UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Primož Vranešič

RAZVOJ OBDELAVE ZAVAROVALNIH POLIC V ZAVAROVALNIŠKI POSLOVNI APLIKACIJI

DIPLOMSKO DELO VISOKOŠOLSKEGA STROKOVNEGA ŠTUDIJA

MENTOR: viš. pred. dr. Igor Rožanc

Ljubljana, 2008

KAZALO

POVZETEK

Diplomska naloga zajema predstavitev zavarovalniške poslovne aplikacije, katere namen je elektronska obdelava življenjskih zavarovanj. Namen je predstavitev tehnologij in orodij uporabljenih pri razvoju aplikacije, s poudarkom na tehnologijah za prikaz podatkov.

Poslovne aplikacije so pomembno področje informacijske tehnologije. Njihov namen je avtomatizirana obdelava podatkov, ki podjetju omogoča, da ne izgublja po nepotrebnem časa in denarja z ročno obdelavo podatkov, vse to pa povečuje produktivnost podjetja in manjša stroške. Zaradi tega razloga lahko v prihodnosti pričakujemo neprestano rast obsega obstoječih in razvoja novih poslovnih aplikacij v podjetjih.

V nalogi predstavimo osnovne pojme elektronskega poslovanja, poslovnih aplikacij in z njimi povezanih konceptov. Seznanimo se z orodji in tehnologijami, ki smo jih uporabljali pri razvoju poslovnih aplikacij. Za razvoj aplikacije smo uporabili veliko osnovnih tehnologij, s katerimi smo se seznanili že med študijem na fakulteti. Poslovna aplikacija temelji na modernem objektnem programskem jeziku Java. Podatke so hranjeni, obdelovani in prenašani z XML datotekami. Veliko je dela z podatkovnimi bazami, zato smo uporabili znanje UML-ja in SQL jezika.

Pred glavnim delom diplomske naloge smo predstavili poslovno aplikacijo, ki je narejena v programskem jeziku Java. S tem smo prikazali širši problem, to je delovanje aplikacije v zavarovalniškem informacijskem sistemu. Na ta problem je v končni fazi vezan ožji problem, problem obdelave tiskanja polic, ki smo ga podrobneje razčlenili kasneje.

Poslovno aplikacijo smo podrobno opisali. Opisali smo njeno funkcionalnost, način uporabe, prednosti in omejitve. Jedro aplikacije je zavarovalna polica, ta zajema pogoje pod katerimi je sklenjeno življenjsko zavarovanje med zavarovalnico in zavarovalcem police. Aplikacija večinoma obdeluje probleme finančne probleme, vsebuje pa tudi dele, ki skrbijo za izpisovanje podatkov. Ti so namenjeni bodisi zaposlencem podjetja bodisi strankam podjetja.

Namen aplikacije razvite v sklopu diplomske naloge je izpis podatkov o zavarovalnih policah. Izpisani podatki so rezultat predhodnjih obdelav začetnih vhodnih podatkov. V sklopu te obdelave smo naredili tudi opis podatkovnega modela, ki obsega načrt podatkovne baze, ter opis nekaterih entitetnih tipov ter njihovih relacij.

Za konec sledi primerjava uporabljenih tehnologij za prikaz podatkov in ugotovitve, pridobljene med pisanjem diplomske naloge.

1 UVOD

Živimo v času informacijskih tehnologij, ki omogočajo podjetjem učinkovitejše in cenovno ugodnejše rešitve za ravnanje z podatki. To jim omogoča lažje nastopanje na trgu in elektronsko poslovanje z drugimi podjetji, kar tudi povečuje učinkovitost in niža stroške. V ta namen se danes veliko vlaga v razvoj poslovnih aplikacij, v prihodnosti pa bo vlaganj še več.

1.1 Namen diplomske naloge

- na splošno predstaviti poslovne aplikacije in njihov razvoj,
- opisati tehnologije in rešitve za razvoj poslovnih aplikacij,
- predstaviti že obstoječo javansko poslovno aplikacijo,
- razviti in opisati primer nadgradnje zavarovalnišle poslovne aplikacije z obdelavo za prikaz podatkov ter
- primerjati tehnologijo, ki je bila uporabljena pri razvoju aplikacije z drugimi možnimi tehnologijami.

1.2 Cilji diplomske naloge

- spoznati celoten postopek razvoja poslovne aplikacije,
- ugotoviti prednosti in slabosti predstavljenih tehnologij, rešitev in postopkov, ki smo jih uporabili pri razvoju poslovne aplikacije ter
- izdelati aplikacijo za zajem in predstavitev podatkov o življenjskih zavarovanjih.

1.3 Struktura diplomske naloge

Najprej bomo predstavili osnovne značilnosti poslovnih aplikacij in elektronskega poslovanja. Podrobneje bomo predstavili razvoj zavarovalniške poslovne aplikacije. Predstavili bomo metodologijo in programske pakete, ki so uporabljeni pri njenem razvoju, njeno funkcionalnost v podjetju, njeno uporabo in cilje.

V drugem poglavju bomo predstavili tehnologije, ki so uporabljene pri razvoju in omenili današnje trende in smernice pri uporabi novih tehnologij.

Tretje poglavje bo namenjeno podrobnemu opisu razvoja zavarovalniške poslovne aplikacije. Opisali bomo specifikacije, poslovni model in celotno poslovno aplikacijo. Omenili bomo nekatere programske pakete, ki so uporabljeni pri aplikaciji.

V četrtem poglavju je predstavljena sama poslovna aplikacija za podporo življenjskih zavarovanj. Skozi njen opis in namen bomo dobili občutek, kaj je v splošnem naloga poslovne aplikacije in na kak način je realizirana.

Peto poglavje bo jedro diplomske naloge. Namenjeno je podrobnemu opisu nadgradnje zavarovalniške poslovne aplikacije z obdelavo za tiskanje zavarovalnih polic.

V zadnjem poglavju bomo primerjali uporabljeno tehnologijo, pri razvoju obdelave tiskanja polic z ostalimi tehnologijami, ki bi lahko bile uporabljene za razvoj te obdelave.

2 POSLOVNE APLIKACIJE IN ELEKTRONSKO POSLOVANJE

V tem poglavju bomo spoznali osnovne pojme in principe elektronskega poslovanja in ideje uporabe poslovnih aplikacij. Na kratko bomo opisali zgodovino elektronskega poslovanja in uporabe poslovnih aplikacij. Skušali bomo ugotoviti njihovo vlogo danes in obete v prihodnosti.

2.1 Terminologija

Podatek je predstavitev dejstva, koncepta ali instrukcije na način, ki je primeren za komunikacijo, interpretacijo ali obdelavo s strani človeka ali stroja.

Informacija je pomen, ki ga človek pripiše podatkom (*interpretacija*).

Informacijski sistem je množica medsebojno odvisnih komponent (strojne opreme, programske opreme, ljudi), ki zbirajo, obdelujejo, hranijo in porazdeljujejo podatke in s tem podpirajo delovne procese v organizaciji.

Poslovna inteligenca (angl. business intelligence, krat. BI) je poslovno ravnateljski izraz, ki opisuje aplikacije in tehnologije, ki se uporabljajo za zbiranje, dostop in analizo podatkov in informacij o delovanju podjetij.

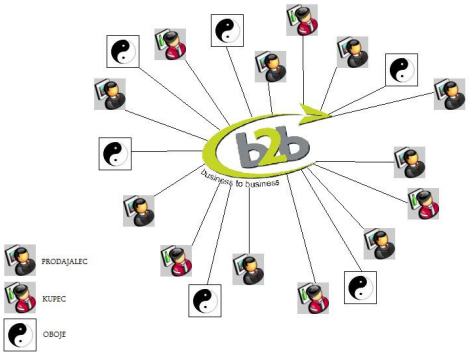
Poslovna aplikacija je aplikacija za uporabo v poslovnem procesu.

Elektronsko poslovanje (angl. Electronic business) se nanaša na izvajanje poslovnih transakcij, na upravljanje odnosov s strankami in na komuniciranje tako znotraj podjetja kot med različnimi podjetji, kupci in državno upravo [1].

B2B poslovanje (angl. Business-to-Business) je pojem, ki se uporablja za elektronsko poslovanje med podjetji [1] (slika 1).

B2C poslovanje (angl. Business-to-Customer) pomeni poslovanje med podjetji in končnimi kupci oziroma potrošniki [1].

B2E poslovanje (angl. Business-to-Employee) pomeni poslovanje podjetja z njegovimi zaposlenci [1].



Slika 1: Poslovanja med podjetji kupci in podjetji prodajalci

2.2 Informacijski sistem, poslovne aplikacije in elektonsko poslovanje

Preden se lotimo uporabe poslovne aplikacije, se moramo zavedati ozadja, zaradi katerega so poslovne aplikacije nastale. Poslovne aplikacije so se razvile kot del informacijskega sistema z namenom izboljšanja učinkovitosti obdelave podatkov v danem informacijskem sistemu.

Informacija se poraja iz podatkov v sistemu, ki ga imenujemo informacijski sistem. Informacijski sistem zbira, obdeluje, analizira, shranjuje in posreduje informacije za določen namen. Vsak sistem ima vhode in izhode. Vhod v informacijski sistem predstavljajo bodisi podatki ali navodila, kaj naj sistem dela. Izhod iz sistema pa so rezultati obdelav ali analiz, ki so posredovani uporabnikom [2].

Sestavine informacijskih sistemov so [2]:

- strojna oprema,
- programska oprema,
- podatki,
- postopki in
- ljudje.

Informacijske sisteme delimo na [3]:

- formalne in
- neformalne.

Druga delitev razmejuje informacijske sisteme na [3] :

- računalniško podprte in
- računalniško nepodprte.

Glede na namen uporabe lahko delimo informacijske sisteme na sledeče vrste [3]:

- Sistemi za upravljanje delovnih procesov (angl. Workflow System, krat. WS),
- transakcijski informacijski sistemi (angl. Transaction Processing System, krat. TPS),
- upravljalno-ravnateljevalni informacijski sistemi (angl. Management Information System, krat. MIS),
- odločitveni sistemi (angl. Decision Support System, krat. DSS),
- ekspertni sistemi (angl. Expert System, krat. ES).

V splošnem informacijske sisteme delimo na [2]:

- **Transakcijsko informacijske sisteme**.
  TPS zbira, obdeluje in hrani podatke o ponavljajočih vsakodnevnih poslovnih transakcijah (poslovnih dogodkih, npr. sprejem naročil, izstavljanje računov).V vsakem trenutku mora zagotavljati točne podatke za potrebe uporabnikov znotraj in zunaj podjetja. Običajno TPS ustvarjajo veliko količino podatkov, ki so osnova za operativno odločanje (npr. ali se naročene in prejete količine ujemajo). Z nedelovanjem se lahko prekine izvajanje temeljnih aktivnosti v podjetju. Je vir podatkov za vse “višje” nivoje IS in mora biti zanesljiv, varen in robusten.

- **Poslovno inteligenčni sistemi (BI)**.
  BI zagotavlja informacije, ki so potrebne za upravljanje organizacije. Informacije, ki jih zagotavlja, omogočajo sprejemanje odločitev, ki omogočajo bolj učinkovito delovanje organizacije. Osnovne značilnosti BI so, da se ukvarja z načrtovanjem in nadzorom organizacije. Vhod predstavljajo podatki, ki so shranjeni v TPS, primerja dejansko stanje z načrtovanim, proizvaja poročila, rezultati se uporabljajo znotraj organizacije, podatki so agregirani, sumirani in ne nujno zelo natančni.

Razvoj informacijskega sistema lahko strnemo v 4 medsebojno povezane faze [3]:

- analiza (tu gre predvsem za analizo uporabniških zahtev),
- načrtovanje (zasnova),
- izvedba (programiranje, takšna in drugačna realizacija aplikacij) in
- vpeljava (izobraževanje končnih uporabnikov glede uporabe IS sistema).

Kot je razvidno so poslovne aplikacije del informacijskega sistema, ki omogočajo izvajanje osnovnih dejavnosti.

Elektronsko poslovanje je del informacijskega sistema, ki ga izvajajo poslovne aplikacije znotraj informacijskega sistema. Najpomembnejši del elektronskega poslovanja je B2B poslovanje, katerega začetki segajo slaba štiri desetletja nazaj. Z velikimi vlaganji v informacijsko tehnologijo in v usposobljenost kadrov so podjetja dosegla visoko stopnjo avtomatizacije internega poslovanja, vendar je njihova komunikacija z poslovnim okoljem še vedno temeljila na klasičnem načinu. Za medsebojno komunikacijo se je še vedno uporabljala navadna pošta, pri čemer je obstajala nevarnost napak. Z prihodom interneta so postale fizične razdalje med podjetji zanemarljive. Znotraj informacijskih sistemov so se razvile poslovne aplikacije, ki so avtomatizirale medsebojno komunikacijo med podjetji. Vse to je doprineslo k zmanjševanju stroškov in napak [1]. Današnji trendi spodbujajo uporabo XML jezika v medsebojni komunikaciji, kar bo v prihodnje pomenilo poenotenje in izdelavo novih (izboljšanje starih) orodij za delo z XML formati, npr. razvoj transformacijskih orodji kot je že danes XSLT.

3 TEHNOLOGIJE IN ORODJA

V naslednjem poglavju bomo na kratko predstavili tehnologije in orodja, ki smo jih uporabili pri razvoju obstoječe javanske aplikacije in pri nadgradnji zavarovalniške poslovne aplikacije v sklopu diplomske naloge.

3.1 Podatkovni strežniki

3.1.1 Sistem za upravljanje podatkovne baze IBM DB2

IBM DB2 je družina upravljalskih sistemov za relacijske podatkovne baze, ki jih IBM nudi za uporabo na številnih različnih platformah. Raziskave kažejo, da je DB2 poleg Oracla vodilni na tržišču podatkovnih strežnikov, prav tako pa je tudi njegova zmogljivost. Ta lastnost se še posebej kaže pri velikih relacijskih podatkovnih bazah, čemur je DB2 tudi primarno namenjen [4,5].

