UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Leon Ropoša

Uporaba dokumentnega sistema Alfresco pri izdelavi delovnega toka za potrjevanje pogodb

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM

PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. 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: 00087/2011
Datum: 01.04.2011

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

Kandidat: **LEON ROPOŠA**

Naslov: **UPORABA DOKUMENTNEGA SISTEMA ALFRESCO PRI IZDELAVI DELOVNEGA TOKA ZA POTRJEVANJE POGODB**

USING THE ALFRESCO DOCUMENT SYSTEM IN CONSTRUCTION OF WORK FLOW FOR CONTRACTS APPROVAL

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Alfresco je odprtokodni dokumentni sistem, ki predstavlja zanimivo alternativo plačljivim dokumentnim sistemom. Predstavite funkcionalnost in uporabo sistema pri izdelavi aplikacije, ki temelji na delovnem toku za potrjevanje pogodb v podjetju. Pri tem izpostavite predvsem možnosti, ki jih sistem ponuja za rešitev specifičnih zahtev uporabnikov.

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 Leon Ropoša, z vpisno številko 63060241, sem avtor diplomskega dela z naslovom:

UPORABA DOKUMENTNEGA SISTEMA ALFRESCO PRI IZDELAVI DELOVNEGA TOKA ZA POTRJEVANJE POGODB

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca
- 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 14.10.2011 Podpis avtorja:

ZAHVALA

Mentorju viš. pred. dr. Igorju Rožancu se zahvaljujem za pomoč in vodenje pri izdelavi diplomske naloge.

Zahvaljujem se tudi podjetju Agila d.o.o., ker mi je omogočilo izdelavo diplomskega dela, še posebej mag. Janezu Hrovatu za strokovno pomoč pri izdelavi diplomske naloge.

Posebna zahvala gre mojim staršem, ker so mi omogočili študij in me podpirali pri študiju.

Kazalo

SEZNAM KRATIC:

- AMP (ang. Alfresco Module Package) – paket Alfresco modula
- API (ang. Application Programming Interface) – aplikacijski programski vmesnik
- CIFS (ang. Common Internet File System) – splošni internetni datotečni sistem
- DMS (ang. Document Management System) – sistem za upravljanje z dokumenti
- DN (ang. Distinguished Name) – prepoznavno ime
- ECM (ang. Enterprise Content Management) - sistem za upravljanje poslovnih vsebin
- FTL (ang. FreeMarker Template Language) - FreeMarker jezik za predloge
- FTP (ang. File Transfer Protocol) – protokol za prenos datotek
- JBPM (ang. JBoss Business Process Management) – JBoss orodje za upravljanje poslovnih procesov
- JDK (ang. Java Development Kit) – Java razvojno okolje
- JDPL (ang. Java Process Definition Language)- Java jezik za definiranje procesov
- JRE (ang. Java Runtime Environment) - Java izvršno okolje
- JSP (ang. Java Server Pages) – Java strežniške strani
- LDAP(ang.Lightweight Directory Access Protocol) - preprost protokol za dostop do imenika
- OCR (ang. Optical Character Recognition) – optično prepoznavanje črk
- OMR (ang. Optical Mark Recognition) – optično prepoznavanje znakov
- RTE (ang. Rich Text Editor) – bogati urejevalnik teksta
- SSO (ang. Single Sign On) – enkratna prijava
- TCP/IP (ang. Transmission Control Protocol/ Internet Protocol) protokol nadzora prenosa/internetni protokol
- URL (ang. Uniform Resource Locator) - enolični krajevnik vira
- WAR (ang. Web Application Archive) – arhiv spletne aplikacije
- WebDAV (ang. Web-based Distributed Authoring and Versioning) – porazdeljena avtorizacija in verzioniranje, ki temeljita na spletu

POVZETEK

Diplomska naloga opisuje razvoj delovnega toka za potrjevanje pogodb v dokumentnem sistemu Alfresco. Alfresco je brezplačen in odprtokodni dokumenti sistem. Je spletna aplikacija napisana v Javi. Delovni tok, ki je opisan v diplomski nalogi, je bil razvit za enega izmed naših naročnikov, z namenom olajšati postopek potrjevanja pogodb. Poskrbeli smo za elektronsko shrambo pogodb in z razvojem lastnega delovnega toka podprli poslovni proces potrjevanja pogodb.

Najprej smo morali določiti uporabniške skupine, ki bodo del delovnega toka in mape, kjer se bodo shranjevali dokumenti. Nato smo morali ustrezno konfigurirati sistem Alfresco za postavitev delovnega toka. V sistem smo vključili dostop z WebDAV protokolom, LDAP dostop, Sharepoint protokol in Tesseract OCR. Za boljšo uporabniško izkušnjo smo prevedli vmesnik in dodali pogled v repozitoriju, ki prikaže zgolj metapodatke pogodb.

Nato je sledil razvoj delovnega toka. V Alfrescu se za definicijo delovnih tokov uporablja jezik JDPL, ki temelji na XML datotečnem formatu. V modelu delovnega toka smo definirali naloge, ki bodo nastopale v delovnem toku, medtem ko smo v procesu delovnega toka definirali zaporedje izvajanja nalog. Med razvojem delovnega toka smo uporabljali tudi programska jezika Javo in Javascript. Na koncu smo še ustrezno oblikovali obrazce nalog delovnega toka. Končni rezultat diplomske naloge je dobra in zlahka nadgradljiva rešitev.

Ključne besede:
Dokumentni sistem, elektronska shramba dokumentov, Alfresco, poslovni proces, delovni tok

ABSTRACT

The following thesis describes the development of workflow for contracts approval in document management system Alfresco. Alfresco is an open source and license free document management system and a web application written in Java. The workflow described in the thesis, was developed for one of our customer in order to simplify the process of approving contracts. We supported the electronic storage of documents and with the development of a custom workflow, we also supported the business process of contracts approval.

First we had to define the user groups, that will take part in workflow and folders, where the documents of the workflow will be stored. Then we had to configure Alfresco for the development of the workflow. We included in system access with WebDAV protocol, LDAP access, Sharepoint protocol and Tesseract OCR. We also translated the interface in Slovene. We added a view in the repository, that only shows the metadata of contracts.

After the configuration we started developing the workflow. Worflows in Alfresco are defined with the XML based language JDPL. In the model of the workflow we defined tasks, that will appear in the workflow. In the process of the workflow, we defined order in which the tasks will take place. During the development of the workflow we used programming languages Java and Javascript. In the end we also visually designed the forms of workflow's tasks. The end result of the thesis is a good and easily upgradable solution.

Keywords:

Document management system, electronic document storage, Alfresco, business process,workflow

1. UVOD

V diplomskem delu smo želeli predstaviti dokumentni sistem Alfresco in podrobno opisati, kako ustvarimo delovni tok v Alfrescu. Delovni tok predstavljen v diplomi, temelji na dejanskem problemu. Pri nekem naročniku so imeli težave pri potrjevanju pogodb med svojimi zaposlenimi, saj je postopek potekal v večji meri prek elektronske pošte med zaposlenimi. Imeli so tudi težave pri shranjevanju pogodb in so poleg postopka za potrjevanje, potrebovali tudi učinkovit način za elektronsko shrambo pogodb.

Za rešitev tega problema bi lahko razvili lastno aplikacijo ali pa bi uporabili eno od že razvitih aplikacij. Tovrstna aplikacija bi temeljila na zunanjem datotečnem sistemu in podatkovni bazi. Bila bi dokaj enostavna, saj naše podjetje nima dovolj virov, da bi razvilo kompleksnejšo aplikacijo. Tudi, če bi imeli dovolj virov za kompleksnejšo aplikacijo, se kljub vloženemu trudu in času ne bi približali kvaliteti že razvitih programskih rešitev. Končen izdelek bi najbrž bil dokaj slab.

Namesto tega smo se odločili za uporabo ene izmed že razvitih rešitev. Izbrati smo morali nek dokumentni sistem, ki omogoča upravljanje z dokumenti in podpira delovne tokove. Izbirali smo med komercialnim in brezplačnim dokumentnim sistemom.

Če bi izbrali komercialen dokumentni sistem, bi pridobili tudi podporo razvijalcev in končni produkt bi bil najverjetneje kvalitetnejši. Vendar so v praksi komercialni dokumentni sistemi predragi zaradi plačila licenc, zato smo se odločili za uporabo brezplačen sistem. Izbrali smo dokumentni sistem Alfresco.

Alfresco je brezplačen in odprtokoden sistem kar pomeni, da imamo dostop do celotne programske kode in jo lahko spreminjamo glede na naše potrebe. Ker je Alfresco kompleksen sistem, bi verjetno porabili kar precej časa najprej za učenje in potem še za razvoj. Končni produkt bi zagotovo bil boljši kakor, če bi sami razvili aplikacijo. S pridobljenim znanjem bi potem v prihodnje lahko hitreje razvijali druge rešitve, ki temeljijo na Alfrescu.

2. DOKUMENTNI SISTEM

2.1. Kaj je dokumenti sistem

Dokumentni sistem (ang. DMS Document Management System) je računalniški sistem ali nabor programske opreme, ki se uporablja za shranjevanje in upravljanje elektronskih dokumentov [4]. Dokumentni sistemi so običajno del sistema za upravljanje poslovnih vsebin (ang. ECM - Enterprise Content Management). Alfresco kot ECM aplikacija omogoča naslednje funkcionalnosti:

- upravljanje z dokumenti (ang. Document management),
- upravljanje s spletnimi vsebinami (ang. Web content management),
- upravljanje z zapisi (ang. Records management) in
- upravljanje s slikami (ang. Image management).

2.2. Komponente dokumentnega sistema in njihova izvedba v Alfrescu

2.2.1. Metapodatki

Metapodatki (ang. ,etadata) so informacije, ki opisujejo neke podatke, vendar pa niso nujno del samih podatkov. V primeru dokumentnega sistema so metapodatki informacije o dokumentu shranjenem v repozitoriju (ang. repository). Poleg podatkov, ki so vezani na vsebino obstajajo metapodatk, ki niso vezani na samo vsebino dokumenta in opisujejo denimo velikost dokumenta, format dokumenta ali pa čas kreiranja in spreminjanja dokumenta.

Poleg tega imamo lahko tudi metapodatke, ki so vezani na samo vsebino dokumenta. Običajno so to podatki, ki smo jih sami definirali. V primeru pogodbe so to na recimo datum veljavnosti pogodbe, vrednost pogodbe, naslov pogodbe in pogodbeni partnerji.

V dokumentnem sistemu nam metapodatki omogočajo lažje iskanje po dokumentih in lažji pregled nad vsebino dokumentov. Dokumentni sistemi nam tudi omogočajo, da prenesemo metapodatke na dokument s kopiranjem delov teksta iz dokumenta ali pa s prenosom podatkov iz slike dokumenta z uporabo OCR aplikacije.

V Alfrescu lahko metapodatke dodamo z dodajanjem aspekta na dokument ali pa dokument dobi metapodatke s tipom vsebine (ang. Content type). Vsak dokument, ki se ustvari v Alfrescu, namreč pripada določenemu tipu vsebine. Glavna razlika med definiranimi metapodatki je, da so metapodatki, ki jih definira tip vsebine, trajni in bodo zmeraj del dokumenta tudi, če bodo brez vsebine. Metapodatke, ki jih definira aspekt pa lahko dodajamo ali pa odvzemamo iz dokumenta z dodajanjem in odvzemanjem aspektov. Dokument zmeraj pripada natančno enemu tipu vsebine, ima pa lahko poljubno število aspektov.

2.2.2. Integracija

Dokumentni sistemi nam pogosto omogočajo direktno integracijo dokumentnega sistema v druge aplikacije. V praksi to pomeni, da lahko znotraj aplikacije dostopamo do vsebine repozitorija, to vsebino potem uredimo ter jo nato shranimo nazaj v repozitorij kot novo verzijo. Vse to lahko počnemo brez zapuščanja aplikacije. Dokumentni sistemi si pri tem pomagajo s standardi kot so LDAP, CIFS in WebDAV [1].

Alfresco omogoča integracijo z aplikacijami: Outlook Express, Liferay, Microsoft Office, Drupal, Lotus Quickr, Joomla in drugimi.

2.2.3. Zajemanje dokumentov

Zajemanje dokumentov (ang. capturing) obdela slike dokumentov v dokumente, ki vsebujejo tekst. Tako dobimo na primer iz slike skeniranega dokumenta v formatu tiff datoteko pdf, ki vsebuje tekst, katerega lahko prekopiramo v nek drug dokument ali pa lahko iščemo zaporedje besed znotraj teksta tega dokumenta. Za omogočanje te funkcionalnosti dokumentni sistemi uporabljajo aplikacije za slikovno prepoznavanje črk - OCR (ang. Optical Character Recognition). Dokumentni sistemi imajo lahko OCR aplikacijo že vgrajeno ali pa izvedemo integracijo dokumentnega sistema z zunanjo OCR aplikacijo.

Pri določenih tipih dokumentov moramo uporabiti tudi aplikacije za slikovno prepoznavanje znakov - OMR (ang. Optical Mark Recognition). Te aplikacije prepoznavajo človeške označbe na sliki [13]. Primer so označbe odgovorov na testih. Na testih so za izbiro posameznega odgovora na voljo krožci za vsak različen odgovor. Ko študent pobarva nek krožec, izbere ustrezen odgovor, kar OMR aplikacija s procesiranjem zazna.

V dokumentnih sistemih uporabljamo tudi aplikacije za zajemanje podatkov iz faktur (ang. invoice capture), ki delujejo z uporabo OCR. Z njimi razberemo specifične podatke iz dokumentov, za katere je značilno, da se določena informacija pojavi vedno na istem mestu. Primer tega bi bila položnica. Tako bi aplikacija za zajemanje podatkov iz faktur iz skenirane položnice lahko razbere vrednost položnice ali pa podatke prejemnika in nalogodajalca. Ti podatki se potem posredujejo dokumentnemu sistemu, ki te podatke doda v ustrezne metapodatke dokumenta. Tak pristop deluje seveda le na dokumentih s predpisano obliko.

Primeri nekaterih OCR aplikacij, ki jih lahko integriramo v Alfresco so: Ocropus, Tesseract OCR, Intelliant OCR, ABBYY FineReader in drugi.

Primeri aplikacij za zajemanje podatkov iz faktur, ki jih lahko integriramo z Alfrescom: Kofax Document Capture, Ephesoft.

2.2.4. Sodelovanje

Sodelovanje (ang. collaboration) v dokumentnem sistemu omogoča skupinsko delo več uporabnikov tako, da pri tem ni ogrožena konsistentnost vsebine dokumentov. Osnovna definicija sodelovanja pomeni, da imajo lahko dostop do določene vsebine v repozitoriju samo uporabniki, katerim je ta vsebina namenjena.

Prav tako imajo samo nekateri uporabniki možnost spreminjanja vsebine. Uporabnik, ki lahko spreminja vsebino, mora seveda imeti tudi dostop do nje. Možno je tudi, da ima uporabnik dostop do vsebine, nima pa pravic, da bi to vsebino spreminjal. Kadar nek uporabnik vsebino ureja, morajo imeti drugi uporabniki blokiran dostop do te vsebine, da dokument ostane usklajen.

Naprednejše oblike sodelovanja dovoljujejo uporabnikom gledanje in spreminjanje vsebine istočasno. Posebna oblika sodelovanja so označbe in komentarji o vsebini. Z njimi uporabniki podajo svoje mnenje o vsebini brez spreminjanja dejanske vsebine dokumentov ali popravljanja metapodatkov. Ključno pri komentarjih in označbah je, da se ohranjajo. K boljšemu sodelovanju prispeva tudi verzioniranje – beleženje sprememb, ki so se zgodile na vsebini dokumentov.

V Alfrescovem repozitoriju imamo na vsaki mapi (ang. space) možnost določanja pravic za dostop do te mape (ang. rules). Določimo lahko, za koga velja ta pravica in koliko pravic ima na tej vsebini. Stopnjo pravic na vsebini določimo z uporabo vlog (ang. roles).

Vloge so v Alfrescu že definirane s seznamom pravic, ki določa, kaj je uporabniku znotraj mape dovoljeno in potem kaj več ni. Da lahko uporabnik vsaj vidi in bere vsebino v neki mapi mora obvezno imeti določeno neko vlogo na tej mapi. Če na mapi nima nobene vloge, nima dostopa do mape in je sploh ne vidi v repozitoriju.

Ko v Alfrescu uporabnik ureja vsebino, lahko drugi uporabniki to vsebino vidijo, ne morejo pa je več spreminjati, saj se vsebina zaklene. Alfresco namreč ne omogoča uporabnikom urejanja vsebine istočasno. Ko uporabnik konča z urejanjem, dokument odklene in ga lahko začne urejati drugi uporabnik.

Vmesnik Alfresco Share nam omogoča veliko dodatnih, bolj naprednih možnosti za sodelovanje. Z uporabo spletišč (ang. sites) nam močno olajša upravljanje dostopa uporabnikov do map. Namesto, map in pravic v repozitoriju lahko ustvarimo novo spletišče in potem zgolj dodajamo v spletišče uporabnike kot člane spletišča.

Spletišča omogočajo tudi veliko dodatnih načinov komuniciranja med svojimi člani. Tako imamo na voljo funkcionalnosti:

- slovar pojmov (ang. Wiki): v njega vnesemo pojme in razlago teh pojmov,
- preprosti blog ,
- koledar prihajajočih dogodkov in
- preprost spletni forum (ang. Discussions) za uporabnike spletišča.

2.2.5. Verzioniranje dokumentov

Verzioniranje dokumentov (ang. versioning) je podobno procesu verzioniranja, ki se uporablja pri razvoju programske opreme. Z uporabo verzioniranja beležimo spremembe v vsebini dokumenta. Tako s spreminjanjem dokumenta dobimo novo verzijo dokumenta, pomembno pa je tudi, da se starejše verzije dokumenta ohranjajo. S tem imamo pregled, kako se je spreminjala vsebina dokumenta v repozitoriju dokumentnega sistema skozi čas.

Dokumentu v Alfrescu lahko omogočimo verzioniranje, če ima ta aspekt `Versionable`. Ko dodamo aspekt na dokument se ustvari začetna verzija dokumenta (ang. initialVersion). V Alfrescu verzioniranje poteka avtomatsko, ob spremembi vsebine dokumenta se ustvari nova verzija dokumenta, ki postane aktualna verzija dokumenta.

Na dokumentu se ohrani vsebina vseh starejših verzij. Kadarkoli lahko pogledamo vsebino dokumenta v starejših verzijah. Zmeraj tudi lahko na dokumentu izvedemo akcijo povrni (ang. revert) – vsebina dokumenta se povrne v starejšo verzijo. Pri tem se ustvari nova verzija dokumenta, ki ima enako vsebino kot starejša verzija.

