpload sends id parameter to server on each upload.	Andrei Markavtsov	03.03.2001	Andrei Markavtsov	Major	Open	UNRESOLVED	29/Jan/09 05:57 AM	11/Feb/09 10:58 AM
RF-6266	FileUpload: AJAX polling problems	Andrei Markavtsov	03.03.2001	Nick Belaevski	Major	Open	UNRESOLVED	19/Feb/09 12:16 PM	24/Feb/09 11:56 AM
RF-6143	inplaceSelect: is not expanded in FF	Anton Belevich	03.03.2001	Tsikhon Kuprevich	Critical	Open	UNRESOLVED	11/Feb/09 09:29 AM	12/Feb/09 06:58 AM

« « 1 2 3 » »

Slika 4. Gradnik `dataTable` omogoča napreden prikaz podatkov v tabelarični obliki

2.3.2 Tehnologija AJAX

S kratico AJAX [1, 17] poimenujemo skupek tehnologij, ki jih uporabljamo za izdelavo interaktivnih spletnih aplikacij.

Glavna prednost uporabe AJAX-a je v tem, da lahko posodobimo vsebino trenutno prikazane spletne strani (oz. del nje) asinhrono. S tem omogočimo hitrejše delovanje aplikacij, saj se

med odjemalcem in strežnikom pretaka manj podatkov. Istočasno pa uporabnikom omogočimo, da neprekinjeno interagirajo z aplikacijo, saj ni potrebno čakati da se po izvedeni akciji stran ponovno v celoti naloži.

AJAX tehnologija se v spletnih aplikacij uporablja pri:

- preverjanju veljavnosti vnesenih podatkov na formah (v realnem času),
- ponujanju možnih vrednosti na podlagi delnega vnosa (autocompletion; npr. pri vnosu besedila v iskalnik),
- periodičnem osveževanju strani (npr. borzni indeksi) ter
- pridobivanju podatkov na zahtevo, kjer ni potrebno osveževanje celotne strani.

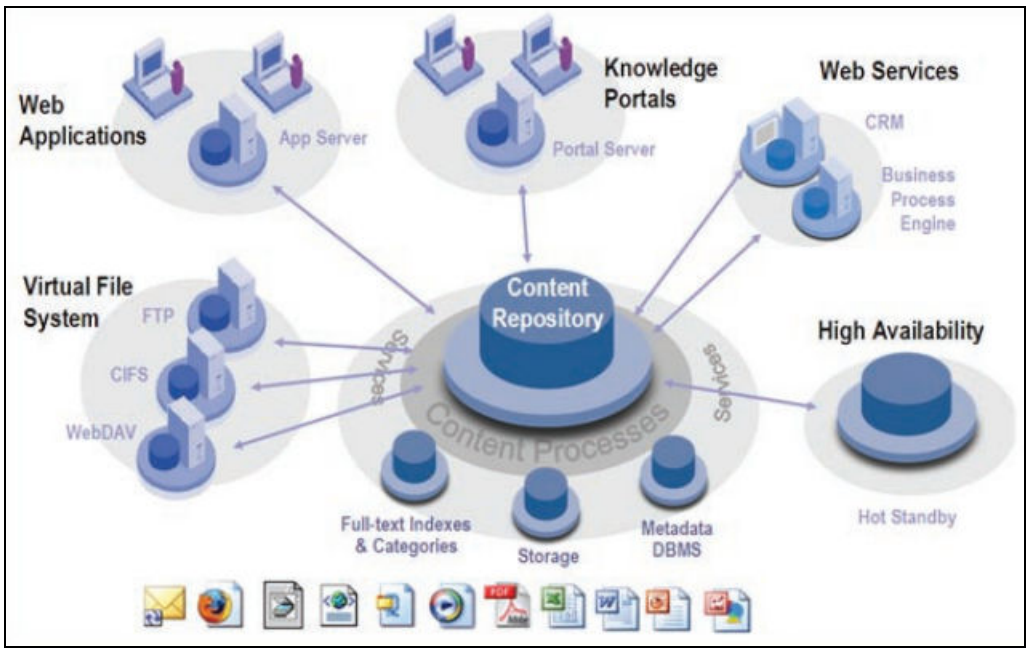
2.4 Sistem Alfresco ECMS

Alfresco ECMS [6] je vodilni odprtokodni sistem za upravljanje z vsebinami oz. dokumenti. Eden od ustanoviteljev, John Newton, je bil soustanovitelj komercialne rešitve Documentum [9], ki je še danes med vodilnimi na tem področju, razvojno ekipo pa sestavljajo še ugledni strokovnjaki, ki so med drugim sodelovali pri razvoju sodobnih sistemov za upravljanje z vsebinami: FileNet [10], OpenText [16] in Interwoven [8].

Alfresco temelji na programskem jeziku Java ter vgrajuje številne odprtokodne rešitve in arhitekturo, ki temelji na globalno sprejetih standardih. Na ta način uporabnikom zagotavlja stabilnost ter široko podporo tudi s strani spletnih skupnosti, ki niso neposredno sodelovale pri razvoju sistema [4].

Glavne lastnosti so:

- številne možnosti integracije s sistemom: spletne storitve, mape v skupni rabi, vtičniki za Microsoft Office... (slika 5),
- povezava z imenikom Active Directory oz. podpora za LDAP protokol,
- enkratni vpis (angl. single sign-on) – uporabniku se ni potrebno eksplicitno prijavljati v sistem, ampak za to skrbi domenski krmilnik (angl. domain controller),
- avtomatično indeksiranje in iskanje vsebin z uporabo knjižnice Apache Lucene [7],
- možnost pretvorbe med različnimi datotečnimi formati in
- neodvisnost od operacijskega sistema (Java) in podatkovne baze (Hibernate).



Slika 5. Dostop do vsebin v sistemu Alfresco je možen preko različnih protokolov

Bistveno prednost pred ostalimi sistemi predstavlja tudi način licenciranja. Plačljiva je namreč samo podpora, sam produkt pa je brezplačen. To se nam je zdelo še posebej zanimivo, saj smo želeli produkt preizkusiti skozi daljše obdobje.

2.5 Spletne storitve

Spletne storitve (angl. web services) [3, 25] predstavljajo sodoben način integracije različnih sistemov. Aplikacije so lahko razvite v različnih programskih jezikih. Sodobne aplikacije, še posebej v objektno usmerjenem programiranju, težijo k modularnosti. Določene funkcije so namreč uporabne na različnih mestih, zato jih je smotrno ponuditi tudi ostalim sistemom. Funkcionalnost, ki je napisana v programskem jeziku Java, želimo na primer uporabiti še v aplikaciji, ki temelji na .NET tehnologiji. Ponovni razvoj poslovne logike bi bil neracionalen, saj bi si na ta način otežili vzdrževanje – vsakič, ko bi se poslovna logika spremenila, bi bilo potrebno posodobiti obe veji razvoja. Razvoj spletnih storitev izhaja ravno iz želje po konsolidaciji poslovne logike in njeni ponovni uporabnosti (angl. reusability).

Spletna storitev je način komunikacije med dvema različnima aplikacijama preko omrežja. Pri tem se preko HTTP protokola izmenjujejo XML sporočila.

Platformo spletnih storitev lahko razdelimo na tri večja področja, in sicer:

- SOAP: protokol za prenos podatkov,
- WSDL: jezik, ki opisuje argumente metod in strukture parametrov spletnih storitev in
- UDDI: služi za izpis razpoložljivih spletnih storitev.

Spletne storitve izpostavljajo samo podpise metod (angl. method signatures), ki so na voljo. O sami implementaciji pa odjemalec ne izve ničesar. Včasih niti tega ne, v katerem programskem jeziku je zaledna metoda napisana.

2.6 SMB/CIFS protokol

CIFS protokol omogoča izmenjavo datotek preko računalniških omrežij. Preko njega odjemalci zahtevajo dostop do datotek na strežnikih. Temelji na protokolu SMB, ki ga uporabljajo osebni računalniki oz. delovne postaje ne glede na operacijski sistem.

2.6.1 Knjižnica jCIFS

JCIFS [23] je odprtokodna knjižnica, ki implementira omrežni protokol CIFS/SMB v programskem jeziku Java. Med drugim nam omogoča, da iz javanske aplikacije dostopamo do map v skupni rabi (angl. shared folders) na oddaljenem strežniku. Poleg tega nudi tudi podporo za NTLM avtentikacijo, ki je potrebna za dostop do takšne mape.

2.7 Sistem COM in komponente ActiveX

COM [15] je platformno neodvisen, porazdeljen, objektno usmerjen sistem za izdelavo aplikacijskih komponent na Microsoft operacijskih sistemih. Take komponente lahko med seboj komunicirajo in interagirajo. COM objekte se lahko razvije v različnih programskih jezikih, najenostavneje pa z uporabo objektno usmerjenih programskih jezikov kot sta C++ ali C#. V COM družino štejemo tudi COM+, DCOM (Distributed COM) in ActiveX komponente.

COM se na primer uporablja tudi v paketu Microsoft Office za interakcijo med različnimi aplikacijami (npr. za uvoz tabelaričnih podatkov iz aplikacije Excel v Word).

ActiveX kontrole [14] so komponente (ali objekti), ki jih lahko uporabimo za dostop do določene funkcionalnosti obsežnejše aplikacije. ActiveX kontrole se lahko izdelajo v

programskem jeziku, ki podpira razvoj COM komponent, npr. C++, Borland Delphi, Visual Basic, .NET programski jeziki (C#/VB.NET).

V Java okolju lahko ActiveX kontrole primerjamo z applet-i.

2.7.1 Knjižnica JACOB

Dostop do COM objektov iz programskega jezika Java je možen z uporabo različnih knjižnic. Odprtokodna knjižnica JACOB (JAva-COm Bridge) [11, 24] je primer take knjižnice, ki je na voljo preko licence LPGL.

Knjižnica JACOB nam omogoča klic COM objektov iz Jave. Za izvedbo klicev (angl. native calls) v COM in Win32 knjižnice uporablja ogrodje JNI.

Knjižnico smo uporabili pri izdelavi komponente za kreiranje PDF datotek.

3 Primer uporabe aplikacije za trženje bančnih storitev

Za lažje razumevanje problema bomo v tem poglavju prikazali praktični primer uporabe aplikacije za trženje bančnih storitev pri procesu priprave ponudbe.

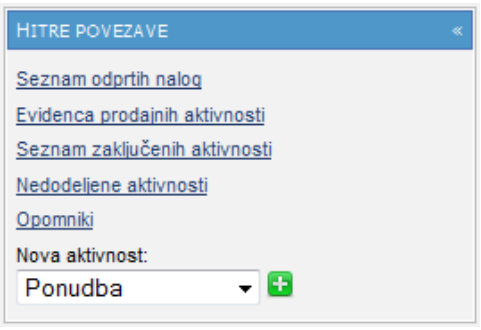
3.1 Vnos podatkov

Prvi korak, ki ga moramo storiti, če želimo pripraviti ponudbo, je izbira komitenta. To lahko storimo na več načinov, najpogostejši pa je ta, da v iskalnik vnesemo ime in priimek, ter nato med vsemi rezultati izberemo ustreznega. Poleg imena in priimka lahko kot kriterij iskanja uporabimo še davčno oz. matično številko, ID komitenta in podobno.

Po izbiri komitenta se nam prikaže stran z osnovnimi podatki o komitentu (slika 6). Na ta način se prepričamo, da je to res oseba, kateri želimo poslati ponudbo.