IBM ponuja DB2 v številnih različicah, tako da kupcem omogoča, da ne plačujejo za tisto funkcionalnost, ki jo ne bodo rabili. Ena izmed bolj popularnih različic je prav gotovo DB2 UDB (angl. Universal Database). Ta združuje relacijsko objektno usmerjeno podatkovno bazo, optimizirane povpraševalne tehnike in paralelno procesiranje. Poleg že naštetega nudi tudi aplikacije z grafičnim uporabniškim vmesnikom (angl. Graphical User Interface, krat. GUI) za upravljanje in podporo Javi (angl. Java Database Conectivity, krat. JDBC). Poleg DB2 UDB poznamo še vrsto drugih različic: DB2 DWE (angl. Data Warehouse Enterprise Edition), DB2 PE (angl. Personal Edition), DB2 ESE (angl. Enterprise Server Edition) in druge [5].

DB2 lahko upravljamo tako iz ukazne vrstice kot tudi z vmesnikom GUI, ki je izveden kot klient aplikacija v Javi in ima poleg številnih uporabnih funkcij tudi čarovnike (angl. wizards) za manj izkušene uporabnike. Ena izmed takih aplikacij je IBM DB2 Control Center, ki ga bomo podrobneje spoznali v nadaljevanju. Upravljanje iz ukaznega načina zahteva nekaj več znanja, vendar omogoča večjo prilagodljivost in uporabo skript [6].

Obstaja množica vmesnikov API, ki jih DB2 podpira: .NET CLI, Java, Python, Perl, PHP, Ruby on Rails, C, C++, REXX, PL/I, COBOL, RPG, FORTRAN in še veliko drugih. DB2 prav tako podpira razvoj v Eclipse in Visual Studio .NET razvojnih okoljih.

3.2 Označevalni jeziki

3.2.1 Označevalni jezik XML

XML (angl. Extensible Markup Language) je razširljiv označevalni jezik, ki ga dandanes pogosto srečamo na internetu. Omogoča nam, da na preprosti način opišemo strukturirane podatke. XML je prisoten na veliko področjih računalništva, še posebej na področju komuniciranja aplikacij in strežnikov. Lahko ga tudi razširimo, saj si lahko sami izmislimo imena etiket (angl. tag), s tem na zelo preprost način naredimo strukturo podatkov lahko berljivo in razumljivo. Zaradi te pregledne strukture, je zelo uporaben za povezovanje med sistemi [7] (slika 2).

XML je razdeljen na 3 dele:

- podatkovni del: vanj shranimo podatke v izbrani obliki z željenimi etiketami,
- opisni del: opisuje kaj določena etiketa predstavlja,
- predstavitveni del: z njim oblikujemo izpis podatkov.

Prednosti XML-ja in njegove uporabe so:

- Njegova sintaksa je razumljiva tako za človeka kot tudi za računalnik.
- Podpira UNICODE standard, kar omogoča predstavitev vseh informacij v katerem koli jeziku.
- Zmožen je predstaviti večino računalniških podatkovnih struktur: zapise, sezname in drevesa.
- Samoopisnost omogoča, da XML vsebuje poleg konkretnih podatkov tudi njihovo strukturo.
- Robustnost in logična preverljivost temeljita na mednarodnih standardih, zato je XML velikokrat uporabljen pri shranjevanju dokumentov.
- Hierarhična struktura je primerna za večino tipov dokumentov.
- Zaradi platformne neodvisnosti je XML imun na spremembe v tehnologiji.

Šibkosti jezika XML pa so:

- Njegova sintaksa je preobširna, kar pogosto vpliva na njeno berljivost in učinkovitost v aplikacijah.
- Razčlenjevalniki (angl. parser) morajo biti izvedeni tako, da lahko preverjajo vgnezdene strukture in preverjajo morebitne napake v sintaksi oziroma v samih podatkih. To lahko vodi do visokih stroškov razvoja.
- Modeliranje nehierarhično urejenih podatkovnih struktur zahteva dodaten napor.
- Prenašanje XML-ja v relacijsko ali objektno usmerjene paradigme (npr. podatkovne baze) je pogosto zamudno in neučinkovito.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ARTICLES SYSTEM
"D:\Projects\Clients\XML\Contents\Temp\articlelist.dtd">
<?xml-stylesheet type="text/xsl"
href="D:\xmltohtml.xslt" ?>
<ARTICLES>
    <ARTICLE>
        <ARTICLEDATA>
            <TITLE>XML Demystified</TITLE>
            <AUTHOR>Jaidev</AUTHOR>
        </ARTICLEDATA>
    </ARTICLE>
    <ARTICLE>
        <ARTICLEDATA>
            <TITLE>XSLT Demystified</TITLE>
            <AUTHOR>X S Cel Tea </AUTHOR>
        </ARTICLEDATA>
    </ARTICLE>
    <ARTICLE>
        <ARTICLEDATA>
            <TITLE>C# Demystified</TITLE>
            <AUTHOR>Aleksey N</AUTHOR>
        </ARTICLEDATA>
    </ARTICLE>
</ARTICLES>
```

Slika 2: Primer xml datoteke

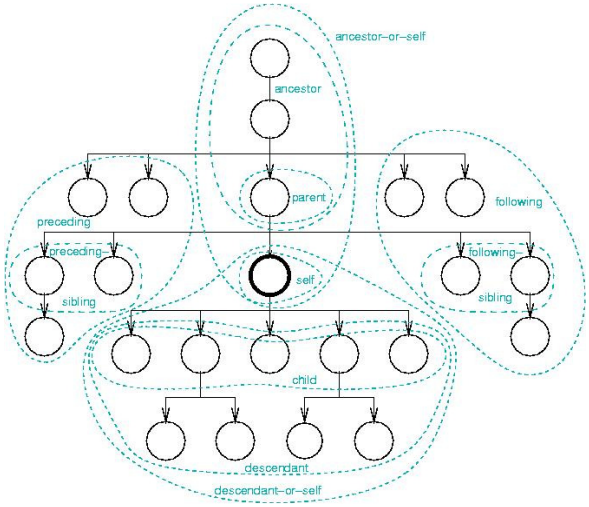
3.2.2 Označevalni jezik XSL

XSL (angl. Extensible Staylesheet Language) [8] je označevalni jezik, ki temelji na XML jeziku in je razvit iz strani konzorcija W3C (World Wide Web Consortium). Razvit je tako, da odpravi nekatere slabosti XML-a. Pri XML-u se ne uporablja v naprej definiranih etiket, XSL pa uporablja natanko predefirane etikete z namenom, da z vsako etiketo poda navodila jeziku za transformacijo XSLT. XSL dejansko opisuje, kako naj bo XML prikazan (slika 3).

Slika 3: XSL stilni list

3.2.3 Jezik XPath

XPath je jezik, ki se uporablja za iskanje informacij v XML dokumentu. Uporablja se za navigacijo skozi elemente in atribute XML dokumenta, preko relacij oče-sin (slika 4). Vsebuje preko 100 vgrajenih funkcij. Med drugim ima funkcije za delo z nizi, številskimi podatki, datumi, logičnimi vrednostmi in raznimi podatkovnimi strukturami. Jezik je narejen za uporabo skupaj z transformacijskim jezikom XSLT [9].



Slika 4: Relacije znotraj XML dokumenta

3.2.4 Označevalni jezik XSL-FO

XSL-FO, XSL[12] oblikovni elementi (angl. XSL Formating Objects) je označevalni jezik, ki se ga uporablja kot vmesno stopnjo za generiranje drugih tipov dokumentov. Je del XSL jezika. Glavna ideja je, da se XML dokument preko XSLT transformacije oblikuje v XSL-FO dokument. Ko je ta dokument kreiran se ga preda aplikaciji imenovani FO procesor, ta pa ga pretvori najpogosteje v PDF ali PS format, lahko pa tudi v kaj drugega. XSLT jezik je bil razvit ravno za XSL-FO transformacije, vendar je bil nato razširjen za podporo bolj splošnim transformacijam. Glavni označevalni koncept je prevzet iz CSS-ja.

Prednosti XSL-FO-ja in njegove uporabe so[12]:

- Ker je v uporabi XML jezik, so za kreiranje XSL-FO potrebne le XSLT transformacije.
- Večina funkcionalnosti je prevzeta iz CSS, zato je relativno preprost za uporabo.
- Obstaja veliko odprtokodnih implementacij, zaradi česar so z uporabo orodja majhni stroški.
- Kreiran je bil za delo z različnimi svetovnimi jeziki in ima za njih dobro lokalno podporo.

Glavna težava jezika XSL-FO je ta, da je bil v osnovi kreiran za izdelavo oblikovno nezapletenih dokumentov, navodil, poslovnih dokumentov in faktur. Za oblikovno bolj zapletene dokumente pa se ne obnese dobro, ker veliko število oblikovnih elementov ni bilo podprto v implementaciji [12].

3.3 Transformacijska orodja XSLT

Orodje XSLT [10] (angl. Extensible Staylesheet Language Transformation), se uporablja za pretvorbo XML dokumenta v kakršen koli drug dokument, lahko XML, XTML, tekstni dokument, ipd. Jezik XSLT je sposoben narediti takšno pretvorbo z pomočjo vmesnih XSL datotek, lahko pa tudi z uporabo XPath-a.

Stilni list je dokument XML, ki uporablja elemente iz slovarja XSLT za opisovanje pretvorb. Element vsakega stilnega lista (angl. stylesheet) je element `<xsl:stylesheet>`, katerega vsebina je množica pravil za opis transformacije, ki se naj izvedejo. Vsako pravilo v stilnem listu vsebuje pridruženi vzorec jezika XPath, ki se ujema z vozlišči v izvornem dokumentu, nad katerim se naj to pravilo uporabi. Vsako pravilo, poimenovano model (angl. template), je predstavljeno z elementom `<xsl:template>` z atributom `match="pattern"` in njegovo pridruženo vrednostjo jezika XPath.

Klasični XSLT model za pretvorbe predstavlja (slika 5):

- enega ali več XML izvornih dokumentov,
- enega ali več XSLT stilnih listov,
- procesor za transformacijo XSLT ter
- enega ali več izhodnih dokumentov v poljubnih formatih.

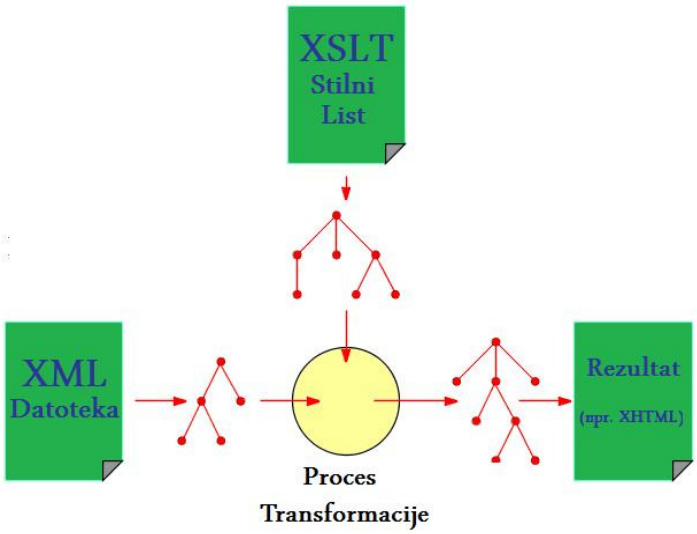
Operacije pri pretvorbi so naslednje [11]:

- Ciljni elementi in atributi v modelu se kreirajo v ciljnem drevesu. Ciljni elementi in atributi, ki niso iz imenskega prostora XSLT, se obravnavajo kot literali in se nespremenjeni pojavijo v ciljnem drevesu.

- Vsi modeli z atributno vrednostjo oblike `{XPathExpr}`, ki so vsebovani znotraj literalnih atributivnih vrednosti, se nadomestijo z vrednostjo njihovega izraza XPath.

- Vsi elementi v imenskem prostoru XSLT se obdelujejo v zaporedju dokumentov. Element `<xsl:value-of>` se obdela in se nadomesti s tekstovnim vozliščem, ki vsebuje vrednost izraza XPath v njegovem atributu `select`.

Osnovna operacija je naslednja. Ko se vozlišče v izvornem drevesu ujema z vzorcem pravila, se vsebina tega pravila kreira v ciljnem drevesu. S pomočjo izvornega drevesa in stilnega lista izvede procesor XSLT pretvorbo, opisano s pravili v stilnem listu. V okviru obdelave listov vozlišč izvornega drevesa se kreirajo fragmenti ciljnega drevesa. Pri obdelavi tekočega vozlišča se upoštevajo vsa možna pravila, ki se z vozliščem ujemajo, nato pa se izbere in uporabi najustreznejše pravilo. Obdelava se prične z listo vozlišč, ki obsega le koren dokumenta. Procesor poišče model, ki se ujema s tem korenskim vozliščem (tipično pravilo, ki se ujema z `match="/"`) in kreira vsebino modela v ciljnem drevesu ob upoštevanju treh osnovni procesnih korakov, opisanih v predhodnih vrsticah. Če model vsebuje elemente iz imenskega prostora XSLT, ki izberejo naslednja vozlišča za procesiranje, potem se postopek rekurzivno nadaljuje, dokler niso obdelana vsa vozlišča. Ko je procesiranje zaključeno, predstavlja ciljno drevo ciljni dokument, proizveden s to pretvorbo.

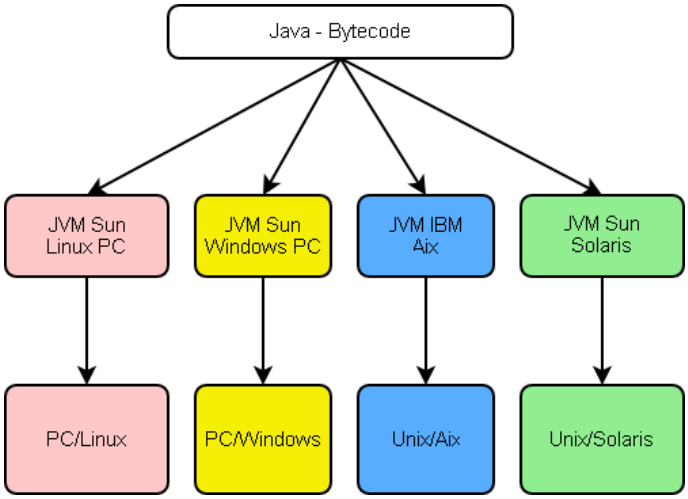


Slika 5: Postopek transformacije z XSLT

Glavna slabost XSLT je ta, da zaradi rekurzivne narave preiskovanja volišč lahko pride do zelo počasne pretvorbe pri XML datotekah, ki vsebujejo veliko število podatkov.

3.4 Programski jezik Java

Java je objektno usmerjen programski jezik, ki so ga razvili v podjetju Sun v začetku 90-ih. V nasprotju z konvencionalnimi programskimi jeziki, ki so v splošnem narejeni tako, da izvorno kodo z prevajalnikom prevedejo v strojno kodo ali pa se izvorna koda tolmači že med izvajanjem, je pri Javi ta postopek drugačen in to je ena njenih posebnosti. Izvorno kodo prevedemo v vmesno kodo (angl. bytecode), ki jo nato poganja Javin navidezni stroj (angl. Java Virtual Machine). S tem dosežemo neodvisnost od platforme (slika 6), kar pomeni, da se programi v Javi lahko izvajajo brez ponovnega prevajanja na vseh platformah.



Slika 6: Vmesna koda se z pomočjo JVM lahko uporabi na vseh platformah

Sintaksa je zelo podobna sintaksi, ki jo imata C in C++, vendar je enostavnejša in programerju prijaznejša za uporabo. Pogosto se zgodi, da ljudje povezujejo Javo s skriptnim jezikom JavaScript, vendar razen podobnosti v sintaksi in imenu nimata skupnih lastnosti.

Programerji, ki so ustvarili Javo, so si zadali pet glavnih ciljev:

- Objektno usmerjeno programiranje
- Neodvisnost od platforme
- Vgrajena podpora za mrežno in medmrežno komunikacijo
- Izvajanje aplikacij oddaljenih sistemov na varen način
- Enostavnost uporabe

Java je danes zelo popularen programski jezik. Kot pomembno pozitivno lastnost moramo omeniti še relativno enostavnost uporabe in to, da ima Java ogromno vgrajenih funkcij in procedur, ki lahko bistveno skrajšajo razvoj aplikacije. Samodejno sproščanje pomnilnika (angl. Automatic Garbage Collector) je prav tako lastnost, ki jo pogrešamo pri marsikaterem drugem programskem jeziku. Gre za to, da Java med samim izvajanjem pobriše podatke v pomnilniku, ko jih ne potrebuje več. S tem je olajšano delo programerja, ki mu za to ni potrebno skrbeti [13].

V splošnem ločimo štiri vrste aplikacij, ki jih lahko razvijemo v Javi:

- **Samostojne aplikacije** so tiste, ki uporabljajo samo Javine vire in njen virtualni stroj JVM. Navadno so to aplikacije brez grafičnega uporabniškega vmesnika, ki jih poženemo kar iz ukazne vrstice.
- **Apleti** so programi, ki so vloženi v druge aplikacije, navadno v spletno stran in se nato prikazujejo v spletnih brskljalnikih.
- **Servleti** so strežniške aplikacije, ki so namenjene izvedbi zahtev, ki jih dobijo od uporabnikov preko HTTP zahtev.
- **Okenske aplikacije** so tiste, ki za svoje delovanje uporabljajo Javine knjižnice za grafične vmesnike: Swing, AWT, SWT,...

Brez dvoma je Java, podobno kot večina programskih jezikov danes izgubila neposreden stik z platformo. Prav na tem področju je izgubila morda še nekaj več zaradi njene arhitekture in virtualnega stroja JVM, ki sicer dovoljuje platformno neodvisnost, vendar sočasno tudi onemogoča stik z njo. Kritik je Java deležna tudi na področju zmogljivosti in porabe spomina, ki je znatno večja kot pri jezikih C in C++.

3.5 Pristopa pri razvoju aplikacij

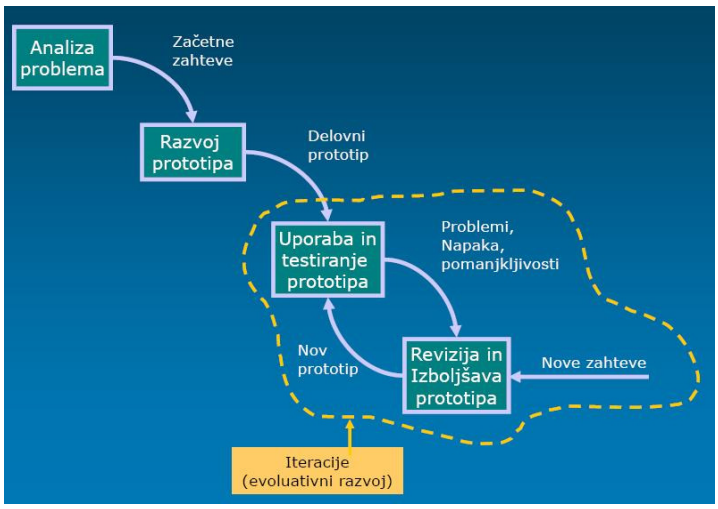
Za razvoj informacijskih sistemov, ki postajajo vedno bolj zahtevni, so v zadnjih desetletjih nastale številne metodologije, katerih namen je vodenje razvijalcev skozi predpisane faze, opredelitev potrebnih tehnik, ki se uporabljajo v posameznih fazah razvoja in zagotavljanje nadzora nad projektom izgradnje informacijskega sistema tako v časovnem kot tudi vsebinskem vidiku. Metodologije so do pojava objektno orientiranih metodologij ločevale podatkovni in procesni vidik. S pojavom objektno orientiranih metodologij se podatkom priključijo operacije za manipulacijo z njimi tako, da je kontrola nad spreminjanji podatkovnih struktur zagotovljena na enem mestu – znotraj objekta.

Obstaja več različnih pristopov, dostikrat pa se v praksi uporabi tudi kombinacija več pristopov. Dva pristopa smo podrobneje opisali, ker sta bila pri razvoju aplikacije, ki je opisana v diplomski nalogi, največ uporabljena.

3.5.1 Evolucijski razvoj prototipov

Evolucijski razvoj prototipov [14] (angl. Evolutionary Prototyping) je pristop za razvoj aplikacij. Osnovna ideja pristopa je v postopni nadgradnji prototipa do te mere, da na koncu dobimo delujoči program. Spremembe izvedemo glede na povratne informacije končnih uporabnikov. Navadno razvijemo uporabniški vmesnik in iz tega postopoma celotno aplikacijo (slika 7).

Prednost tovrstnega pristopa je v večji učinkovitosti (skrajšamo čas razvoja in izboljšamo vidnost napredka), večji možnosti za uspeh v prvem poskusu in na dolgi rok. Evolucijski razvoj prototipov uporabimo takrat, ko so določeni deli aplikacije slabo definirani in rešitev ni popolnoma jasna. Vendar uporaba tovrstnega pristopa tudi povečuje tveganje zaradi neučunkovite porabe časa, nerealističnih pričakovanj glede izvedbe, cene, slabega načrtovanja in težjega vzdrževanja.



Slika 7: Evolucijski razvoj prototipov

3.5.1 Tehnika miniaturnih mejnikov

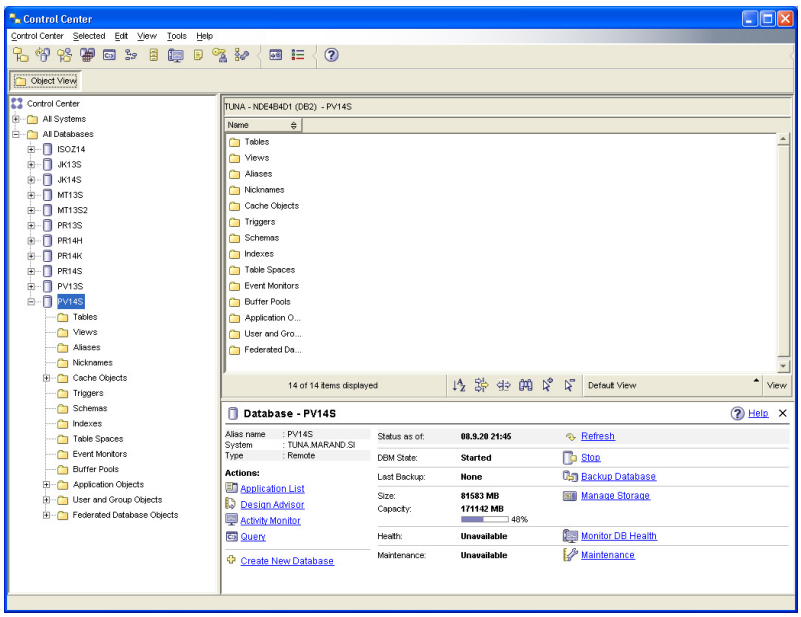
Tehnika miniaturnih mejnikov (angl. Miniature Milestones) je prav tako pristop za razvoj aplikacij. Pristop poudarja spremljanje razvoja aplikacije v kratkih intervalih (poznavanje stanja razvoja v vsakem trenutku). Pri razvoju programa si zastavimo več mejnikov, ki jih moramo izpolniti. Vsak mejnik predstavlja večjo spremembo ali dodano funkcionalnost v aplikacije in pa datum, do kdaj naj bi bila sprememba izvedena.Tako odpravimo nenadzorovane in nezaznane zamude pri razvoju aplikacije, saj vedno vemo, kako se razvoj aplikacije pokriva z načrtom [14].

Prednosti miniaturnih mejnikov so predvsem manjše število napak, manjša količina kodiranja, večje zaupanje uporabnikov ter razvijalcev in za 40-85% zmanjšano tveganje pri razvoju [14].

Z uporabo miniaturnih mejnikov si razvijalci postavijo lastne mejnike že v zgodnji fazi razvoja aplikacije. Mejniki naj bi bili posamezni deli aplikacije, ki naj ne bi zajemali več kot dan ali dva dela. Prav tako naj bi mejniki zajemali vse aktivnosti in bili označeni z "opravljeno" ali "neopravljeno".

3.6 Upravljanje podatkovnih baz IBM DB2 Control Center

IBM DB2 Control Center je orodje, ki je napisano v programskem jeziku Java in je na voljo bodisi kot samostojna aplikacija ali pa kot spletna aplikacija. Orodje je namenjeno upravljanju DB2 podatkovnega strežnika preko grafičnega uporabniškega vmesnika (slika 8), kar zelo olajša delo v primerjavi z delom v ukazni vrstici [15].



Slika 8: DB2 Control Center – uporabniški vmesnik

Z njim lahko:

- dodajamo DB2 UDB sisteme, primerke (angl. instance) in podatkovne baze ter njihove objekte
- upravljamo z podatkovnimi bazami in njihovimi objekti: ustvarjamo, odstranjujemo ter urejamo podatkovne baze, tabele, poglede, indekse, prožilce (angl. triggers) in sheme
- upravljamo z uporabniki DB2 podatkovnega strežnika in njihovimi pravicami
- upravljamo s podatki: nalagamo, izvažamo, uvažamo, reorganiziramo in si na podlagi obstoječih podatkov lahko ustvarimo tudi statistiko
- izvajamo preventivna vzdrževalna dela za obnovitev podatkov v tabelah
- upravljamo s povezavami na DB2 bazo

3.7 Razvojno okolje

Razvojno okolje (angl. Integrated Development Environment, krat. IDE) je v računalništvu orodje, ki nudi programerjem veliko možnosti za razvoj programov. Ponavadi IDE združuje urejevalnik izvorne kode, prevajalnik ali interpreter (ali oboje) in ponavadi tudi razhroščevalnik. IDE tipično predstavlja program, v katerem poteka ves razvoj. Ta program nudi veliko možnosti za spreminjanje, prevajanje, dodeljevanje avtorskih pravic in razhroščevanje programov.

Uporaba vizualnega programiranja se stalno povečuje. Vizualni IDE-ji omogočajo uporabnikom, da oblikujejo nove programe kar s premikanjem programskih gradnikov, ali da z uporabo vozlišč kreirajo diagrame, ki jih je nato možno prevesti ali tolmačiti [16].

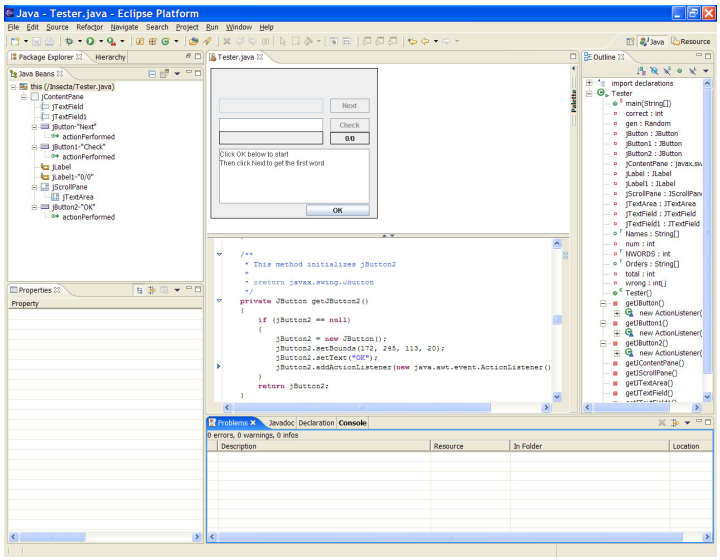
Prednosti IDE-jev so:

- Barvanje kode
  - večja preglednost
  - lažje razhroščevanje
- Samodejno dopolnjevanje kode
  - imena metod, spremenljivk in razredov
  - for zanke, if stavki in drugi jezikovni konstrukti
- Upravljanje z viri
  - boljši pregled nad datotekami in mapami
  - vgrajeni urejevalniki različnih formatov
- Orodja za predogled
  - predogled izdelka (npr. v spletnem brskalniku)
- Razhroščevanje in manipulacija kode
  - sprotno zaznavanje napak v kodi
  - samodejen in ročen refactoring
- Močen iskalni pogon
  - zmogljiv
  - regularni izrazi

3.7.1 Razvojno okolje Eclipse

Primer razvojnega okolja je program Eclipse. Arhitekturno Eclipse uporablja vstavke (angl. plug-ins). Vstavki predstavljajo funkcionalnost Eclipsa, vključno z sistemom za izvajanje (angl. runtime system), ki temelji na vstavku poimenovanem Equinox. Edina izjema je Eclipsovo izvajalno jedro (angl. runtime kernel).

Eclipse (slika 9) je bil prvotno namenjen za razvijalce v Javi in vsebuje Javanska razvojna orodja (angl. Java Development Tools, krat. JDT). Sistem vstavkov omogoča Eclipsu prilagodljivost, med drugim možnost, da se ga razširi za uporabo pri drugih programskih jezikih, kot na primer za C ali Python. Omogoča tudi delo z omrežnimi aplikacijami, upravljanje podatkovnih baz in nudi orodja za delo z verzijami [17].



Slika 9: Razvojno okolje Eclipse

3.7.2 Orodje za upravljanje z verzijami in TortoiseSVN

Orodje za upravljanje z verzijami (angl. Subversion, krat. SVN) ima za nalogo hraniti trenutne in pretekle verzije datotek, izvorne kode, web strani in dokumentacije. Je naslednik sistema za sočasno delo z verzijami (angl. Concurrent Versions System, krat. CVS).

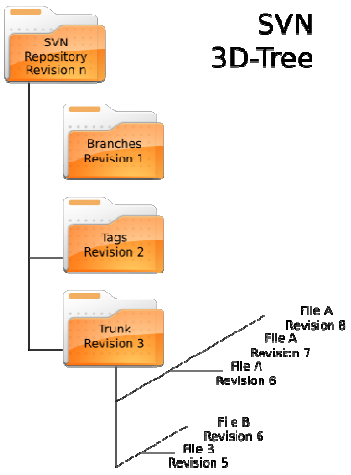
SVN je dobro poznano orodje v odprtokodni skupnosti in je uporabljeno na mnogih odprtokodnih projektih, kot so Apache, KDE, GNOME, Free Pascal, GCC.

Notranjo zgradbo sistema SVN predstavlja več knjižnic, ki so urejene kot plasti (angl. layers). Vsaka izvaja specifično nalogo in omogoča razvijalcem, da kreirajo lastna orodja na željenem nivoju specifikacije in kompleksnosti.

Plasti sistema SVN:

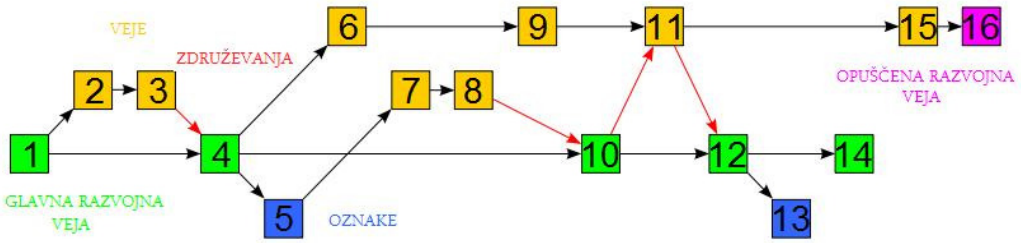
- plast datotečnega sistema se nahaja na najnižjem nivoju. Implementira verzioniran datotečni sistem, ki hrani uporabniške podatke.
- plast repozitorijev, ki je zgrajena okoli plasti datotečnega sistema, vključuje veliko pomožnih funkcij, ki se uporabljajo ob izvedbi določenih akcij nad datotečnim sistemom, kot so recimo akcija potrditve (angl. commit). Plast repozitorijev skupaj z plastjo datotečnega sistema predstavlja datotečni vmesnik (angl. filesystem interface).
- plast mod_dav_svn je plast, ki definira mrežne protokole za delo z Apache 2.
- plast repozitornega dostopa (angl. repository access layer) skrbi za dostop do repozitorija tako lokalno, kot omrežno. Ta plast poimenuje repozitorije kot enolične označevalnike vira (angl. Universal Resource Locator, krat. URL).
- Najvišji nivo predstavlja plast klienta. Ta plast abstraktira dostop do repozitorija in nudi standardne funkcionalnosti, kot so prijava uporabnika, primerjanje verzij, ipd.