Vmesnik Alfresco Share nam omogoča tudi, da pri verzioniranju označimo, gre za manjšo (ang. minor version) ali za večjo spremembo (ang. major version). Pri tem se uporablja podobno številčenje kot pri razvoju programske opreme. Na primer, če je trenutna verzija dokumenta 1.1, bi ob manjši spremembi postala nova verzija dokumenta 1.2, ob večji spremembi pa bi dobili verzijo 2.0.

2.2.6. Shramba vsebine in kasnejši dostop do te vsebine

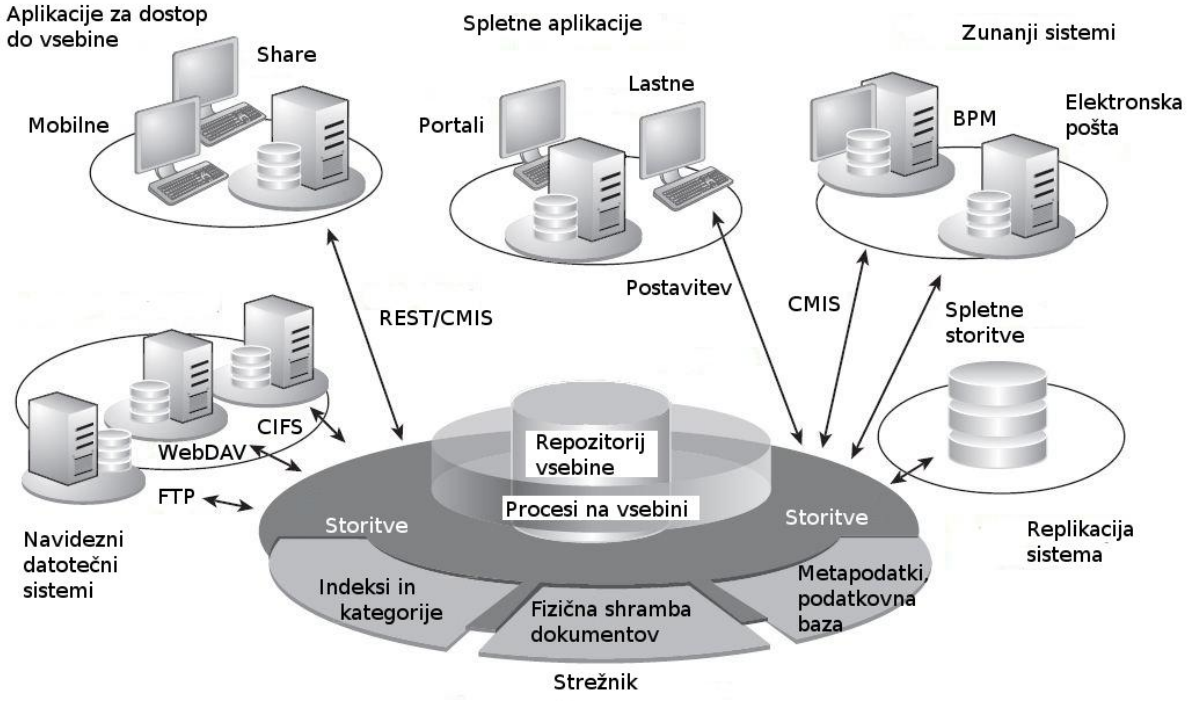
Shramba (ang. storage) obravnava, kako je vsebina shranjena v dokumentnem sistemu. Shramba vključuje upravljanje s shranjeno vsebino, kje točno, koliko časa in kako je shranjena v elektronski obliki oziroma fizično na disku. Obravnava tudi brisanje vsebine, recimo kaj se zgodi z vsebino ko jo pobrišemo in kako se pobrisana vsebina na koncu uniči.

Dokumentni sistem mora dokumente hraniti tako, da je kasnejše dostopanje do njih čim lažje in čim bolj učinkovito. Pri dostopu do vsebine (ang. retrieval) nam pomagata indeksiranje dokumentov in možnost iskanja po vsebini.

Privzeta namestitev Alfresca ima arhitekturo, ki je tipična za spletno aplikacijo(slika 1). Naš vmesnik je spletni brskalnik, z njim dostopamo do aplikacijskega strežnika (ang. application server). Znotraj aplikacijskega strežnika so Alfrescove aplikacije, ki omogočajo upravljanje z dokumenti, zapisi in spletnimi vsebinami, ter Alfrescov repozitorij. Alfrescov repozitorij omogoča uporabniku naslednje storitve [3]:

- definiranje strukture vsebine – modeliranje,
- ustvarjanje, brisanje in urejanje vsebine ter pripadajoče metapodatke in razmerja,

- nastavljanje dovoljenj na mapah,
- poizvedbe po vsebini,
- sprožitev dogodkov ob spremembah,
- zaklepanje vsebine,
- verzioniranje vsebine,
- dodajanje večjezičnosti vsebine in vmesnika,
- uvoze in izvoze vsebine,
- prikaz vsebine in
- izvajanje pravil in akcije na vsebini.



Slika 1: Podrobnejši prikaz arhitekture Alfresca.

Repozitorij v Alfrescu temelji na naslednjih dveh konstruktih: vozliščih (ang. nodes) in vsebini (ang. content).

Vozlišča omogočajo metapodatke in strukturo vsebine. Lastnosti se uporabljajo v vozliščih za določanje metapodatkov. Vozlišča so sestavljena tako, da so med sabo v določenem razmerju. Med sabo so lahko v podrejenem razmerju (starš in otrok), ali pa v enakovrednem razmerju. Strukturo vozlišč in razmerja med vozlišči opisujejo modeli vsebine (ang. content models).

Vsebina so dokumenti, ki jih uporabniki shranimo v repozitorij, kot recimo pdf ali wordove datoteke. Za lažje sledenje vsebini in lažji kasnejši dostop do nje Alfresco uporablja indeksiranje s knjižnico Apache Lucene.

Vsebina in vozlišča so shranjena ločeno. Vozlišča so shranjena v zunanji podatkovni bazi, ki je logični arhiv dokumentnega sistema. Vsebina je shranjena v fizičnem arhivu dokumentnega sistema in leži nekje na računalniku. Shranjena je znotraj mape izven Tomcat strežnika, ki poganja Alfresco, privzeto ime mape je `alf_data`. V tej mapi je vsebina shranjena znotraj mape `contentstore`. Vsebina je shranjena hierarhično v podmapah, ki so urejene glede na datum in čas vnosa. Ta vsebina je namenjena zgolj za Alfrescovo interno uporabo, ne moremo pa je pregledovati prek računalnika.

Mapa `alf_data` poleg mape `contentstore` vsebuje še mapi `lucene-indexes` in `backup-lucene-indexes`, ki hranita indekse ter mapo `contentstore.deleted`. Znotraj mape `contentstore.deleted` je shranjena izbrisana vsebina. Vsebina postane v Alfrescu izbrisana, ko ne obstaja več nobeno vozlišče, ki bi kazalo na to vsebino. Takrat se vsebina prestavi v mapo `contentstore.deleted`. Ta mapa ima enako hierarhično strukturo podmap kot `contentstore`. Vsebino mape `contentstore.deleted` pobriše Alfresco po določen času, (glede na nastavitve), lahko pa jo pobrišemo tudi sami.

2.2.7. Indeksiranje vsebine

Uporablja se za sledenje dokumentom v repozitoriju. Indeksiranje obstaja predvsem za podporo dostopu do dokumentov. Najbolj preprosta oblika indeksiranja je raba enoličnega identifikatorja za dokumente v repozitoriju. Bolj napredne oblike indeksiranja so z uporabo metapodatkov ali pa z uporabo delov dejanske vsebine dokumenta v repozitoriju.

Indeksiranje v Alfrescu temelji na brezplačni odprtokodni knjižnici Lucene [11], ki je napisana v Javi. Lucene indeksi se ustvarjajo avtomatsko ob dodajanju vsebine in se shranjujejo v fizičnem arhivu v mapi `lucene-indexes`.

Lucene omogoča indeksiranje po celotnem tekstu (ang. full text indexing), kar pomeni, da se indeksi ne ustvarjajo zgolj za metapodatke vsebine, temveč tudi po vsebini dokumentov. Zaradi tega je Lucene neodvisen od datotečnega formata, dokler le ta vsebuje tekst, ki ga lahko računalnik bere.

Alfresco v mapo `backup-lucene-indexes` periodično prepisuje Lucene indekse iz `lucene-indexes`. Zaradi tega lahko v primeru težav zaradi napak v indeksih zaženemo Alfresco v načinu `Full Index Recovery`, kjer nastavimo indekse na prejšnje stanje, kot je zapisano v mapi `backup-lucene-indexes`.

2.2.8. Iskanje po vsebini

Ko so dokumenti shranjeni v repozitoriju se pojavi potreba po tem, da jih lahko poiščemo. Četudi so varno shranjeni v repozitoriju, je lahko pot do njih dolga ali pa se je mogoče več ne spomnimo. Prav tako nas mogoče ne zanima le določen dokument, temveč skupina dokumentov, ki vsebuje določeno vsebino, določene metapodatke ali pa je določenega datotečnega formata.

V takih primerih si pomagamo z iskanjem po vsebini repozitorija. Za učinkovito iskanje je potrebno imeti možnost iskanja po imenih, specifičnih metapodatkih, ali po dejanski vsebini dokumenta.

Oba vmesnika nam omogočata iskanje tako po strukturi dokumenta (recimo imenu dokumenta), formatu dokumenta, tipu vsebine, času ustvarjanja dokumenta kot po metapodatkih dokumenta (tudi takih, ki smo jih sami definirali). To je osnovna oblika iskanja v Alfrescu, ki je namenjena za potrebe navadnega uporabnika v sistemu.

Naprednejšo oblika iskanja najdemo v Alfrescu v administrativnih orodjih in v programski kodi, ki dostopa do vsebine repozitorija. Alfresco nam v ta namen omogoča rabo dveh poizvedovalnih jezikov, Lucene[15] in XPath [15]. Oba poizvedovalna jezika delujeta na podoben način. Omogočata iskanje v obliki poizvedb (ang. query), v katerih določimo, po kateri vsebini dokumentov naj iščemo. Omogočata iskanje po tipih, lastnostih, aspektih, identifikatorjih vozlišč in dejanski vsebini dokumentov. Prav tako omogočata naprednejše oblike poizvedb z logičnimi operatorji.

2.2.9. Delovni tok

Organizacije imajo svoja opravila opisana s poslovnimi procesi. Delovni tok (ang. workflow) je izvedba poslovnega procesa v dokumentnem sistemu. Sestavljajo ga aktivnosti in odločitveni mehanizmi. Ima definiran začetek in konec ter vse prehode med posameznimi aktivnostmi. Vsebuje vse korake, ki se morajo izvesti za uspešen zaključek nekega opravila ali procesa.

Matematično bi lahko delovni tok prikazali kot usmerjen graf. Vozlišča bi v takem grafu prikazovala določene aktivnosti v delovnem toku, usmerjene povezave, možne poti med temi vozlišči oziroma prikaz, kako si aktivnosti v delovnem toku sledijo. Dokumentni sistem nadzoruje pretok dokumentov skozi vnaprej definirane delovne tokove, uporabniki pa s pomočjo vmesnika sprejemajo odločitve, ki vplivajo na pretok dokumentov po delovnem toku.

Alfresco ima dve vrsti delovnih tokov:

- preproste delovne tokove (ang. simple workflow) in
- napredne delovne tokove (ang. advanced workflow).

Preprosti delovni tok je vezan na vsebino (ang. content oriented). Določimo ga tako, da ga dodamo kot pravilo v mapo, ki se izvede, ko določena vrsta vsebine prispe v mapo. Preprosti delovni tok je zmeraj vezan na zgolj en dokument in za vsak dokument, se ustvari lasten delovni tok. Ima zgolj začetno in končno stanje, vmesnih stanj ni.

Vsebuje vsaj en korak, odobritev vsebine (ang. approve) in lahko včasih še korak, za zavrnitev vsebine (ang. reject), torej največ dva koraka. V vsakem izmed teh korakov določimo, v katero mapo se dokument premakne oziroma prepiše. Preprost delovni tok je tako v celoti vezan na strukturo map v repozitoriju. Vsebina se mora premakniti v drugo mapo, da se odraža sprememba v stanju.

Z večjim številom preprostih delovnih tokov, ki si sledijo eden za drugim, lahko simuliramo poslovni proces z več stanji. To dosežemo tako, da se po koraku dokument premakne v mapo, v kateri to povzroči zagon novega preprostega delovnega toka. Vendar smo pri tem še vedno omejeni na zgolj dva možna izhoda iz stanja, po procesu se lahko premikamo samo nazaj in naprej.

Preprosti delovni tok tudi ne pozna pojma naloge (ang. task). Vsebino lahko potrdi ali zavrne vsak, ki ima dostop do mape. Nalogo v preprostem delovnem toku lahko simuliramo le tako, da pravice za dostop do mape omejimo na zgolj določene uporabnike ali skupine. Dokument je istočasno lahko samo v enem preprostem delovnem toku. Logiko in odločitve lahko v delovni tok dodamo le s pravili na mapah. Zaradi mnogih omejitev moramo pri malce bolj zapletenih problemih uporabiti napredni delovni tok (ang. advanced workflow).

Napredni delovni tok je vezan na naloge (ang. task oriented). Naloga je stanje v procesu, v katerem mora vršilec naloge izbrati eno izmed ponujenih tranzicij, ki vodi v naslednje stanje v procesu. Število tranzicij ni omejeno, kot pri preprostem delovnem toku. Naloga je lahko vezana na zgolj enega uporabnika ali pa ima več uporabnikov.

Ker je napredni delovni tok vezan na naloge, je neodvisen od strukture map in nima omejitev glede števila dokumentov, ki nastopajo v njem. Tako se lahko zgodi, da ne vsebuje nobenega dokumenta. Prav tako dokumenti niso omejeni glede števila delovnih tokov, ki jim pripadajo. Napredni delovni tok ima zmeraj natančno eno začetno stanje in vsaj eno končno stanje ter vsaj eno tranzicijo med dvema stanjema. Uporabnik, ki je zagnal napredni delovni tok, je pobudnik delovnega toka (ang. initiator).

Slabost naprednih delovnih tokov (v primerjavi s preprostimi) je, da jih ne moremo zagnati s pomočjo v mapo prihajajočih dokumentih. To omejitev lahko obidemo tako, da nastavimo pravilo, da se ob prihodu dokumenta v mapo zažene skripta, ki zažene napredni delovni tok in prispeli dokument doda v delovni tok.

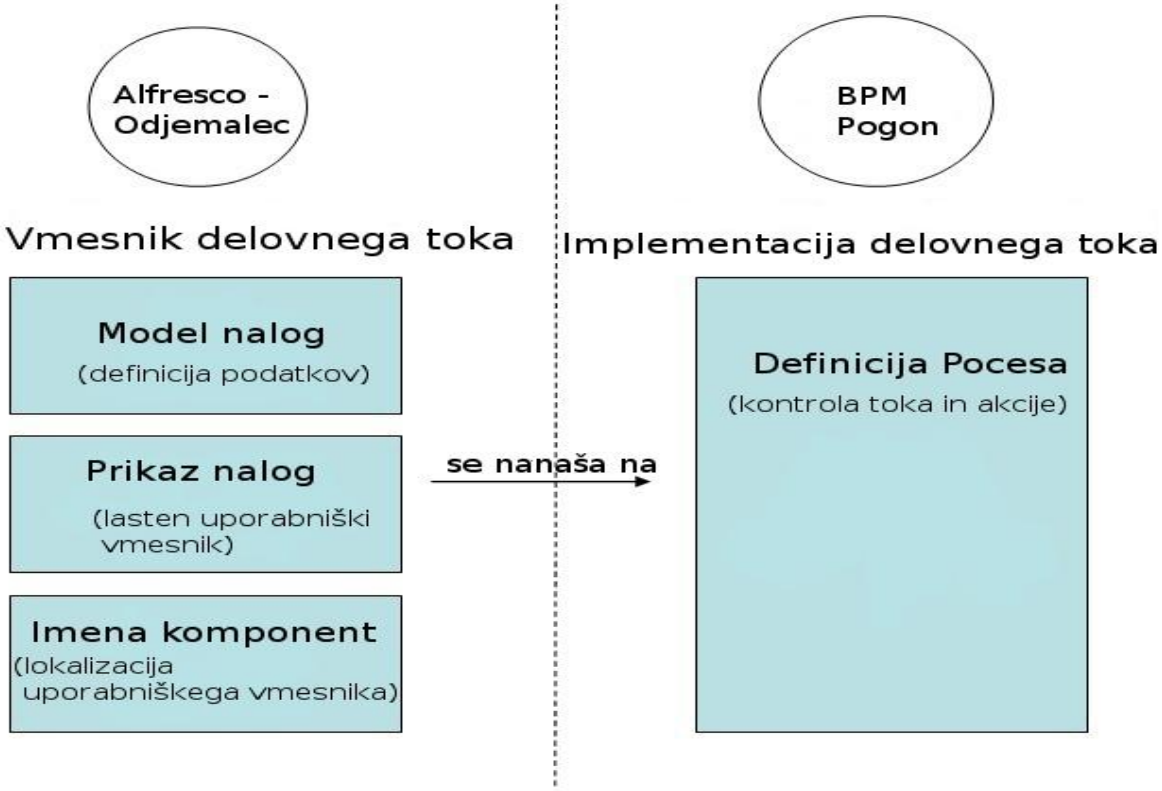
Alfresco vsebuje dva že pripravljena napredna delovna tokova:

- Adhoc Task – delovni tok, v katerem pobudnik delovnega toka določi nekemu drugemu uporabniku naj opravi neko nalogo in
- Review & Approve – delovni tok, v katerem vršilci nalog pregledujejo dokumente, ki jih je pobudnik delovnega toka dodal vanj in jih potem odobrijo ali zavrnejo.

Če pa je naš problem bolj zapleten, lahko ustvarimo lasten napredni delovni tok (ang. custom advanced workflow). Na sliki 2 vidimo lahko grafični prikaz komponent delovnega toka.

Napredni delovni tok sestavljajo naslednje komponente:

- definicija procesa (ang. process definition),
- model nalog (ang. task model),
- konfiguracija prikaza nalog in
- imena komponent v delovnem toku (ang. resource bundle).



Slika 2: Grafični prikaz komponent naprednega delovnega toka v Alfrescu

Definicija procesa je datoteka, ki opisuje stanja in tranzicije v delovnem toku. Tranzicije so prehodi med stanji. Izvede jih lahko sistem ali pa uporabnik, ko se v nalogi odloči za eno izmed ponujenih tranzicij.

Procesi v Alfrescu so definirani z jezikom JDPL (Java Process Definition Language) [7], to je oblika zapisa v jeziku XML. JPDL je jezik, ki temelji na matematičnem konceptu grafov (ang. graph-based execution language). Alfresco lahko ta standard uporablja, ker ima integriran pogon za delovne tokove JBPM (JBoss Business Process Management). JBPM je odprtokoden, napisan v Javi ter omogoča izvajanje poslovnih procesov, ki so definirani z JPDL.