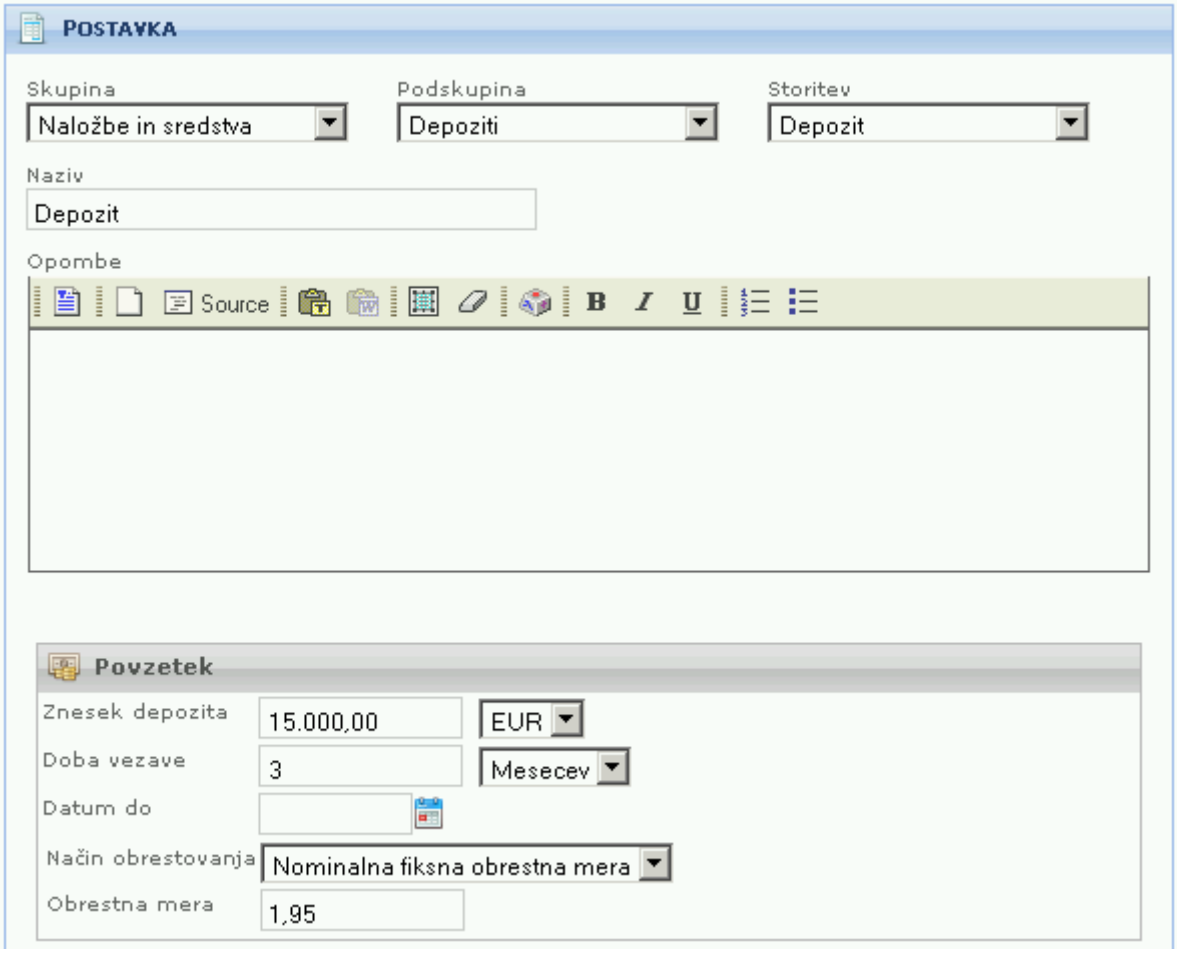
Slika 6. Prikaz osnovnih podatkov o komitentu

Nato v desnem robu iz nabora izberemo opcijo `Ponudba`, kliknemo ikono + (slika 7), ter tako prikličemo formo za vnos ponudbe.

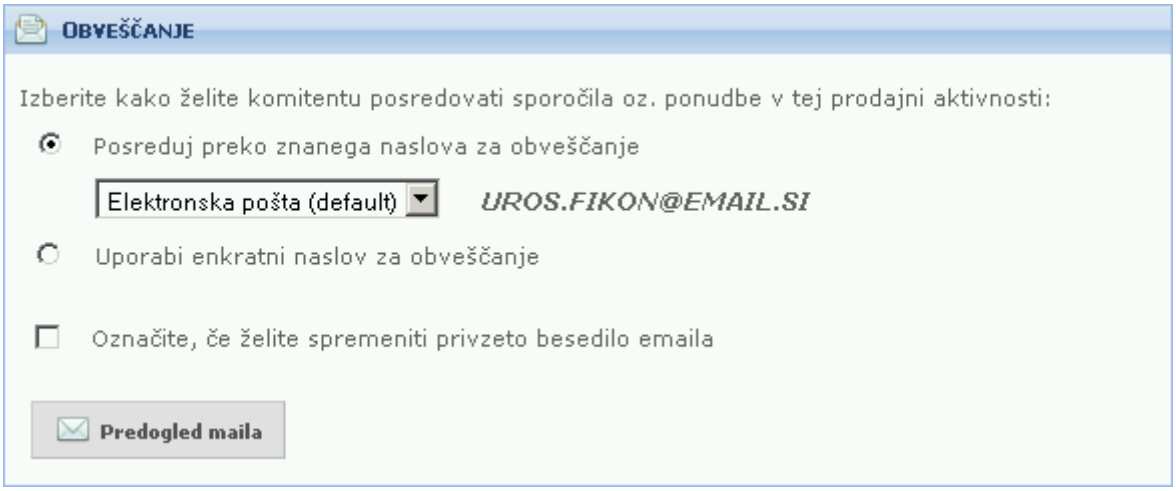


Slika 7. Formo za kreiranje ponudbe prikličemo s klikom na ikono +

Na njej se prikažejo vnosna polja za podatke, ki jih potrebujemo pri pripravi ponudbe. Nekatera polja imajo že vnaprej določene vrednosti glede na vlogo (rolo) uporabnika aplikacije. Osredotočili se bomo predvsem na osrednji del zaslona, kjer vnašamo podatke o storitvi (slika 8), ki jo ponujamo, ter podatke o obveščanju (preko katerega kanala želimo ponudbo posredovati – slika 9).

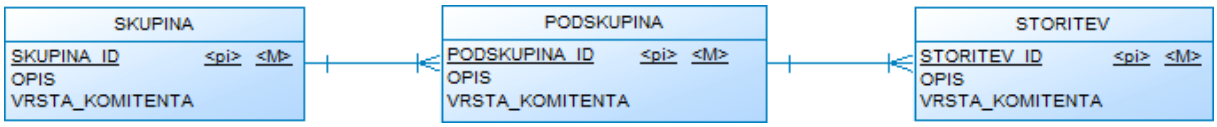


Slika 8. Osrednji del forme za pripravo ponudbe



Slika 9. Izbira kanala preko katerega bo komitentu poslana ponudba

Izbire skupina, podskupina in storitev predstavljajo tri nivoje, v katere so razdeljene storitve oz. produkti, ki jih banka trži. Nivoji so medsebojno odvisni: vsaka storitev pripada natanko eni podskupini, vsaka podskupina pa natanko eni skupini (slika 10).



Slika 10. Konceptualni model z medsebojnimi relacijami objektov skupina, podskupina in storitev

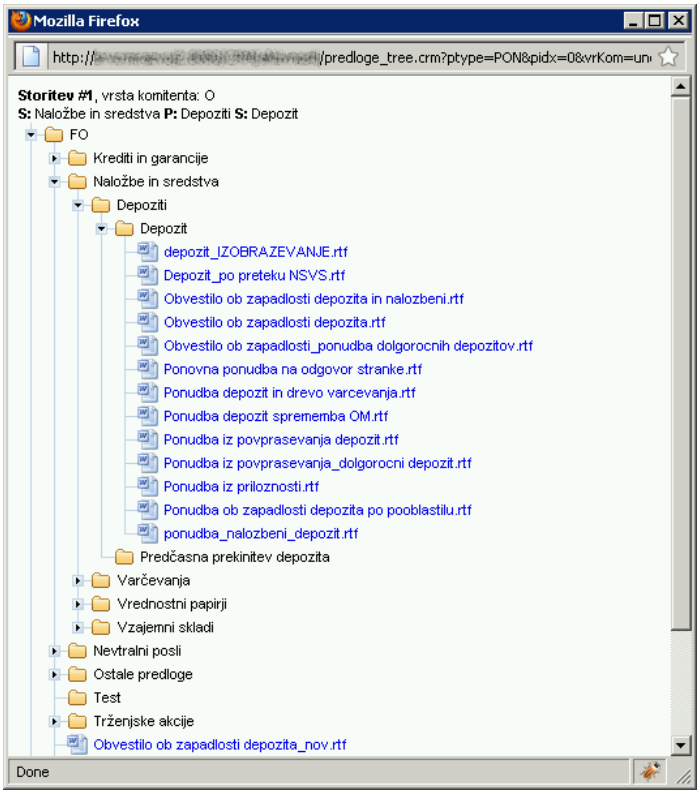
Ob izbiri storitve, se nam na formi prikaže še dodatna skupina polj imenovana `Povzetek`, kamor lahko vnesemo pogoje, pod katerimi izbranemu komitentu ponujamo storitev. Povzetek je lahko:

- splošen: vsebuje le najosnovnejša polja (npr. znesek, valuta, obdobje, tarifa),
- specifičen: poleg osnovnih polj vsebuje tudi polja, ki so za izbrano storitev še posebej pomembna (npr. vrsta zavarovanja pri kreditnih poslih ali oznaka vrednostnega papirja)

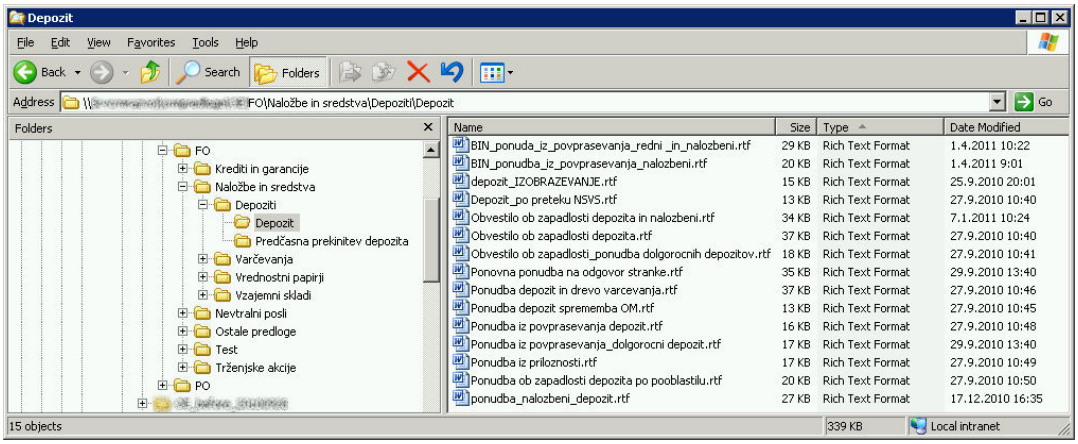
V nekaterih primerih povzetka sploh ne prikažemo, saj ponujena storitev ne predvideva posebnih pogojev (npr. ponudba za debetno kartico je enaka za vse komitente).

3.2 Izbira predloge

Po končanem vnosu pogojev za storitev, moramo izbrati ustrezno predlogo (angl. template) za ponudbo. To storimo tako, da kliknemo na gumb `Izbira predloge`. Odpre se nam novo okno, kjer omogočimo brskanje po skupni mapi, kjer so shranjene predloge (slika 11). Z uporabniškega vidika se brskanje izvaja enak način, kot če bi do skupne mape dostopali preko Raziskovalca (Windows Explorer; slika 12), kar je za uporabnika zelo intuitivno.



Slika 11. Brskanje po predlogah v spletni aplikaciji s pomočjo komponente `<rich:tree>`



Slika 12. Brskanje po predlogah z uporabo aplikacije Windows Explorer

Poleg tega je komponenta implementirana tako, da uporabnika avtomatično postavi na mapo, ki vsebuje predloge za izbrano storitev.

Za isto storitev je lahko na voljo več predlog, ki se med seboj razlikujejo po vsebini. Izbira ustrezne je prepuščena uporabniku na podlagi internih navodil. Skrbnik predlog pa je odgovoren, da predloge poimenuje nedvoumno. Uporabniki morajo namreč na podlagi naziva razumeti kaj je vsebina predloge.

V kolikor uporabnik ne najde ustrezne predloge, je dolžan o tem obvestiti podporno službo, ki poskrbi, da se predloga za dano storitev izdela.

3.3 Priprava dokumenta

Po izbiri ustrezne predloge se izvede poslovna logika, ki izdela končni dokument. Postopek se izvede v več korakih.

3.3.1 Razčlenitev dokumenta

Najprej moramo izbrati ustrezno ogrodje (angl. skeleton) predloge. Ogrodja se med seboj razlikujejo po kontaktnih podatkih in podpisu. Ločimo tri ogrodja: za komitente spletne banke, za uporabnike, ki delajo v Klicnem centru, ter za uporabnike poslovnih enot. Primer ogrodja je prikazan v poglavju 4.3.1 Dinamična izdelava RTF predloge (slika 22).

Vnašanje vrednosti v predlogo smo si zamislili z uvedbo prostornikov (angl. placeholder) oz. oznak, katere se ob procesiranju nadomestijo z ustrezno vrednostjo iz nabora podatkov, ki so bili poslani preko forme. Nabor oznak je vnaprej definiran in usklajen z uporabniki, ki skrbijo za pripravo predlog. Zaradi enostavnejšega vzdrževanja in zmanjšanja možnosti napak, smo postavili še nekaj pravil:

- oznake se začnejo in končajo z dvema številskima znakoma (angl. hash, `##`)
- oznake so navedene z velikimi črkami, pri čemer šumniki niso dovoljeni
- v kolikor je oznaka sestavljena iz več besed, so le-te med seboj povezane s podčrtajem

Primeri dovoljenih oznak:

- `##NAZIV##`
- `##DATUM_ROJSTVA##`
- `##ST_TRGOVALNEGA_RACUNA##`

Primeri nedovoljenih oznak:

- `##IME IN PRIIMEK##,`
- `##ODPLAČILNA_DOBA##`

Poleg oznak, ki služijo uporabnikom, smo uvedli še nekaj sistemskih oznak, in sicer:

- `##CONTENT##` - označuje mesto v ogrodju, ki ga nadomestimo z vsebino ponudbe
- `##VSEBINA_PONUDBE_BEGIN##` - označuje začetek vsebine ponudbe
- `##VSEBINA_PONDUBE_END##` - označuje konec vsebine ponudbe

Sistem za pripravo predloge najprej poskrbi, da se pravilno zlijeta (angl. merge) ogrodje in vsebina izbrane predloge. V naslednjem koraku pa poskrbi, da se "uporabniške" oznake nadomestijo s podatki ponudbe.