SVN datotečni sistem lahko opišemo kot trodimenzionalni datotečni sistem (slika 10). Samo drevo direktorija je dvodimenzionalno, medtem ko mu tretjo dimenzijo dodajo revizije. Vsaka revizija v SVN sistemu ima svoj lasten koren (angl. root), ki se uporablja za dostop do vsebine dane revizije [18].



Slika 10: SVN 3D drevo

SVN uporablja meddatotečni model vejitev prevzet iz orodja Perforce. Ta predpisuje uporabo vejitev (angl. branching) in uporabo oznak (angl. tagging) (slika 11). Vejitev je zmožnost, da se izolira spremembe nad delom aplikacije v neodvisni smeri razvoja glede na preostalo aplikacijo. Označevanje pa je zmožnost povezovanja dodatne informacije kot naprimer oznako verzije nad določeno revizijo. Praviloma se uporablja eno glavno vejo imenovano povezovalna veja (angl. trunk branch), v katero se v končni fazi uveljavi vse spremembe, ki so bile narejene v ostalih razvojnih vejah. Temu se reče operacija združevanja (angl. merge) [18].



Slika 11: SVN razvojno drevo

TortoiseSVN je plast klienta orodja SVN. Implementiran je kot lupina Microsoft Windows operacijskega sistema. Je odprtokodni program izdan pod GNU GPL licenco. Leta 2007 je bil nagrajen s strani SourceForge.net skupnosti kot najboljše orodje ali pomožni program za razvijalce [19].

Lastnosti TortoiseSVN :

- Integriran v Windows lupino.
- Lahko je uporabljen z ali brez IDE okolja.
- Označevanje nad ikonami kaže takojšnji vpogled nad tem, katere datoteke so se spreminjale in katere so ostale nespremenjene (glede na repozitorij).
- Uporabniški vmesnik je podprt za 28 različnih jezikov.
- Omogoča izvajanje sprememb nad pisarniškimi dokumenti (Microsoft Word, Excel).

3.8 Orodja za vodenje projektov

3.8.1 Atlassian Jira

Vsaka aktivnost v podjetju je lahko opisana ter nadzorovana z programskim orodjem Atlassian Jira, ki je napisano v Javi. Jira je namenjena distribuciji opravil ter spremljanju napredka le-teh. Orodje lajša delo celotnemu oddelku, posebej pomembno pa je na nivoju projektne skupine. V njem se beležijo opravila (*angl. tasks*), roki, kdaj so bili izdani in do kdaj morajo biti končani deli nalog ali celotne naloge. Vse aktivnosti, ki se jih opravi na delu nad projektom, se hrani v Jiri. Grafični uporabniški vmesnik Jire je realiziran preko brskljalnika.

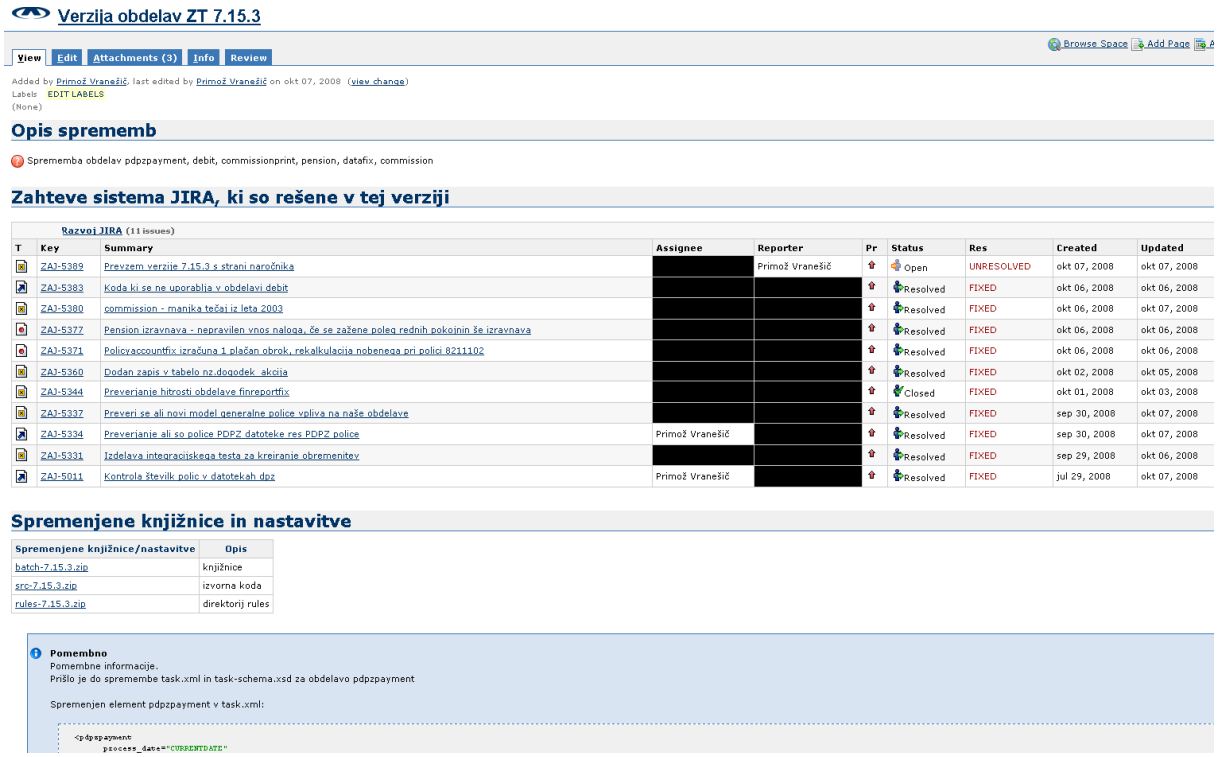
Vsako opravilo v tem programskem orodju ima tudi osebo naročnika, ki je zahtevala izvedbo in tudi pooblaščenca, ki mora rešiti zahtevo. Sistem omogoča hitro komunikacijo znotraj projektne skupine in/ali naročniki, saj zmanjšuje potrebo po fizičnem kontaktu med "naročnikom" in "izvajalcem", ter premošča časovne in krajevne ovire pri sporočanju

informacij. Komunikacija poteka po principu: naloga je bila izdana, sprejeta, opravljena, preverjena ter (morebiti) na koncu ponovno odprta zaradi neželene funkcionalnosti, oziroma zaprta, kar pomeni da je bila pravilno končana oziroma zaprta z določenim statusom iz podanega nabora.

Jira omogoča tudi direktno povezavo do SVN sistema, tako da je ob vsakem opravilu v brskljalniku možen vpogled v tiste datoteke, ki so bile zaradi danega opravila spreminjane.

3.8.2 Atlassian Confluence

Orodje Confluence je prav tako kot Jira na nivoju grafičnega uporabniškega vmesnika narejeno za delo preko brskljalnika in napisano v programskem jeziku Java. Uporablja se v poslovnih okoljih z namenom hranjenja poslovnega znanja. V sistem se beleži opise funkcionalnosti aplikacij, specifikacije zahtev uporabnikov, vodi se evidenco predaj verzij ipd. Za opisovanje se uporablja wiki format. Po navadi se ga uporablja v kombinacij z Jiro, kar povečuje njegovo moč predstavitve znanja. Na ta način se lahko na primer v strani, ki opisuje predano verzijo in njene nove funkcionalnosti poveže dejanska opravila, ki so bila predana ob verziji (slika 12).



Slika 12: Primer confluence strani ob predaji verzije

4 PREDSTAVITEV JAVANSKE POSLOVNE APLIKACIJE ZA OBDELOVANJE ŽIVLJENJSKIH ZAVAROVANJ

V naslednjem poglavju bomo spoznali javansko poslovno aplikacijo za obdelovanje življenjskih zavarovanj. Opisali bomo, kako je potekal razvoj in sama izvedba te aplikacije, kakšne tehnologije in rešitve uporablja, katera orodja so bila uporabljena pri razvoju ter namen aplikacije.

4.1 Terminologija

Zavarovalna polica je sestavni del zavarovalne pogodbe. V zavarovalni polici morajo biti zato navedeni vsi podatki, ki omogočajo sklenitev zavarovanja. Zavarovalna polica vsebuje bistvene elemente zavarovalne pogodbe: stranke, zavarovalni predmet oz. interes ali zavarovano osebo, označbo zavarovanih nevarnosti, navedbo splošnih in posebnih pogojev, ki so sestavni del zavarovalne pogodbe, izračun zavarovalne premije, trajanje zavarovanja, kraj sklenitve pogodbe ter podpis pogodbenih strank. Zavarovalno polico sme začasno nadomeščati potrdilo o začasnem kritju.

Zavarovalec je oseba, ki sklene zavarovalno pogodbo z zavarovalnico in se zaveže plačevati zavarovalno premijo.

Zavarovanec je tista oseba, ki je nevarnostni objekt, torej od katere smrti, invalidnosti, okvare zdravja, obolelosti ali doživetja je odvisno izplačilo zavarovalnine.

Upravičenec je oseba, ki je z zavarovalno pogodbo določena, da ji je zavarovalnica v primeru zavarovalnega primera dolžna plačati zavarovalnino.

Zavarovalna vsota je dogovorjen denarni znesek, ki predstavlja zgornjo mejo obveznosti zavarovalnice, če nastane zavarovalni primer. Z zavarovalno pogodbo se lahko dogovori, da se zavarovalna vsota med trajanjem zavarovanja revalorizira zaradi ohranjanja tiste realne vrednosti, ki jo predstavlja ob sklenitvi zavarovanja.

Zavarovalna premija je znesek, ki ga zavarovalec plača po zavarovalni pogodbi zavarovalnici. Zavarovalna premija ni enovit pojem, temveč se deli na čisto (funkcionalno) premijo in režijski dodatek. Iz zavarovalne premije se plačujejo sedanje in prihodnje škode, pokrivajo stroški preventive ter stroški poslovanja zavarovalnice.

Obdelava je del aplikacije, katerega namen je, da iz vhodne množice podatkov, ki so lahko podatki iz podatkovne baze ali tekstne datoteke, kreira po točno določenih pravilih novo izhodno množico podatkov, ki se lahko vstavijo v podatkovno bazo ali izpišejo v datoteko.

Revizijska sled je sestavljena iz vidnih podatkov in informacij, ki omogočajo identificirati posamezne transakcije, določiti njihovo veljavnost, pravilnost in celovitost izkazovanja v računovodskih izkazih ter dejanskost izvedbe kontrol. Namen revizijske sledi je zagotavljanje točnosti in nepotvorljivosti podatkov, dokumentiranost okolja, v katerem so transakcije nastale, predstavitev zgodovinskega zaporedja transakcij o poslovnem dogodku, nadzor nad izvedbami kontrol, zagotavljanje in nadzor nad usklajenostjo rešitve z zakonodajo. Revizijska sled je orodje revizorjev, kontrolorjev in linijskih vodij.

Bilančni mesec je mesec, v katerem se izvršijo denarne transakcije podjetja in opravi knjigovodski del poslovanja podjetja.

4.2 Namen

Osnovni namen razvoja poslovne aplikacije je bil ustvariti delujočo poslovno aplikacijo za ravnanje z življenjskimi zavarovanji. Ker imamo več različnih vrst življenjskih zavarovanj, ki se med seboj po pravilih razlikujejo, je razvoj potekal kot razvoj evolucijskih prototipov. Poslovna aplikacija se je nenehno razvijala na vseh vidikih kode, predvsem zaradi dodajanja podpore novim tipom življenjskih zavarovanj in razširjanja podpore obstoječim. Na začetku razvoja so bile podprte najnujnejše obdelave (kot so naprimer obdelava za kreiranje položnic in obdelovanje vplačil), sčasoma pa so bile dodane tudi obdelave z nižjimi prioritetami, vendar prav tako pomembne (kot so recimo obdelave za popravljanje podatkov v bazi, testiranje pravilnosti ostalih obdelav, kreiranje izpisnih list za agencijske delavce ipd.).

Ker se je že na začetku vedelo, da bodo vse te obdelave nekoč implementirane v poslovni aplikaciji, je bilo zastavljeno dobro ogrodje (angl. framework), ki omogoča takšno razširljivost. To ogrodje bo podrobneje opisano v nadaljevanju.

4.3 Primer poslovne aplikacije

Preden si bomo podrobneje pogledali tehnologijo, ki je v ozadju, si bomo pogledali za kakšne naloge je bila aplikacija razvita.

4.3.1 Delovanje obstoječe aplikacije v zavarovalniškem informacijskem sistemu

Potek denarnih tokov v zavarovalniškem poslovnem sistemu in podpora zavarovanjem je izrazito ciklična. Nad zavarovalnimi policami se v vsakem bilančnem mesecu ponavlja ustaljeno zaporedje akcij. Izvršitev določene akcije v danem mesecu je predpogoj za začetek izvrševanje naslednje akcije, ki jo je potrebno tipično izvesti nad zavarovalno polico. Opisana poslovna aplikacija je zato narejena transakcijsko.

Najpomembnejši del zavarovalniškega informacijskega sistema, katerega del je aplikacija, zajema finančni del poslovanja. Opisane naloge, so v osnovi preproste, vendar se pri realizaciji izkažejo za dokaj zahtevne. Razlog za to je v tem, da vsaka obdelava načeloma podpira vse vrste izdelkov, za vsak izdelek pa veljajo drugačna pravila, ki so določena preko zakonov o zavarovanjih. Zato sem opisal osnovne naloge pomembnejših obdelav v zaporedju kot si sledijo, nisem pa se spuščal v podrobnosti delovanja.

Prva naloga aplikacije pred tekočim bilančnim mesecem je, da za pretekli bilančni mesec opravi še vse preostale finančne transakcije, kot so: obračun upravljalskih stroškov, obračun premij za smrt, konverzija prehodnih enot, izračun matematičnih rezervacij, ipd. Na ta način aplikacija zagotovi pravilno stanje police v tekočem bilančnem mesecu. Poleg tega opravi še avtomatsko stornacijo zavarovalnih polic in avtomatsko zapiranje opominov. Vse to je narejeno preko obdelave imenovane `aktuarska obdelava`. Razlog za to obdelavo je preprost. Namreč, šele na zadnji delovni dan bilančnega meseca so znani vsi podatki, ki se uporabijo pri obdelavi.

Nato se glede na pogodbo tvorijo obroki in obremenitve za posamezne zavarovalne police. Ti so pomembni, saj jih potrebujejo vse obdelave, ki ravnajo s prenašanjem denarja. Temu sledi sklop obdelav, ki imajo isti namen-obremeniti zavarovalce polic.

Ko je ta del obdelav zaključen, je potrebno ugotoviti, katere zavarovalne police so še veljavne v skladu z pogodbo. Možno je, da se zaradi različnih razlogov zavarovalec police ne drži pogodbe, ali pa jo ni več zmožen izpolniti. Take primere rešujeta dve obdelavi. Obdelava za stornacijo polic in obdelava za kreiranje opominov policam imenovani `cancellation` in `remindercreation`. Obdelava za kreiranje opominov v osnovi preveri vsa vplačila in vse obroke in glede na njihovo razliko določi dolg zavarovalca. Če ta dolg presega zakonsko predpisani dolg, se takemu zavarovalcu preko pošte pošlje opomin z zneskom trenutnega dolga, ki ga je dolžen izplačati zavarovalnici. Obdelava za stornacijo za take police, ki jim je bil že poslan opomin preveri, ali je bil ta plačan ali ne. V primeru izpolnitve obveznosti popravi podatke o policah in omogoči nadaljevanje obdelovanja takšnim zavarovalnim policam v sistemu. Za tiste police, ki pa nimajo poravnanih obveznosti in je pretekla zakonska meja za poravnavo, obdelava glede na skupno vsoto vplačil ali zamrzne sredstva na polici in prepreči nadaljevanje obdelovanja, dokler niso obveznosti pokrite, ali pa odvzame vsa sredstva in prekine zavarovanje.

V kolikor ima zavarovalec sklenjeno plačevanje police preko fakture, evidenčne fakture ali trajnega naloga, zavarovalec ne nakazuje neposredno denarja zavarovalnici, ampak zavarovalnica dobi ta denar preko podjetja, v katerem je zavarovalec zaposlen ali pa preko zavarovalčeve banke, ki nakazuje dogovorjeno zavarovalno premijo. Zaradi različnih tipov plačevanja in velike razlike v pravilih med tipi zavarovanj ta sklop obdelujejo tri različne obdelave. Njihova naloga je, da pripravijo podatke, ki se pošljejo podjetjem in bankam, da se ve, koliko denarja je potrebno trgati stranki zavarovalnice.

Ko banke ali podjetja prejmejo od zavarovalnice podatke o zneskih, do katerih je zavarovalnica upravičena, nakažejo denar zavarovalnici. Lahko se zgodi, da zavarovalec police ne more izplačati dela ali celotnega zneska. Iz tega razloga zavarovalnica ob nakazilu dobi tudi podatke v elektronski obliki, o tem koliko denarja je kateri zavarovalec nakazal zavarovalnici. Te podatke prevzamejo obdelave, ki skrbijo za knjiženje vplačil v sistemu zavarovalnice. Obdelave tudi ugotovijo nepravilna nakazila in v tem primeru posredujejo podatke o nepravilnostih zadolženim osebam v zavarovalnici. V določenih primerih neplačil pošljejo strankam tudi položnice.

Ta na novo kreirana vplačila je nato potrebno v sistemu zavarovalnice pravilno obdelati. Glede na pogodbo se razdeli znesek vplačil na rizike in sklade kakor je sklenjeno po pogodbi, poračuna se stroške upravljanja, matematične rezervacije in obračuna premije.

Nato sledi zopet sklop večih obdelav, ki pošljejo podatke o uspešnosti opravljene transakcije bankam in podjetjem, ki so nakazala denar zavarovalnici.

Ko je ta del zaključen, sledijo še obdelave za pomoč knjigovodstvu. Te obdelave zberejo podatke o vsem denarju, ki se trenutno nahaja v sistemu, in omogočijo zavarovalnici filtriran vpogled v sredstva, na podlagi katerih vedo, koliko sredstev imajo in kako so razporejena oziroma koliko nerazporejenih sredstev še imajo na voljo za vlaganje v sklade.

Poleg teh obdelav obstaja še kopica drugih, ki se lahko zaganjajo dnevno ali mesečno. Praviloma se iz podatkovne baze ne sme ničesar brisati, saj se s tem izgublja zgodovino. Vse

spremembe, ki se izvedejo nad bazo, se izvedejo tako, da se nepravilne zapise razveljavi glede na pravila, ki veljajo in v take tabele doda spremembe.

Vse spremembe v sistem dodaja obdelava imenovana `alteration`. Postopek spreminjanja je realiziran preko treh tabel: tabele `akcija`, tabele `dogodek` in tabele `dogodek_akcija`. Tabela `dogodek` vsebuje vse dogodke, ki se lahko izvedejo nad bazo, tabela `akcija` določa spremembo, ki se zgodi nad bazo, tabela `dogodek_akcija` pa predpisuje, katere akcije sproži določen dogodek.

Zavarovanja za zavarovalnico sklepajo zavarovalni agenti. Ti so lahko zaposleni znotraj same zavarovalnice, lahko pa so zaposleni pri agencijah, ki sodelujejo z zavarovalnico. Zavarovalni agenti so plačani glede na uspešnost sklepanja pogodb. Od zavarovalnice dobivajo vsak mesec izračun in izpis provizije, ki jim je izplačana. To je prav tako podprto kot obdelava aplikacije.

Obstaja še ena pomembnejša obdelava, ki bi jo v tej točki omenil. To je obdelava `dursreport`, ki pripravi podatke in poročila za Davčno upravo Republike Slovenije. Sama obdelava uporablja veliko stvari, ki sem jih uspel uporabiti tudi pri razvoju obdelave za tiskanje zavarovalnih polic. Temelji na zajemu podatkov iz baz in ustreznem filtriranju teh podatkov.

4.3.2 Uporabljene tehnologije in orodja

Za izvedbo aplikacije smo izbrali programski jezik Java. Izbiro smo podprli s številnimi prednostmi, ki jih ta jezik ponuja (opisano v poglavju 3.3). Prav tako smo želeli aplikacijo dobro strukturirati, kar Java omogoča.

Za izdelavo poslovne aplikacije smo uporabili orodje Eclipse. Zelo nam je bil v pomoč sintaktični analizator, ki je odkril večino napak že pred samim prevajanjem. Poleg tega smo Eclipse uporabili tudi za kreiranje XML in XSL datotek.

Za shranjevanje podatkov smo si izbrali IBM-ov podatkovni strežnik DB2. Izbrali smo ga zato, ker se DB2 uporablja v podjetju za katerega je bila ta aplikacija razvita. Prav tako smo pri izboru upoštevali vse prednosti, ki jih DB2 ponuja.

4.3.3 Knjižnice

Apache Log4j je na Javi temelječe pomožno orodje za beleženje. Primarno se uporablja kot orodje za razhroščevanje.

`Db2jcc` je knjižnica, ki omogoča priklop Javanske aplikacije na DB2 bazo.

Drools knjižnice vsebujejo orodja za delo s poslovnimi pravili (angl. business rule engine). Pravila so ponavadi Excelove preglednice (slika 13), pri katerih ena vrstica vsebuje pogoje in akcije, ki se izvedejo glede na pogoje. Vrstica predstavlja pravilo.

	RuleTable Leaders Commission				
	CONDITION	CONDITION	CONDITION	ACTION	ACTION
	$w: WorkerData			$w	$w
	leaderWorkerType	subordinateWorkersCount >=	subordinateWorkersCount <=	getLeaderCommissionParametersData().setLeadersCommissionFactorMap($1,$2);	getLeaderCommissionParametersData().setLeadersCommissionPercentageMap($1,$2);
leaders commission percentage rules	leader type	minimum subworkers	maximum subworkers	leader factor	leader percentage (%)
	1	1	7	1,7	1,15
	1	8	10	1,8	1,15
	1	11	15	1,10	1,15
	1	16	20	1,12	1,15
	1	21	2000000	1,15	1,15
	2	1	2000000	2,75	2,3

Slika 13: Primer poslovnih pravil v Excelovi preglednici

Hibernate knjižnice omogočajo objektno preslikovanje za jezik Java. V osnovi je glavna naloga Hibernate-a mapiranje javanskih razredov v tabele, ki se nahajajo v podatkovni bazi. To je narejeno preko SQL povpraševanj (angl. SQL Queries). V javanskem razredu se nahajajo getter in setter metode, ki so vezane na posamezna polja tabele in s pomočjo teh metod dostopamo do določenih polj in jih spreminjamo.

Saxon knjižnice predstavljajo orodja za izvedbo XSLT transformacij (opisano v poglavju 3.4).

`Dom4j` je knjižnica, ki omogoča Javi, da na prepreprosti način rokuje z XML in pa HTML datotekami. Nudi polno podporo za delo z DOM modelom (angl. Document Object Model, krat. DOM).

4.3.4 Hierarhija programske kode

Že ob začetku razvoja je bilo jasno, da bo bila aplikacija obsežna, zato je bilo potrebno dobro premisliti, kako strukturirati kodo.

Ker gre pri razvoju aplikacije za obsežen projekt, ta ne vsebuje le datotek z izvorno kodo, ampak prav tako datoteke z dokumentacijo in drugimi uporabnimi stvarmi. Zaradi tega je narejen razvoj projekta v standardni datotečni drevesni strukturi (slika 14), ki loči različne tipe datotek v različne mape [20].