Najpomembnejša stanja v procesu oziroma vozlišča v grafu (ang. nodes) so:

- začetno in končno stanje v procesu (ang. start-state in end-state)
- vozlišče za definicijo naloge (ang. task-node), ki definira nalogo za enega ali več uporabnikov v dokumentnem sistemu,
- vozlišče za določitev (ang. decision), v kateri se sistem sam odloči, katero izmed ponujenih tranzicij bo izbral glede na definiran pogoj in
- vejitev (ang. fork) in združitev (ang. join), ki nam omogoča uporabo vzporednih nalog (ang. paralel tasks). To so naloge, ki jih vršilci nalog istočasno opravljajo neodvisno drug od drugega.

V delovni tok lahko na določenih mestih vključimo Java ali Javascript kodo. Ta mesta so pred vstopom in po izstopu iz določenega stanja ter v prehodih med stanji. Vključitev kode v proces nam omogoča programsko spreminjanje vsebine v nalogah, kot tudi spremembe v repozitoriju (recimo ustvarjanje map, spreminjanje pravic na mapah, premikanje dokumentov po repozitoriju in ustvarjanje nove vsebine v repozitoriju).

Model nalog je XML datoteka, ki se uporablja za definicijo nalog, ki jih morajo v delovnem toku izvesti uporabniki. Vsaka naloga je sestavljena iz imena, opisa in informacij, ki so pripete k nalogi (lastnosti, asociacije in aspekti). Sistem in uporabnik tem informacijam določata vsebino in glede na njihovo vsebino sprejemata odločitve v delovnem toku.

V modelu naloge ne definiramo vršilca nalog, to storimo v procesu. Zaradi tega lahko isto nalogo večkrat definiramo v procesu in jo lahko uporabimo v več različnih procesih. Modeli se v Alfrescu poleg določanja informacij o nalogah uporabljajo tudi za določanje metapodatkov vsebine, ki je shranjena v repozitoriju. Pri tem definiramo model na skoraj enak način, kot če bi želeli z njim opisati nalogo.

S konfiguracijo prikaza nalog v brskalniku določimo, kako se bo obrazec z nalogo (ang. form) prikazal končnemu uporabniku. S konfiguracijo prikaza obrazca z nalogo poskrbimo za uporabniku prijazen vizualni izgled, konfiguracija obrazca pa je pomembna tudi iz praktičnih razlogov.

Prvi pomemben razlog je ta, da določimo, katere izmed lastnosti naloge sploh želimo prikazati. Nekatere informacije so za uporabnika nepomembne ali pa jih sistem uporablja le za sprejemanje odločitev v delovnem toku in sploh niso namenjene končnemu uporabniku.

Drugi pomemben razlog je ta, da s tem določimo pravice in obveznosti, ki jih ima uporabnik na vsebini v obrazcu. Vsebino polja v obrazcu lahko spreminja ali pa je vsebina namenjena zgolj za branje (ang. read-only). Če je polje obvezno (ang. mandatory), ga je vedno potrebno izpolniti, v nasprotnem primeru pa ga lahko uporabnik pusti praznega.

Imena komponent v delovnem toku niso obvezna za delovanje delovnega toka, vendar pomembna za prijaznejši in razumljivejši prikaz končnemu uporabniku. Vsaka naloga in vsaka informacija v nalogi je zapisana v obliki predpona:identifikator. Ko v posebnih datotekah s končnico `.properties` določimo neko vrednost v enakem zapisu, vendar dvopičje zamenjamo s podčrtajem, spremenimo imena nalog in polj v nalogah v bolj berljivo obliko. Z uporabo istih datotek tudi določamo imena komponent v vmesniku Alfresca in omogočamo večjezičnost vmesnika.

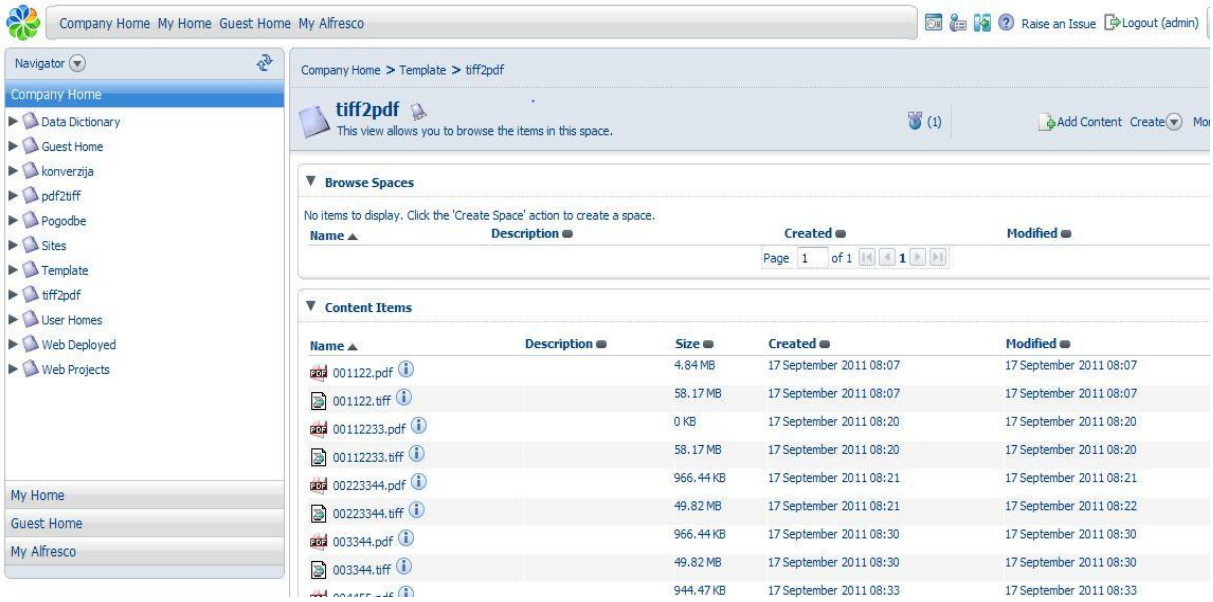
3. KONFIGURACIJA OKOLJA ALFRESCO

3.1. Primerjava vmesnikov Alfresco Explorer in Alfresco Share

Alfresco ponuja uporabnikom dva vmesnika. Prvi je Alfresco Explorer [1], ki je izvirni vmesnik in obstaja že od prve različice Alfresca. Drugi je Alfresco Share [1], ki je novejši, saj obstaja šele od različice 3.2; trenutna različica Alfresca je 3.4.d. Alfresco Share je uporabnikom bolj prijazen in privlačen vmesnik, v njem pa je tudi večji poudarek na skupinskem delu in sodelovanju med uporabniki.

Oba vmesnika dostopata do istega repozitorija, z istimi uporabniki in gesli. Spremembe, ki smo jih naredili preko enega vmesnika, bodo takoj vidne tudi v drugem vmesniku. Imata ločeni avtentikaciji, kar pomeni, da se moramo prijaviti v vsakega posebej. Zaradi tega lahko znotraj istega brskalnika uporabljamo oba vmesnika, lahko tudi z različnima uporabnikoma v sistemu.

Izgled obeh vmesnikov moramo posebej konfigurirati in spreminjati. Vmesnika se zelo razlikujeta po tem, kako sta sestavljena. Alfresco Share uporablja veliko več Javascripta in CSS, uporablja pa tudi FreeMarker jezik za predloge (ang. Template language) [1], katerega Alfresco Explorer sploh ne uporablja. Alfresco Explorer pa za razliko od Alfresco Share uporablja veliko število JSP strani; skoraj vsaka akcija v vmesniku ima lastno JSP stran. Spreminjanje izgleda vmesnika je veliko lažje v Alfresco Share, prav tako imamo tukaj več možnosti za spreminjanje izgleda.

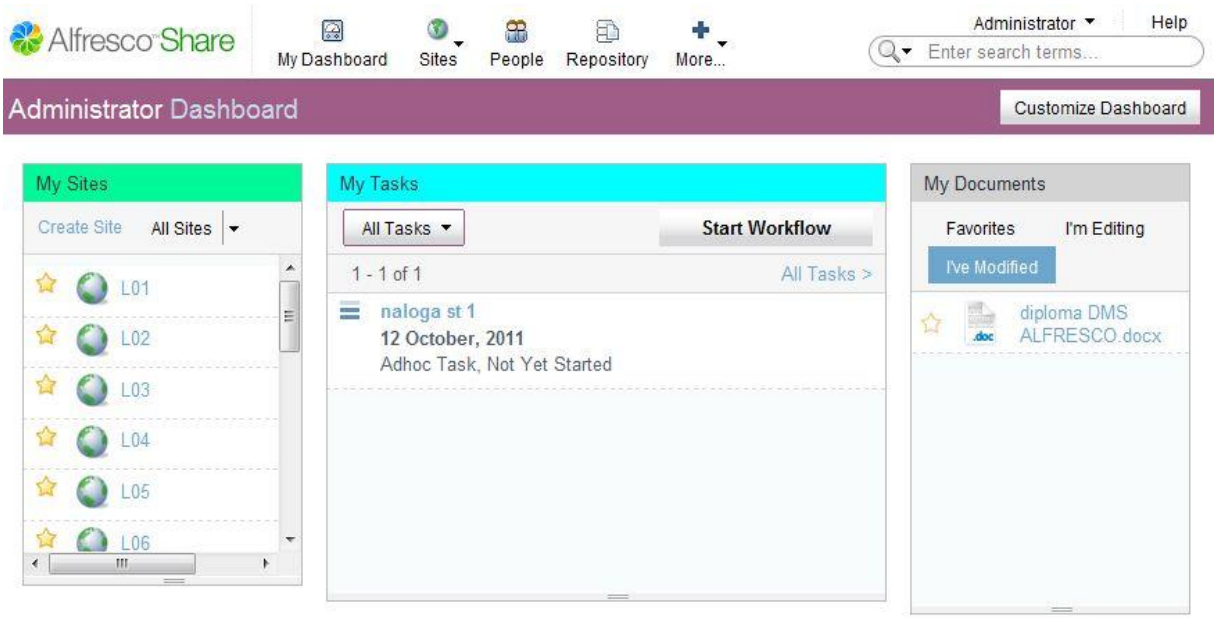


Slika 3: Pogled na repozitorij v vmesniku Alfresco Explorer.

Alfresco Explorer ima v primerjavi z Alfresco Share veliko bolj statičen in vizualno manj privlačen izgled. Uporabniku daje bolj vtis namizne aplikacije kot pa spletne aplikacije. Zelo dobra lastnost vmesnika Alfresco Explorer je, da imamo na levi strani pogled na hierarhično strukturo repozitorija, ki ga po potrebi lahko tudi skrijemo (slika 3).

Vstopna stran Alfresco Shrare – My Dashboard je veliko bolj dinamična. Sestavljena je iz večjega števila polj, ki omogočajo dostop do določenih funkcionalnosti (ang. dashlet). Primeri nekaterih izmed teh polj so:

- My Tasks – seznam nalog uporabnika,
- My Documents – seznam dokumentov, ki jih je uporabnik označil kot priljubljene, ki jih je spreminjal ali pa jih v tistem trenutku ureja in
- My Sites – moja spletišča.

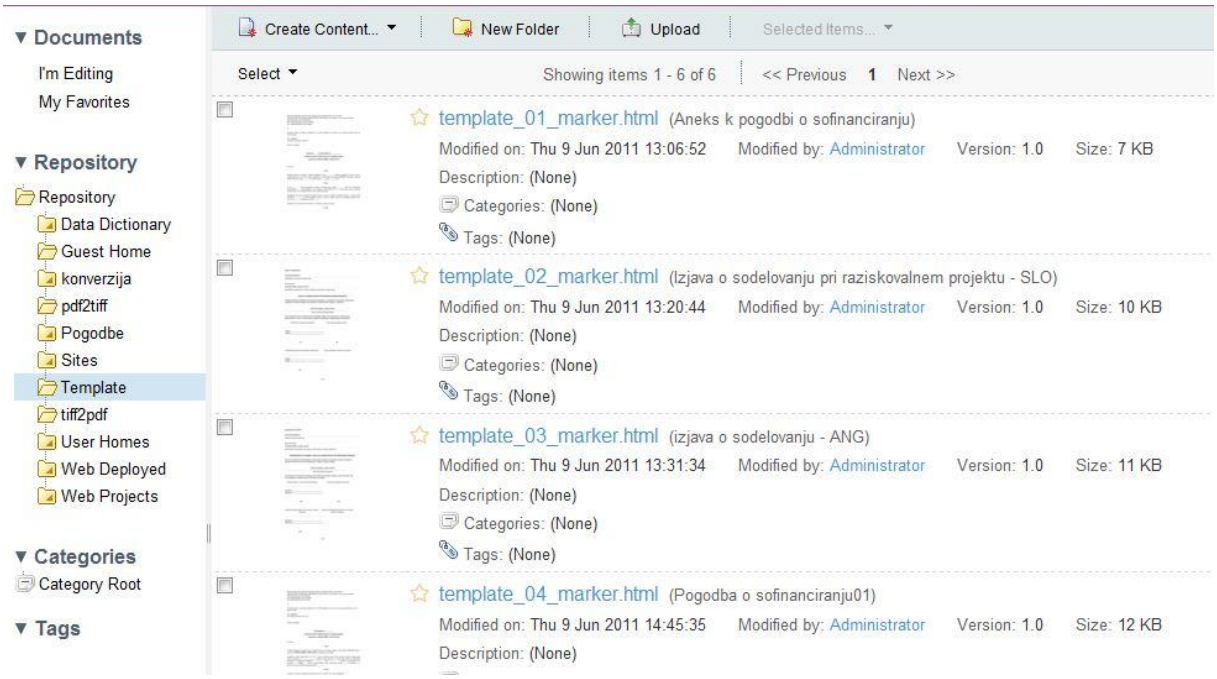


Slika 4: Vstopna stran s polji v vmesniku Alfresco Share.

Posebnost vstopne strani je, da lahko vsak uporabnik sam izbere polja, ki se mu bodo prikazala in kako naj bodo razporejena (slika 4). Slabost vmesnika Alfresco Share pa je, ker nimamo pogleda v repozitorij (slika 5); za dostop do repozitorija moramo celo zapustiti vstopno stran.

V vmesniku Alfresco Explorer moramo obvezno zagnati napredni delovni tok na nekem že obstoječem dokumentu. Zaradi polja My Tasks Alfresco Share nima te omejitve, napredni delovni tok pa lahko zaženemo tako na dokumentu v repozitoriju kot tudi na polju My Task. V drugem primeru gre za delovni tok brez dokumentov. Zaradi tega razloga smo delovni tok za potrjevanje pogodb razvijali zgolj za delo v Alfresco Share vmesniku.

Alfresco Explorer ima tri različne poglede vsebine repozitorija, medtem ko ima Alfresco Share zgolj en pogled. Izmed treh pogledov Alfresco Explorerja je najpomembnejši `Details View` (slika 3). Ta pogled predstavi metapodatke dokumentov v vzporednih kolonah, omogoča pa nam tudi, naraščajoče ali padajoče sortiranje dokumentov po poljubni koloni.

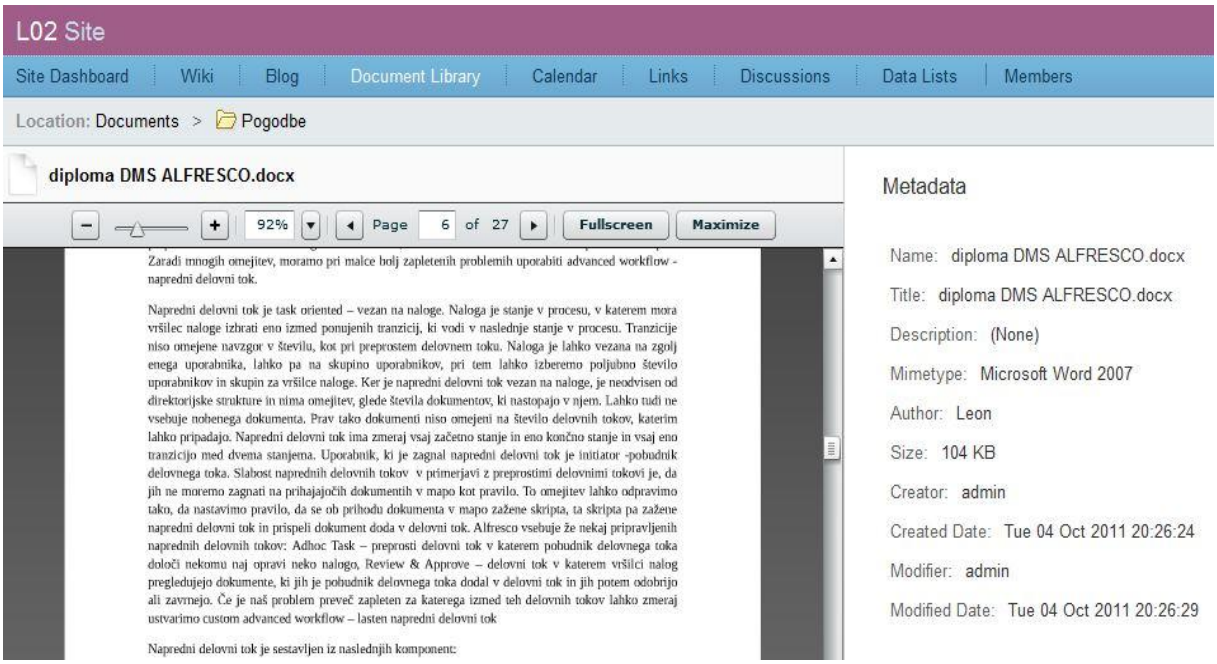


Slika 5: Pogled na repozitorij v vmesniku Alfresco Share.

Velika prednost vmesnika Alfresco Share pred Alfresco Explorerjem pri pregledovanju vsebine repozitorija je njegov predogled dokumentov (ang. previewer), vidimo ga na sliki 6. Ko izberemo nek dokument v repozitoriju, lahko poleg metapodatkov dokumenta vidimo tudi predogled vsebine dokumenta na levi strani. Če bi želeli videti vsebino dokumenta v vmesniku Alfresco Explorer, bi morali dokument obvezno naložiti na računalnik.

Če smo prijavljeni v vmesnika kot administrator, imamo dostop tudi do administrativnih orodij. Pri teh se boljše izkaže vmesnik Alfresco Explorer, saj so orodja v tem primeru bolj pregledna in ponujajo več funkcionalnosti (slika 7).

Alfresco Share omogoča veliko večjo podporo sodelovanju in skupinskem delu z uporabo spletišč. Vendar pa je veliko teh naprednejših funkcij, predvsem različni načini komunikacije med člani spletišč, za potrebe našega delovnega toka nepomembnih. Kljub temu pa so delovne mape iz delovnega toka za potrjevanje pogodb znotraj spletišč, za vsak oddelek namreč obstaja lastno spletišče.