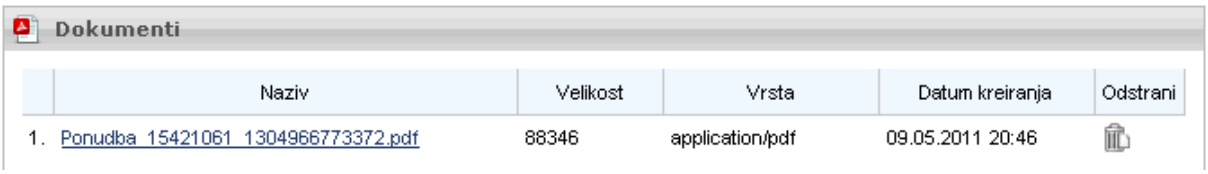
Tako ustvarjena datoteka, se uporabniku odpre v privzetem urejevalniku datotek tipa RTF, kjer lahko ponudbo po potrebi še dopolni oz. pregleda.

3.3.2 Nalaganje, pretvorba v format PDF in arhiviranje

Ko imamo tako pripravljeno končno RTF datoteko z vsebino ponudbe sledi pretvorba v PDF/A format. Tak format je primeren za dolgoročno elektronsko hrambo dokumentov, saj ustreza standardom, ki jih arhiviranje dokumentov določa. Izdelavo PDF datoteke izvedemo s klicem spletne storitve `Word2PDF`.

Po pretvorbi v PDF format se datoteko shrani še v sistem za upravljanje dokumentov Alfresco. Tudi pri tem postopku se poslužujemo tehnologije spletnih storitev.

V tej fazi moramo shraniti tudi metapodatke dokumenta (identifikacijsko številko, naziv in velikost datoteke, MIME tip,...), ki jih bomo potrebovali, ko bomo želeli dokument prikazati na uporabniškem vmesniku (slika 13).



Slika 13. Prikaz pripetega dokumenta na uporabniškem vmesniku

3.4 Distribucija dokumenta

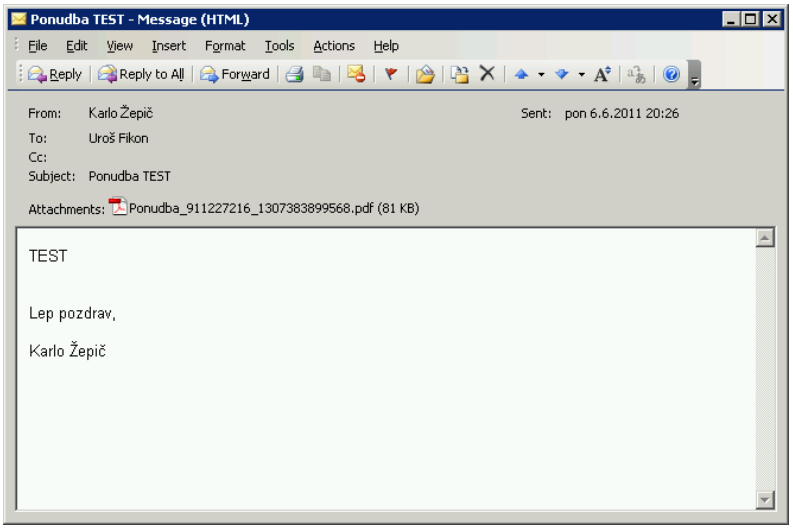
Ko je dokument pripravljen, moramo v razdelku »Obveščanje« le še izbrati izhodni kanal ga želimo posredovati komitentu. Nabor kanalov je omejen le na tiste, za katere imamo veljaven naslov. V kolikor za izbrani kanal nimamo podatka o naslovu, tega ne prikažemo v naboru (npr. če v podatkovni bazi nimamo podatka o elektronskem naslovu izbranega komitenta, potem ne ponudimo kanala elektronska pošta).

Po navadi imamo na voljo štiri izhodne kanale:

- spletna banka (v kolikor izbrani komitent uporablja storitev spletnega bančništva),
- elektronska pošta,
- navadna pošta in
- osebna vročitev na bančnem okencu, če ponudbo pripravljamo ob prisotnosti komitenta (ta kanal je vedno na voljo).

Če želimo dokument posredovati osebno na bančnem okencu, je dovolj, da ga natisnemo na lokalnem tiskalniku. V primeru navadne pošte je proces enak, s tem da kuvertiranje izvedemo ročno. V primeru masovne ponudbe, ko ponudbo pošiljamo večjemu številu komitentov, je proces avtomatiziran. Ponudba se namreč posreduje v tiskanje zadolženemu operaterju, kjer se poskrbi tudi za avtomatično kuvertiranje.

V primeru elektronske pošte, bo komitent ponudbo prejel na izbrani elektronski naslov v obliki PDF priponke (slika 14).



Slika 14. Elektronska pošta s pripeto ponudbo.

V kolikor pa komitent uporablja storitev spletnega bančništva, bo lahko do ponudbe dostopal neposredno iz spletne banke, kjer bo na ponudbo lahko tudi odgovoril (slika 15).

Slika 15. Pregled prejete ponudbe v spletni banki

4 Razvoj aplikativne rešitve

Tipičen proces, ki ga moramo podpreti, bi torej sestavljali sledeči koraki:

1. komitent pride na bančno okence in se zanima za določeno storitev
2. bančni uslužbenec (referent) komitentove zahteve zabeleži v aplikacijo
3. na podlagi zahtev in trenutno veljavnih pogojev referent pripravi ustrezno ponudbo
4. referent ponudbo posreduje komitentu bodisi v tiskani obliki neposredno v enoti oz. preko navadne pošte, ali v elektronski obliki preko enega od (elektronskih) kanalov: elektronska pošta ali spletna banka

Točki 1 in 2 sta opcijski, saj lahko referenti v primeru proaktivne prodaje (v kolikor komitent ustreza pogojem ciljne skupine za določeno akcijo) tudi sami začnejo proces kreiranja ponudbe, ne da bi komitent to zahteval.

Proces priprave ponudbe želimo začrtati tako, da bo za uporabnike čim bolj enostaven, ter da ga bodo osvojili kar se da hitro. Moramo se namreč zavedati, da je rešitev namenjena širokemu spektru uporabnikov (tako po izobrazbenem kot po starostnem kriteriju). To pomeni, da moramo uporabnikom ponuditi kar se da intuitivno rešitev, ki jih bo, če je le mogoče, do cilja pripeljala z osnovnim računalniškim znanjem.

Koraki, ki nas pripeljejo do dokumenta so sledeči:

1. izbira komitenta
2. izbira storitve, ki jo želimo tržiti ter vnos ustreznih podatkov
3. izbira naslova za dostavo (opcijsko)
4. izbira predloge
5. izdelava dokumenta
6. dopolnitev ponudbe (opcijsko)
7. nalaganje datoteke
8. izdelava PDF/A datoteke
9. uvoz datoteke v DMS
10. pošiljanje ponudbe preko različnih kanalov

V tem primeru predstavljata največji izziv točki 4 in 5, kateri bomo v nadaljevanju tudi detajlno opisali in predstavili njihovo izvedbo.

4.1 Vnosne forme

Kot rečeno so forme razvite z uporabo JSF komponent in njihovih razširitev, ki jih ponujata knjižnici RichFaces in ogrodje Seam. Predstavitveni sloj je izveden v jeziku XHTML, ki je nekoliko bolj restriktiven v primerjavi s starejšim HTML-jem. Pri prvem namreč aplikacijski strežnik javi napako, v kolikor vsebuje stran sintaktične napake.

Prva stvar, ki jo moramo pri pripravi ponudbe definirati, je storitev. Za to smo v vnosni formi predvideli tri kombinirane sezname (angl. combo box), ki predstavljajo trinivojsko segmentacijo storitev. Ob izbiri kateregakoli nivoja se nabor vrednosti v ostalih dveh seznamih ustrezno zmanjša. Na ta način smo uporabnikom želeli olajšati delo pri izbiri storitve.

Z izbiro storitve se avtomatično definirata tudi ustrezna skupina in podskupina. Enako se ob izbiri podskupine izbere ustrezna skupina, nabor storitev pa se zmanjša na tiste, ki so podrejene izbrani podskupini.

Poslovno logiko, ki skrbi za polnjenje in osveževanje nabora vrednosti te skupine kombiniranih seznamov, smo združili v samostojnem Java razredu oz. Seam komponenti - `StoritevAction`.

Slika 16 prikazuje del kode, ki skrbi za generiranje prikaza na uporabniškem vmesniku.

```
<a:region id="PodatkiStoritve">
...
<h:selectOneMenu id="skupina" value="#{ponudbaFormAction.skupina}">
    <s:selectItems value="#{storitevAction.getSkupine()}" var="_skupina"
label="#{_skupina.opis}" noSelectionLabel="-- Prosimo izberite --" />
    <a:support event="onchange"
action="#{storitevAction.refreshComboGroupData()}" reRender="PodatkiStoritve"
status="globalStatus" />
    <s:convertEntity />
</h:selectOneMenu>
...
</a:region>
```

Slika 16. Programska koda, ki generira kombinirani seznam na uporabniškem vmesniku.

Element `<h:selectOneMenu>` je standardna JSF komponenta, ki predstavlja HTML gradnik kombinirani seznam. Parameter `value` predstavlja lastnost (angl. property) na zaledni komponenti (angl. backing bean), kamor se shrani izbrana vrednost iz seznama.

Za generiranje nabora vrednosti seznama smo uporabili Seam-ova gradnika `<s:selectItems>` in `<s:convertEntity>`. Prvi omogoča, da kot zalogo vrednosti seznama uporabimo kar tipizirano listo javanskih objektov, npr. `List<Skupina>`. Te nato z ukazom `<s:convertEntity/>` avtomatično pretvorimo v listo objektov tipa `SelectItem`, ki jih element `<h:selectOneMenu>` po JSF standardu pričakuje.

V zgornjem primeru se lista objektov razreda `Skupina` napolni dinamično z ukazom `#{storitevAction.getSkupine()}` neposredno v XHTML datoteki. Do posameznega elementa v listi lahko nato dostopamo preko spremenljivke navedene z atributom `var`. Za izpis naziva skupine tako uporabimo ukaz `#{_skupina.opis}`. Ukaz po specifikaciji izvede klic javne metode `getOpis()` na razredu, ki ga predstavlja naša spremenljivka `_skupina`, to pa je istoimenski razred `Skupina` (slika 17).

```
@Entity
@Table(name="SKUPINA")
public class Skupina implements Serializable {

    private int id;
    private String opis;

    @Id
    @Column(name="SKUPINA_ID", nullable=false)
    public int getId() {
        return id;
    }

    @Column(name="OPIS", length=75, nullable=false)
    public String getOpis() {
        return opis;
    }

    ...
}
```

Slika 17. Definicija razreda Skupina z ORM anotacijami.

Kot že omenjeno želimo ob spremembi vrednosti v posameznem kombiniranem seznamu osvežiti zalogo vrednosti v ostalih dveh. Ker ob tem ni želimo osveževati celotne strani, se bomo poslužili tehnologije AJAX, ki jo omogoča knjižnica RichFaces. Gradnike te knjižnice po navadi označujemo s predlogo `a` (oz. `a4j`). To definiramo v korenskem elementu XHTML

datoteke, kjer opredelimo tudi vse ostale imenske prostore (angl. namespaces) katere bomo uporabljali na strani (slika 18).

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:a="http://richfaces.org/a4j"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:rich="http://richfaces.org/rich"
                xmlns:s="http://jboss.com/products/seam/taglib"
                xmlns:ui="http://java.sun.com/jsf/facelets">
```

Slika 18. Definicija uporabljenih imenskih prostorov v XHTML datoteki

Uporabili bomo element `<a:support>`, ki osnovnemu gradniku doda AJAX funkcionalnost. Dogodek, ki sproži asinhroni klic bo v našem primeru sprememba vrednosti seznama (atribut `onChange`).

Ob izbiri vrednosti v kombiniranem seznamu pokličemo torej metodo, ki je definirana v atributu `action`. V našem primeru je to metoda `refreshComboGroupData()` na Seam komponenti `storitevAction` (slika 19).