```
myproject\
    src\
    build\
    reports\
    etc\
    doc\
    lib\
```

Slika 14: Standardna datotečna drevesna struktura

Direktorij `src\` (slika 14) je v modelu standardne datotečne drevesne strukture namenjen izvorni kodi.

Poslovna aplikacija je namenjena za opravljanje za opravljanje veliko različnih stvari, ki pa jih lahko poenotimo. V splošnem imamo problem, ki ga mora aplikacija obdelati, rešitev problema pa je rezultat. Torej imamo tri korake. Iz tega vidika gledamo na vse obdelave, ki jih podpiramo, naj bo to kreiranje obrokov ali pa tiskanje zavarovalnih polic. V Javi je realizacija tega problema dokaj lahko rešljiva z uporabo abstraktnih razredov.

Iskanje problemov definira razred `AbstractProblemProducer`. Problem pri nas pomeni nekaj, kar je potrebno obdelati in preveriti, ali so vsi pogoji izpolnjeni, da pride do dejanskih sprememb v bazi oziroma do katerega koli rezultata.

Reševanje problemov je definirano preko razreda `AbstractProblemSolver`. Ta definira metode za reševanje problema.

Razredi ki implementirajo vmesnik (angl. interface) `Iresult`, vsebujejo podatke o rezultatu obdelovanja problema.

Vsaka obdelava razširi oziroma implementira vse tri razrede, zato kljub veliki različnosti posameznih obdelav obstaja poenoten sistem, ki skrbi za vse različne obdelave.

Naša aplikacija ima vstopni razred imenovan `WorkerManager`. Njegova naloga je, da se preko njega zganjajo želene obdelave, da inicializira vse potrebne strukture, ki skrbijo za dostop do baze ali za branje XML datotek, in omogoča tudi ravnanje z napakami.

4.3.5 Nastavitve programa

Za nastavljanje programa uporabljamo XML datoteke. V njih so definirane obdelave (slika 15) in vhodni parametri, ki so potrebni za izvajanje programa, vključno s podatki o podatkovni bazi, podatki o tem, ali naj se sproti z obdelavo izpisuje revizijsko sled, dostopi do osebnih podatkov in ostalimi vhodnimi podatki.