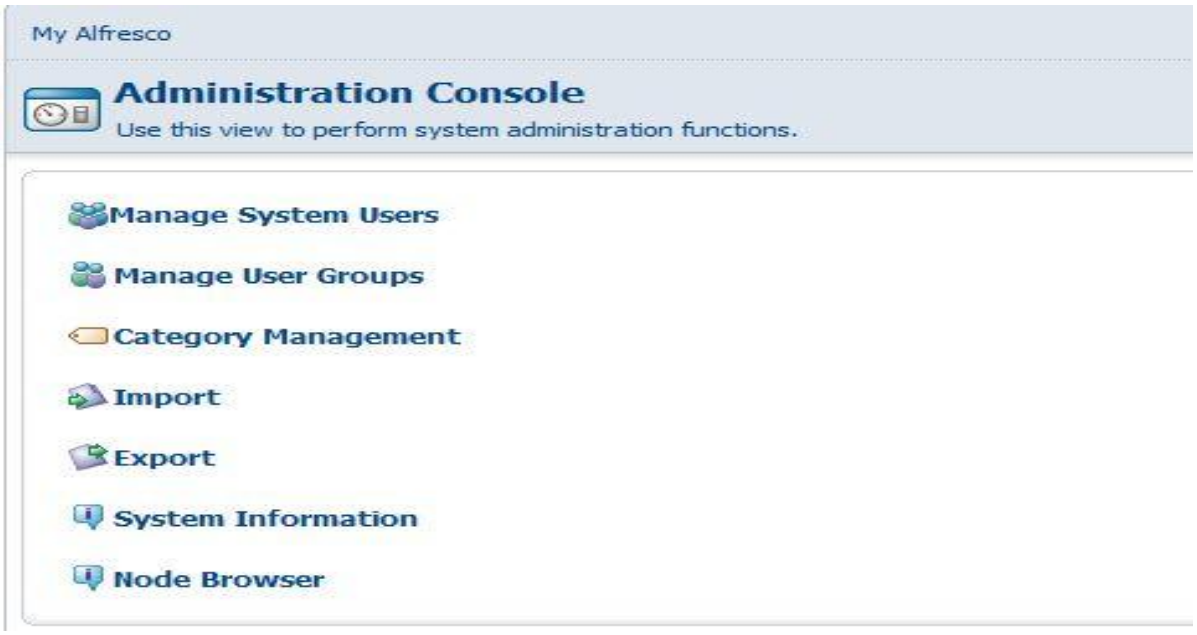
Slika 6: Predogled dokumenta v vmesniku Alfresco Share.

Obstajajo tri vrste spletišč glede na to, kako lahko uporabniki postanejo njihovi člani [1]. Vsem pa je skupno to, da na njih lahko delajo in jih pregledujejo zgolj člani spletišč. Obstajajo:

- javna spletišča (ang. public sites), kjer se vsak uporabnik, ki ni član spletišča, lahko po želji pridruži oziroma zapusti spletišče,
- zasebna spletišča (ang . private sites), ki se jim uporabniki lahko pridružijo le, če jih doda v spletišče skrbnik spletišča (ang. site manager). Skrbnik spletišča je tisti uporabnik, ki je ustvaril spletišče in ima možnost dodajanja novih skrbnikov in
- upravljana spletišča (ang. moderated sites), ki so nekje vmes med javnimi in zasebnimi spletišči. Vsak uporabnik lahko poda zahtevo o pridružitvi k spletišču, to zahtevo pa skrbnik spletišča potrdi ali zavrne.

Spletišča so znotraj pogleda v repozitoriju vidna zgolj kot en skupek map. Funkcionalnost spletišč pride do izraza, ko dostopamo do njih prek vmesnika Alfresco Share v posebnem pogledu, ki je namenjenem zgolj za vsebino spletišč. Prav tako je zelo priporočljivo, da se vsebina znotraj spletišč spreminja le znotraj tega, spletiščem namenjenega pogleda.

Zaključek primerjave je, da je vmesnik Alfresco Share veliko primernejši za navadnega uporabnika, ki bo zgolj uporabljal delovni tok. Vmesnik Alfresco Explorer po drugi strani je bolj primeren za administratorje zaradi boljših administrativnih orodij. Kadar želimo podrobneje pregledovati vsebino repozitorija pa je najboljše, da uporabljamo kar oba. Alfresco Explorer ima boljši pregled po vsebini repozitorija, Alfresco Share pa je zelo koristen zaradi svojega predogleda vsebine dokumentov v repozitoriju.



Slika 7: Administrativna orodja v vmesniku Alfresco Explorer.

3.2. Namestitev okolja Alfresco

Alfresco je spletna aplikacija, ki je napisana v Javi, JSP in Javascriptu. Ker je odprtokodna aplikacija imamo dostop do celotne programske kode in jo lahko po želji poljubno spreminjamo. Na voljo imamo brezplačno različico (ang. community edition) in komercialno različico (ang. enterprise edition), s katero dobimo še podporo razvijalcev.

Zaradi uporabe programskega jezika Java potrebujemo nameščeno javansko izvajalno okolje (ang. Java Runtime Environment – JRE) ali razvojno okolje (ang. Java Development Kit – JDK). Ker je Alfresco spletna aplikacija, ni odvisna od izbire operacijskega sistema, vendar za svoje delovanje nujno potrebuje strežnik in zunanjo podatkovno bazo.

Za lažjo namestitev Alfresca imamo na voljo namestitveni program, ki poleg Alfresca, namesti še strežnik Apache Tomcat[1] in sistem za upravljanje s podatkovnimi bazami MySq[1]. Namestitveni program tudi poveže te komponente tako, da lahko Alfresco zaženemo brez dodatnega nastavljanja.

V primeru, da želimo Alfresco namestiti na obstoječi strežnik, ki že ima sistem za upravljanje s podatkovnimi bazami, uporabimo `.war` datoteko (ang. Web Application Archive. To je skupek kode Alfresca, ki jo potem strežnik razpakira in tako dobimo spletno aplikacijo. V tem primeru lahko uporabimo tudi kateri drug strežnik, ki je sposoben procesirati JSP kodo, recimo JBoss [3]. Prav tako lahko namesto MySql uporabimo recimo PostgreSQL, Microsoft SQL Server, Oracle ali kateri drugi sistem za upravljanje s podatkovnimi bazami.

Namestitveni program poleg tega namesti še dodatne komponente, ki sicer niso obvezne za delovanje Alfresca, vendar pa omogočajo ključne funkcionalnosti. To so Open Office [1], ImageMagick [1] in swftools [3].

Open Office je brezplačna in odprtokodna alternativa pisarniškemu paketu Microsoft Office. Alfresco uporablja OpenOffice za pretvorbo dokumentov s tekstom v repozitoriju iz enega formata v drugega, recimo iz `.txt` v `.pdf`. Za pretvorbo slik v repozitoriju uporabljamo ImageMagick. ImageMagick je skupaj s SWF Tools tudi ključen pri predogledu dokumentov.

ImageMagick je odprtokodni programski paket, ki se uporablja za prikazovanje, pretvorbo in urejanje rastrskih slik. Alfrescu omogoča rokovanje s sliko za predogled. SWF Tools je odprtokodni programski paket, ki vsebuje orodja za ustvarjanje in manipulacijo `.swf` datotek (format datotek okolja Adobe Flash). Alfresco uporablja orodje pdf2swf [3] znotraj SWF Tools za predogled PDF datotek, PDF datoteko pretvori v format, ki ga lahko Adobe Flash bere. Predogled datotek v Alfrescu deluje, če ima uporabnik v svojem brskalniku še pravilno nameščen Flash Player.

3.3. Opis poteka delovnega toka potrjevanja pogodb

V tem delovnem toku vodje oddelkov in oba vodja skupnih služb potrjujejo pogodbo, ki jo je pobudnik delovnega toka poslal v potrjevanje. Pogodba v delovnem toku lahko že obstaja v repozitoriju ali pa jo ustvarimo iz predloge (ang. template). Ker lahko v Alfrescu delovni tok zaženemo na obstoječem dokumentu ali brez dokumenta, velja naslednje pravilo: če delovni tok zaženemo na dokumentu, ki je že v repozitoriju, to šteje kot obstoječa pogodba, v nasprotnem primeru pa se pogodba ustvari iz predloge. Tako uporabniku ni potrebno izbirati med vrsto pogodbe.

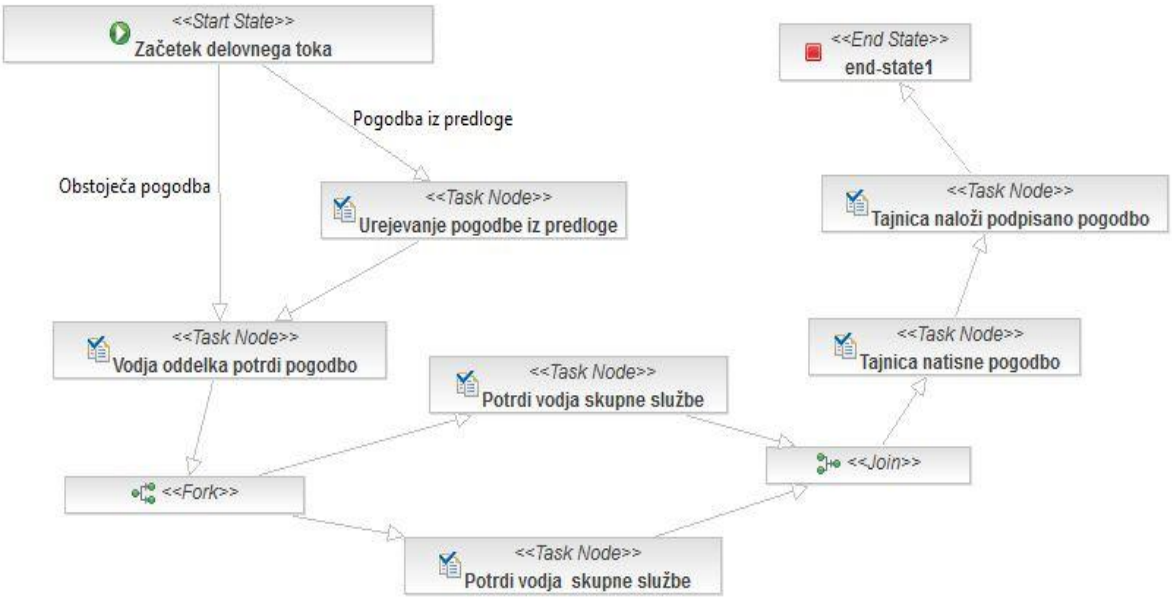
V obeh primerih je prva naloga enaka: uporabnik mora vpisati metapodatke o dobaviteljih. Vpis podatkov je še posebej pomemben pri pogodbi iz predloge, saj se ti podatki vpišejo v novo nastalo pogodbo.

V primeru pogodbe iz predloge sledi dodatna naloga za pobudnika. Novo nastalo pogodbo mora pregledati in jo po potrebi ustrezno dopolniti. Potem gre pogodba v potrjevanje, pri čemer je delovni tok je enak za obe vrsti pogodb.

Pogodbe potrjujejo trije zaposleni in sicer vodja oddelka pobudnika in vodje obeh skupnih služb. Ob zavrnitvi katerega izmed treh dobi predlagatelj nalogo, da pogodbo popravi in jo ponovno pošlje v potrjevanje. Prvi potrjuje vodja oddelka. Če on potrdi, sta naslednja na vrsti za potrjevanje oba vodja skupnih služb. Tukaj nastopi vzporedna naloga. Oba vodja skupnih služb dobita ločeni nalogi, pri katerih morata pogodbo potrditi ali zavrniti. Ko oba zaključita nalogo, se delovni tok nadaljuje.

V primeru potrditve je na vrsti še tajnica, ki ima še dve zaporedni nalogi v delovnem toku. V prvi nalogi mora pogodbo natisniti in jo poslati v podpis direktorju. Ker se vse to izvede ročno, ta naloga v sistemu izgleda tako, da tajnica zgolj potrdi, da je nalogo opravila.

Potem sledi še zadnja naloga, v kateri tajnica doda v delovni tok še dokument s podpisano pogodbo. Delovni tok se tako zaključi, v repozitoriju se shranita tako delovna kot podpisana različica pogodbe. Na sliki 8 lahko vidimo še grafični prikaz poteka delovnega toka.



Slika 8: Diagram poteka delovnega toka.

3.3.1. Razporeditev uporabniških skupin za delovni tok

Za vsak oddelek obstaja lastna uporabniška skupina, v kateri so vsi člani tega oddelka. Poleg teh uporabniških skupin obstajajo še skupine, za vodje oddelkov, ki imajo v svojem imenu besedo `Management`.

Za vsak oddelek obstaja lastna skupina `Management`. V skupinah `Management` ni nobenih članov, ima pa vsaka dve podskupini. Ena izmed teh podskupin je `vodja oddelka`, druga pa `vodja oddelka limited`. Obe podskupini dobita isto nalogo med potekom delovnega toka, saj se naloga dodeli članom skupine `Managament`. Podskupina namreč vedno podeduje pravice in naloge starševskih skupin.

Skupini `vodja oddelka` in `vodja oddelka limited` se razlikujeta v pravicah, ki jih imata na nekaterih mapah v repozitoriju. Dejanski vodja oddelka je zmeraj v skupini, ki ga določa kot vodjo oddelka, skupina `vodja oddelka limited` pa je večino časa prazna. Ko nekdo nadomešča vodjo oddelka, dodamo to osebo v mapo `vodja oddelka` ali `vodja oddelka limited` glede na to, če želimo, da ta oseba zgolj opravlja naloge vodje ali pa, da ima tudi dostop do vseh map kot vodja oddelka. Dejanski vodja oddelka tudi v času nadomeščanja ostane v mapi, ki ga določa kot vodjo oddelka.

Poleg teh skupin sta za delovni tok pomembni še skupini `tajništvo` in skupina z namestniki vodij skupnih služb; to so uporabniki, katerim lahko vodje skupnih služb delegirajo svojo nalogo, da jo opravijo namesto njih.

3.3.2. Pravice uporabnikov in skupin na mapah v repozitoriju

Na vsaki mapi v repozitoriju Alfresca lahko določimo pravice, tako za skupine kot za posamezne uporabnike. Na voljo imamo pet različnih vlog, ki določajo obseg pravic uporabnika na mapi [1].

- Consumer je najnižja vloga, z njo lahko uporabnik zgolj vidi mapo in vsebino mape.
- Editor ima poleg pravic vloge Consumer še pravice, ki mu omogočajo urejanje vsebine mape.
- Contributor ima poleg pravic vloge Consumer še pravice, za dodajanje nove vsebine v mapo.
- Collaborator združuje pravice vlog Editor in Contributor.
- Coordinator je najvišja možna vloga. Z njo imamo poleg pravic vloge Collaborator še pravice, ki nam dovoljujejo brisanje vsebine v mapi. S tem imamo v tej mapi identične pravice kot administrator Alfresca.

V primeru, ko nobena izmed teh pet vlog ne zadošča našim potrebam, lahko ustvarimo lastno vlogo. Pri mapah lahko določimo tudi ali mapa podeduje pravice starševske mape. Pravice lahko na identičen način določimo tudi na posameznih datotekah v mapi. V Alfrescu je privzeto, da imajo vsi uporabniki (z izjemo administratorja) v vseh mapah vlogo Consumer, le v svoji domači mapi (ang. User Home) imajo vlogo Coordinator.

3.3.3. Razporeditev map za delovni tok

Vsak oddelek je predstavljen s svojim spletiščem. Dostop do spletišča imajo člani in vodja tega oddelka, vodji obeh skupnih služb, njihovi namestniki in tajništvo. Vsi imajo na spletišču vlogo Collaborator.

Med izvajanjem delovnega toka se v spletišču v mapi `Pogodbe` ustvari začasna mapa, kjer bodo shranjene pogodbe za ta delovni tok. Za vsakega pobudnika delovnega toka se ustvari različna mapa, ki ima ime enako uporabniškemu imenu pobudnika. To mapo lahko vidi zgolj pobudnik delovnega toka, ostali člani istega oddelka do nje nimajo dostopa. Do mape imajo dostop tudi člani vseh ostalih skupin v spletišču, torej vodja oddelka, tajnica, vodji skupnih služb in njuni namestniki. V primeru, da vodja oddelka pobudnika delegira svojo nalogo nekomu drugemu v oddelku, dobi tudi ta dostop do te mape. Po koncu delovnega toka se ta začasna mapa izbriše, razen v primeru, če uporabnik izvaja še nek drug delovni tok, kar pa se v praksi zelo redko zgodi.

Ob koncu delovnega toka se pogodba v mapi prenese v posebno mapo, ki je v repozitoriju izven spletišč. V tej mapi so pogodbe razvrščene v podmapah glede na oddelek. Dostop do posamezne podmape imajo tega oddelka, vodji skupnih služb in tajništvo. Vsi imajo zgolj

vlogo Consumer, saj se pogodb po koncu delovnega toka(potem ko so podpisane), ne sme več spreminjati.

3.3.4. Dodatne zahteve naročnika

Poleg samo izdelave delovnega toka, smo od naročnika prejeli še naslednje zahteve, katere smo morali implementirati:

- Prevod obeh vmesnikov v slovenščino.
- Podpora vstopni pisarni za dostop s skenerjem
- Olajšanje postopka urejanja pogodb v Wordu
- Dostop s protokolom LDAP
- Integracija OCR pogona v Alfresco
- Metapodatki vezani samo na pogodbe
- Poseben prikaz metapodatkov za pogodbe
- Možnost pisanja komentarjev v vsaki nalogi
- Pošiljanje elektronskim sporočil uporabnikom, ko so na vrsti za neko nalogo

3.4. Izdelava prevoda za Alfresco

Trenutno je na voljo zgolj prevod za vmesnik Alfresco Explorer, pa še ta le za starejšo verzijo. Vendar pa to ni težava, saj je izdelava prevoda oziroma omogočanje večjezičnosti v Alfrescu preprosto. To nam omogočajo tekstovne datoteke z končnico `.properties`. V teh datotekah lahko za vsako komponento v vmesnikih Alfresco Explorer in Alfresco Share določimo njihov zapis v vmesniku, isto lahko naredimo tudi za vsebino modelov, za tipe, aspekte in lastnosti.

Pomembno je, da med koncem imena datoteke in končnico dodamo oznako jezika (ang. locale). To je zapis, ki nam določi želeno obliko jezika v vmesniku uporabnika. Zapišemo jo v naslednji obliki `cs_CZ` (primer za češčino). Prvi dve črki se obvezno napišeta z malimi črkami in določita jezik (v tem primeru češčina) zadnji dve, ki se pišeta z velikimi črkami pa definirata državo, v tem primeru Republika Češka. Oznaka jezika za Slovenščino je `sl_SI`.

V mapo `tomcat/shared/classes/alfresco/messages` shranimo prevod za Alfresco Explorer in v mapo `tomcat/shared/classes/alfresco/web-extension` prevod za Alfresco Share.