```
public void refreshComboGroupData() {

    PonudbaFormAction form = (PonudbaFormAction)
Component.getInstance("ponudbaFormAction");

    if (form.getStoritev() != null) {
        storitve = entityManager.createQuery("Select st from Storitev st where
st.storitevId = :storitevId").setParameter("storitevId",
form.getStoritev().getId()).getResultList();

podskupine = entityManager.createQuery("Select p from Podskupina p where
p.storitevId = :storitevId").setParameter("storitevId",
form.getStoritev().getId()).getResultList();

skupine = entityManager.createQuery("Select sk from Skupina sk where
sk.podskupinaId in (:podskupine)").setParameter("podskupine",
podskupine).getResultList();
    }
    else if (form.getPodskupina() != null) {
        ...
    }
    else if (form.getSkupina() != null) {
        ...
    }
}
```

Slika 19. Implementacija metode za osvežitev nabora vrednosti kombiniranega seznama

Metoda preveri kateri nivo je bil izbran, na podlagi vrednosti atributov storitev, podskupina in skupina v Seam komponenti `ponudbaFormAction`, ter ustrezno napolni ostala dva seznama. Objekti se s pomočjo ORM preslikave (v našem primeru se poslužujemo knjižnice Hibernate), preberejo iz podatkovne baze. Ključni objekt pri delu s podatkovno bazo predstavlja razred `EntityManager`. Uporablja se ga za kreiranje, odstranjevanje in iskanje vseh entitet, ki so definirane v naši aplikaciji. Razredi, ki predstavljajo entitete, so označeni z anotacijama `@Entity` in `@Table`.

Ko se asinhroni klic metode zaključi, moramo le še osvežiti ustrezen del strani. Razdelek, ki ga želimo osvežiti je naveden v atributu `reRender` elementa `<a:support>`. V našem primeru je to blok kode označen z identifikatorjem `PodatkiStoritve`, ki zajema le del forme vezan na storitev (slika 8):

- kombinirane sezname Skupina, Podskupina in Storitev in
- skupino polj, ki ga predstavlja rubrika Povzetek (konkretni podatki o storitvi, ki jo želimo ponuditi komitentu).

Sledi izbira ustrezne predloge za ponudbo.

4.2 Izbira predloge

Datotečni sistem je običajno predstavljen z drevesno strukturo. Trenutno izbrana mapa predstavlja koren, vsebovane datoteke so listi, (neprazne) podmape pa so vozlišča, ki lahko spet vsebujejo enake podelemente. Te tri entitete se lahko znotraj posameznega vozlišča ponovijo. Brskanje po taki podatkovni strukturi zato izvajamo rekurzivno.

Pri izdelavi uporabniškega vmesnika se bomo poslužili komponente `<rich:tree>`, ki je namenjena hierarhični prestavitvi podatkov, v našem primeru drevesne strukture.

Poleg tega bomo uporabili še `<rich:recursiveTreeNodesAdaptor>`, ki omogoča definiranje podatkovnih modelov, ter rekurzivno obdelavo vozlišč.

Definiramo še podatkovne strukture oz. razrede, ki jih pri tem potrebujemo, in sicer:

- `FileSystemNode`: predstavlja posamezen element drevesa oz. datotečnega sistema (mapo ali datoteko),

- `FileSystemBean`: hrani podatke o avtentikaciji za dostop do datotečnega sistema ter poskrbi za inicializacijo drevesne strukture na posameznem nivoju ter
- `FileSystemTreeStateAdvisor`: preverja status vozlišč; na vmesniku predstavlja razprtje ali zaprtje poddrevesa.

Podatki, ki jih o posameznem vozlišču hranimo v podatkovni strukturi `FileSystemNode` so:

- naziv,
- absolutna pot (na datotečnem sistemu),
- zastavico, ki nam pove ali vozlišče predstavlja mapo ali datoteko ter
- seznam podvozlišč (v primeru mape).

Poleg tega v razredu definiramo še dve pomožni metodi. Eno potrebujemo za prikaz ustrezne ikone na drevesu, drugo pa za pravilno delovanje v brskalniku Internet Explorer - v njem je potrebno šumnike v imenu datoteke nadomestiti s HTML ubežnimi znaki (angl. escape characters).

Drevo inicializiramo s klicem funkcije `getSourceRoots()` razreda `FileSystemBean`. Le-ta poskrbi, da se najprej kreira instanca objekta `NTLMPasswordAuthentication` (del knjižnice jCIFS), ki v nadaljevanju skrbi za avtentikacijo pri branju elementov datotečnega sistema, na katerem gostujejo predloge. Poleg tega iz konfiguracijske datoteke prebere še dva ključna podatka:

- lokacijo mape, ki predstavlja koren našega drevesa ter
- seznam tipov datotek (predlog), ki jih dovolimo uporabljati – uporabniška zahteva je bila namreč, da je dovoljeno uporabljati le datoteke tipa RTF in PDF.

Ko se inicializacija objekta konča, preidemo k branju. Branje poteka tako, da se kreira instanca razreda `FileSystemNode`, ki predstavlja določeno vozlišče, ki nato poišče neposredne potomce s klicem metode `getNodes()`. V kolikor je kateri od otrok hkrati tudi vozlišče, ob kliku nanj rekurzivno ponovi celoten postopek.

Metoda `getNodes()` je prikazana v prilogi A.

Spremenljivka `rootDir` predstavlja trenutno izbrano vozlišče. V kolikor je to mapa, preberemo neposredne otroke vozlišča, ter jih shranimo v dve listi (`files` in `dirs`, ki

predstavljata datoteke in mape). Pri branju datotek pa moramo upoštevati še prej omenjeno dejstvo, da je primernega tipa. Kot zanimivost naj omenimo, da smo med razvojem ugotovili še eno težavo: zaklepanje datoteke. Ob urejanju RTF datotek v urejevalniku MS Word se kreirajo začasne datoteke, ki povedo, da je originalna datoteka v urejanju. Na srečo lahko take datoteke identificiramo z enostavnim pogojem (njihovo ime se začne z nizom `~$`) in jih lahko potemtakem tudi enostavno izločimo.

Ko smo končali s prebiranjem potomcev, te shranimo v polje `children`. V kolikor izbrano vozlišče nima otrok, vrnemo polje z nič elementi.

Sedaj imamo vse elemente, ki jih potrebujemo za izdelavo grafičnega uporabniškega vmesnika (slika 20).

```
<rich:tree switchType="ajax" stateAdvisor="#{fileSystemTreeStateAdvisor}">
    <rich:recursiveTreeNodesAdaptor roots="#{fileSystemBean.sourceRoots}"
var="_item" nodes="#{_item.nodes}">
        <rich:treeNode iconLeaf="#{_item.icon}">
            <s:fragment rendered="#{_item.directory}">
                <h:outputText value="#{_item.fileName}"/>
            </s:fragment>
            <s:fragment rendered="#{not _item.directory}">
                ...
            </s:fragment>
        </rich:treeNode>
    </rich:recursiveTreeNodesAdaptor>
</rich:tree>
```

Slika 20. Programska koda, ki skrbi za prikaz komponente `<rich:tree>`

Gradniku `<rich:recursiveTreeNodesAdaptor>` moramo določiti najmanj tri atribute, in sicer:

1. `roots` (seznam korenskih vozlišč na trenutnem nivoju drevesa),
2. `var` (vozlišče is seznama `roots`, ki ga trenutno obdelujemo)
3. `nodes` (seznam podrejenih vozlišč trenutno izbranega vozlišča; vrednost tega seznama se ob kliku na mapo prenese v `roots`)

Posamezen element drevesa na vmesniku je predstavljen z gradnikom `<rich:treeNode>`. Za boljšo uporabniško izkušnjo smo posamezne elemente okrasili z ustrezno ikono. Tako smo vozliščem, ki predstavljajo direktorije pred naziv dodali ikono v obliki mape, ostalim datotekam pa ikono, ki predstavlja tip datoteke (RTF oz. PDF). S tem smo želeli ohraniti

doslednost s prikazom v Raziskovalcu (Windows Explorer), katerega referenti uporabljajo pri vsakdanjem delu z računalnikom.

4.3 Priprava datoteke

Naslednji korak v procesu je pridobivanje datoteke. Datoteko generiramo tako, da z miško kliknemo na naziv datoteke. Takrat se sproži klic servleta, ki izvede logiko za generiranje datoteke. Servlet je enostaven javanski razred, ki skrbi za obdelavo HTTP zahteve (angl. HTTP request). V večini primerov je odgovor na tako zahtevo HTML stran, ki se prikaže uporabniku - v našem primeru pa gre za dokument tipa RTF ali PDF.

Ob klicu moramo servlet-u posredovati še nekaj parametrov, ki so potrebni za uspešno generiranje dokumenta, in sicer:

- `path`: absolutna pot datoteke (predloge) na oddaljenem strežniku
- `userId`: ID referenta, ki pripravlja ponudbo
- `komId`: ID komitenta, za katerega se pripravlja ponudbo
- `sid`: ID seje (session ID) prijavljenega uporabnika, da lahko dostopamo do objektov, ki so shranjeni v seji in jih potrebujemo pri pripravi ponudbe

Pri pošiljanju parametrov preko URL naslova, moramo posebno pozornost posvetiti šumnikom, saj se lahko pri prenosu popačijo. Da se temu izognemo, uporabimo statično metodo `encode()` iz razreda `java.net.URLEncoder`, s katero poskrbimo, da so poslani znaki šifrirani z UTF-8 kodnim naborom (angl. character set), katerega priporoča tudi združenje World Wide Web Consortium. Tako se recimo niz `'Naložbeno varčevanje'` spremeni v `'Nalo%C5%BEbeno%20var%C4%8Devanje'`.

Niz parametrov shranimo v spremenljivko `_params` z uporabo pomožne komponente `<ui:param>` (slika 21), katero nato uporabimo pri klicu servlet-a preko JavaScript funkcije `Predloge.download()`.

```
<ui:param name="_params"
value="uuid=#{_item.escapedPath}&amp;userId=#{workingData.prijavljeniReferentId
}&amp;komId=#{workingData.izbraniKomitentId}&amp;sid=#{session.id}" />
<a href="#" onclick="Predloge.download('#{request.contextPath}','#{_params}');
">#{_item.fileName}</a>
```

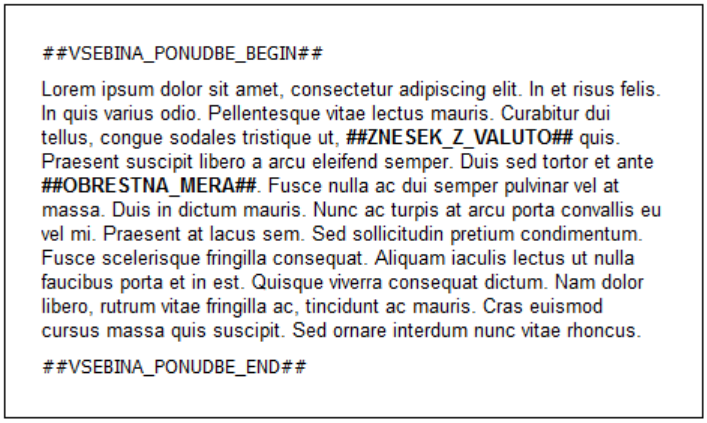
Slika 21. Posredovanje parametrov z uporabo komponente `<ui:param>`

Servlet najprej poskrbi za branje in preverjanje pravilnosti parametrov posredovanih preko HTTP zahteve. V kolikor je kateri neustrezen, aplikacijski strežnik zahtevo zavrne in uporabniku prikaže stran z napako. V nasprotnem primeru najprej dekodira parameter `path`, preko klica statične metode `java.net.URLDecoder.decode()` z enakim kodnim naborom kot je bil pred tem kodiran, v našem primeru UTF-8.

4.3.1 Dinamična izdelava RTF predloge

Kot že rečeno ponudbo sestavljajo predloga in podatki na njej. Predloga se dinamično tvori iz dveh ločenih RTF datotek, ki predstavljata ogrodje (slika 22) in vsebino ponudbe (slika 23).

Slika 22. Primer RTF datoteke, ki predstavlja ogrodje ponudbe



Slika 23. Primer RTF datoteke, ki predstavlja vsebino ponudbe

Vsebino ponudbe določa predloga, ki jo je uporabnik izbral v prejšnjem koraku. Uporabljeno ogrodje pa določajo nekateri parametri kot na primer podatek če komitent uporablja spletno banko. V tem primeru se glava predloge in podpis razlikujeta od predloge, ki je namenjena komitentom, ki se ne poslužujejo spletnega poslovanja.

Vsebino oddaljene datoteke pretvorimo v polje bajtov (angl. array of bytes) ter jo v taki obliki pošljemo v nadaljnje procesiranje. Prišli smo namreč do koraka, ko je potrebno izdelati vsebino datoteke.

Tudi v tem primeru vsebino datoteke, ki predstavlja ogrodje, pretvorimo v spremenljivko tipa polje bajtov in jo skupaj z vsebino predloge pošljemo v nadaljnje procesiranje.

Ko imamo definirano ogrodje in vsebino ponudbe, lahko generiramo predlogo. V nadaljevanju je opisan postopek, ki nas pripelje do tega. Manipulacijo datotek in vso logiko, ki skrbi za delo z RTF datotekami, smo združili v razredu `RTFTemplateHelper`.

Prvo stvar pri procesu izdelave predloge je analiziranje vsebine ponudbe. To storimo tako, da si prebrano vsebino iz prejšnjega koraka shranimo v začasno datoteko na strežniku, ter iz nje preberemo le vrstice, ki nas zanimajo. Za dostop do datoteke uporabimo razred `BufferedReader` [12]. Na ta način je branje vsebine datoteke učinkovitejše, saj ob vsakem branju preberemo količino podatkov, ki je definirana z velikostjo medpomnilnika. S tem omejimo število dostopov do datoteke oz. vhodnega toka. Uporabljeni algoritem je opisan v nadaljevanju.

Beremo posamezne vrstice v datoteki dokler ne zasledimo oznake `##VSEBINA_PONUDBE_BEGIN##`. V naslednji iteraciji, tj. naslednji vrstici, poiščemo indeks elementa `{` (zaviti oklepaj). Tega potrebujemo, ker se po RTF standardu stili oz. slogi označujejo z zavitimi oklepaji. Če tega ne bi upoštevali, se končna datoteka ne bi prikazovala pravilno. V primeru, da je vrstica prazna, preberemo naslednjo, sicer preberemo vsebino vrstice od prvega znaka `{` do konca vrstice. Z naslednjo iteracijo začnemo prebirati celotne vrstice, vse dokler ne zasledimo oznake `##VSEBINA_PONUDBE_END##`. V tem primeru preberemo vsebino vrstice od začetka do prvega znaka `{`, ki bi predstavljal nov slog.

Programska izvedba algoritma je prikazana v prilogi B.

Tako prebrano vsebino ponudbe shranimo v spremenljivko `vsebinaPonudbe` tipa `String`.

Sedaj z nekoliko enostavnejšim, a podobnim algoritmom obdelamo še datoteko, ki predstavlja ogrodje. Prebiramo posamezne vrstice, dokler ne zasledimo oznake `##CONTENT##`, katero enostavno nadomestimo z vrednostjo spremenljivke `vsebinaPonudbe`, ki smo jo ustvarili pred tem. Na ta način dobimo končno verzijo predloge, katero moramo le še napolniti s podatki ponudbe.

4.3.2 Pridobivanje podatkov za ponudbo

V okviru servleta moramo dostopati še do podatkov, ki so shranjeni v seji. Zaradi tega smo morali v prejšnjem koraku posredovati tudi ID seje trenutnega uporabnika. Vendar samo to ni dovolj. Spletna aplikacija namreč nima privzetega mehanizma, ki bi hranil podatke o aktivnih sejah. Tega je potrebno implementirati. Zaradi tega smo izdelali nov razred `SessionListener`, ki v mapi hrani referenco do vseh aktivnih sej aplikacije na aplikacijskem strežniku.

V osnovni konfiguraciji spletne aplikacije (datoteki `web.xml`), nato še registriramo razred, ki smo ga pravkar izdelali. To storimo z uporabo vozlišča `<listener>` (slika 24).