```
<creationcause
  process_date="CURRENTDATE"
  use_isoz="false"
  locale="sl_SI"
  export_dir="c:/nzzbatch/export/creationcause"
  changes="BOTH">
  <premium_cutoff amount="5" currency="EUR"/>
  <policy pattern_operator="==" policy_id_pattern="8207371"/>
</creationcause>
```

Slika 15: Vhodni parametri obdelave `creationcause`

Poleg XML datotek uporablja program tudi tekstovne datoteke (običajno z končnico `.properties`), ki vsebujejo vrednosti spremenljivk. S pomočjo te datoteke lahko uporabnik aplikacije sam nastavlja nekatere ključne besede, ki se uporabijo pri obdelavah. Če podjetje nastopa na dveh različnih tržiščih, lahko preprosto kreira izpisne liste v dveh različnih jezikih. Vse, kar moramo narediti, je le to, da prevedemo besede v datoteki `.properties` v ustrezni jezik.

Del obdelav v aplikaciji uporablja drools knjižnic, zato uporabljamo tudi veliko XML datotek s poslovnimi pravili.

Program prav tako uporablja XSL datoteke, ki predpisujejo pretvorbe podatkov.

4.3.6 Vhodni podatki

Poleg vseh naštetih datotek z nastavitvami aplikacija vhodne podatke dobi tudi iz podatkovne baze. Zavarovalnice strankam omogočajo različne načine plačevanja zavarovalne premije. Možno je plačevanje preko fakture, trajnika, direktna nakazila ipd. Zato zavarovalnice veliko

poslujejo z bankami. Rezultati elektronskega poslovanja so različne datoteke z novimi podatki o vplačilih. Te datoteke tudi nastopajo kot vhodni podatki poslovne aplikacije.

4.3.7 Izhodni podatki

Rezultati obdelav se zapisujejo v podatkovno bazo in ali datoteke. Aplikacija kreira tekstovne datoteke in datoteke v PDF formatu za izpis različnih poročil ter XML datoteke za uporabo v informacijskem sistemu. Vsaka obdelava kreira tudi datoteko z podatki o poteku transakcije z končnico `.log`.

4.3.8 Obravnavanje in delo z napakami

Med izvajanjem programa lahko pride do velikega števila različnih napak zaradi izpada strežnika, napačne vhodne datoteke, napake na bazi in podobno.

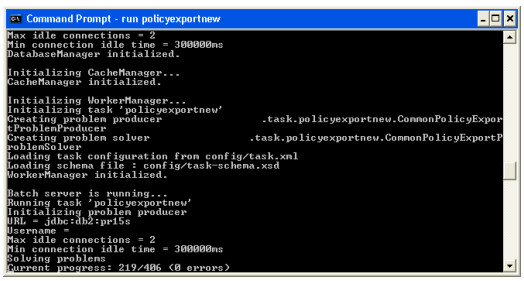
Da bi se temu izognili te napake lovimo in ob posamezni napaki ukrepamo napaki primerno. Določene obdelave zahtevajo, da se vse, kar se obdeluje, izvede brez napake, v nasprotnem primeru izvedemo ukaz razveljavitve transakcije (angl. rollback). Določene obdelave pa dovolijo, da se kakšen problem obdela z napako, in se samo za ta problem naredi razveljavitev, za pravilno obdelane probleme pa se izvede potrditev transakcije (angl. commit).

Edino, kar lahko za uporabnika naredimo v primeru napake je to, da uporabnika obvestimo ter napako čim temeljiteje pišemo v `*.log` datoteke. Velikokrat naletimo na pogoste napake, ki jih pri izpisu oštevilčimo z standardno številko napake. Vse opise in rešitve tipičnih napak imamo na sistemu Confluence. Stranki se v tem primeru ni potrebno obračati na nas, ampak lahko sama reši takšne napake.

Tako zagotavljamo usklajene in zanesljive podatke v bazi.

4.4 Uporaba poslovne aplikacije

Naša poslovna aplikacije je namenjena za zaganjanje iz ukazne vrstice (slika 16). Fizično gre za hierarhijo direktorijev in datotek. Stranke same spreminjajo vse datoteke, ki vsebujejo vhodne parametre. Zato je postopek nameščanja novih verzij relativno preprost. Strankam se pošlje vse notranje knjižnice aplikacije in nova pravila (drules). Aplikacija vsebuje določene obdelave, ki se zaganjajo ob datumih, ki niso znani v naprej. Takšne obdelave se zaganjajo ročno. Pred tem se v ustrezne datoteke nastavitev vnesejo podatki. Številne obdelave pa se zaganjajo dnevno. Poganjajo se preko pripravljenih skript. Za te obdelave se običajno vhodni podatki iz datotek nastavitev ne spreminjajo.



Slika 16: Zaganjanje obdelave `policyexportnew` iz ukazne vrstice

5 IZDELAVA OBDELAVE ZA IZPIS PODATKOV

V prejšnjem poglavju smo spoznali javansko poslovno aplikacijo za obdelovanje življenjskih zavarovanj. V naslednjem poglavju bomo predstavili potek razvoja izdelave obdelave za izpis podatkov o zavarovalni polici ki je sestavni del zavarovalne pogodbe. Za to bomo spoznali tudi področje sklepanja življenjskih zavarovanj.

5.1 Opis problema

Poslovna aplikacija sodi na področje življenjskega zavarovalništva in v okvir tega sodi tudi izpis zavarovalnih polic za življenjska zavarovanja. V tem delu bomo predstavili možnosti uporabe tehnologij in možnosti rešitev.

5.2 Specifikacija obdelave tiskanja zavarovalnih polic

Tiskani primer zavarovalne police je pravni dokument. Je del zavarovalne pogodbe. Zavarovalna polica vsebuje vse, kar je bilo dogovorjeno ob sklenitvi pogodbe. Na njej je zapisan sklenitelj zavarovanja, zavarovane osebe, upravičenci ob škodnih primerih, riziki za katere so zavarovane osebe zavarovane, znesek zavarovalne premije, znesek zavarovalne vsote in drugo.

Zavarovalnica podpira sklenitve več različnih vrst zavarovanj: naložbena življenjska zavarovanja, pokojninska zavarovanja ipd. Vsaka vrsta zavarovanja ima lasten nabor polj z podatki, ki se izpisujejo na polico (slika 17, slika 18). Nekatera polja so pri različnih vrstah zavarovanja skupna (kot na primer ime zavarovalca police), druga pa so posebna.

Zavarovalnica je imela za določeno vrsto zavarovanj že pripravljen sistem za elektronsko poslovanje. Ta sistem sprejme XML datoteko in kreira PDF datoteko z zavarovalno polico, ki gre nato v tiskanje. Vendar sistem ni nadgrajen s podporo za vse vrste zavarovanj. Za nekatera zavarovanja se zato uporablja tudi aplikacijo, ki sprejme tekstovno datoteko s podatki o zavarovanjih in iz te kreira PDF datoteko z zavarovalno polico.

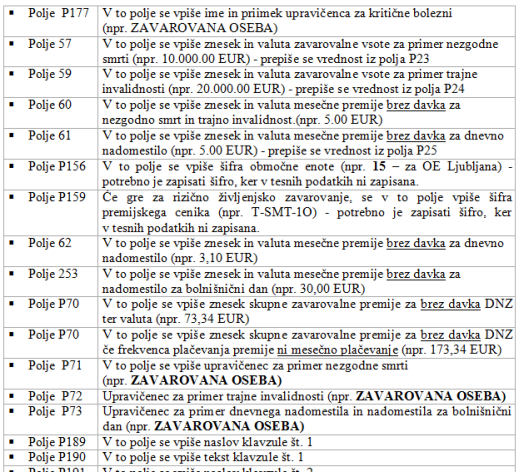
Obdelava za tiskanje zavarovalnih polic mora zato podpirati dva različna načina kreiranja izhodne datoteke. Znotraj teh dveh načinov mora izpisovati točno določena polja glede na vrsto zavarovanja. Zahtevano je bilo še, da se za različene tipe naložbenega zavarovanja kreira posebne datoteke, ki vsebujejo podatke le o tem tipu.

Vsi vhodni podatki, ki jih obdelava potrebuje se nahajajo v podatkovni bazi.

Zavarovalno polico se tiska s podatki, ki so veljavni na datum začetka zavarovanja police. Vendar se lahko zgodi, da želi po začetku zavarovanja zavarovalec police dopolniti pogodbo ali pa ugotovi, da je na polici prišlo do napake, in zahteva popravek podatkov. Obdelava za tiskanje polic mora podpirati vse tri dogodke.

Možno je tudi, da se obdelava uporabi za tuje trge, zato mora imeti sistem, ki uporabnikom omogoča preprosto prevajanje tekstov.

Če se zgodi, da se določene zavarovalne police ne more iztisniti (ali zaradi pomankljivih podatkov v bazi ali napravilnih kombinacij podatkov), to ne sme vplivati na druge zavarovalne police, ki se pravilno natisnejo.



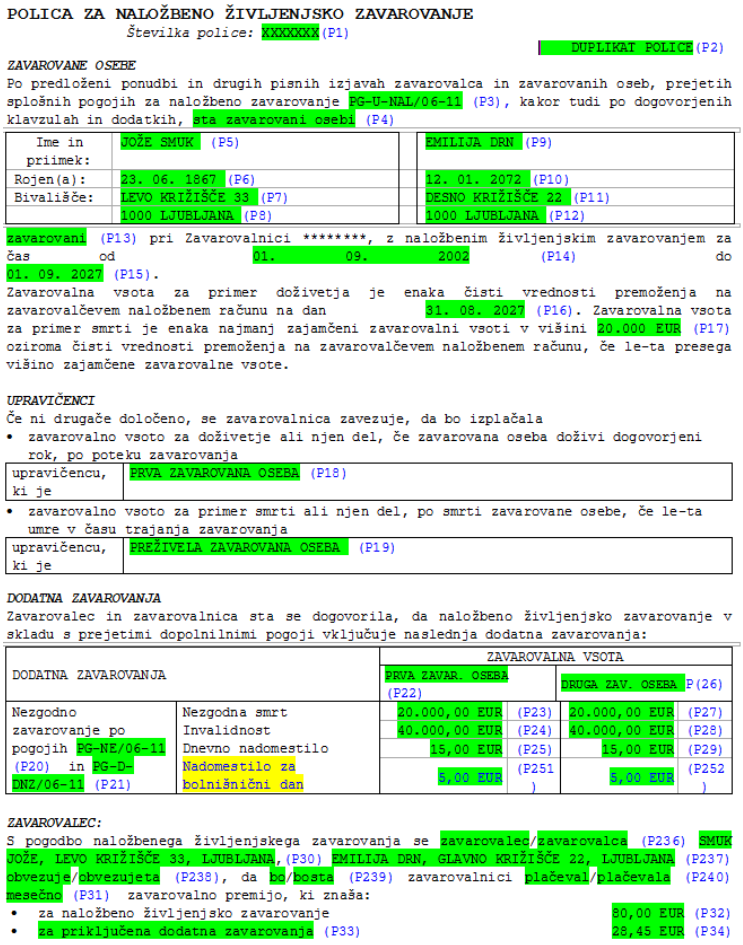
Polje P177	V to polje se vpiše ime in priimek upravičenca za kritične bolezni (npr. ZAVAROVANA OSEBA)
Polje 57	V to polje se vpiše znesek in valuta zavarovalne vsote za primer nezgodne smrti (npr. 10.000.00 EUR) - prepiše se vrednost iz polja P23
Polje 59	V to polje se vpiše znesek in valuta zavarovalne vsote za primer trajne invalidnosti (npr. 20.000.00 EUR) - prepiše se vrednost iz polja P24
Polje 60	V to polje se vpiše znesek in valuta mesečne premije brez davka za nezgodno smrt in trajno invalidnost.(npr. 5.00 EUR)
Polje 61	V to polje se vpiše znesek in valuta mesečne premije brez davka za dnevno nadomestilo (npr. 5.00 EUR) - prepiše se vrednost iz polja P25
Polje P156	V to polje se vpiše šifra območne enote (npr. **15** – za OE Ljubljana) - potrebno je zapisati šifro, ker v tesnih podatkih ni zapisana.
Polje P159	Če gre za rizično življenjsko zavarovanje, se v to polje vpiše šifra premijskega cenika (npr. T-SMT-1O) - potrebno je zapisati šifro, ker v tesnih podatkih ni zapisana.
Polje 62	V to polje se vpiše znesek in valuta mesečne premije brez davka za dnevno nadomestilo (npr. 3,10 EUR)
Polje 253	V to polje se vpiše znesek in valuta mesečne premije brez davka za nadomestilo za bolnišnični dan (npr. 30,00 EUR)
Polje P70	V to polje se vpiše znesek skupne zavarovalne premije za brez davka DNZ ter valuta (npr. 73,34 EUR)
Polje P70	V to polje se vpiše znesek skupne zavarovalne premije za brez davka DNZ če frekvenca plačevanja premije ni mesečno plačevanje (npr. 173,34 EUR)
Polje P71	V to polje se vpiše upravičenec za primer nezgodne smrti (npr. **ZAVAROVANA OSEBA)**
Polje P72	Upravičenec za primer trajne invalidnosti (npr. **ZAVAROVANA OSEBA)**
Polje P73	Upravičenec za primer dnevnega nadomestila in nadomestila za bolnišnični dan (npr. **ZAVAROVANA OSEBA)**
Polje P189	V to polje se vpiše naslov klavzule št. 1
Polje P190	V to polje se vpiše tekst klavzule št. 1

Slika 17: Primer specifikacije posameznih polj, uporabljenih pri tiskanju

5.3 Podatki vezani na tiskanje zavarovalne police

- Podatki vezani na polico:
  - identifikacijska številka police,
  - datum izdaje police,
  - datum začetka zavarovanja,
  - datum konca zavarovanja.
- Podatki o zavarovalcu:
  - EMŠO,
  - ime,
  - priimek,
  - datum rojstva,
  - naslov.
- Podatki o zavarovancu:
  - EMŠO,
  - ime,
  - priimek,
  - datum rojstva,
  - naslov,
  - riziki, ki so del zavarovalne pogodbe,
  - skladi, ki so del zavarovalne pogodbe,
  - upravičenec za primer škode.
- Podatki o stroških police:
  - upravljalski stroški,
  - stroški vodenja,
  - izstopni stroški.
- Zneski:
  - zavarovalna vsota,
  - zavarovalna premija,
  - tabele rasti premoženja v primeru skladov,
  - spremembe nevarnostnih faktorjev glede časovno lestvico.

Zgoraj navedeni podatki so predmet pogodbe med zavarovalnico in zavarovancem police. Podatki se pošljejo zavarovalcu in mu služijo kot pravni dokument o sklenitvi zavarovanja.



POLICA ZA NALOŽBENO ŽIVLJENJSKO ZAVAROVANJE

Številka police: XXXXXXX (P1)

DUPLIKAT POLICE (P2)

ZAVAROVANE OSEBE

Po predloženi ponudbi in drugih pisnih izjavah zavarovalca in zavarovanih oseb, prejetih splošnih pogojih za naložbeno zavarovanje PG-U-NAL/06-11 (P3), kakor tudi po dogovorjenih klavzulah in dodatkih, sta zavarovani osebi (P4)

Ime in priimek:	JOŽE SMUK (P5)	EMILIJA DRN (P9)
Rojen(a):	23. 06. 1867 (P6)	12. 01. 2072 (P10)
Bivališče:	LEVO KRIŽIŠČE 33 (P7)	DESNO KRIŽIŠČE 22 (P11)
	1000 LJUBLJANA (P8)	1000 LJUBLJANA (P12)

zavarovani (P13) pri Zavarovalnici ********, z naložbenim življenjskim zavarovanjem za čas od 01. 09. 2002 (P14) do 01. 09. 2027 (P15).

Zavarovalna vsota za primer doživetja je enaka čisti vrednosti premoženja na zavarovalčevem naložbenem računu na dan 31. 08. 2027 (P16). Zavarovalna vsota za primer smrti je enaka najmanj zajamčeni zavarovalni vsoti v višini 20.000 EUR (P17) oziroma čisti vrednosti premoženja na zavarovalčevem naložbenem računu, če le-ta presega višino zajamčene zavarovalne vsote.

UPRAVIČENCI

Če ni drugače določeno, se zavarovalnica zavezuje, da bo izplačala

- zavarovalno vsoto za doživetje ali njen del, če zavarovana oseba doživi dogovorjeni rok, po poteku zavarovanja

upravičencu, ki je	PRVA ZAVAROVANA OSEBA (P18)

- zavarovalno vsoto za primer smrti ali njen del, po smrti zavarovane osebe, če le-ta umre v času trajanja zavarovanja

upravičencu, ki je	PREŽIVELA ZAVAROVANA OSEBA (P19)

DODATNA ZAVAROVANJA

Zavarovalec in zavarovalnica sta se dogovorila, da naložbeno življenjsko zavarovanje v skladu s prejetimi dopolnilnimi pogoji vključuje naslednja dodatna zavarovanja:

DODATNA ZAVAROVANJA		ZAVAROVALNA VSOTA			
		PRVA ZAVAR. OSEBA (P22)		DRUGA ZAV. OSEBA P(26)	
Nezgodno zavarovanje po pogojih PG-NE/06-11 (P20) in PG-D-DNZ/06-11 (P21)	Nezgodna smrt	20.000,00 EUR	(P23)	20.000,00 EUR	(P27)
	Invalidnost	40.000,00 EUR	(P24)	40.000,00 EUR	(P28)
	Dnevno nadomestilo	15,00 EUR	(P25)	15,00 EUR	(P29)
	Nadomestilo za bolnišnični dan	5,00 EUR	(P251)	5,00 EUR	(P252)

ZAVAROVALEC:

S pogodbo naložbenega življenjskega zavarovanja se zavaroval**ec**/zavarovalca (P236) SMUK JOŽE, LEVO KRIŽIŠČE 33, LJUBLJANA, (P30) EMILIJA DRN, GLAVNO KRIŽIŠČE 22, LJUBLJANA (P237) obvezuje/obvezujeta (P238), da bo/bosta (P239) zavarovalnici plačeval/plačevala (P240) mesečno (P31) zavarovalno premijo, ki znaša:

- za naložbeno življenjsko zavarovanje 80,00 EUR (P32)
- za priključena dodatna zavarovanja (P33) 28,45 EUR (P34)

Slika 18: Primer tiskanja police z variabilnimi polji označenimi z zeleno barvo

5.4 Podatkovni model

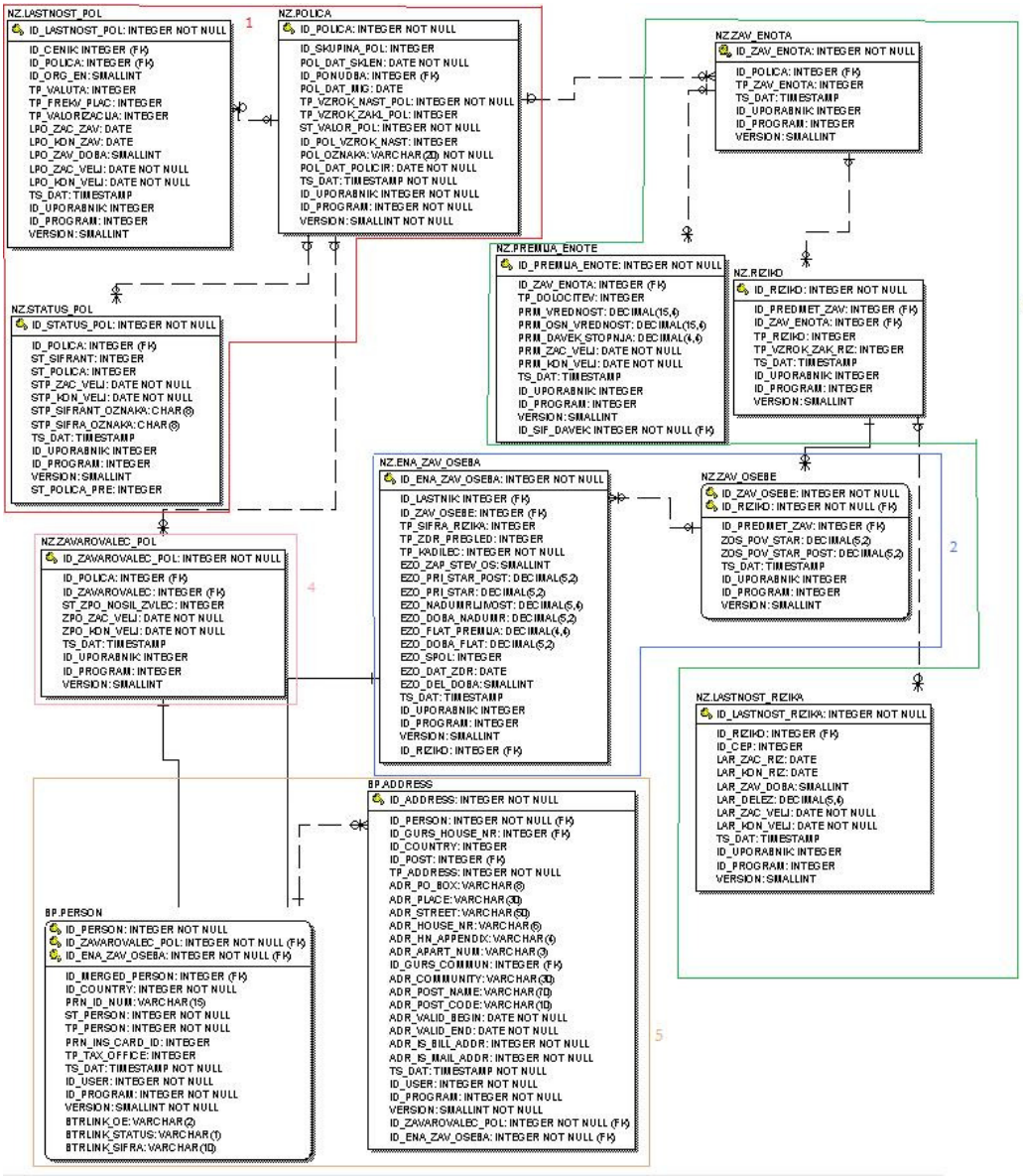
Konceptualni model podatkovne baze (slika 19), prikazuje logična razmerja med entitetnimi tipi. Posamezni entitetni tip vsebuje ime, enolični ključ in atribute ter je povezan z drugimi entitetnimi tipi. Tabele, ki so predstavljene, so najpogosteje uporabljane tebele pri obdelavi tiskanja polic.

V nadaljevanju bomo podrobneje spoznali pomembnejše entitetne tipe konceptualnega modela, ki je prikazan na sliki 18 ter razmerja med njimi.

1. **`Polica`, `Lastnost_Pol` in `Status_Pol`**

   Enolični ključ entitetnega tipa `Polica` je `ID_Polica`, praktično pomeni zaporedno številko sklenjenega zavarovanja v sistemu. Na entiteto `Polica` sta vezani dve entiteti `Status_Pol` in `Lastnost_Pol`, ki določata stanje zavarovalne police v zavarovalnici. S pomočjo teh dveh entitet vemo, ali je

polica v določenem trenutku primerna za tiskanje ali ne. Polica se tiska takrat, ko se je zavarovanje že začelo in je polica v aktivnem statusu.



Slika 19: Konceptualni model podatkovne baze

2. `Zav_Osebe` in `Ena_Zav_Oseba`

Na zavarovalni polici imamo zavarovane osebe, ki so zavarovane za določene rizike ali sklade. Zavarovane osebe dobimo preko entitete `Zav_Osebe`. Za posamezen riziko ali sklad na polici dobimo preko entitete `Riziko` podatek o tem, kdo je ali so zavarovane osebe za ta riziko ali sklad. Preko entitete `Ena_Zav_Oseba` nato dobimo točno število zavarovanih oseb (vključno z njihovimi tujimi ključi `id_person`). Iz entitete `Person` lahko dobimo točne podatke kdo te osebe so.

3. `Zav_Enota`, `Premija_Enote`, `Riziko` in `Lastnost_Rizika`

Preko entitete Zav_Enota lahko dobimo vse rizike in sklade, ki so vezani na zavarovalno polico. Pravilo je takšno, da ima vsak riziko svojo zavarovalno enoto, med tem ko imajo skladi skupno zavarovalno enoto. Vsaka polica ima natanko eno zavarovalno enoto za sklade in poljubno število zavarovalnih enot za rizike. Porazdelitev zavarovalne premije med skladi na polici je določena z entiteto Premija_Enote. Polje PRM_Vrednost z poljem LAR_Delez iz entitete Lastnost_Rizika določi, kako se porazdeli premija iz polja PRM_Vrednost. Entiteta Lastnost_Rizika tudi določa trenutno stanje rizika na polici.

4. **Zavarovalec_Pol**

   Entiteta Zavarovalec_Pol določa osebo, ki je sklenila zavarovanje. S povezavo na entiteto Person lahko natanko ugotovimo, za katero osebo gre.

5. **Person in Address**

   Entiteti Person in Address vsebujeta vse podatke o osebah, ki so priključene na zavarovalno polico.

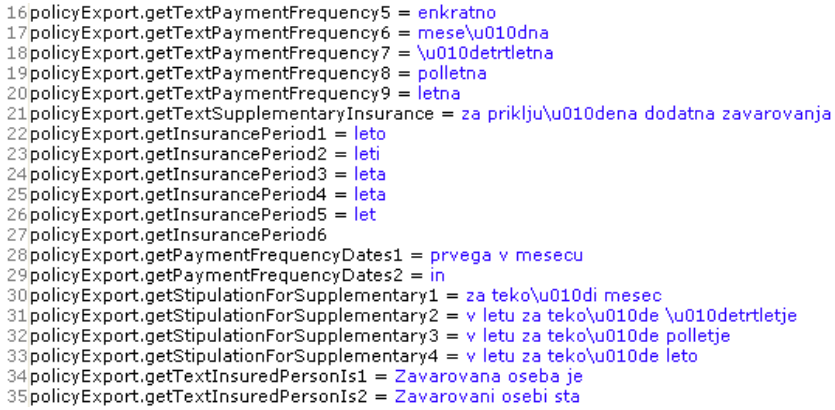
5.5 Izdelava obdelave za tiskanje polic

Zaradi različnih zahtev, ki izhajajo iz specifikacije in napovedi, da se bodo tudi v prihodnje dodajali modeli za izpis novih tipov zavarovalniških polic, je bilo potrebno razviti obdelavo čim splošnejše. Tako bi bilo mogoče brez spremembe v začetku razvite kode nadgraditi aplikacijo. Na roko pri obdelavi nam je šlo dejstvo, da vsaj branje podatkov ni imelo zapletene poslovne logike, je pa zgradba podatkovne baze dokaj prepletena. Podatki, ki se jih tiska, so po večini iz tabel, ki opisujejo rizike, osebne podatke, stanje premoženja v skladih, te tabele in povezave med njimi pa so obsežne. Obdelava po specifikaciji aplikacije zaporedoma obdeluje po eno zavarovalno polico. Zaradi teh dveh dejstev in počasnejšega dostopa do podatkovne baze je bila najboljša možnost, da se za vsako polico prebere vse podatke iz tabel in se potem glede na tip zavarovalne police določi, katere podatke se prikaže.

Ker se na polici izpisujejo spremenljivi teksti (glede na to za katero državo se izpisuje zavarovalne police), je bilo potrebno narediti še datoteke, ki vsebujejo tekste, nato pa ustrezne tekste prenesti v aplikacijo. Java vsebuje razred ResourceBundle, ki je namenjen natanko za to opravilo (slika 21).

Vključno z potrebnimi podatki za izpis je bilo potrebno pridobiti podatke, ki so potrebni za krmiljenje obdelave. To so podatki, ki med drugim tudi določajo državo, za katero se izpisuje podatke (slika 20).

Slika 20: Podatki ki so potrebni za krmiljenje obdelave tiskanja polic



Slika 21: Primer datoteke z teksti

Ker obdelava vsebuje veliko količino podatkov in XML omogoča uporabniku razumljivo strukturiranje podatkov, je bil prvi korak pri izdelavi aplikacije zapis podatkov v XML datoteko (slika 22). Ta pa je nato služila nadaljnemu delu z podatki. V Javi smo uporabili paket `org.w3c.dom`, s katerim je možno narediti podatkovno strukturo XML-a, poleg tega smo dali možnost uporabniku, da določi (če želi), da se mu posebej naredi ta datoteka, kar omogoča vmesno preverjanje prebranih podatkov in s tem lažjo obravnavo napak.

Ko je kreiran notranji XML, se odpre veliko poti za končno izdelavo tekstovne ali nove XML datoteke. Lahko se ročno v Javi napiše del kode, ki nato preko standardnega izhoda kreira željeno obliko podatkov. Možna je tudi uporaba plačljivih orodij, ki preberejo notranjo datoteko in zmorejo tudi preko uporabniškega vmesnika ustrezno zgraditi pravila pod katerimi se kreira obliko nove datoteke.