Oba vmesnika se razlikujeta tudi po tem, kako izberemo določen jezik. Pri vmesniku Alfresco Share je jezik odvisen od privzete oznake jezika v brskalniku. Pri vmesniku Alfresco Explorer pa jezik določimo ob prijavi z izbiro jezika v spustnem meniju. Jezike, ki se pojavijo v spustnem meniju določimo v datoteki `web-client-config-custom.xml`.

3.5. Podpora vhodni pisarni z uporabo WebDav

Naloga vhodne pisarne je skeniranje fizičnih dokumentov, ki bodo shranjeni v dokumentnem sistemu. Za učinkovito delo vhodne pisarne se dokumente takoj po skeniranju shrani v Alfrescov repozitorij. Zato moramo uporabiti enega izmed protokolov za dostopanje do vsebine v repozitoriju.

Na voljo imamo protokole:

- CIFS,
- WebDAV in
- FTP.

FTP omogoča zgolj prenos vsebine v repozitorij, medtem ko ostala dva, poleg prenosa vsebine, omogočata tudi urejanje vsebine v repozitoriju. CIFS nam omogoča več funkcionalnosti kot WebDAV, vendar je tudi bolj zahteven za nastavitev.

Pri izbiri je ključno predvsem, katere protokole skener sploh podpira. Ker skener naročnika podpira FTP in WebDAV, smo se na koncu odločili za uporabo WebDAV. Pri uporabi protokola WebDAV ni potrebna nobena dodatna konfiguracija, pa tudi sama raba je zelo preprosta. Potrebno je, da zgolj vpišemo v skenerju URL naslov v naslednji obliki:

`http://ime_strežnika:port_stežnika/alfresco/webdav`,

nato pa še pot do naše mape, na primer, če bi želeli v mapi `User Homes` dostopati do mape `uporabnik`, bi bil url naslov

`http://ime_strežnika:port_stežnika/alfresco/webdav/User Homes/uporabnik`.

Poleg tega se moramo še prijaviti s pravilnim uporabniškim imenom in geslom, in imeti kot prijavljeni uporabnik imeti v mapi zadostne pravice za dodajanje vsebine.

3.6. Urejanje pogodb med delovnim tokov v Word datoteki

Pri delovnem toku za pogodbe je ključna funkcionalnost urejanja pogodb v `.doc` datoteki. Potem ko se iz predloge ustvari pogodba, jo je potrebno vsaj enkrat dopolniti. V Alfrescu bi za urejanje obstoječega dokumenta v tem formatu morali ta dokument najprej prenesti na računalnik, ga popraviti lokalno, nato pa ga, naložiti kot novo verzijo tega dokumenta. S takim pristopom se nam ohranjajo verzije dokumenta, vendar je tak pristop za delovni tok preveč zamuden.

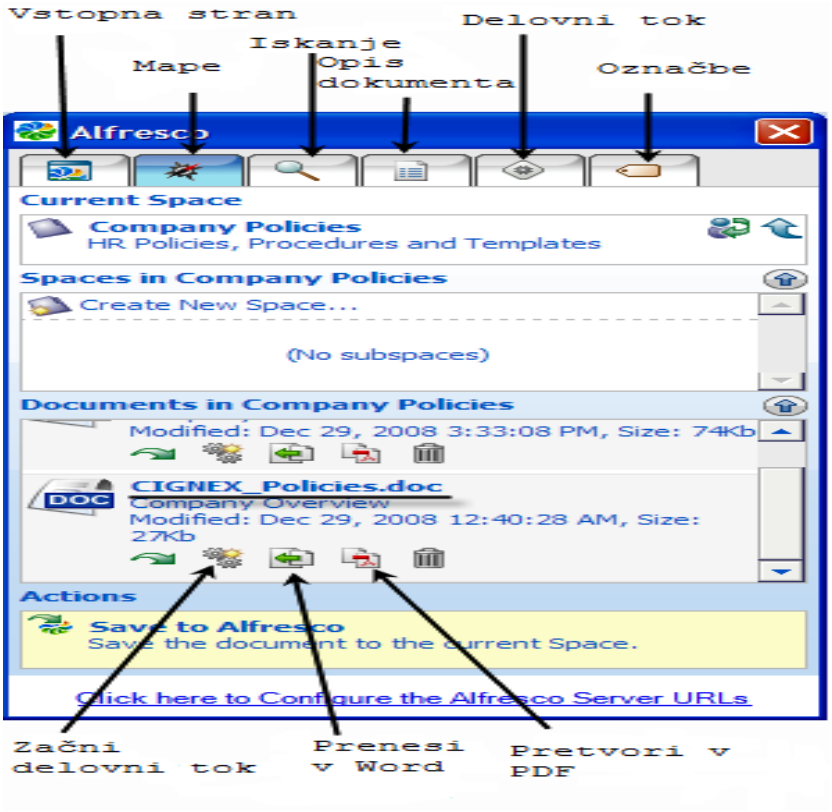
V Alfrescu lahko znotraj repozitorija urejamo samo preproste dokumente (HTML in XML). Da bi to težavo odpravili, bi lahko uporabili eno izmed naslednjih rešitev.

3.6.1. Microsoft Office dodatek

Ta dodatek nam omogoča, da med delom s programi Microsoft Office dostopamo do Alfrescovega repozitorija. Namestitev dodatka je preprosta, preprosta je tudi konfiguracija, saj moramo nastaviti zgolj lokacijo Alfresco strežnika in uporabniško ime ter geslo za dostop. Dodatek je treba namestiti na vsakem odjemalcu posebej.

Ta dodatek nam omogoča, da urejeni dokument, shranimo neposredno v Alfrescov repozitorij. Tudi dokumente, ki so trenutno v Alfrescovem repozitoriju, zaklenemo in odpremo za urejanje (ang. check-out), po urejanju pa jih shranimo nazaj v repozitorij [1]. Prikaz ostalih funkcionalnosti lahko vidimo na sliki 9.

Dodatek je koristen pri shranjevanju novega dokumenta v repozitorij, vendar se je pri urejanju dokumenta pokazal kot preveč neroden za uporabo. V ta namen obstaja boljša rešitev: namestitev Sharepoint protokola v Alfresco.



Slika 9: Podrobnejši prikaz funkcionalnosti, ki jih Microsoft Office dodatek omogoča.

3.6.2. Sharepoint protocol v Alfrescu

Alfresco ima od različice Labs 3a SharePoint protokol, ki omogoča Microsoft Office Suite aplikacijam (Word, PowerPoint, Excel, itd.), možnost sprotnega urejanja z Alfrescom, na enak način kot bi s SharePointom. V praksi nam to omogoča, da imamo tudi za Office dokumente akcijo na voljo `Edit Online`. Ta funkcija je sicer na voljo zgolj za vmesnik

Alfresco Share in to le za dokumente znotraj spletišč. To pa v našem primeru ni problem, saj so delovne verzije pogodb zmeraj v spletiščih.

Sharepoint protokol je že del namestitvenega programa, ki vzpostavi Alfresco z vsemi potrebnimi komponentami, vendar ga je za uporabo potrebno še posebej namestiti. Dobimo ga v obliki AMP datotek (Alfresco Module Package). To so skupek različnih datotek, ki dodajo v Alfresco nove funkcionalnosti. Sharepoint protokol se nahaja v modulu `vti-module`. Za razliko od Microsoft Office dodatka, ki ga je potrebno namestiti pri vsakem odjemalcu, tukaj opravimo namestitev le na strežniškem delu, kar pomeni zgolj eno namestitev. AMP datoteke se namestijo z uporabo orodja Module Management Tool [1].

Sharepoint protokol je preprost za uporabo. Ko gledamo Office dokument, se nam pojavi dodatna možnost `Edit Online`, pod pogojem, da imamo na dokumentu pravice za urejanje dokumenta. Ko to izberemo nas preusmeri v ustrezni Office urejevalnik za ta tip dokumenta.

Ko shranimo dokument v urejevalniku, se spremembe avtomatsko shranijo tudi v dokumentu v repozitoriju. Pri tem se izvede tudi verzioniranje, pri čemer se stara verzija dokumenta ohrani, popravljena pa postane aktualna verzija dokumenta.

Pri Sharepoint protokolu se nam je pojavila omejitev, ki je vezana na brskalnik uporabnika. Protokol namreč deluje zgolj v brskalniku Internet Explorer. V brskalniku Mozzila Firefox smo to težavo rešili tako da smo uporabili dodatek IE Tab 2 [6]. Ta dodatek nam omogoča, da trenutno stran prikažemo z uporabo brskalnika Internet Explorer.

Uporaba dodatka je enostavna. V vrstici stanja imamo poseben gumb, s katerim preklapljamo med pogledom z Internet Explorer in standardnim pogledom v Mozzila Firefox. Slabost tega pristopa je, da se je potrebno med preklopi ponovno prijaviti v Alfresco.

3.7. LDAP konfiguracija

LDAP (**Lightweight Directory Access Protocol**) je programski protokol, ki se uporablja za poizvedovanje in spreminjanje imeniških storitev in teče preko TCP/IP [10]. LDAP imenik je niz predmetov z atributi, ki je organiziran na logični in hierarhični način. Vnos v imeniku sestavlja skupek atributov, ki so določeni s shemami in imajo neko ime in eno ali več vrednosti. Vsak vnos ima enolični identifikator (ang. DN - Distinguished Name).

Zelo preprost primer LDAP imenika je navaden telefonski imenik. Njegova uporaba v praksi omogoča, da se nam je potrebno prijaviti zgolj enkrat za več različnih aplikacij. Tako bi se na primer prijavili zgolj v domeno z našim uporabniškim imenom, potem pa bi imeli dostop še do vseh ostalih aplikacij, pri katerih bi uporabljali isto uporabniško ime.

Alfresco ima na voljo več različnih načinov avtentikacije, ki so razdeljeni v različne avtentikacijske podsisteme (ang. Authentication Subsystems) [2]. Na voljo so nam naslednji podsistemi:

- alfrescoNtlm – standardna Alfresco avtentikacija – prijava z uporabniškim imenom in geslom,
- ldap - LDAP avtentikacija z OpenLDAP implementacijo protokola,
- ldap-ad - LDAP avtentikacija z ActiveDirectory implementacijo protokola,
- passthru - avtentikacija prek Windows domain serverja,
- kerberos – avtentikacija prek Kerberosa in
- external – avtentikacija prek zunanjega SSO(ang. Single sign-on) mehanizma.

V primeru, da bi vklopili LDAP avtentikacijo, ne bi imeli več dostopa prek **s**tandardne Alfresco avtentikacije. Tako razvijalci sistema ne bi mogli več dostopati do Alfresca, saj nismo vpisani v LDAP imenik naročnika. Zato imamo pri konfiguraciji avtentikacijskih sistemov možnost členjenja različnih avtentikacijskih sistemov. Primer avtentikacijske verige (ang. authenticathion chain):

```
authentication.chain=alfrescoNtlm1:alfrescoNtlm,passthru1:pass
thru,ldap1:ldap-ad
```

Zapis je v naslednji obliki: `ime_instance, tip_avtentikacijske_instance: ....`

Vsaki instanci nato posebej določimo nastavitve. To nam omogoča, da imamo več načinov avtentikacije istega tipa, recimo uporabljamo dva ločena LDAP imenika. Pomemben je tudi vrstni red avtentikacij v verigi, Alfresco namreč poskuša vzpostaviti avtentikacijo na način, ki se najprej pojavi v verigi.

3.8. Dodaten pogled v repozitoriju vmesnika Alfresco Explorer

Ena izmed dobrih lastnosti vmesnika Alfresco Explorer je njegov pogled `Details View` v repozitoriju. Ta pogled nam omogoča zelo dober pregled nad lastnostmi dokumenta. Še pomembneje je da nam omogoča sortiranje dokumentov po lastnostih. Problem se pojavi v tem, da ta pogled prikaže zgolj nekaj osnovnih lastnosti dokumenta, ne prikaže pa lastnosti, ki so lastne zgolj dokumentom, ki predstavljajo pogodbe.

Ena zelo preprosta rešitev tega problema je, da bi popravili datoteko `browse.jsp`. V tej datoteki so določeni prikazi vseh različnih pogledov v repozitoriju Alfresco Explorerja. Tak pristop pa bi imel dve veliki pomanjkljivosti:.

1. Prva pomanjkljivost je v tem, da bi spreminjali že obstoječe Alfrescove datoteke. Ob nadgradnjah sistema bi se spremembe v teh datotekah izgubile, saj bi jih zamenjale datoteke iz nove različice Alfresca.
2. Druga pomanjkljivost pa je v tem, da bi pogled `Details View` zmeraj prikazoval lastnosti pogodb v celotnem repozitoriju, ne glede na to ali bi ti dokumenti sploh imeli

te lastnosti oziroma ali bi si uporabnik to sploh želel. Z dodajanjem novih vrst dokumentov bi potrebovali tudi različne poglede za razlikovanje med dokumenti.

Zato potrebujemo boljšo rešitev, ki bi nam v Alfrescu dodala nov pogled, ki bi bil soroden pogledu `Details View` in bi prikazoval tudi lastnosti pogodb, hkrati pa ne bi spreminjal že obstoječih datotek Alfresca.

Najprej moramo definirati nov pogled. Vsak pogled v Alfresco Explorerju je definiran z lastnim Java razredom. Poglede, ki jih imamo na voljo, določimo v mapi `tomcat/shared/classes/alfresco/extension` znotraj datoteke `web-client-config-custom.xml`.

Ker bo naš nov pogled v sestavi identičen pogledu `Details View`,a bo prikazoval drugačne lastnosti, bomo za naš pogled uporabili kodo, ki definira `Details View`. Edina sprememba bo pravzaprav drugačna vrednost konstante `VIEWMODEID`, ki definira ključ pogleda.

Sedaj moramo definirati prikazovanje novega pogleda v datoteki `browse.jsp`. Namesto spreminjanja že obstoječe datoteke v Alfrescu ustvarimo lasten `browse.jsp`, ki je zgolj dopolnitev originalne datoteke. Med pogledi, ki so na voljo za izbiro v spustnem meniju, dodamo naš lasten pogled, nato pa definiramo še prikazovanje našega pogleda na podoben način, kot so v datoteki definirani obstoječi pogledi.

Na koncu moramo še določiti, da se bo pri prehodih med različnimi `.jsp` stranmi, ki definirajo vmesnik Alfresco Explorerja, namesto obstoječega `browse.jsp` uporabil naš `browse.jsp`. To naredimo v datoteki `faces-config-custom.xml`, ki leži v mapi `tomcat/shared/classes/alfresco/extension`. Pri tem prekopiramo Alfrescove izvirne prehode med stranmi (ang. navigation), ki se nahaja v datoteki `faces-config-navigation.xml` in zamenjamo vse pojavitve izvirnega `browse.jsp` z našim `browse.jsp`.

3.9. Uporaba tipov vsebine ali aspektov za metapodatke pogodbe.

Pri dodajanju lastnih metapodatkov na dokument imamo dve možnosti:

1. ustvarimo nov tip vsebine, ki bi bil otrok `cm:content`, ki je privzeti tip vsebine za dokumente v repozitoriju ali
2. ustvarimo lasten aspekt s podatki za pogodbo; ta aspekt bi potem dodali na dokument, ki bi predstavljal pogodbo, dokument bi bil še vedno tipa vsebine `cm:content`.

Tipi vsebine in aspekti so po sestavi zelo podobni. Oboji definirajo lastnosti na isti način. Vsaka lastnost predstavlja določen metapodatek, ki je lahko različnega tipa, celo število, decimalno število, tekst, datum, ipd…. .

Nadalje lahko lastnost še dodatno definiramo, recimo določimo če je obvezna, nastavimo omejitve glede vrednosti (na primer zgolj določene dolžine), nastavimo privzeto vrednost lastnosti in podobno. Obojim lahko nastavimo prikaz in izgled, v poljih za urejanje metapodatkov in na podatkih dokumenta.

Po drugi strani pa tipi vsebine definirajo vsebino, ki lahko obstaja v Alfrescu, medtem ko se aspekti uporabljajo predvsem za dodajanje dodatne funkcionalnosti tej vsebini. Močno se razlikujejo tudi po tem, kako jih dodamo v dokument. Tip vsebine lahko definiramo samo na začetku, ko dokument ustvarimo oziroma ga prenesemo v repozitorij. Potem ga ne moremo več spremeniti in zmeraj ima lastnosti, ki mu jih je tip vsebine definiral.

Aspekti so veliko bolj fleksibilni. Na dokument jih lahko dodamo kadarkoli, enako velja tudi za odstranitev iz dokumenta, seveda vse pod pogojem, da imamo na dokumentu pravice za urejanje. Poleg tega lahko aspekte dodajamo tudi programsko med potekom delovnega toka,ali pa s pomočjo pravil, ki jih nastavimo na določeno mapo v repozitoriju.

Za katerega izmed teh dveh pristopov se bomo odločili je na koncu odvisno od tega, kaj nam definirani metapodatki pomenijo. Če predstavljajo nekaj, s čimer želimo ta dokument ločiti od ostalih ali mu dati nek poseben pomen, uporabimo tip vsebine. Če predstavljajo podatke, ki jih lahko vsebujejo dokumenti, ki imajo za nas popolnoma različen pomen, uporabimo aspekte.

V našem primeru smo uporabili tip vsebine, saj je pogodba za nas ključen dokument in jo želimo posebej izpostaviti. Primer za uporabo aspekta bi bil določanje, ali se dokument nanaša na tujino ali je ne. Ta lastnost ni nujno vezana zgolj na pogodbe, temveč na veliko število različnih dokumentov, recimo na primer ponudbe, račune ali naročilnice.

Slabost uporabe aspektov bi se v tem primeru pokazala tudi pri iskanju po metapodatkih. Ker smo uporabili tip vsebine, imamo možnost omejiti naše iskanje zgolj na dokumente, ki so tipa pogodba. Z uporabo aspekta bi naše iskanje lastnosti potekalo po vseh dokument, ki so tipa `cm:content`, ali pa imajo `cm:content` za prednika.

Imamo tudi dva posebna tipa metapodatkov, ki ju dodamo na dokument z dodajanjem ustreznih aspektov. To so označbe (ang. tags) in kategorije. Definirajo ju aspekta `Tagable` oziroma `Classifiable`.

Označbe in kategorije se razlikujejo po sestavi; označbe nimajo nobene posebne sestave, kategorije pa imajo drevesno strukturo. Primer kategorij predstavljajo celine, znotraj tako imamo kategorije Evropa kategorijo Srednja Evropa, znotraj te kategorije pa Republika Slovenija.

Označbe in kategorije se razlikujejo tudi po tem, kako jih ustvarimo. Novo označbo lahko ustvari vsak, za dodajanje in urejanje kategorij pa je potrebno imeti administratorske pravice. Označbe in kategorije so v vmesniku Alfresco Share posebej prikazane in nam omogočajo lažje in hitrejše iskanje po dokumentih ter dodatno klasifikacijo vsebine.