```
<listener>
    <listener-class>com.ufikon.utils.SessionListener</listener-class>
</listener>
```

Slika 24. Registracija razreda `SessionListener` v konfiguracijski datoteki `web.xml`.

Na ta način se ob vsaki kreirani seji v mapo shrani referenca do le-te. Dostop do posamezne seje je nato na enostaven način možen preko parametra `sid`, ki smo ga omenili na začetku poglavja (slika 25).

```
HttpSession session = SessionListener.getInstance().getSession(sid);
```

Slika 25. Pridobivanje seje z uporabo identifikatorja preko objekta `SessionListener`.

Ko imamo referenco na sejo uporabnika, lahko do objektov, ki so shranjeni v njej (angl. session-scoped komponente), dostopamo preko metode `getAttribute(String name)`. Seam komponente, ki so definirane na nivoju seje, so dostopne kar preko svojega imena. Tega določa notacija `@Name` (slika 26).

```
@Scope(ScopeType.SESSION)
@Name("rtfDownloadHelper")
public class RTFDownloadHelper implements Serializable {
    ...
}
```

Slika 26. Seam komponente poimenujemo z anotacijo `@Name`.

Do komponente `rtfDownloadHelper` lahko torej dostopamo na način, ki ga prikazuje slika 27.

```
RTFDownloadHelper rtfDownloadHelper = (RTFDownloadHelper)
session.getAttribute("rtfDownloadHelper");
```

Slika 27. Pridobivanje reference na Seam komponento preko objekta HttpSession.

Na komponenti imamo shranjene konkretne podatke o storitvi (povzetek), ter naslov za distribucijo dokumenta, ki jih shranimo v objekt `templateData`. Na isti objekt hkrati shranimo še matične podatke o komitentu (razred `Komitent`), ter podatke o referentu, ki ponudbo pripravlja (razred `Referent`), kot prikazuje slika 28.

```
Povzetek povzetek = rtfDownloadHelper.getPovzetek();
NaslovZaObvescanje naslov = rtfDownloadHelper.getNaslovZaObvescanje();

Referent referent = entityManager.find(Referent.class, userId);
Komitent komitent = entityManager.find(Komitent.class, komId);

RTFTemplateDataMap templateData = RTFTemplateHelper.getDataForPonudba(referent,
komitent, naslov, povzetek);
```

Slika 28. Priprava objekta, ki hrani podatke za pripravo ponudbe.

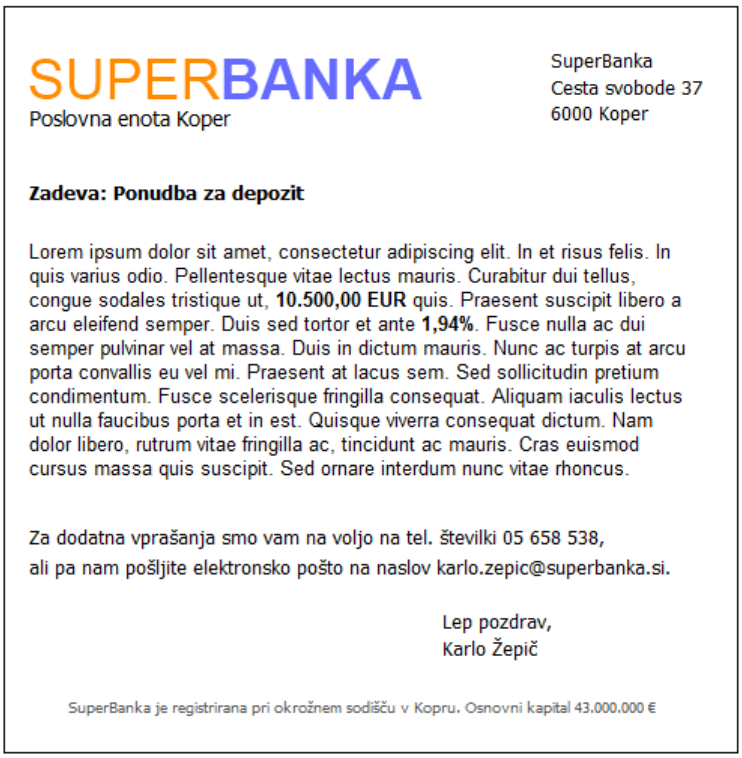
Na podlagi teh podatkov generiramo objekt tipa `HashMap<String, String>`, ki vsebuje formatirane podatke, potrebne za izpis v dokumentu, kot so: ime in priimek, naslov, znesek, obrestna mera, ipd. Podatke shranimo v mapo pod ustreznim ključem, ki predstavlja ime oznake v predlogi (glej poglavje 3.3.1 Razčlenitev dokumenta). Podrobnosti implementacije bomo v tem primeru izpustili. V tabeli 1 lahko vidimo nekaj zapisov, ki jih taka mapa vsebuje.

Tabela 1. Vsebina mape s podatki za ponudbo

Ključ	Vrednost	Opis
##KOMITENT_NAZIV##	Uroš Fikon	Naziv komitenta
##KOMITENT_NASLOV##	Turki 3	Naslov komitenta
##KOMITENT_POSTA##	6276 POBEGI	Poštna številka in kraj
...	...	...
##PE_NAZIV##	Poslovna enota Koper	Naziv poslovne enote, kjer se pripravlja ponudbo
##PE_NASLOV##	Cesta svobode 37	Naslov poslovne enote
##REFERENT_NAZIV##	Karlo Žepič	Naziv referenta, ki pripravlja ponudbo
##REFERENT_EMAIL##	Karlo.zepic@superbanka.si	E-naslov referenta, ki pripravlja ponudbo
...	...	...
##ZNESEK_Z_VALUTO##	10.500,00 EUR	Znesek in valuta za storitev
##OBRESTNA_MERA##	1,94%	Obrestna mera za storitev
...	...	...

4.3.3 Izdelava končnega RTF dokumenta

Ko imamo na voljo tako predlogo kot podatke, lahko izdelamo končni dokument (slika 29).



SUPERBANKA
Poslovna enota Koper

SuperBanka
Cesta svobode 37
6000 Koper

Zadeva: Ponudba za depozit

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In et risus felis. In quis varius odio. Pellentesque vitae lectus mauris. Curabitur dui tellus, congue sodales tristique ut, **10.500,00 EUR** quis. Praesent suscipit libero a arcu eleifend semper. Duis sed tortor et ante **1,94%**. Fusce nulla ac dui semper pulvinar vel at massa. Duis in dictum mauris. Nunc ac turpis at arcu porta convallis eu vel mi. Praesent at lacus sem. Sed sollicitudin pretium condimentum. Fusce scelerisque fringilla consequat. Aliquam iaculis lectus ut nulla faucibus porta et in est. Quisque viverra consequat dictum. Nam dolor libero, rutrum vitae fringilla ac, tincidunt ac mauris. Cras euismod cursus massa quis suscipit. Sed ornare interdum nunc vitae rhoncus.

Za dodatna vprašanja smo vam na voljo na tel. številki 05 658 538,
ali pa nam pošljite elektronsko pošto na naslov karlo.zepic@superbanka.si.

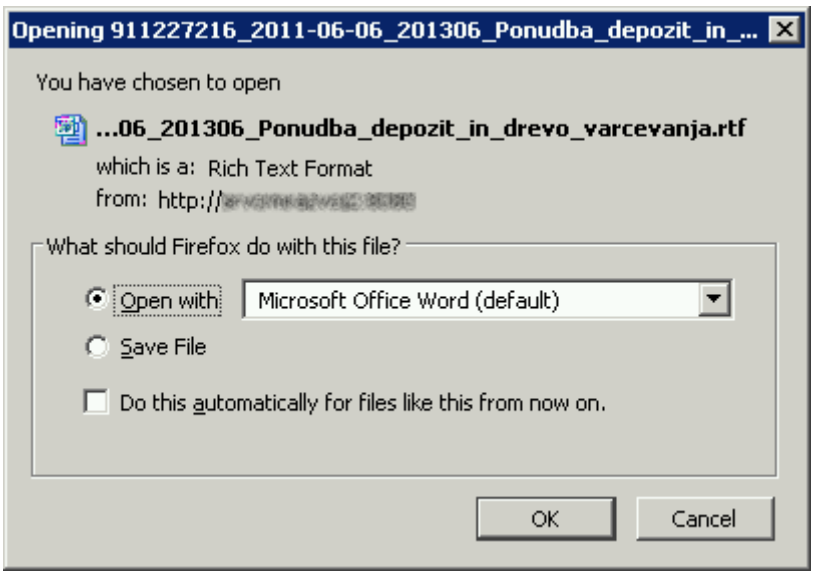
Lep pozdrav,
Karlo Žepič

SuperBanka je registrirana pri okrožnem sodišču v Kopru. Osnovni kapital 43.000.000 €

Slika 29. Primer izdelane ponudbe, kakršno prejme komitent

V osnovi uporabimo enak algoritem kot pri kreiranju osnovne predloge, in sicer:

1. preberemo posamezno vrstico RTF predloge
2. preverimo ali vrstica vsebuje katerega od ključev mape s podatki (torej niz znakov `##VAR##`)
3. v kolikor najdemo niz znakov, ki ustreza kateremu od ključev, ga zamenjamo z njegovo vrednostjo v mapi
4. preberemo naslednjo vrstico
5. korake od 1 do 4 ponavljamo, dokler ne pridemo do konca datoteke
6. uporabniku omogočimo da si tako generiran dokument shrani na lokalni disk (slika 30), in ga po potrebi še ročno dopolni



Slika 30. Dialog za shranjevanje končne RTF datoteke

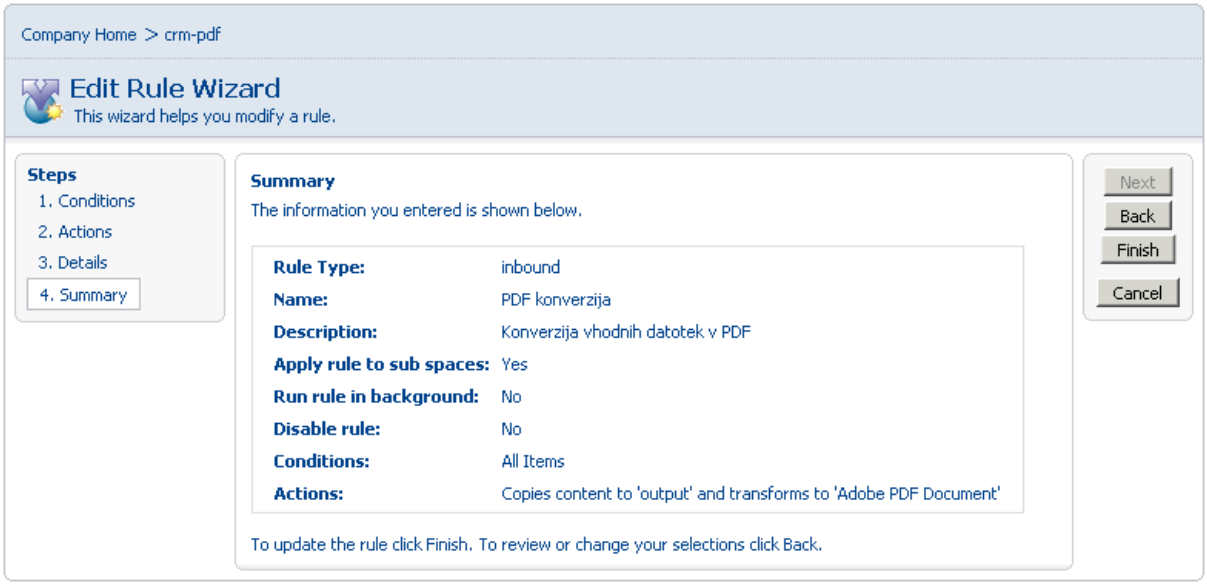
Zaradi varnosti in zahteve po uporabi standardnih rešitev je potrebno dokument pretvoriti še v PDF format.

4.4 Pretvorba v PDF/A format

Za PDF/A format smo se odločili, ker je to eden od standardnih formatov za dolgoročno hranjenje oz. arhiviranje elektronskih dokumentov. PDF/A format je definiran z ISO standardom 19005-1:2005 [26], ki je stopil v veljavo s 1. oktobrom 2005. Bistvena značilnost take datoteke je, da vsebuje vse pisave, ki se v dokumentu uporabljajo. Zaradi tega so običajno te datoteke večje in zasedajo več diskovnega prostora.

4.4.1 OpenOffice server

V prvi fazi smo za generiranje PDF dokumentov uporabili Alfresco-v osnovni mehanizem za pretvorbo datotek, ki temelji na OpenOffice strežniku. Princip delovanja je tak, da v grafičnem vmesniku sistema Alfresco definiramo vhodno mapo, kamor bomo shranjevali datoteke, katere bi želeli pretvoriti v format PDF. Nato kreiramo novo mapo, v katero bo sistem shranil pretvorjene datoteke. Sledi še kreiranje pravila za pretvorbo. To se na zelo enostaven način uredi s pomočjo čarovnika (slika 31).



Slika 31. Čarovnik za urejanje akcij nad mapami

Ob tem je potrebno še pravilno nastaviti OpenOffice strežnik. To storimo tako, da najprej namestimo paket OpenOffice. Nato v ukazni vrstici operacijskega sistema sprožimo ukaz z ustreznimi parametri, ki opredelijo, da aplikacija deluje v ozadju.

Ob dvigu aplikacijskega strežnika lahko v konzolnem oknu opazimo izpis, da se je Alfresco povezal z OpenOffice-om.

Komunikacija med našim sistemom za podporo prodaji in sistemom z upravljanje z dokumenti poteka preko spletnih storitev. Alfresco namreč že v osnovi nudi programski vmesnik (API) za spletne storitve. V osnovi torej iz naše aplikacije pošljemo dokument z metapodatki v Alfresco. Med metapodatki je tudi unikaten naziv datoteke, ki ga bomo potrebovali za pridobivanje PDF dokumenta. Ob pretvorbi se namreč osnovni dokument kopira v novo mapo in spremeni končnico, npr. iz `Ponudba_12345_1243523434.rtf` v `Ponudba_12345_1243523434.pdf`. Iskanje po repozitoriju datotek poteka z uporabo knjižnice Apache Lucene, ki indeksira metapodatke o shranjenih datotekah in tako omogoča hitrejše iskanje.

Pri kreiranju PDF dokumenta se pokličeta dve metodi: `addRTFContentToRepository()` in `getContentByFileName()`. Prva shrani RTF dokument v DMS, druga pa vrne metapodatke o PDF verziji poslanega dokumenta. Med enim in drugim klicem, se izvede še pretvorba formata.

Vrnjeni metapodatki druge metode nam omogočajo prikaz povezave (naziv datoteke, velikost, format,...) in priklic dokumenta (ID dokumenta v sistemu, njegova lokacija,...) iz prodajne aplikacije.

Po določenem času in vse večjemu dnevnemu obsegu števila generiranih dokumentov, smo se odločili, da poskusimo najti alternativno rešitev za proces pretvorbe dokumentov v PDF obliko. Slaba stran našega sistema je bila namreč velikost končne datoteke (dvostranski dokument v PDF obliki je namreč zasedel kar okrog 400 KB).

4.4.2 Komponenta Word2Pdf

Alternativo smo našli v komercialni rešitvi, Print2PDF [19], ki deluje kot navidezni tiskalnik. To pomeni, da moramo v primeru RTF datotek, le-te odpreti v ustreznem programu, klikniti opcijo "Natisni...", izbrati ustrezen tiskalnik, ter določiti še nekaj opcij kot na primer naziv izhodne datoteke.

Da bi ta proces avtomatizirali smo razvili nov razred, poimenovan `Word2PDF` in ključno metodo `createPDF()`. Predvidevali smo, da bo funkcionalnost uporabna še za kakšno drugo aplikacijo, kjer se pripravljajo PDF dokumenti, zato smo jo izpostavili kot spletno storitev. Metoda `createPDF()` sprejme kot vhodni parameter vsebino RTF datoteke v obliki polja bajtov, in v primeru uspešne pretvorbe vrne rezultat polje bajtov, ki pa tokrat predstavlja PDF datoteko. V primeru neuspeha ostane rezultat nedefiniran (`null`).

Postopek kreiranja PDF datoteke v grobem sestavlja pet korakov:

1. pretvorba vhodnega parametra tipa polje bajtov v RTF datoteko
2. odprtje te datoteke v programu Microsoft Word
3. izbira ustreznega tiskalnika
4. "tiskanje" dokumenta (priprava PDF datoteke)
5. pretvorba PDF datoteke v polje bajtov

Prvi in zadnji korak smo v naši aplikaciji že implementirali ob branju predlog, ostanejo nam torej še trije.

4.4.2.1 Razvoj komponente in izdelava spletne storitve

V kolikor bi aplikacijo razvijali na Microsoft platformi (.NET, C++, C#,...), bi bila rešitev najbrž trivialna. V našem primeru pa smo morali poiskati ustrezen "most", ki smo ga našli v knjižnici JACOB. Z njeno pomočjo smo razvili klic aplikacije Word kot ActiveX komponente.

Pri tem smo upoštevali še določene nastavitve (ime izhodne datoteke, zagon aplikacije Word v ozadju,...). Jedro metode je prikazano v prilogi D.

Upoštevali smo tudi dejstvo, da gre lahko pri pretvorbi kaj narobe in se proces ne zaključi pravilno. V tem primeru bi lahko prišlo do težav, ker bi na strežniku sčasoma zmanjkalo resursov. Težavi smo se izognili tako, da smo metodi dodelili največji dovoljen časovni okvir (angl. timeout), ki ga ima na voljo, da pretvorbo izvede do konca. V kolikor metoda v danem časovnem okviru ne bi uspela izvesti pretvorbe v PDF, bi sistem zahtevo zavrnil, in v vsakem primeru sprostil resurse. Kot privzeto časovno omejitev smo določili 15 sekund - meritve so namreč pokazale, da so za pretvorbo dokumentov v povprečju potrebne maksimalno tri sekunde.

Razred smo definirali kot SLSB, ki je ena od implementacij standarda EJB, temelja Java EE okolja. SLSB komponente ne shranjujejo nobenega podatka o stanju, ki bi se navezovalo na točno določenega odjemalca, od tod jim tudi ime (angl. stateless session bean). Poleg tega, obstaja več enakovrednih instanc objektov, in odjemalec ne ve vnaprej, kateri mu bo na voljo. Izkoristil bo tistega, ki mu ga bo ponudil aplikacijski strežnik. Komponenta bo izvedla svojo nalogo, posredovala rezultat odjemalcu, in se nato vrnila v bazen (angl. pool), kjer bo čakala na novo zahtevo.

EJB komponente so sestavljene iz vmesnika in razreda, ki ta vmesnik implementira. Kreirali smo torej vmesnik `Word2Pdf`, in v njem definirali podpis (angl. signature) metode `createPdf()`, ki bo skrbela za pretvorbo.

Da bi funkcionalnost omogočili tudi ostalim sistemom, smo morali komponento izpostaviti kot spletno storitev. To smo storili s pomočjo anotacij v uporabljenih Java razredih. Java anotacije so se prvič pojavile v verziji 1.5, in so se sčasoma razširile na vse več področij. Na tak način se izognemo konfiguraciji z uporabo XML datotek, ki postajajo vedno kompleksnejše kot napreduje razvoj.

Razredu `Word2PdfImpl`, smo torej dodali nekaj anotacij (slika 32), in tako definirali SLSB, ki je hkrati dostopen kot spletna storitev.