```
<PolicyData>
   <policyId>6100274</policyId>
   <GeneralPolicyDesignation></GeneralPolicyDesignation>
   <InsuranceStartDate>
      <day>01</day>
      <month>01</month>
      <year>1999</year>
      <MonthLong>januar</MonthLong>
   </InsuranceStartDate>
   <InsuranceStopDate>
      <day>31</day>
      <month>12</month>
      <year>2023</year>
      <MonthLong>december</MonthLong>
   </InsuranceStopDate>
   <InsurancePeriod>25 let</InsurancePeriod>
   <Pricelist>PICC</Pricelist>
   <InsuranceConditionCode>PG-PKI/98-05</InsuranceConditionCode>
   <Currency>SIT</Currency>
   <PremiumType>AMOUNT</PremiumType>
   <PaymentFrequencyMultiplier>1.0</PaymentFrequencyMultiplier>
   <OrganizationUnitData>
      <Name>MURSKA SOBOTA</Name>
      <Id>8</Id>
      <OrganizationUnitCounter>476</OrganizationUnitCounter>
   </OrganizationUnitData>
   <PersonData>
      <InsurerPersonData>
         <FirstCustomerData>
            <BirthDate>
               <day>02</day>
               <month>09</month>
               <year>1961</year>
               <MonthLong>september</MonthLong>
            </BirthDate>
            <PersonType>FIZIČNA OSEBA</PersonType>
```

Slika 22: Del vmesne XML datoteke

V našem primeru smo se odločili za odprtokodno orodje za izdelavo pretvorb. Uporabili smo orodje XSLT, ki v kombinaciji z XSL datotekami (slika 23) zmore izdelati poljubno obliko tekstovnih in XML datotek. To orodje se je izkazalo za zelo učinkovito in je omogočilo hitri razvoj aplikacije. V primeru, ko se je dodalo novo vrsto izdelka, je bilo večinoma potrebno le narediti novo XSL datoteko, ki opisuje pravila, pod katerimi se pretvarjajo podatki. Za to orodje smo se odločili tudi zato, ker se podatki namenjeni kasnejši uporabi za izdelavo dokončne police in imajo enostavno strukturo prikaza.

Ker se uporablja tiskanje polic v različnih državah, se uporablja tudi različne tiskarje. Ti imajo različne programe, ki jih uporablja za dokončen tisk polic. Zato tudi kreiramo različne vrste datotek, kar pa je z XSL pretvorbami trivialno.Tekstne datoteke vsebujejo le podatke, ki so ločeni z # znaki, med tem ko XML datoteke vsebujejo strukturirane podatke skladno z XML pravili. Datoteke, ki jih kreiramo, se nato pošljejo zavarovalnici, ta pa jih pošlje tiskarju, ki jih dokončno pretvori z uporabo oblikovnega programa v obliko, kot jo dobijo zavarovalci.