3.10. Lastna pretvorba vsebine v Alfrescu

Ko vhodna pisarna skenira pogodbe v Alfrescov repozitorij nastane težava, ker ne moremo iskati po vsebini pogodb, saj pogodbe še niso šle skozi prepoznavo besed iz slike. Ta poskrbi, da dobimo iz skenirane slike tekst, ki je razumljiv računalniku (ang. machine readable text) [12]. OCR smo zato morali integrirati v Alfresco. To smo naredili z uporabo lastne pretvorbe vsebine (ang. custom content transformation).

Pretvorba vsebine je postopek, v katerem se vsebina pretvori iz enega formata v drugega. Alfresco ima veliko pretvorb že vgrajenih, predvsem z uporabo OpenOffice in ImageMagick. Seveda lahko zmeraj dodamo tudi lastno pretvorbo.

Pretvorbe po navadi sprožimo s pravili na mapah; vsebina, ki vstopi v mapo, se ob vstopu pretvori v neki drugi format. Tako dobimo dve datoteki z enakim imenom in z različno končnico, ker sta v različnih formatih. Pretvorbe lahko sprožimo tudi programsko v kodi znotraj delovnega toka ali pa znotraj skript, ki se ob določenih dogodkih zaženejo.

Da v Alfresco dodamo novo pretvorbo, moramo najprej ustvariti XML datoteko znotraj mape `tomcat/shared/classes/alfresco/extension`, ki novo pretvorbo definira. Ime datoteke se mora obvezno končati s `-transformers-context`. Znotraj te datoteke nato definiramo tri oznake `<property>` z različnimi imeni [9].

V `<property>` oznaki z imenom `explicitTransformations` definiramo datotečni format vira (ang. source mimetype) in ciljni datotečni format (ang. target mimetype). Pretvorba se bo izvedla na datotečnem tipu, kot je definiran v viru ,rezultat bo shranjen v tipu, kot je definiran v cilju.

Druga oznaka `<property>` z imenom `checkCommand` preveri, če sistem lahko izvede pretvorbo. Preveri se obstoj potrebnih komponent ali skripte z ukazi. Tretja oznaka `<property>` z imenom `transformCommand` pa vsebuje ukaze oziroma mesto zunanje skripte z ukazi, ki izvedejo dejanski postopek pretvorbe. Pomembno je, da v ukazu oziroma v skripti uporabimo kot argument spremenljivki `${source}` - vir in `${target}` – cilj pretvorbe.

V našem primeru smo želeli pretvoriti skeniran PDF brez teksta v PDF s tekstom. Pri tem je nastala težava, saj sta v tem primeru začetni in končni tip enaka, kar pomeni, da se v praksi pretvorba ne bi izvedla. Zato smo morali pretvorbo razdeliti na dva dela natančneje na dve ločeni pretvorbi; prvi del poteka iz `pdf` v `tiff` in drugi del poteka iz `tiff` v `pdf` format.

V prvem delu smo z uporabo ImageMagick in ukazom `convert` pretvorili pdf datoteko v 8-bitno tiff datoteko. OCR pogon, ki smo ga uporabili v drugem delu pretvorbe (Tesseract OCR), deluje namreč zgolj na 8 bitnih slikah formata tiff.

Nato sledi še drugi del pretvorbe. Najprej z uporabo ukaza `tiffsplit` iz `LibTIFF` knjižnice, tiff datoteko razdelimo na več enostranskih tiff datotek. Tesseract OCR namreč ne zna delati s tiff datotekami, ki imajo več kot eno stran. Nadalje na vsaki izmed teh tiff datotek poženemo Tesseract OCR.

Tesseract OCR lahko tiff datoteke pretvori v dva različna datotečna formata s tekstom [16]. Prvi format je navadni tekst brez oblikovanja (ang. plain text). Drugi pa je .html datoteka zapisana v standardu hOCR [16]. hOCR je standard s katerim v .html datoteke zapišemo (poleg teksta, ki smo ga dobili z OCR obdelavo) tudi oblikovanje, stile in postavitev teksta na sliki.

Iz html datoteke nato naredimo pdf datoteko s tekstom z uporabo ukaza `hocr2pdf` iz knjižnice `ExactImage`. Ukaz `hocr2pdf` sprejme dva argumenta, hOCR datoteko in tiff sliko, iz katere smo dobili hOCR datoteko. Tako ima končen pdf dokument, enak izgled kot tiff slika oziroma skenirana pogodba. Dejanski tekst znotraj pogodbe pa se zaradi napak in nenatančnosti OCR pogona lahko razlikuje od teksta, ki ga na pogodbi vidimo.

Na koncu dobljene PDF datoteke (za vsako stran dobimo eno PDF datoteko) združimo v končno PDF pogodbo z uporabo ukaza `pdfjoin` iz knjižnice `PDFjam`.

4. IZDELAVA LASTNEGA NAPREDNEGA DELOVNEGA TOKA V ALFRESCU

4.1. Model delovnega toka

Model delovnega toka nam določa naloge, ki se bodo izvedle med izvajanjem delovnega toka. Model je napisan v XML datoteki, ki jo v repozitoriju shranimo v mapo `Company Home/Data dictionary/Models`.

Najprej moramo definirati predpono modela (ang. prefix). To predpono bomo uporabljali za vse, kar bomo definirali znotraj delovnega toka. Če bo na primer predpona delovnega toka definirana kot »moje«, bo ime naloge v delovnem toku »moje:prvaNaloga«.

V modelu lahko tudi uvozimo vsebino, ki je že bila definirana v drugih modelih. Uvoze ustvarimo z oznako `<import>`, te oznake pa so znotraj oznake `<imports>` na začetku modela. Za potrebe delovnega toka je potrebno uvoziti vsaj tri modele. To so Dictionary Model, Content Model in Business Process Model.

Dictionary Model (s predpono `d`) definira različne vrste podatkovnih tipov. Primeri podatkovnih tipov, ki jih model definira, so:

- `d:int` in `d:long`, za cela števila,
- `d:float` in `d:double` za decimalna števila,
- `d:date` za datum,
- `d:datetime` za datum in čas,
- `d:boolean` za boolovo logiko, resnično ali neresnično,
- `d:category` za referenco do neke kategorije,
- `d:content` za vsebino v repozitoriju in
- `d:any` za poljubni podatkovni tip.

Content Model (s predpono `cm`) definira lastnosti vsebine v repozitoriju. Content Model definira vsebino, ki je v repozitoriju, lastnosti te vsebine (recimo ime, osebo v repozitoriju in lastnosti te osebe) ter nekatere že vgrajene aspekte v Alfresco, kot na primer:

- `Versionable`,
- `Dublin Core`,
- `Classifiable in`

- `Taggable`.

Business Process Model s predpono `bpm` pa definira stvari, ki so bistvene za delovanje delovnega toka. Uporabljajo ga tudi vsi že obstoječi delovni tokovi v Alfrescu. Nekatere izmed pomembnih stvari v delovnem toku, ki jih Business Process Model definira, so:

- `bpm:package`, ki definira dokumente, ki jih vključimo v delovni tok,
- `bpm:task`, ki definira nalogo in njene lastnosti in
- `bpm:assignee`, `bpm:assignees`, `bpm:groupAssignee`, ki definirajo vršilca naloge oziroma vršilce nalog.

Vsak tip v modelu, določimo z oznako `<type>` in predstavlja točno eno določeno nalogo, ki jo pa lahko v procesu uporabimo večkrat. Vsakemu izmed tipov moramo določiti tudi prednika v oznaki `<parent>`.

Za prvo nalogo je pomembno, da je njen prednik `bpm:startTask`, saj s tem določimo, da bo to začetna naloga. Za ostale naloge je potrebno, da so njihovi starši naloge, ki se bodo v delovnem toku izvedle pred njimi, saj s tem naloga podeduje vse do takrat definirane lastnosti, v nasprotnem primeru pa bi bile v nalogi te lastnosti nedefinirane. V tem primeru se model zaradi prednikov tipov uporablja kot model naloge (ang. Task model).

Če bi želeli, da se model uporablja kot model, za definicijo vsebine (ang. Content Model), bi morali za tipe v modelu uporabiti druge starše (recimo tip `cm:content)`, ki definira vsebino repozitorija. Razen tega modele, ki predstavljajo vsebino in naloge, definiramo na enak način.

Vsak tip vsebuje neke lastnosti, ki jih dodamo z oznako `<property>`. V primeru modela naloge te lastnosti predstavljajo polja, ki jih vpisujemo in urejamo v obrazcu naloge. V lastnosti je najpomembneje definirati, kateremu podatkovnemu tipu pripada ta lastnost. To dodamo znotraj oznake `<type>` v lastnosti. Vsi možni podatkovni tipi so definirani v Dictionary Modelu.

Poleg tega lahko za posamezno lastnost definiramo še nekaj drugih oznak:

- `<title>` definira privzeto ime lastnosti v obrazcih za naloge,
- `<mandatory>` definira, če je to obvezna lastnost ali ne. Ima dve možni vrednosti (`true` ali `false)`, privzeta vrednost je `false`,
- `<multiple>` nam pove, ali lahko neka lastnost zavzame več kot samo eno vrednost. Tudi ta oznaka ima na dve možni vrednosti (`true` ali `false`), privzeta vrednost je `false` in
- `<default>` definira privzeto vrednost lastnosti, ko se lastnost prvič prikaže v delovnem toku.

V modelu lahko v tipih vsebine poleg lastnosti definiramo tudi asociacije. Z uporabo asociacij dosežemo, da tipi vsebine kažejo na stvar, ki je že bila definirana v nekem modelu, na primer uporabnika v Alfrescu ali dokument v repozitoriju.

Asociacije postavimo znotraj oznake `<associations>`. Posamezno asociacijo definiramo z oznako `<association>` oziroma `<child-association>`. Razlika med njima je, da prva asociacija samo kaže na neko drugo stvar (ang. reference association), druga vrsta asociacije pa dela kazano vsebino, odvisno od vsebine, ki se nanjo sklicuje (ang. child association). To pomeni, da sta v razmerju prednik - naslednik, če izbrišemo starša bo izbrisan tudi otrok.

Primer asociacije prednik - naslednik sta mapa in vsebina, ki leži v tej mapi. Mapa je definirana s tipom `cm:folder`, znotraj tipa pa je oznaka `<child-association>` z imenom `cm:contains`, ki kaže na vsebino, ki je shranjena v tej mapi.

Asociacij po navadi ne uporabljamo pogosto, če definiramo model delovnega toka, ker so asociacije zmeraj vezane na neko že obstoječo vsebino. Zato pa so bolj pogoste v modelih, ki definirajo vsebino v repozitoriju.

Veliko uporabnih asociacij za delovne tokove, je že definiranih v obstoječih modelih v Alfrescu. Tako so na primer vršilci nalog definirani z uporabo asociacij, asociacija je tudi `bpm:package`, saj z njim kažemo na že obstoječe dokumente, ki so shranjeni v repozitoriju.

Vsaka asociacija vsebuje dve oznaki: `<source>` vsebina iz katere se sklicujemo oziroma vir in `<target>` vsebina na katero se sklicujemo oziroma cilj. V vsaki izmed teh dveh oznak sta oznaki `<many>` in `<mandatory>`. Prva oznaka definira če v neki asociaciji nastopa več virov ali ciljev, druga pa pove, če je vir oziroma tarčo potrebno obvezno definirati. Kadar je `<mandatory> true,` je treba asociacijo zmeraj definirati, če uporabljamo tip, ki jo vsebuje. Znotraj oznake `<target>` nastopa tudi oznaka `<class>`, znotraj te oznake vpišemo vsebino, na katero se ta asociacija sklicuje, torej ime izbranega tipa vsebine.

Poleg tipov lahko v modelu nastopajo tudi aspekti. Definiramo jih znotraj oznake `<aspects>` z oznakami `<aspect>`. Tako kot tipi vsebine so sestavljeni iz lastnosti in asociacij, torej vsebino znotraj oznak aspektov definiramo na isti način.

Kot že omenjeno pa se aspekti od tipov vsebine močno razlikujejo po uporabi. Aspekte uporabljamo tedaj, ko definiramo modele za vsebino, redko pa v modelih, ki definirajo delovne tokove. V teh modelih praviloma definiramo samo naloge, te pa se zmeraj definirajo s tipi vsebine. Primer uporabe aspektov v delovnem toku je definicija vršilca oziroma vršilcev nalog, saj so ti namreč definirani znotraj aspektov.

Ker so aspekti namenjeni razširitvi funkcionalnosti vsebine, ki jo določajo tipi, lahko že v modelu dodamo tipu vsebine nek obstoječi aspekt. S tem bo aspekt zmeraj del tipa vsebine. To storimo tako, da znotraj tipa dodamo oznako `<mandatory-aspects>` in znotraj te oznake naštejemo aspekte, katere tip vsebuje.

4.1.1. Omejitev v modelu in lastna dinamični omejitev

Omejitev (ang. constraint) dodatno omejuje možne vrednosti, ki jih lahko uporabnik določi za neko lastnost. Omejitve uporabljajo tako tipi vsebine kot aspekti.

Omejitve so definirane po definiciji predpone znotraj oznake `<constraints>`, torej pred definicijo tipov vsebine in aspektov. Lastnosti se potem sklicujejo na različne v modelu že definirane omejitve. To določimo znotraj oznake `<constraints>` v lastnosti. Ena lastnost ima lahko več omejitev, ena omejitev pa lahko nastopa v več lastnostih.

Alfresco nam ponuja štiri vrste že pripravljenih omejitev [1]:

- REGEX primerja vrednost lastnosti z nekim regularnim izrazom, in preveri ali se vrednost lastnosti ujema z regularnim izrazom. Glede na definicijo te omejitve, je od vrednosti parametra `requiresMatch` odvisno, če je lastnost veljavna, ko se njena vrednost ujema oziroma ne ujema z regularnim izrazom,
- LIST vrednost lastnosti omeji na nekaj že definiranih in nespremenljivih lastnosti. Polje take lastnosti se v obrazcu naloge po navadi prikaže kot spustni meni,
- MINMAX se uporablja za omejevanje številskih lastnosti. S tem, ko določimo zgornjo mejo lastnosti (parameter `maxValue`) in spodnjo mejo lastnosti, (parameter `minValue`), določimo številski interval v kateremu mora biti vrednost lastnosti in
- LENGTH se uporablja za določitev dolžine neke lastnosti. Parameter `minLength` določi najmanjše možno število črk v lastnosti, parameter `maxLength` pa največje možno število črk v neki lastnosti.

Poleg uporabe že obstoječih omejitev lahko napišemo tudi lastno omejitev. Delovni tok za potrjevanje pogodb ima lastnost, ki predstavlja pogodbenega partnerja. Pogodbeni partner so shranjeni v zunanji podatkovni bazi, v katero se tudi redno vnašajo novi pogodbeni partnerji.

Ker želimo lastnost omejiti zgolj na točno določene pogodbene partnerje, moramo za to uporabiti omejitev. Ker želimo tudi, da je omejitev dinamična (seznam partnerjev se vsakič prebere iz baze), moramo uporabiti lastno omejitev (ang. custom constraint).

Omejitev je v Alfrescu definirana v Javanskem razredu. Ta razred, definira omejitev tako, da implementira vmesnik `org.alfresco.service.cmr.dictionary.Constraint`. Poleg tega razred z omejitvijo tudi razširja (ang. extends) enega izmed razredov, ki že definira omejitev. Razred, ki določa najbolj osnovno stopnjo omejitve, se imenuje `org.alfresco.repo.dictionary.constraint.AbstractConstraint`.

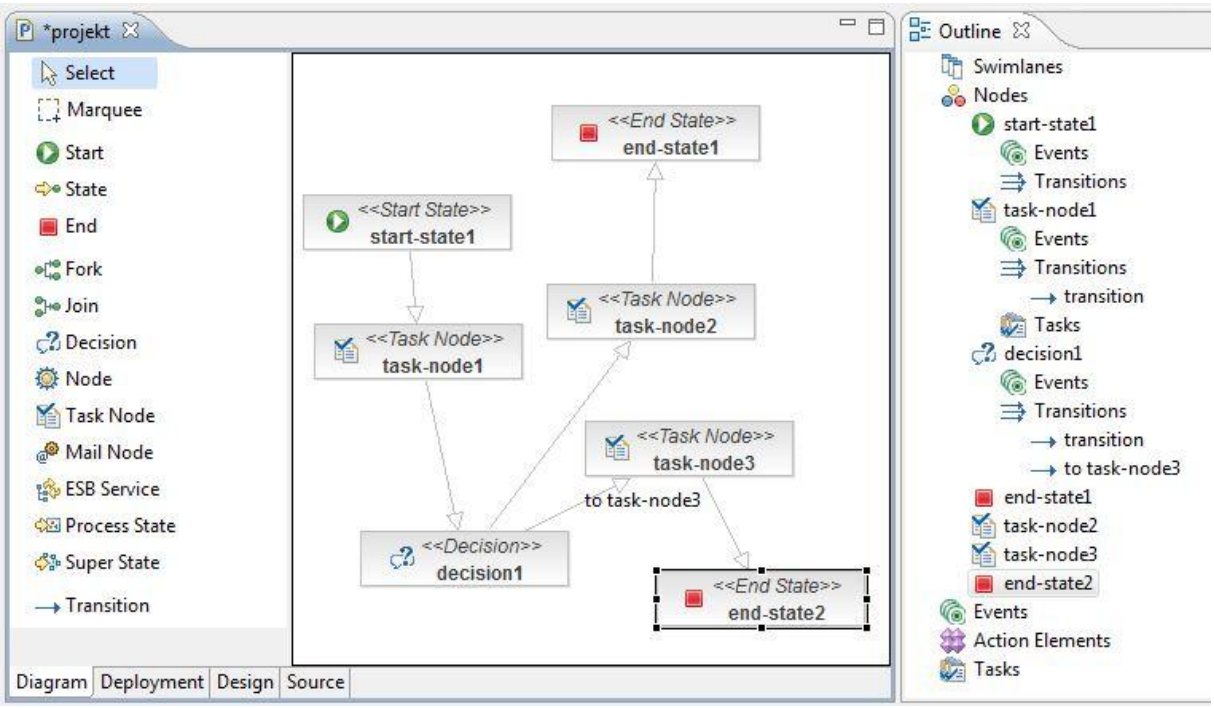
Kateri razred bo naša omejitev razširila, je odvisno od tega, kako bi želeli lastnost omejiti. Naša omejitev je zelo podobna omejitvi `LIST` saj definira seznam točno določenih vrednosti. Razlika je le v tem, da v naši omejitvi seznama ne definiramo sami v modelu, temveč se

napolni z zapisi iz podatkovne baze. Za to bomo v naši omejitvi razširjali razred omejitve `org.alfresco.repo.dictionary.constraint.ListOfValuesConstraint`.