```
@Stateless
@WebService(name="Word2PdfWS", serviceName="Word2PdfWS",
portName="Word2PdfWSSoap")
@WebContext(contextRoot="/WS", urlPattern="/Word2PdfWS")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class Word2PdfImpl implements Word2Pdf, Serializable {

    @WebMethod(operationName="createPdf")
    @WebResult(name="pdfFile")
    public synchronized byte[] createPdf(@WebParam(name="inputFile") byte[]
inputFile) {
        ConversionResult rt = createPdf(inputFile);
        if ( rt.getStatus() != 0 ) {
            return null;
        }
        else {
            return rt.getContent();
        }
    }
    ...
}
```

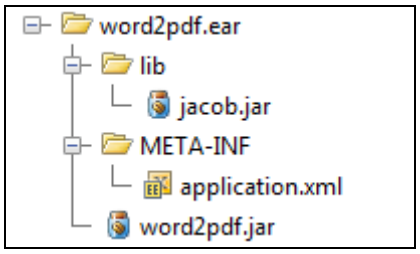
Slika 32. Konfiguracija spletne storitve z uporabo anotacij

Pomen posameznih anotacij, ki smo jih pri tem uporabili, je sledeč:

- `@Stateless` EJB vsebniku (angl. container) pove, da gre za SLSB
- `@WebService` označuje, da razred implementira spletno storitev
- `@WebContext` definira osnovni kontekst oz. URL naslov spletne storitve v spletni aplikaciji
- `@SoapBinding` opredeljuje način povezovanja spletne storitve na SOAP protokol
- `@WebMethod` metodo izpostavi spletni storitvi; preko spletne storitve so dostopne samo javne metode označene s to anotacijo
- `@WebParam` omogoča, da vhodnemu parametru dodelimo nekatere lastnosti, ki bodo na voljo v WSDL datoteki, med drugim ime parametra, za lažje razumevanje s strani odjemalcev
- `@WebResult` podobno kot `@WebParam`, tokrat za rezultat metode

4.4.2.2 Objava spletne storitve

Pred objavo na aplikacijskem strežniku JBoss, moramo servis še umestiti v spletno aplikacijo. To storimo tako, da razrede prevedemo, ter v ustrezni strukturi (slika 33) shranimo v EAR arhiv z imenom `word2pdf.ear`.



Slika 33. Struktura EAR arhiva spletne aplikacije Word2Pdf

V njej lahko opazimo datoteke:

- `jacob.jar` - knjižnica JACOB potrebna za komunikacijo z aplikacijo Microsoft Word
- `word2pdf.jar` - prevedeni Java razredi (SLSB `Word2Pdf` in pomožni razredi)
- `application.xml` - konfiguracijska datoteka, ki aplikacijskemu strežniku JBoss pove, da gre za java EE aplikacijo, katere module vsebuje in kje se nahajajo (slika 34)

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                 http://java.sun.com/xml/ns/javaee/application_5.xsd"
             version="5">

    <display-name>Word2Pdf</display-name>

    <module>
        <ejb>word2pdf.jar</ejb>
    </module>

</application>
```

Slika 34. Vsebina datoteke `application.xml`, ki opisuje aplikacijo Word2Pdf

Za pravilno delovanje knjižnice JACOB je potrebna še datoteka `jacob.dll`, katero moramo umestiti v globalno dostopno mapo (`path`), ali pa v mapo, ki jo specificiramo v nastavitvah Java navideznega stroja aplikacijskega strežnika. V našem primeru smo jo odložili kar v

mapo, kjer je nameščen operacijski sistem. Uporabili bomo operacijski sistem Microsoft Windows Server 2003 (ker potrebujemo aplikacijo Microsoft Word).

Sedaj moramo samo še kreirati novo instanco strežnika JBoss in objaviti našo aplikacijo `Word2Pdf`. Kreiranje nove instance strežnika JBoss pomeni kreirati novo mapo, ki vsebuje konfiguracijo za vse komponente, ki jih želimo v okviru aplikacijskega strežnika ponuditi odjemalcem. V našem primeru naredimo kopijo mape `default` in jo poimenujemo `word2pdf`. Nato iz nje izločimo vse servise, ki jih ne potrebujemo. Ta korak sicer ni obvezen, ampak s tem prihranimo nekaj spomina in procesorske moči ter zapremo morebitne varnostne luknje.

Nato v mapo `deploy` kopiramo arhivsko datoteko `word2pdf.ear`, ki smo jo izdelali v prejšnjem koraku.

