UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Dejan Stopar

Upravljanje poslovnih procesov

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Marko Bajec

Ljubljana, 2009

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: 00445/2009
Datum: 05.04.2009

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

Kandidat: **DEJAN STOPAR**

Naslov: **UPRAVLJANJE POSLOVNIH PROCESOV**
BUSINES PROCESS MANAGEMENT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

V diplomskem delu naj kandidat poda pregled nad področjem upravljanja poslovnih procesov. V tem okviru naj razvrstiti pojme in tehnologije po logičnih področjih ter zgodovinsko opredeliti in ovrednotiti pomembnost tehnologij in standardov, ki se na tem področju uporabljajo. Cilj naloge naj bo opredeli tiste vidike upravljanja poslovnih procesov, ki so najbolj pomembni z vidika informatika oziroma razvijalca programske opreme.

Mentor:

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:

prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Dejan Stopar,

z vpisno številko 63020144,

sem avtor diplomskega dela z naslovom:

Upravljanje poslovnih procesov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Marka Bajca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne Podpis avtorja:

Zahvala

Največja zahvala gre njej. Moji mami Lilijani. Njej, ki je trdo delala in se odrekala zame. Njej, ki ni nikoli niti za trenutek podvomila vame in v moje cilje.

Zahvalil bi se rad moji sestri Kristini, ki je kljub moji občasni muhavosti in trdoglavosti, pravtako verjela vame in me venomer spodbujala.

Posebna zahvala gre punci Ivani, ki mi je pri končanju študija nudila pomoč, podporo in ogromno ljubezni.

Kazalo

Slike

Tabele

Povzetek

Poslovni procesi so ključni del vsake združbe. Predvsem v večjih združbah je optimizacija poslovnih procesov in hitrost, s katero združba spreminja in uspešno uvaja nove procese, ključnega pomena za uspešno delovanje in konkurenčno prednost na trgu. Tako je v zadnjih letih ena izmed najbolj vročih tem upravljanje poslovnih procesov in z njo povezane metode, tehnologije in programske rešitve. Disciplina upravljanja procesov ne pogojuje uporabo tehnologije, vendar pa je v času razcveta računalništva, interneta in vsesplošne globalizacije najbolj logični korak pri optimizaciji poslovnih procesov prav avtomatizacija s pomočjo informacijske tehnologije.

Namen diplomske naloge je podati pregled nad področjem upravljanja poslovnih procesov, razvrstiti pojme in tehnologije po logičnih področjih, zgodovinsko opredeliti in ovrednotiti pomembnost tehnologij in standardov. Cilj je podati tiste informacije o upravljanju poslovnih procesov, ki so najbolj pomembne z vidika informatika oziroma razvijalca programske opreme.

V začetnem poglavju opredelimo ključne pojme, ki se v povezavi s tem obširnim področjem pojavljajo, opišemo metode, pomembne organizacije, standarde in tehnologije. Tretje poglavje je namenjeno jeziku za modeliranje poslovnih procesov BPMN, četrto pa jeziku za izvajanje poslovnih procesov BPEL. V petem poglavju, praktičnem delu diplomske naloge, prikažemo še primer implementacije z uporabo odprtokodnega sistema za upravljanje s poslovnimi procesi imenovanega jBPM. Pri tem prikažemo bistvene korake, ki so potrebni za avtomatizacijo poslovnega procesa, od modeliranja, načrtovanja do zagona procesa. Zaključno poglavje podaja sklepne ugotovitve ter predstavi bodoče spremembe in smernice na področju upravljanja s poslovnimi procesi.

Ključne besede:

BPM, upravljanje, procesi, BPEL, BPMN, SOA, jBPM, Seam

Abstract

Business processes play a crucial role in every company. Especially in major companies the optimization of business processes and the speed of changing and introduction of new processes is crucial for successful operation and competitive position on market. Business process management and its methods, technologies and program solutions have been one of the most popular issues in recent years. The discipline of process managing does not condition the use of technologies, but in the age of computers, internet and universal globalization the most logical step is the automation with the help of information technologies.

The purpose of the diploma work is to hand a survey of the fields of business process management, to classify the notions and technologies, to give a historical determination and to evaluate the importance of technologies and standards. The intention is to offer those information about business process management which are most relevant from the point of view of information scientist or software developer.

The first part defines the key notions that occur in association with this large field, then it describes the methods, important organizations, standards and technologies. The third chapter deals with the business process modelling language BPMN and the fourth one treats the business process execution language BPEL. The fifth chapter, the practical part of the diploma work, shows an example of the implementation with the use of an open source system for the business process management called jBPM. We show the essential steps, needed for the automation of business processes from process modelling, design to execution. The last part hands the conclusions and sets the future changes and the guidelines in the field of business process management.