V Java razredu moramo še prebrati pogodbene partnerje iz podatkovne baze in te določiti kot seznam vrednosti v omejitvi. To naredimo z metodo `setAllowedValues`, ki je definirana v razredu `ListOfValuesConstraint` in sprejme za argument seznam vrednosti (podatkovni tip `List`). Če pri razvoju lastne omejitve potrebujemo dodatne parametre, te dodamo v razpršeno tabelo (ang. HashMap) `params`, ki definira imena parametrov omejitve.

4.2. Proces delovnega toka

Pri izdelavi procesa delovnega toka si lahko pomagamo z orodjem JBoss jBPM Process Designer [14]. To je grafični urejevalnik za izdelavo delovnih tokov v orodju Eclipse(slika 10). V tem urejevalniku lahko istočasno grafično urejamo proces in ročno vpisujemo kodo procesa. Spremembe so takoj vidne v kodi oziroma v diagramu procesa.



Slika 10: Orodje JBoss jBPM Process Designer.

Z urejevalnikom lahko zgolj uredimo osnovni skelet delovnega toka, vozlišča in povezave med vozlišči. Dodatne elemente delovnega toka (kot so vršilci nalog in akcije v delovnem toku, Java in Javascript koda), moramo nato sami ročno dodati.

Koda procesa je sestavljena iz oznak, ki predstavljajo posamezna vozlišča. Oznaka `<start-state>` predstavlja prvo vozlišče, ki definira začetno nalogo. Oznaka `<end-state>` predstavlja končno vozlišče in je zaključek delovnega toka. Delovni tok mora imeti vsaj eno oznako `<end-state>`, lahko jih ima tudi več.

Naloge v delovnem toku oziroma vozlišča z nalogami definiramo z oznako <task-node>. Tranzicije ležijo znotraj oznak vozlišč na koncu. Oznaka <transition> definira posamezno tranzicijo, z atributom to določimo v katero vozlišče naj tranzicija vodi. V ta namen ima vsako vozlišče atribut name, ki je ključ vozlišča.

Znotraj <start-state> in <task-node> definiramo nalogo z oznako <task>, ki naj se izvede v tem vozlišču. Atribut name v oznaki predstavlja identifikator naloge iz modela delovnega toka.

V vsaki nalogi moramo definirati vršilca naloge. Znotraj oznake <task> je oznaka <assignment>, znotraj nje pa nastopi ena izmed dveh oznak. Oznaka <actor> se uporabi v primeru, kadar je naloga vezana na samo enega uporabnika, oznaka <pooledactors> pa, ko je naloga vezana na skupino uporabnikov.

Znotraj teh oznak nato določimo vršilce. To lahko naredimo z uporabo Javascript kode, ki nam vrne določenega uporabnika, (funkcija people.getMembers(uporabniško ime uporabnika)) ali pa določeno skupino, (funkcija people.getGroup (ime skupine, začne se s tekstom GROUP_ in nato nadaljuje z identifikatorjem skupine)).

Drugi način za določanje vršilcev je z uporabo asociacij, ki so definirane v Business Process Model. To so:

- bpm:assignee eden uporabnik za vršilca naloge,
- bpm:assignees več uporabnikov za vršilca naloge,
- bpm:groupAssignee ena skupina za vršilca naloge in
- bpm:groupAssignees več skupin za vršilca naloge.

Vrednost teh asociacij se mora že prej določiti v delovnem toku. Po navadi jih na začetku določi kar pobudnik delovnega toka. Ne glede na, to katerega izmed teh načinov uporabimo, moramo vrednost znotraj oznak <actor> in <pooledactors> začeti z #{ in končati z }.

Poleg definiranja vršilcev nalog znotraj oznake <task> lahko enako storimo tudi v oznakah <swimlane>. To so oznake, ki jih definiramo izven vozlišč delovnega toka. Znotraj teh oznak je definicija vršilcev enaka kot znotraj oznak <task>, uporabimo tudi iste oznake <assignment>, <actor> in <pooledactors>. Vsaki oznaki <swimlane> določimo identifikator z atributom name. Ko želimo nato definirati vršilce naloge z uporabo <swimlane> oznake, v oznaki <task> vpišemo v atribut swimlane ime te oznake.

Poseben primer predstavlja definicija pobudnika delovnega toka za vršilca naloge, saj takrat znotraj oznake <assignment> ne postavimo nobene oznake.

Ko smo določili strukturo delovnega toka, naloge in vršilce nalog, je na vrsti dodajanje logike in obnašanja v delovni tok. Način za dosego tega je uporaba akcij. Akcije določimo z oznako

<action> in predstavljajo mesta v delovnem toku kjer izvedemo neko programsko kodo v Javi ali pa Javascriptu.

Oznaka <action> ima atribut class, ki definira, če gre za Java ali Javascript kodo. Če gre za javansko kodo je vrednost atributa class enaka imenu datoteke z Java kodo, ko je vrednost atributa class org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript pa znotraj oznake <action> vpisujemo Javascript kodo. Java koda je za razliko od JavaScript kode vedno v zunanji class datoteki.

Oznake <action> lahko v procesu postavimo le na točno določena mesta. Eno mesto je znotraj tranzicij (torej znotraj oznak <transition>), drugo mesto pa je znotraj oznak <event>.

Oznake <event> predstavljajo nek dogodek, ki se zmeraj zgodi ob nekem točno določenem času. Atribut type v oznaki predstavlja vrsto dogodka, ki določa, kdaj se ta dogodek izvrši. Vsako vozlišče v procesu definira dva dogodka:

- node-leave, ko zapustimo vozlišče in
- node-enter, ko vstopimo v vozlišče.

Poleg tega imamo na voljo še 4 dodatne dogodke, kadar je to vozlišče z nalogo. Takrat lahko znotraj oznake <task> dodamo še naslednje vrste dogodkov:

- task-create, se izvede, ko se naloga ustvari,
- task-start, se izvede, ko nalogo zaženemo,
- task-assign, se izvede, ko se naloga dodeli nekemu uporabniku, pogosto kadar vršilec nalogo dodeli nekomu drugemu (ang. reassign) in
- task-end, se izvede tedaj ko je naloga zaključena.

V delovnem toku potrjevanja pogodb smo večkrat uporabili akcije obeh vrst tako z Javascriptom kot tudi z javasnko kodo. Tako smo na primer uporabili JavaScript kodo v <start-state> za dogodek node-leave, za določanje vrednosti lastnosti bpm:workflowDescription. Ta lastnost definira ime naloge, ki bo vidno v seznamu nalog.

Zahteva naročnika je bila, da bo to ime enako naslovu pogodbe, naslov pogodbe pa že definira obvezna lastnost v prvi nalogi. Zato ne potrebujemo polja za vnos bpm:workflowDescription in vrednost bpm:workflowDescription nastavimo kar na vrednost lastnosti naslova pogodbe. Že definiranim lastnostim, kot so bpm:workflowDescription in procesnim spremenljivkam lahko nastavimo vrednost v vseh oznakah <action> ne glede na mesto, kjer se pojavijo. Procesne spremenljivke so

spremenljivke, ki jih definiramo znotraj procesa delovnega toka za pomoč pri obnašanju in logiki v delovnem toku. Končni uporabnik jih ne vidi.

Lastnostim, ki smo jih definirali znotraj našega modela lahko nastavimo vrednost zgolj v dogodkih, ki se sprožijo zaradi nalog, (torej oznake `<event>` v oznaki `<task>`).Vrednost lastnosti nastavi metoda `taskInstance.getVariable` (ime_lastnosti). Vrednost lastnosti dobimo z metodo `taskInstance.setVariable` (ime_lastnosti, vrednost).

Lahko se zgodi, da moramo znotraj JavaScript kode kdaj izvesti opravilo, za katerega bi potrebovali pravice administratorja, oziroma trenutni vršilec naloge nima dovolj pravic za to opravilo. Primer takih opravil je denimo urejanje pravic dostopa do mape in vsebine. Takrat znotraj oznake `<action>` dodamo oznako `<runas>` z vrednostjo `admin`. S tem smo dosegli, da lahko programsko izvedemo JavaScript kodo z enakimi pravicami kot administrator.

Ko uporaba JavaScripta ne zadostuje za naše potrebe, uporabimo znotraj akcije javansko kodo. Eden izmed takih primerov je dostop do zunanje baze znotraj akcije. Javascript tega, namreč ne omogoča.

V delovnem toku potrjevanja pogodb se, potem ko v prvi nalogi pobudnik izbere enega ali več pogodbenih partnerjev, z uporabo Java kode določijo ostali podatki partnerjev, kot so naslov, zastopnik in številka tekočega računa. Ti podatki so namreč shranjeni v zunanji bazi.

Java razredi, ki se uporabljajo znotraj akcij, morajo obvezno razširjati razred `org.alfresco.repo.workflow.jbpm.JBPMSpringActionHandler`. Koda, ki se bo izvedla znotraj neke akcije, leži znotraj javanskega razreda v metodi `public void execute(ExecutionContext executionContext)`. Znotraj te metode nato definiramo vrednosti lastnosti. To počnemo na podoben način kot v Javascriptu, torej z uporabo metode `setVariable` znotraj razreda `TaskInstance`.

Zelo pomembno vozlišče v delovnem toku je vozlišče za odločitev, ki ga določimo z oznako `<decision>`. Iz vozlišča `<decision>` vedno vodita vsaj dve tranziciji. Odločitev, po kateri izmed tranzicij se bo nadaljeval delovni tok, ne sprejme uporabnik temveč sistem, glede na pogoj, ki je definiran znotraj `<decision>` vozlišča.

Pogoj za izbiro tranzicije določimo z uporabo oznake `<condition>` znotraj posamezne tranzicije v oznaki `<decision>`. Znotraj oznake `<condition>` nato določimo pogoj. Pomembno je da se pogoj začne z `#{` in konča z `}`. Znotraj `<decision>` `lahko` pustimo eno tranzicijo tudi brez pogoja. Ta tranzicija se bo izbrala, ko ne bo zadoščeno nobenemu izmed pogojev v ostalih tranzicijah.

Pri določanju pogojev v tranzicijah moramo paziti na vrstni red pogojev, oziroma tranzicij s pogoji, da se pogoji med seboj ne prekrivajo. Zato je primer dobre prakse za določanje pogojev, da ob vstopu v vozlišče `<decision>` v dogodku `node-enter` izvedemo preverjanje vseh pogojev, ki določajo izbiro tranzicije. Znotraj tega dogodka tudi definiramo

spremenljivko, njena vrednost določa, katera izmed tranzicij se bo izbrala. Zdaj moramo samo paziti, da se spremenljivka nastavi tako, da bo imela eno izmed točno določenih vrednost oziroma da se spremenljivki zmeraj določi neka vrednost. Nato znotraj oznak `<condition>` zgolj preverimo, če je spremenljivka enaka vrednosti, ki določa, da je bila izbrana ta tranzicija. Da to deluje mora biti znotraj posameznih oznak `<condition>` različna vrednost, s katero to spremenljivko preverjamo.

4.2.1. Vzporedna naloga v delovnem toku

Ko vodja oddelka potrdi pogodbo, sta naslednja na vrsti za potrjevanje vodja skupnih služb. Pogodba je potrjena samo takrat, ko sta jo potrdila oba vodja skupnih služb, drugače je zavrnjena. Ker sta ti dve potrjevanji neodvisni med seboj, ni potrebe, da bi nastopila ena naloga pred drugo. Odločitev v eni nalogi ne vpliva na potek in izid druge naloge. Zato tudi potekata ti dve nalogi v delovnem toku vzporedno in ne zaporedno.

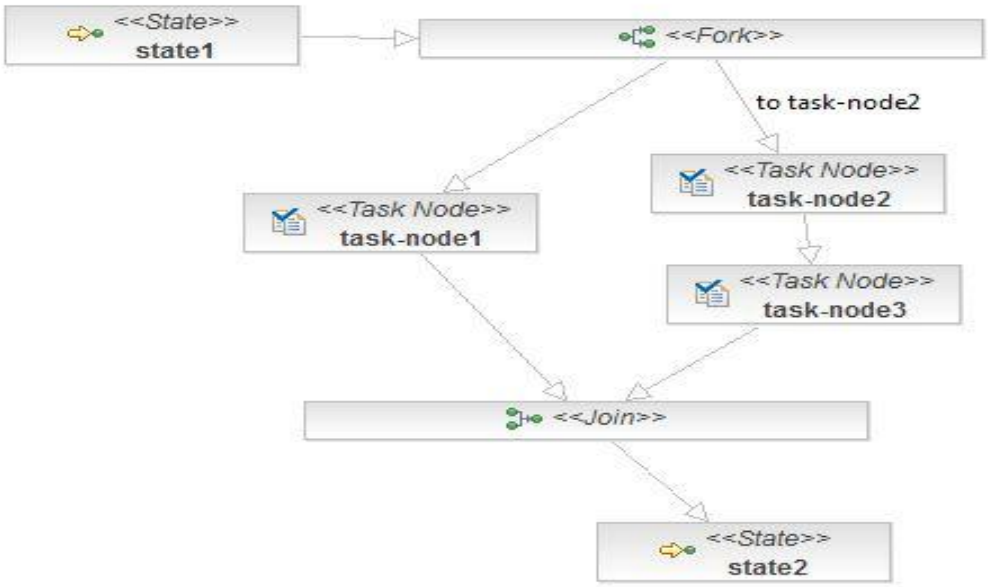
Vzporedne naloge v delovnem toku določimo na dva načina. Prvi način je z uporabo konstrukta `foreach` [8]. Konstrukt `foreach` v Alfrescu definiramo kot akcijo razreda `org.alfresco.repo.workflow.jbpm.ForEachFork`. V tej akciji sta dve oznaki.

Prva oznaka je `<foreach>`, v kateri se v zanki sprehodimo skozi neko skupino uporabnikov v Alfrescu. Druga oznaka je `<var>`. Znotraj nje poimenujemo spremenljivko, ki predstavlja seznam uporabnikov v skupini. Potem v naslednji nalogi vstavimo to spremenljivko znotraj oznake `<actor>`. S tem smo za vršilca naloge določili skupino uporabnikov, ki bodo to nalogo izvedli neodvisno drug od drugega.

Konstrukt `foreach` je preprost za uporabo, vendar ne zadošča pri kompleksnejših primerih vzporednih nalog. V našem primeru bi bil neučinkovit, saj sta vodji skupnih služb v ločenih skupinah, pa tudi v modelu imamo definirani ločeni nalogi za vsakega vodjo posebej. V takih primerih namesto foreach uporabimo oznaki `<fork>` in `<join>` (slika 11).

Oznaka `<fork>` razveji potek delovnega toka v več ločenih vzporednih poti, oznaka `<join>` pa te poti nazaj združi. Prednost takega pristopa je, da lahko znotraj vzporednih poti definiramo povsem različen potek nalog in drugih vozlišč za vsako izmed teh poti. Tako v našem primeru v oznaki `<fork>` definiramo dve tranziciji. Vsaka vodi do naloge za vodjo ene izmed skupnih služb. Obe tranziciji iz teh nalog se združita v oznaki `<join>`.

Ker je pogodba potrjena zgolj takrat, ko sta oba vodja potrdila pogodbo, moramo za to dodati še dve stvari. Procesna spremenljivka beleži kolikokrat je bila pogodba potrjena. Povečamo jo vsakič za 1 znotraj tranzicije ko se zgodi potrditev. Oznaka `<decision>` nastopi za oznako `<join>` in ustrezno nadaljuje potek delovnega toka.



Slika 11: Grafični prikaz uporabe vozlišč <fork> in <join>.

4.2.2. Prenos komentarjev med nalogami delovnega toka

V nalogah je potrebno, da bo vsak vršilec naloge lahko podal svoj komentar. To velja posebej za vodje, da lahko podajo svoje mnenje o pogodbi. Pomembno je tudi, da se lahko ti komentarji prenašajo med nekaterimi nalogami, saj so zelo verjetno namenjeni vršilcu naslednje naloge. Poleg tega ima pobudnik delovnega toka vedno pregled nad vsemi komentarji za vsako nalogo.

Business Process Model definira lastnost `bpm:comment`, ki predstavlja komentar v delovnem toku. Prednost te lastnosti je, da je lepo vidna v pregledu delovnega toka. Problem te spremenljivke je, da se njena vrednost nastavi vsakič na prazno, ko se ustvari nova naloga, torej se vrednost lastnosti ne prenaša med nalogami. Da odpravimo to omejitev, si pomagamo z dodatno lastnostjo, ki predstavlja komentar v prejšnji nalogi.

Kljub temu da se vrednost `bpm:comment` vsakič izbriše, se vsi v delovnem toku do takrat vpisani komentarji shranjujejo v posebno spremenljivko. Znotraj JavaScripta je to spremenljivka `token.comments`. Zadnji vpisan komentar v tej spremenljivki je komentar z največjim indeksom, vrednost tega indeksa je tako velikost `token.comments` zmanjšana za 1. Ko se ustvari nova naloga, vpišemo v lastnost, ki predstavlja zadnji komentar, zadnji zapis v `token.comments`.

Da dodamo uporabniku možnost izbire, če naj se komentar posreduje naslednjemu v delovnem toku, smo dodali še lastnost tipa `d:boolean`. V obrazcu se prikaže kot polje s kljukico, ki določa, ali naslednji v delovnem toku vidi ta komentar ali ne. Ne glede na to izbiro se komentarji shranjujejo v `token.comments` in so vidni pobudniku med pregledom delovnega toka.

4.2.3. Ustvarjanje pogodbe med potekom delovnega toka

V delovnem toku lahko pogodba obstaja že na začetku, lahko pa se ustvari med delovnim tokom iz ene izmed predlog. Predloge so datoteke formata `.doc` in pobudnik izbere eno na začetku delovnega toka. Potem sledi še urejanje pogodbe s funkcijo `EDIT ONLINE` in dopolnitev vsebine, ki je specifična za to pogodbo. Sem sodijo podatki kot so ime partnerja, naslov partnerja, zastopnik partnerja in podobno.

Problem je nastal, ker je bilo velik del specifične vsebine za neko pogodbo, definiran že na začetku, z izbiro enega ali več pogodbenih partnerjev. Vpisovanje teh podatkov bi bil povsem odvečen korak. Tako se je pojavila potreba, da se vsakič dopolni vsebina nove pogodbe iz predloge z že znanimi podatki.

Alfresco ne more ustvariti in spreminjati vsebine formatov kot je `.doc`, to lahko počne le za preproste formate kot so `.xml`, `.html` in `.txt`. Vsebuje pa veliko že vgrajenih pretvorb vsebine. Tako Open Office v Alfrescu omogoča pretvorbo iz `.html` v `.doc` dokument, kar je idealno za naše potrebe. Najprej smo `.doc` dokumente v Open Office shranili kot `.html` dokumente, da bomo pri pretvorbi nazaj v `.doc` dobili najbolj natančen približek izvirniku. Te .html dokumente smo nato shranili v repozitorij kot predloge za pogodbe in v njih določili mesta, kjer se bo vnesla že definirana vsebina.