Sledi še zagon aplikacijskega strežnika. V ukazni vrstici se postavimo v mapo, kjer je nameščen JBoss ter zaženemo skripto `run.bat` z ustreznimi stikali oz. parametri (slika 35).

```
D:\DEVELOPMENT\jboss-eap-4.3\bin>run.bat -c word2pdf -b 0.0.0.0
```

Slika 35. Zagon aplikacijskega strežnika iz ukazne vrstice

V konzolnem oknu nato spremljamo zagon aplikacijskega strežnika, kjer zasledimo tudi informacijo o tem, da je bil naš servis uspešno objavljen (slika 36).

```
INFO  [EARDeployer] Init J2EE application: file:/D:/DEVELOPMENT/jboss-eap-
4.3/server/word2pdf/deploy/word2pdf.ear
INFO  [JmxKernelAbstraction] creating wrapper delegate for:
org.jboss.ejb3.stateless.StatelessContainer
INFO  [JmxKernelAbstraction] installing MBean:
jboss.j2ee:ear=word2pdf.ear,jar=word2pdf.jar,name=Word2PdfWSImpl,service=EJB3
with dependencies:
INFO  [EJBContainer] STARTED EJB: com.ufikon.ws.word2pdf.Word2PdfWSImpl ejbName:
Word2PdfWSImpl
INFO  [EJB3Deployer] Deployed: file:/D:/DEVELOPMENT/jboss-eap-
4.3/server/word2pdf/tmp/deploy/tmp6841711162180043036word2pdf.ear-
contents/word2pdf.jar
INFO  [WSDLFilePublisher] WSDL published to: file:/D:/DEVELOPMENT/jboss-eap-
4.3/server/word2pdf/data/wsdl/word2pdf.ear/word2pdf.jar/Word2PdfWS12167148889253
04084.wsdl
INFO  [DefaultEndpointRegistry] register:
jboss.ws:context=WS,endpoint=Word2PdfWSImpl
INFO  [TomcatDeployer] deploy, ctxPath=/WS, warUrl=.../tmp/deploy/word2pdf.ear-
word2pdf.jar2281413365511923248.war/
INFO  [EARDeployer] Started J2EE application: file:/D:/DEVELOPMENT/jboss-eap-
4.3/server/word2pdf/deploy/word2pdf.ear
```

Slika 36. V log datoteki aplikacijskega strežnika lahko opazimo, da je aplikacija uspešno objavljena.

To lahko preverimo tudi z uporabo brskalnika, in sicer tako, da odpremo naslov `http://localhost:8080/jbossws/services`, kjer vidimo seznam vseh registriranih spletnih storitev (slika 37).

Slika 37. Vmesnik za pregled objavljenih spletnih storitev

S klikom na povezavo `"Endpoint address"`, pa lahko vidimo tudi definicijo spletne storitve, kot to določa standard WSDL (priloga C).

4.4.2.3 Uporaba spletne storitve

Če želimo spletno storitev uporabiti, moramo najprej izdelati objekte, ki omogočajo komunikacijo s spletno storitvijo. To storimo s pomočjo orodja, ki na podlagi WSDL definicije spletne storitve, generira ustrezne razrede. Orodje, ki ga pri tem uporabimo, je odvisno od platforme, ki jo bomo kot odjemalec uporabili. V našem primeru bomo uporabili orodje `wsconsume` [13], ki je del JBoss aplikacijskega strežnika.

V ukazni vrstici se postavimo v namestitveno mapo JBoss aplikacijskega strežnika, ter poženemo ukaz `wsconsume -o stubs http://localhost:8080/WS/Word2Pdf?wsdl`.

Rezultat te operacije nova mapa v delovnem direktoriju z imenom `stubs`, ki vsebuje dve datoteki (`Word2PdfWS.class` in `Word2PdfWS_Service.class`), kateri uporabimo v naši aplikaciji za podporo prodaji, za klic spletne storitve `Word2PdfWS` (slika 38).

```
byte[] rtfFile = ...; // vsebino rtf datoteke shranimo kot polje bajtov

Word2PdfWS_Service ws = new Word2PdfWS_Service();
Word2PdfWS soap = ws.getWord2PdfWSSoap();
byte[] pdfFile = soap.createPdf(rtfFile);
```

Slika 38. Primer klica spletne storitve.

V kolikor pri pretvorbi ni bilo napak, se v spremenljivko `pdfFile` shrani vsebina PDF datoteke, v nasprotnem primeru ostane spremenljivka brez vrednosti (`null`).

4.5 Zapis dokumenta v DMS

Pred sklenitvijo procesa nam je tako ostal samo še korak, kjer zapišemo PDF datoteko v sistem za upravljanje z dokumenti. Alfresco ponuja nekaj spletnih storitev, ki omogočajo interakcijo s sistemom [5]. Med njimi so:

- Authentication - skrbi za prijavo/odjavo iz sistema,
- Repository - skrbi za poizvedovanje in manipulacijo modela,

- Content - skrbi za manipulacijo vsebine,
- Authoring - skrbi za kreiranje vsebine za sodelovanje (forum,...),
- Classification - skrbi za klasifikacijo vsebin in njihove kategorije,
- Administration - urejanje uporabnikov, izvoz/uvoz podatkov,
- Action - skrbi za upravljanje akcij in pravil (JBPM),
- Access Control - upravljanje uporabniških pravic in vlog (angl. roles).

V nadaljevanju bomo za dodajanje dokumenta v repozitorij uporabili spletna servisa `ContentService` in `RepositoryService`.

V grobem je proces tak, da kreiramo referenco na nadrejeno vozlišče. Nato določimo ime datoteke, ki jo želimo dodati. Preverimo, če dokument s takim imenom že obstaja, in v tem primeru zbrišemo staro verzijo. Nato kreiramo CML stavek, ki ustvari nov zapis z metapodatki o datoteki, ki jo želimo dodati. Kot zadnje nam ostane še zapis datoteke v repozitorij.

Podrobna implementacija je prikazana v prilogi E.

Rezultat metode je objekt tipa `AlfrescoDocument`, ki vsebuje sledeče metapodatke:

- ID vozlišča v repozitoriju,
- ime datoteke,
- pot do datoteke preko CIFS protokola,
- vsebinski/MIME tip datoteke (content type oz. MIME type) in
- velikost datoteke.

Te metapodatke nato shranimo v podatkovni model kot del podatkov o ponudbi, saj jih potrebujemo za prikaz dokumenta v spletni aplikaciji oz. za njegovo distribucijo.

4.6 Distribucija dokumenta

Zadnji korak v procesu je distribucija pripravljenih dokumentov. Omejili se bomo le na elektronske kanale, torej elektronsko pošto in spletno banko, ker sta ostala dva trivialna.

4.6.1 Elektronska pošta

Uporabili smo rešitev, ki je že bila na voljo v drugi bančni aplikaciji. To je servis za pošiljanje elektronske pošte imenovan `MailService`. Deluje tako, da na določen interval preverja vsebino mape, če obstaja kakšna datoteka tipa `cfg`. Če odkrije takšno datoteko, jo pretvori v standardni format za elektronsko pošto, ter preko SMTP strežnika posreduje prejemniku. V primeru uspeha, se izvorna `cfg` datoteka prenese v arhivsko mapo.

Datoteka je navadnega tekstovnega tipa in vsebuje minimalni nabor podatkov, potrebnih za kreiranje elektronske pošte (slika 39):

- prejemnik (`From`),
- pošiljatelj (`To`),
- vsebinski tip (`ContentType`),
- velikost (`MessageSize`),
- zadeva (`Subject`),
- vsebina (`[MESSAGE]`) ter
- priponke (`AtnX`)

```
[MAIL]
From=karlo.zepic@superbanka.si
To=uros.fikon@email.si
ContentType=text/html
MessageSize=3551
Subject=Ponudba TEST

[MESSAGE]
TEST

[ATTACHMENTS]
Atn000="\\ALF_SRV\Alfresco\documents\pdfs\Ponudba_911227216_1307383899568.pdf"
```

Slika 39. Vsebina datoteke iz katere se generira elektronska pošta.

V primeru elektronske pošte se torej uporabi atribut `cifsPath` iz objekta `AlfrescoDocument`, ki hrani podatek o lokaciji datoteke na skupni mapi. `MailService` bo tako lahko dostopal do datoteke, ter jo dodal v elektronsko pošto na standarden način, kot priponko.

4.6.2 Spletna banka

V primeru spletne banke je rešitev enaka kot v aplikaciji za podporo prodaji. Na dnu HTML strani, ki predstavlja vsebino ponudbe se prikaže povezava z nazivom in velikostjo datoteke, ter ikono, ki ponazarja vsebinski tip - v našem primeru je to standardna ikona za PDF datoteko (slika 13).

S klikom na povezavo, se pokliče komponenta `AlfrescoDownloadServlet`, ki poskrbi za prikaz dokumenta. Servlet-u moramo poslati še parameter `nodeId`, ki predstavlja ID dokumenta, ki ga želimo prikazati, npr. `http://www.superbanka.si/downloadFile?nodeId=123-234sdfsdf-123asdas-34534`

Ponovno se bomo posluževali spletne storitve `ContentService`, tokrat bomo uporabili metodo `read()`, ki prebere vsebino iz repozitorija. Definirali bomo predikat, ki se navezuje na dokument z danim identifikatorjem, ter prebrali njegovo vsebino (slika 40).

```
ContentServiceSoapBindingStub contentService =
WebServiceFactory.getContentService();
Reference r = new Reference(STORE, nodeId, null);

Content[] readResult = contentService.read(new Predicate(new Reference[]{r},
STORE, null), Constants.PROP_CONTENT);
```

Slika 40. Pridobivanje dokumenta iz Alfresco repozitorija

V kolikor je bil klic metode uspešen, bi morali kot rezultat dobiti polje objektov tipa `Content`. Ker smo za iskanje vsebine uporabili identifikator (ID), je iskani element na prvi poziciji polja, torej element z indeksom 0. Potrebne podatke si shranimo v pomožni razred `AlfrescoContentData`, katerega instanciramo s klicem konstruktorja, ki sprejme štiri parametre: ime datoteke, MIME tip, velikost, in vsebino datoteke (slika 41).

```
if ( readResult != null ) {
    Content doc = readResult[0];

    String mimeType = doc.getFormat().getMimetype();
    long fileLength = doc.getLength();
    String url = doc.getUrl();
    String fileName = url.substring(url.lastIndexOf('/') + 1);

    byte[] bytes = getContentAsByteArray(doc);

    return new AlfrescoContentData(fileName, mimeType, fileLength, bytes);
}
else {
    return null;
}
```

Slika 41. Branje metapodatkov in vsebine dokumenta.

Sledi samo še del kode, ki uporabniku ponudi dialog za odprtje oz. shranjevanje datoteke na lokalni disk. To storimo tako, da objektu `HttpServletResponse` v zaglavju (angl. http response header) eksplicitno definiramo atribut `"Content-disposition"` z vrednostjo `"attachment;filename=\"ime_datoteke.pdf\""`. Na ta način brskalniku prepovemo odprtje datoteke (zaradi performans oz. težav z vtičniki). Poleg tega moramo določiti še vsebinski tip ter velikost datoteke, ki bo posredovana preko objekta `HttpServletResponse` (slika 42).

```
AlfrescoContentData doc = ...;

// nastavimo da se v brskalniku prikaže dialog za prenos datoteke
res.setHeader("Content-disposition","attachment;filename=\"" +doc.getFilename()+
"\"");

// nastavimo mimetype podatkov, ki jih prenašamo
res.setContentType(doc.getMimeType());

// določimo še velikost datoteke, ki se prenaša
res.setContentLength((int) doc.getLength());
```

Slika 42. Nastavitev HTTP zaglavja pred prenosom datoteke.

Ko smo definirali vse nastavitve na objektu `HttpServletResponse` in ko uporabnik potrdi prenos datoteke, sledi še prenos vsebine le-te k odjemalcu. To storimo s pisanjem na izhodni tok (angl. output stream) objekta `HttpServletResponse` (slika 43).

```
// deliver the data
OutputStream outputStream = res.getOutputStream();
outputStream.write(doc.getData());
```

Slika 43. Prenos datoteke (pisanje podatkov na izhodni tok).

S tem korakom se proces vezan na pripravo in distribucijo ponudbe tudi zaključi.

5 Sklepne ugotovitve

Z raziskovalnega vidika je bila izdelava aplikacije izredno zanimiva, saj je zajemala številna področja: od namestitve in konfiguracije aplikacijskega strežnika do postavitve sistema za upravljanje dokumentov. Še posebej pa bi izpostavili številne načine za integracijo sistemov, ki so danes na voljo. Nenazadnje smo med razvojem odkrili številne zanimive odprtokodne rešitve, s katerimi smo si prihranili marsikatero uro programiranja.

Proces priprave ponudbe, ki je bil v tem diplomskem delu opisan, se že več kot leto dni uspešno uporablja v produkcijskem okolju ene od slovenskih bank. Odziv uporabnikov je nadvse spodbuden, saj smo z uvedbo rešitve omenjeni proces bistveno skrajšali. Prednost izbranega pristopa vidimo predvsem v pozitivni uporabniški izkušnji. Končni uporabnik se namreč poslužuje znanih orodij in ne čuti odpora, ki se včasih pojavi pri uvedbi novega informacijskega sistema.