```
</xsl:call-template>
<xsl:text disable-output-escaping="yes">#</xsl:text>
<xsl:call-template name="writeField">
   <xsl:with-param name="inputString">
      <xsl:value-of select="AdditionalData/StipulationsTable/Stipulation[10]"/>
   </xsl:with-param>
</xsl:call-template>

<!-- 209 -->
<xsl:call-template name="writeField">
   <xsl:with-param name="inputString"><xsl:value-of select="
      PrintingData/PolicyTypeCode"/></xsl:with-param>
</xsl:call-template>

<!-- 210 -->
<xsl:if test="exists(AdditionalData/StipulationsTable)">
   <xsl:text disable-output-escaping="yes">1</xsl:text>
</xsl:if>
<xsl:if test="empty(AdditionalData/StipulationsTable)">
   <xsl:text disable-output-escaping="yes">0</xsl:text>
</xsl:if>
<xsl:text disable-output-escaping="yes">#</xsl:text>

<!-- 211 -->
<xsl:call-template name="writeField">
   <xsl:with-param name="inputString">
      <xsl:value-of select="concat(
         PolicyData/PersonData/InsurerPersonData/FirstCustomerData/Name,
         $space,
         PolicyData/PersonData/InsurerPersonData/FirstCustomerData/Surname
         )"/>
   </xsl:with-param>
</xsl:call-template>
```

Slika 23: XSL datoteka, ki se uporabi pri pretvorbi XML datoteke

5.6 Uporaba obdelave tiskanja polic

Obdelava za tiskanje polic se tako kot vse druge obdelave poganja preko ukazne vrstice. Stranke uporabljajo datoteko z nastavitvami (slika 19), preko katere določijo katere tipe zavarovalnih polic izpisujejo, za katero državo gre, izbirajo, če omogočajo tiskanje dvojnikov polic in ostale krmilne podatke. Poleg tega imajo tudi dostop do datotek z prevodi, tako da lahko (če želijo), sami popravljajo spremenljive tekste, ki se dokončno izpišejo na polici.

Obdelava zbere vse potrebne podatke v ustrezne datoteke. Te datoteke se nato pošljejo tiskarju, ki natisne dokončno obliko police. Zato datoteke obdelave tiskanja polic vsebujejo tudi krmilne podatke z navodili za tiskarje, preko katerih ti ugotovijo, za katero vrsto police gre in katero obliko morajo izbrati pri tiskanju.

Ker se bere veliko datotek, je v obdelavo v ozadju vključen tudi sistem napak, ki se zaveda poslovne logike zavarovalnice. Ker gre pri tiskanju polic večinoma za nove police, omogoča ta sistem zgodnje odkrivanje napak, ki so nastali pri vnosih podatkov v bazo ob sklenitvi zavarovanj.

6 ZAKLJUČKI

Na koncu želimo ovrednotiti uporabljena orodja in samo poslovno aplikacijo oz. njeno nadgradnjo

6.1 Orodja

Ključno orodje za izgradnjo aplikacije je IDE Eclipse. Orodje se je izkazalo kot neprecenljivi pripomoček. Zaradi svojega sistema vstavkov omogača povezljivost vseh tehnologij, ki si jih želi razvijalec uporabiti pri razvoju aplikacije. Zelo pomembno dejstvo je, da se gre pri tem za brezplačno orodje, ki ima veliko zaledje razvijalcev, zato so v primeru težav popravki zelo hitro na voljo. Sam vmesnik je tudi prijazen do uporabnika in omogoča razvijalcem hiter pregled nad spremembami v kodi. Primer je recimo uporaba Eclipse-a z vstavkom `subclipse`, ki omogoča povezavo do SVN repozitorija.

Za delo z bazami je bila izbira DB2 primerna. V preteklosti se je namreč uporabljalo MySQL bazo, ki se je pri povečevanju aplikacija izkazala kot slabša izbira. Prednost baze DB2 je velika množica pomožnih upravljalskih programov in dobra podatkovna varnost baze. Res pa je, da je DB2 plačljiv, vendar to za večje podjetje ne bi smelo biti problem.

Za hranjenje podatkov se je XML pokazal kot najboljša izbira. Omogoča jasen in pregleden zapis podatkov, ki so lahko razumljivi človeku. Zaradi tega in ker obstaja kopica orodij za delo z XML dokumenti, priporočam XML za uporabo pri prenosu podatkov med poslovnimi aplikacijami.

Zaradi preprostosti XML in njegove drevesne strukture je tudi uporaba orodij za pretvorbe dokaj preprosta. Omogočajo praktično vse, kar si razvijalec želi pri prikazu podatkov. Večina teh orodij je prav tako brezplačnih. Vendar bi želel omeniti, da smo pri razvoju aplikacije uporabili več jezikov za pretvorbe in med njimi se je gotovo najbolje odrezal protvornik Saxon, ki je bil med vsemi izbirami najhitrejši. Ravno hitrost pretvarjanja zna biti največji problem uporabe jezikov za pretvorbe. Pri naši obdelavi z uporabo XSLT datoteke dolžine 4000 vrstic se je že opazil občuten padec hitrosti pretvorb, ki je bil sicer še vedno na zadovoljivi ravni. Možna je tudi uporaba specifičnejših metod za pretvorbe (kot naprimer orodja XSL-FO ali kakšnih drugih), vendar se je pri razvoju naše aplikacije to izkazalo za nepotrebno.

Razvoj aplikacije vedno predstavlja velik izziv. Po eni strani je potrebno zagotoviti vso funkcionalnost, ki jo posamezno podjetje želi, po drugi strani pa je to potrebno narediti čim hitreje. Izkaže se da se izplača na začetku razvoja dobro analizirati zahteve in se lotiti izgradnje zadeve čim bolj modularno. Na ta način je možno zagotoviti določene vzorce, ki se jih lahko uporabi pri dodajanju vseh funkcionalnostih aplikacije. To olajša razumljivost aplikacije razvijalcem, ko le-ta postane velika, prav tako pa omogoča tudi hitrejši nadaljni razvoj.

Predstavljena aplikacija je skupek obdelav. Vse obdelave uporabljajo skupne pojme za uporabljene objekte in tudi enak vzorec za obdelavo objektov, kar ima za posledico enako zaporedje izvajanja. Taka struktura omogoča tudi v prihodnje relativno preprosto dodajanje novih funkcionalnosti v poslovno aplikacijo.

6.2 Sklep

Poslovne aplikacije so nedvomno eno izmed področij, o katerih bo še veliko govora na področju informacijskih tehnologij. Zaradi vedno večjih potreb po prikazu in izmenjavi podatkov na racionalen in urejen način imajo tovrstne aplikacije že danes veliko možnosti.

Učinkovit razvoj, izvedba ter nadgradnja in upravljanje poslovnih aplikacij je ključnega pomena za njihovo uporabo. Naša diplomska naloga je lahko dober prikaz, kaj potrebuje razvijalec poslovne aplikacije za uspešno izvedbo projekta na tem področju.

7 ZAHVALA

Zahvaljujem se mentorju na Fakulteti za računalništvo in informatiko v Ljubljani viš. pred. dr. Igorju Rožancu za vso pomoč in podporo pri izdelavi diplomske naloge.

Zahvaljujem se staršem, ki so mi omogočili študij in me podpirali, ter sodelavcem za strokovno in moralno pomoč pri pisanju diplomske naloge.

Nazadnje se zahvaljujem še vsem ostalim, ki so kakorkoli prispevali k nastanku te diplomske naloge in pa prijateljem, ki so me pri tem ves čas podpirali.

8 VIRI

[1] *Elektronsko poslovanje*. Vir na spletu:
http://en.wikipedia.org/wiki/Electronic_business

[2] Miro Gradišar, Jurij Jaklič, Tomaž Turk. *Prosojnice pri predmetu Poslovni informacijski sistemi*, Ekonomska fakulteta 2007.

[3] *Informacijski sistem*. Vir na spletu:
http://sl.wikipedia.org/wiki/Informacijski_sistem

[4] DB2 (Database 2 and IBM DB2) Definition. Vir na spletu:
http://www.pitpipe.com/tlist/DB2.html

[5] DB2 Definition. Vir na spletu:
http://www.pcmag.com/encyclopedia_term/0,2542,t=DB2&i=40930,00.asp

[6] IBM DB2. Vir na spletu:
http://en.wikipedia.org/wiki/IBM_DB2

[7] Extensible Markup Language. Vir na spletu:
http://en.wikipedia.org/wiki/Xml

[8] Extensible Stylesheet Language. Vir na spletu:
http://www.w3schools.com/Xsl/xsl_languages.asp

[9] XPath. Vir na spletu:
http://www.w3schools.com/xpath/xpath_intro.asp

[10] Extensible Stylesheet Language Transformations. Vir na spletu:
http://www.w3schools.com/Xsl/xsl_intro.asp

[11] Tomaž Mohorič, Aljaž Zrnec. *Prosojnice Transformacijski spletni jezik. Vir na spletu:*
http://www.ltfe.org/pdf/Transformacijski_spletni_jeziki.pdf.pdf

[12] XSL-FO. Vir na spletu:
http://en.wikipedia.org/w/index.php?title=XSL_Formatting_Objects

[13] Java (*programming language*). Vir na spletu:
http://en.wikipedia.org/wiki/Java_programming_language

[14] A. Leonardis. *Prosojnice za predmet: Razvoj aplikacij*. Fakulteta za računalništvo in informatiko, Ljubljana 2004.

[15] DB2 Control Center overview. Vir na spletu:
http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.cc.doc/db2_udb/intro.htm

[16] Razvojno okolje (IDE). Vir na spletu:
http://en.wikipedia.org/wiki/Integrated_development_environment

[17] Razvojno okolje Eclipse. Vir na spletu:
http://en.wikipedia.org/wiki/Eclipse_(software)

[18] Sistem za delo z verzijami (Subversion). Vir na spletu:
http://en.wikipedia.org/wiki/Subversion_(software)

[19] Klient sistema za delo z verzijami TortoiseSVN. Vir na spletu:
http://en.wikipedia.org/wiki/TortoiseSVN

[20] Primer standardne datotečne strukture. Vir na spletu:
http://www.informit.com/articles/article.aspx?p=30183

IZJAVA O SAMOSTOJNOSTI DELA

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod mentorstvom viš. pred. dr. Igorja Rožanca. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Primož Vranešič