V delovnem toku se izvaja proces ustvarjanja pogodbe tako, da najprej ustvarimo ustrezni `.html` dokument, vnesemo podatke o pogodbi na točno določeno mesto, nato `.html` dokument transformiramo v `.doc` dokument, izbrišemo `.html` dokument in nato na koncu `.doc` dokument dodamo v `bpm:package` in s tem v delovni tok. S tem dobimo novo pogodbo z mnogimi že vpisanimi podatki o partnerjih. Nato je na vrsti še pobudnik, da v naslednji nalogi dopolni in po potrebi tudi oblikuje pogodbo.

4.2.4. Ponovna dodelitev naloge pri skupinah

Ko gledamo neko nalogo v Alfresco Share se nam v primeru ko, je naloga namenjena le enemu vršilcu, nad nalogo prikaže gumb `Reassign`. Ta gumb vršilcu naloge omogoča, da jo dodeli nekomu drugemu.

Kadar je naloga namenjena skupini ali pa več posameznikom, gumb `Reassign` nadomesti gumb `Claim`. Ta gumb zaklene nalogo pred ostalimi uporabniki tako, da je ti ne morejo vršiti, saj bi sicer lahko nalogo istočasno izvedlo več uporabnikov. Ko nek vršilec prevzame nalogo, gumb `Claim` nadomesti gumb `Release to Pool`. S tem nalogo sprostimo, da jo lahko prevzame in izvrši nek drug uporabnik.

Težava nastopi, ker vodje oddelkov in oba vodja skupnih služb v praksi pogosto dajo pogodbo nekomu drugemu, naj jo pregleda namesto njih. To pomeni da bi v delovnem toku v Alfrescu obvezno potrebovali gumb `Reassign`. Vodje oddelkov in skupnih služb so

definirani s tem, da pripadajo ustrezni skupini. Tako moramo v delovnem toku za vršilca naloge navesti ustrezno skupino vodje.

Prvi razlog, zaradi katerega so vodje definirani s skupinami, je, da v primeru zamenjave vodje oddelka ni potrebno popravljati delovnega toka. Zgolj dodamo novega vodjo v skupino in odstranimo starega iz skupine. Drugi razlog pa je da, v primeru dopusta ali odsotnosti vodja, dodamo v skupino osebo, ki ga v tem času nadomešča, ta bo potem oseba prosto izvaja naloge vodje.

Morali smo najti rešitev, kjer bi v primeru, ko je v skupini vodje zgolj ena oseba (dejanski vodja), bil ko vršilec naloge določena oseba in ne skupina. Da bi to dosegli, smo morali za vsako nalogo, ki je vezana na vodje, dodati novo vozlišče naloge, ki vsebuje isto nalogo in tranzicije, s to razliko, da je vezana na osebo in ne na skupino. Pred vsakim izmed teh vozlišč smo morali tako dodati še vozlišče za odločitev, ki preveri, ali je skupini vodje zgolj en član. To ustrezno usmeri potek delovnega toka.

V nalogah kjer je v skupini samo pravi vodja, se vršilec določi znotraj oznake `<actor>`, kot prvi element tabele, ki ga vrne metoda `people.getMembers`, saj ima tabela zgolj enega uporabnika.

4.2.5. Pošiljanje obvestil v obliki emailov

Ena izmed zahtev naročnika je bila tudi, da se znotraj delovnega toka pošiljajo elektronska obvestila vršilcem nalog, da so dobili nalogo za izvršitev oziroma, da je rok za izvršitev naloge potekel. Elektronska sporočila znotraj delovnega toka pošiljamo z uporabo kode v akcijah, torej z JavasScriptom ali Javo.

V JavaScriptu email sporočilo ustvarimo s stavkom `actions.create("mail");`. Rezultat tega stavka shranimo v neko spremenljivko. Znotraj te spremenljivke je dejansko shranjen objekt, ki predstavlja email in ima več lastnosti, ki določajo vsebino elektronskega sporočila. Te lastnosti so:

- `parameters.to` in `parameters.to_many`, predstavljata prejemnika oziroma skupino prejemnikov,
- `parameters.subject` določa zadevo,
- `parameters.text` določa vsebino sporočila in
- `parameters.from` določa naslov pošiljatelja sporočila.

Ko smo določili vsebino sporočila, ga je potrebno še poslati. To naredimo tako, da na spremenljivki, ki predstavlja elektronsko sporočilo, pokličemo metodo `execute` s parametrom `bpm_package`.

Pri vključevanju sporočil v delovni tok smo se soočili z dvema problemoma. Prvi problem je bil, da se obvestila v obliki elektronskih sporočil pošiljajo glede na željo uporabnika. Torej se

uporabnik sam odloči ali želi, da sprejema obvestila v obliki elektronskih sporočil ali ne. Idealna rešitev bi bila, če bi imel uporabnik v svojem profilu polje s kljukico, ki bi to določalo. Nesmiselno bi bilo spreminjati Content Model in nato še konfigurirati izgled vmesnika za naše potrebe.

Tako smo za naše potrebe izbrali eno izmed lastnosti, ki jih Content Model že definira. Na žalost ni nobene lastnosti za uporabnika, ki bila vrste `d:boolean`. Vse lastnosti so tipa `d:text`. Tako smo se morali odločiti za neko lastnost, ki bi, v primeru ko bi imela eno ali več točno določenih vrednosti, vklopila pošiljanje email obvestil uporabniku v delovnem toku.

Idealna lastnost za to je `cm:companyemail`, elektronski naslov podjetja uporabnika. Vsi uporabniki v sistemu imajo v našem primeru vedno le eden elektronski naslov. Ta je pa že definiran v lastnosti `cm:email`, zato ni potrebe po lastnosti `cm:companyemail` in zato smo jo izbrali.

Drugi problem ki se je pojavil, je bil, da moramo uporabnikom poleg obvestil, da so prejeli nalogo, poslati tudi obvestilo, da je naloga zastarala oziroma da je njen rok potekel. Tukaj si pomagamo z oznako `<timer>`. Znotraj te oznake postavimo akcijo, za katero želimo, da se ne izvede takoj, temveč čez nekaj časa. Ključen atribut znotraj oznake `<timer>` je `duedate`, ki določa čez koliko časa se bo akcija izvedla [8]. To je lahko fiksen čas (torej po določenem številu minut, ur ali dni) lahko pa je vezan na neko spremenljivko, ki predstavlja nek datum.

Ker smo v našem primeru želeli poslati email obvestilo, ko nalogi poteče rok, smo za `duedate` določili lastnost `bpm:dueDate` iz Business Process Modela. Ta lastnost predstavlja rok za izvedbo naloge, vsaka naloga v delovnem toku pa ima zmeraj lasten `bpm:dueDate`.

Oznaka `<timer>` ima poleg atributa `duedate` še atribut `transition`. S tem avtomatsko sprožimo eno izmed tranzicij, ki so definirane v tem vozlišču. V našem primeru ga nismo uporabili, saj smo želeli, da uporabnik zmeraj opravi nalogo.

4.3. Vizualno oblikovanje obrazcev za naloge

V Alfrescu obstajata dva različna načina oblikovanja obrazcev za nalogo. Eden je namenjen vmesniku Alfresco Share, drugi pa vmesniku Alfresco Explorer. Zaradi tega imamo lahko prikaz iste naloge popolnoma različen v obeh vmesnikih.

Brez definicije prikaza nalog bi bil v vmesniku Alfrescu Explorer obrazec naloge prazen. V Alfrescu Share pa se nam tudi brez definicije prikaza nalog pokaže obrazec s polji. Vendar ta obrazec prikaže polja za čisto vsako lastnost, ki je bila do takrat definira v delovnem toku, kar

pomeni, da se nam prikažejo tudi polja, ki niso namenjena uporabniku. Poleg tega so polja razvrščena brez kakšnega posebnega reda in obrazec z nalogo tako postane zelo nepregleden.

Ker bo naročnik uporabljal delovni tok zgolj v vmesniku Alfresco Share, smo naloge definirali prikaz obrazcev nalog zgolj za ta vmesnik. Prikaz obrazcev se določi v datoteki `share-config-custom.xml`, znotraj mape `tomcat/shared/classes/web-extension`. Na sliki 12 lahko vidimo obrazec ene izmed oblikovanih nalog.

Definirati moramo prikaz za vsako nalogo posebej. Naloga je tukaj mišljena kot naloga, ki je definirana v modelu delovnega toka in ne kot vozlišče z nalogo v procesu delovnega toka. To pomeni da, če bi se v primeru, ko se pojavi neka naloga, ki se nahaja v več vozliščih v delovnem toku, bi obrazec teh nalog vsakič izgledal enako.

Prikaz posamezne naloge določimo znotraj oznake `<config>`. V tej oznaki moramo obvezno definirati dva atributa `evaluator` in `condition`. `Evaluator` ima vedno vrednost `task-type`, `condition` pa ima vrednost identifikatorja naloge, kot je ta definiran v modelu delovnega toka. Izjema je začetna naloga v procesu, kjer ima `evaluator` vrednost `string-compare`, vrednost `condition` pa je enaka kot identifikator, ki definira proces.

Znotraj oznake `<config>` sta dve oznaki; `<field-visibility>` in `<appearance>`. V `<field-visibility>` določimo znotraj `<show>` oznak atribut `id`, to je atribut, ki predstavlja identifikator lastnosti, za katere želimo, da se prikažejo. V oznaki `<appearance>` pa definiramo izgled teh lastnosti. Z oznako `<field>` definiramo posamezno polje. Na vsakemu izmed teh polj moramo določiti FreeMarker predlogo [3].

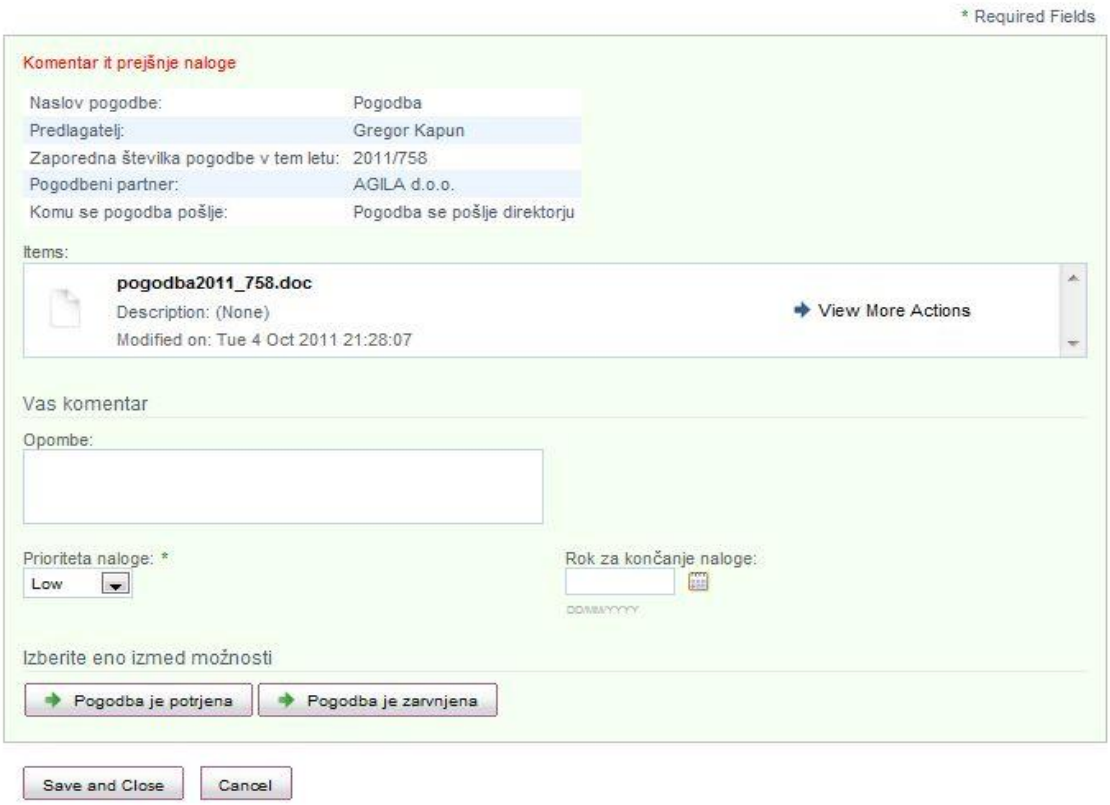
FreeMarker je brezplačen standard. Napisan v javi in je sposoben dinamično generirati html kodo iz posebne predloge shranjene s končnico `.ftl`, glede na vsebino polja in dodatne parametre, ki jih definiramo. FreeMarker datoteko na polju definiramo z oznako `<control>` znotraj oznake `<field>`, v oznaki `<control>` nadalje definiramo obvezne ali pa opcijske parametre z oznakami `<control-param>`.

Alfresco vsebuje kar nekaj `.ftl` datotek, po potrebi lahko vedno napišemo tudi lastno `.ftl` datoteko za prikaz polja. Nekaj primerov `.ftl` datotek Alfresca:

- checkbox.ftl definira polje s kljukico,
- date.ftl pole ki nam prikaže koledar za preprosto izbiro datuma,
- info.ftl prikaže vsebino nekega polja kot navaden tekst tako, da vsebine ni mogoče spreminjati,
- richtext.ftl prikaže polje z Obogatenim urejevalnikom besedila (ang. RTE Rich Text Editor) ,
- selectone.ftl prikaže spustni meni, kjer lahko izberemo eno izmed ponujenih možnosti in
- textarea.ftl in textfield.ftl definirata standardna `.html` polja za urejanje vsebine.

Poleg polj definiramo znotraj oznake `<appearence>` tudi oznake `<set>`. S temi oznakami lahko združujemo polja v vrste po 2 ali 3 polja, saj so polja privzeto postavljena eno pod drugim. Poleg tega lahko tudi določimo naslov nad skupino polj, ki služi za lažje razumevanje funkcionalnosti polj. Določeno polje dodamo v nek set tako, da v oznaki `<field>` definiramo vrednost atributa `set`. Ta služi kot identifikator neke oznake `<set>`.

Oznaka `<constraint-handlers>` znotraj oznake `<field>` vsebuje oznake `<constraint>`, ki definirajo dodatne omejitve na vsebini polja in preverijo, če je vsebina polja veljavna. Te omejitve delujejo na podoben način kot omejitve v modelu delovnega toka. Uporabne so ko želimo na neko polje dodati omejitev, za zgolj neko določeno nalogo. Tako lahko na primer naredimo polje v nalogi obvezno, četudi v modelu lastnost tega polja nima omejitve `MANDATORY`.



Slika 12: Primer oblikovanega obrazca iz delovnega toka za potrjevanje pogodb.

5. SKLEPNE UGOTOVITVE

Alfresco se je pokazal kot povsem primeren za naše potrebe. Razvili smo kvalitetno rešitev. Pokazal pa se je tudi kot zelo zahteven za učenje, saj smo za razvoj porabili veliko več časa kot smo pričakovali. Možen razlog za dolg razvoj je tudi to, da ima Alfresco kot brezplačna odprtokodna aplikacija manj literature in dokumentacije, kot jo imajo podobne komercialne aplikacije.

Med razvojem smo se srečali s kar nekaj problemi, ki so nam vzeli veliko časa. Največ časa smo porabili za integracijo Tesseract OCR v Alfresco in izvedbo pretvorbe vsebine. Več časa smo pri razvoju porabili tudi za samo konfiguracijo sistema, medtem kot za izdelavo delovnega toka ni bilo potrebnega toliko časa. Izdelava osnovne verzije delovnega toka je potekala dokaj hitro, več časa smo porabili za poenostavitve poteka delovnega toka za uporabnike.

Končna rešitev je robustna in hitro nadgradljiva. Če bi v prihodnje razvijali rešitev za drugega naročnika ali nove delovne tokove za istega naročnika, bi zaradi pridobljenega znanja razvoj potekal veliko hitreje. Tako se je na koncu odločitev za izbiro dokumentnega sistema Alfresco pokazala kot pravilna.

Možne so še dodatne izboljšave našega produkta. Ena izmed izboljšav bi bila aplikacija, v kateri bi z uporabo Alfrescovega API vmesnika dodajali in brisali uporabnike v sistemu. Trenutno moramo uporabnike v sistem obvezno dodajati z administracijskimi orodji. Lahko bi aplikacijo razvili tudi tako, da bi sama avtomatsko dodajala in brisala uporabnike v Alfrescu glede na spremembe v neki zunanji bazi o zaposlenih. V primeru dodatnih delovnih tokov bi lahko bilo potrebno tudi integrirati aplikacijo za fakture, recimo če bi imeli še delovni tok za račune. Dobra izbira za tako aplikacijo bi bil Ephesoft [5], ki je tako kot Alfresco spletna aplikacija, brezplačen, odprtokoden in napisan v Javi.

Končni sklep diplomske naloge je, da je danes za nekatere probleme bolje izbrati že obstoječe rešitve in se naučiti dela z njimi, kot pa razvijati lastne rešitve. Izbire je veliko, potrebno je biti le previden pri izbiri in skrbno pretehtati svojo odločitev.

KAZALO SLIK

LITERATURA

[1] Amita Bhandari, Munwar Shariff, Pallika Majmudar, Vinita Choudhary, Alfresco 3 Enterprise Content Management Implementation, Birmingham:Packt Publishing, 2009

[2] Avtentikacijski podsistemi v Alfrescu. Dostopno na: http://wiki.alfresco.com/wiki/Alfresco_Authentication_Subsystems

[3] David Caruana, Mike Farman, John Newton, Kevin Roast , Michael Uzquiano, Professional Alfresco, Indianapolis:Wiley Publishing, 2010

[4] Document Management Systems. Dostopno na: http://en.wikipedia.org/wiki/Document_management_system

[5] Ephesoft. Dostopno na: http://www.ephesoft.com/aboutus/about-ephesoft

[6] Internet Explorer tab 2 add-on. Dostopno na: https://addons.mozilla.org/en-US/firefox/addon/ie-tab-2-ff-36/

[7] JBPM Business Process Management Suite. Dostopno na: http://www.jboss.org/jbpm

[8] Jeff Potts, Alfresco Developer Guide, Birmingham:Packt Publishing, 2009

[9] Lastna transformacija vsebine v Alfrescu. Dostopno na: http://wiki.alfresco.com/wiki/Content_Transformations

[10] LDAP. Dostopno na: http://en.wikipedia.org/wiki/Ldap

[11] Lucene. Dostopno na: http://lucene.apache.org/

[12] Optical Character Recognition. Dostopno na: http://en.wikipedia.org/wiki/Optical_character_recognition

[13] Optical Mark Recognition. Dostopno na: http://en.wikipedia.org/wiki/Optical_Mark_Recognition

[14] Orodje JBoss jBPM Process Designer. Dostopno na: http://www.javaworld.com/javaworld/jw-05-2006/jw-0522-jbpm.html

[15] Poizvedovalna jezika Lucene in XPath v Alfrecu. Dostopno na: http://www.open-source-ecm.com/2008/11/tutorial-alfresco-search-with-lucene.html

[16] Tesseract OCR. Dostopno na: http://en.wikipedia.org/wiki/Tesseract_(software)