Po drugi strani pa je glavna slabost velika mera discipline in natančnosti, ki jo potrebuje pripravljavec predlog, saj sistem ne dopušča večjih nepazljivosti. Problematična je lahko tudi aplikacija, s katero se predlogo pripravlja (npr. Microsoft Word), saj lahko določene sistemske oznake (na primer oznaka za krepke črke) pokvarijo našo oznako v "izvorni kodi" RTF datoteke.

Boljšo alternativo bi v tem primeru predstavljala uporaba kombinacije XML/XSLT datotek, kjer bi v prvih hranili podatke, v drugih pa predstavitveno logiko. Tak način prikaza podatkov je v svetu računalništva zelo razširjen in se uporablja tudi za prikaz elektronskih pogodb in računov. V tem primeru bi večjo težavo predstavljalo naknadno urejanje takšnega dokumenta. Aplikacije, ki to omogočajo, so specifične in niso tako množično razširjene kot npr. Microsoft Word ali OpenOffice Writer, kar bi pri končnih uporabnikih najbrž vzbudilo nejevoljo in posledično odpor do rešitve.

6 Literatura

[1] D. Allen, "Seam in Action", Manning, 2008

[2] J.Jamae, P. Johnson, "JBoss in Action", Manning, 2011

[3] W.C, Richardson, D. Avondolio, S. Schrager, M.W. Mitchell, J. Scanlon, "Professional Java JDK 6 Edition", Wiley Publishing, 2007

[4] M. Shariff, "Alfresco Enterprise Content Management Implementation", Packt Publishing, 2006

[5] Alfresco Content Management Web Services documentation. Dostopno na: http://wiki.alfresco.com/wiki/Alfresco_Content_Management_Web_Services#Services

[6] Alfresco ECMS. Dostopno na: http://www.alfresco.com/

[7] Apache Lucene. Dostopno na: http://lucene.apache.org/java/docs/features.html

[8] Autonomy Interwoven. Dostopno na: http://www.interwoven.com/

[9] EMC corp., Documentum. Dostopno na: http://www.emc.com/products/category/enterprise-content-management.htm

[10] IBM, FileNet Content Manager. Dostopno na: http://www-01.ibm.com/software/data/content-management/filenet-content-manager/

[11] JACOB - Java COM Bridge. Dostopno na: http://sourceforge.net/projects/jacob-project/

[12] Java Platform, Standard Edition 6 API specification – BufferedReader. Dostopno na: http://download.oracle.com/javase/6/docs/api/java/io/BufferedReader.html

[13] JBossWS – wsconsume. Dostopno na: http://community.jboss.org/wiki/JBossWS-Wsconsume

[14] Microsoft, Introduction to ActiveX Controls. Dostopno na: http://msdn.microsoft.com/en-us/library/aa751972(VS.85).aspx

[15] Microsoft, What is COM? Dostopno na: http://www.microsoft.com/com/default.mspx

[16] OpenText corp., ECM suite. Dostopno na: http://www.opentext.com/2/global/products/products-document-management.htm

[17] Oracle, Asynchronous JavaScript Technology and XML (Ajax) With the Java Platform. Dostopno na: http://www.oracle.com/technetwork/articles/javaee/ajax-135201.html

[18] Oracle, JavaServer Faces Technology. Dostopno na: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[19] Print2PDF. Dostopno na: http://www.software602.com/pdf-products/print2pdf/

[20] RichFaces Project Page. Dostopno na: http://www.jboss.org/richfaces/

[21] Seam Framework, JBoss Seam. Dostopno na: http://seamframework.org/

[22] Terracotta, Quartz Job Scheduler. Dostopno na: http://www.quartz-scheduler.org/

[23] The Java CIFS Client Library. Dostopno na: http://jcifs.samba.org/

[24] The JACOB Project: a JAva-COM Bridge. Dostopno na: http://danadler.com/jacob/

[25] Webopedia – Web services. Dostopno na: http://www.webopedia.com/TERM/W/Web_Services.html

[26] Wikipedia – PDF/A. Dostopno na: http://en.wikipedia.org/wiki/PDF/A

7 Priloge

Priloga A: metoda za pridobivanje podrejenih elementov izbrane mape

```
public synchronized FileSystemNode[] getNodes() {
    if ( children == null ) {
        try {
            String root = "smb://" + this.path + '/';
            SmbFile rootDir = new SmbFile(root,
FileSystemBean.ntlmPasswordAuthentication, SmbFile.FILE_SHARE_READ);
            SmbFile[] childNodes = null;
            ArrayList<SmbFile> files = new ArrayList<SmbFile>();
            ArrayList<SmbFile> dirs = new ArrayList<SmbFile>();
            if ( rootDir.isDirectory() ) {
                childNodes = rootDir.listFiles();
                for ( SmbFile sf : childNodes ) {
                    if ( sf.isDirectory() ) {
                        dirs.add(sf);
                    }
                    else if ( sf.isFile() ) {
                        String fileName = sf.getName();
                        String ext =
StringUtils.substringAfterLast(fileName.toLower(), ".");
                        if ( FileSystemBean.allowedExtensions.contains(ext) ) {
                            if ( fileName.startsWith("~$") ) {
                                continue;
                            }
                            files.add(sf);
                        }
                    }
                }
            }
            if ( childNodes != null ) {
                int actualChildrenNumber = dirs.size() + files.size();
                children = new FileSystemNode[actualChildrenNumber];

                for ( int i = 0; i < dirs.size(); i++ ) {
                    String nodePath = dirs.get(i).toString();
                    if ( nodePath.endsWith("/") ) {
                        nodePath = nodePath.substring(0, nodePath.length() - 1);
                    }
                    children[i] = new FileSystemNode(nodePath, true);
                }

                int start_idx = dirs.size();
                for ( int i = 0; i < files.size(); i++ ) {
                    String nodePath = files.get(i).toString();
                    if ( nodePath.endsWith("/") ) {
                        nodePath = nodePath.substring(0, nodePath.length() - 1);
                    }
                    children[i + start_idx] = new FileSystemNode(nodePath, false);
                }
            }
            else {
                children = CHILDREN_ABSENT;
            }
        }
        catch ( Exception e ) {
            log.error("An error occured while reading subnodes: " + e, e);
        }
    }
    return children;
}
```

Priloga B: algoritem za branje vsebine RTF predloge

```
...

StringBuilder tmp = new StringBuilder();
String line;
boolean doFetchContent = false;
boolean firstLineThatMatches = false;
boolean skipFetchOnFirstLineThatMatches = false;
while ( (line = in.readLine()) != null ) {
    if ( firstLineThatMatches ) {
        int idx = line.indexOf("{");
        if ( idx == -1 ) {
            continue;
        }
        line = line.substring(idx);
        tmp.append(line).append(Constants.NEW_LINE);
        firstLineThatMatches = false;
        doFetchContent = true;
        skipFetchOnFirstLineThatMatches = true;
    }

    if ( line.contains("##VSEBINA_PONUDBE_BEGIN##") ) {
        firstLineThatMatches = true;
    }
    else if ( line.contains("##VSEBINA_PONUDBE_END##") ) {
        line = line.substring(0, line.indexOf("{"));
        tmp.append(line).append(Constants.NEW_LINE);
        doFetchContent = false;
    }

    if ( doFetchContent ) {
        if ( skipFetchOnFirstLineThatMatches ) {
            skipFetchOnFirstLineThatMatches = false;
        }
        else {
            tmp.append(line).append(Constants.NEW_LINE);
        }
    }
}
return tmp.toString();

...
```

Priloga C: vsebina WSDL datoteke spletne storitve Word2Pdf

```
<definitions name="Word2PdfWS"
             targetNamespace="http://word2pdf.ws.ufikon.com/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://word2pdf.ws.ufikon.com/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <types/>
    <message name="Word2PdfWS_createPdfResponse">
        <part name="pdfFile" type="xsd:base64Binary"/>
    </message>
    <message name="Word2PdfWS_createPdf">
        <part name="inputFile" type="xsd:base64Binary"/>
    </message>
    <portType name="Word2PdfWS">
        <operation name="createPdf" parameterOrder="inputFile">
            <input message="tns:Word2PdfWS_createPdf"/>
            <output message="tns:Word2PdfWS_createPdfResponse"/>
        </operation>
    </portType>
    <binding name="Word2PdfWSBinding" type="tns:Word2PdfWS">
        <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="createPdf">
            <soap:operation soapAction=""/>
            <input>
                <soap:body namespace="http://word2pdf.ws.ufikon.com/"
use="literal"/>
            </input>
            <output>
                <soap:body namespace="http://word2pdf.ws.ufikon.com/"
use="literal"/>
            </output>
        </operation>

    </binding>
    <service name="Word2PdfWS">
        <port binding="tns:Word2PdfWSBinding" name="Word2PdfWSSoap">
            <soap:address location="http://lt6730b:8080/WS/Word2PdfWS"/>
        </port>
    </service>
</definitions>
```

Priloga D: jedro metode za generiranje PDF datoteke

```
...

ActiveXComponent wrd = null;

try {
    Variant backgroundPrinting = new Variant(true);
    Variant appendPrint = new Variant(false);
    Variant printingRange = new Variant(-1);
    Variant outputFileName = new Variant(Utils.TEMP_DIRECTORY + pdfFileName);
    Object[] printArgs = {backgroundPrinting, appendPrint, printingRange,
outputFileName};

    wrd = new ActiveXComponent("Word.Application");
    wrd.setProperty("Visible", new Variant(false));
    wrd.setProperty("ActivePrinter", "Print2PDF");

    Dispatch docs = wrd.getProperty("Documents").toDispatch();
    Dispatch doc = Dispatch.call(docs, "Open", inFileName).toDispatch();
    Dispatch app = wrd.getProperty("Application").toDispatch();

    Dispatch.callN(app, "PrintOut", printArgs);
    long startPrintingTime = System.currentTimeMillis();

    // čakamo, da se printanje izvede do konca
    while ( Dispatch.call(app, "BackgroundPrintingStatus").getInt() > 0 ) {
        if ( System.currentTimeMillis() - startPrintingTime < Utils.DEFAULT_TIMEOUT
) {
            Thread.sleep(Utils.WAIT_TIME);
        }
        else {
            throw new RuntimeException("Timeout reached while printing " +
inFileName);
        }
    }
    Dispatch.call(doc, "Close");

    ...

}
catch ( Exception e ) {
    log.error("Napaka pri klicu metode 'createPDF': " + e, e);
    res = new ConversionResult(-1);
}

finally {
    if ( wrd != null ) {
        try {
            wrd.invoke("Quit");
        }
        catch ( Exception e ) {
            log.error("Napaka: " + e, e);
            res = new ConversionResult(-1);
        }
    }
}

...
```

Priloga E: metoda za dodajanje dokumenta v Alfresco repozitorij preko spletne storitve

```
...

// Create a reference to the parent where we want to create content
Store storeRef = new Store(Constants.WORKSPACE_STORE, "SpacesStore");

ParentReference parentNode = new ParentReference(storeRef, null, getPdfFolder(),
Constants.ASSOC_CONTAINS, null); // pdfFolder:
/app:company_home/cm:documents/cm:pdfs

RepositoryServiceSoapBindingStub repositoryService =
WebServiceFactory.getRepositoryService();
ContentServiceSoapBindingStub contentService =
WebServiceFactory.getContentService();

NamedValue[] contentProps = new
NamedValue[]{Utils.createNamedValue(Constants.PROP_NAME, fileName)}; // set
filename
NamedValue[] titledProps = new
NamedValue[]{Utils.createNamedValue(Constants.PROP_TITLE, fileName)}; // set title

// Assign name
parentNode.setChildName(CONTENT_NS + fileName);
CMLAddAspect addAspect = new CMLAddAspect(Constants.ASPECT_TITLED, titledProps,
null, "1");
CML cml = new CML();

// check if file already exist - if so, delete it and create a new one
Query query = new Query(Constants.QUERY_LANG_LUCENE, "PATH:\"" + getPdfFolder() +
"/cm:" + fileName + '\"');
ResultSet rs = repositoryService.query(storeRef, query, false).getResultSet();
if ( rs.getTotalRowCount() > 0 ) {
    ResultSetRow[] rsRows = rs.getRows();
    Reference reference = new Reference(storeRef, rsRows[0].getNode().getId(),
null);
    Predicate predicate = new Predicate(new Reference[]{reference}, null, null);
    CMLDelete delete = new CMLDelete(predicate);
    cml.setDelete(new CMLDelete[]{delete});
    repositoryService.update(cml);
}

// Construct CML statement to create content node
// Note: Assign "1" as a local id, so we can refer to it in subsequent CML
statements within the same CML block
CMLCreate create = new CMLCreate("1", parentNode, parentNode.getUuid(),
Constants.ASSOC_CONTAINS, null, Constants.PROP_CONTENT, contentProps);
// Construct CML Block
cml = new CML();
cml.setCreate(new CMLCreate[]{create});
cml.setAddAspect(new CMLAddAspect[]{addAspect});
// Issue CML statement via Repository Web Service and retrieve result
// Note: Batching of multiple statements into a single web call
UpdateResult[] result = repositoryService.update(cml);
Reference content = result[0].getDestination();

ContentFormat format = new ContentFormat(MIMETYPE_PDF, Constants.ENCODING_UTF8);

// Write the content
contentService.write(content, Constants.PROP_CONTENT, inputFile, format);

...
```