Key words:

BPM, management, processes, BPEL, BPMN, SOA, jBPM, Seam

Poglavje 1

Uvod

Upravljanje poslovnih procesov (ang. Business Process Management - v nadaljevanju BPM) je trenutno eno izmed najbolj vročih področij v današnji industriji programske opreme. Ponaša se z mnogimi proizvajalci in standardi, nenavadno poimenovanimi akademskimi teorijami in širokim spektrom potencialnih uporabnikov, od raziskovalcev do inženirjev, poslovnih analitikov, poslovnih arhitektov in menedžerjev. Z večanjem zanimanja je bilo v sklopu BPM-ja integriranih veliko metodologij in paradigem na področjih teorije upravlja organizacije, računalniške znanosti, matematike, lingvistike in filozofije, kar je povzročilo, da je BPM postal meddisciplinarna "teorija v praksi". Prav zaradi vpletenosti vseh teh tehnologij, akterjev in znanosti se raziskovanje in prakticiranje sooča s podvajanjem in nerazumevanjem.

To pa ne pomaga računalniškim inženirjem, ki bi to področje hoteli spoznati in razumeti. Pogost problem, s katerim se BPM sooča, je pomanjkanje univerzalne terminologije [4, 11, 12]. Pojmi se uporabljajo ohlapno in največkrat je odgovor na vprašanje, kaj sploh je BPM, odvisen od sogovornikov in konteksta razprave [7]. Tak primer je enačenje poslovnih procesov s spletnimi storitvami (ang. Web Services) ali pa zmeda med pojmi BPM, BPR (ang. Business Process Reengineering) in WfM (ang. Workflow Management). Napačno tolmačenje je prispevalo tudi k neusklajenim (ali, še slabše, napačnim) implementacijam na področju upravljanja procesov. Dodatno kompleksnost povzroča množica disciplin, metod, tehnologij in polstandardov, ki so velikokrat vezane na posameznega ponudnika programske rešitve. Samo za oris naj omenimo nekaj pogostih pojmov, ki se omenjajo v povezavi z BPM: že omenjenim SOA, WfM, BPR lahko dodamo še BPI, EAI, BPEL, BPMN, BPDM, WSFL, PDL, BAM, Six Sigma, Lean itn. Vsa ta zmeda in pomanjkanje pregleda je mogoče za začetnika na tem področju prehuda, zato je lahko

BPM neupravičeno spregledan kot še ena modna muha ali marketinški trik.

Namen diplomske naloge je zato zlasti podati uvod v vse ključne sfere BPM-ja. Poudarek je na pojmih in računalniških tehnologijah, ki so zanimive s perspektive računalniškega inženirja ali študenta računalništva. V prvem poglavju se bomo osredotočili na definicije BPM-ja in ostalih povezanih pojmov. Pokazali bomo, kakšne so njihove relacije, in med drugim tudi, kako v sklop BPM-ja sovpadata SOA in WfM. Poglavji dve in tri opisujeta standarda BPMN in BPEL. Prvi je standard pri modeliranju poslovnih procesov, medtem ko se drugi uporablja pri definiranju procesov, ki jih lahko izvaja procesni pogon. Pri obeh jezikih podamo kratko zgodovino in navedemo razloge, ki so privedli do sprejetja obeh standardov. Kljub temu, da obstaja množica drugih jezikov, sta BPMN in BPEL dandanes splošno sprejeta kot *de facto* standarda na področjih BPM in SOA. Oba sta del ključnih korakov življenjskega cikla BPM - modeliranja in izvajanja.

Ker nobena teorija ne zdrži brez prakse, smo se odločili implementirati praktičen primer procesa. Pri tem smo uporabili tehnologije, ki so prosto dostopne. Za ogrodje in procesni pogon smo izbrali jBPM v povezavi z JBoss Seam ogrodjem; vse skupaj pa se je izvajalo na JBoss aplikacijskemu strežniku. Za modeliranje procesa smo si pomagali z vtičnikom *jPDL Process Designer* za programsko orodje Eclipse.

Poglavje 2

Koncepti BPM

2.1 Poslovni proces

Upoštevajoč tradicionalne poglede Fredericka W. Taylorja na procese na področju znanstvenega vodenja, lahko moderne in eksplicitne definicije pojma *poslovni proces* zaznamo v zgodnjih 90. letih prejšnjega stoletja na področju BPR (ang. Business Process Re-engineering)[1]. Temeljna dela Hammerja in Champya [1] definirajo poslovni proces kot skupek aktivnosti, ki sprejmejo enega ali več vrst vhodov in ustvarijo izhod, ki je pomemben za stranko. Poslovni proces ima cilj, nanj pa delujejo dogodki iz zunanjega sveta ali iz drugih procesov. Definicija je trdna, kljub njeni preprosti formi, saj povzame vse možne permutacije tokov poslovnih procesov v realnosti. Namesto, da gledamo na poslovne procese samo kot na "skupek aktivnosti", pa moramo pri tem vendarle upoštevati še sistematičnost in zaporedje aktivnosti v prostoru in času.

V svojem delu Davenport [2] poudarja, da mora tako strukturo nujno podpirati informacijska tehnologija. Poslovni proces definira kot strukturirano, merljivo množico aktivnosti, ki imajo kot rezultat nek izhod, ki je namenjen določeni stranki ali trgu. Poudarek je na tem, *kako* je delo opravljeno znotraj združbe, v nasprotju s klasičnim poudarkom *na čemu*. Proces je tako specifično zaporedje delovnih aktivnosti v prostoru in času, z začetkom in koncem, in jasno opredeljenimi vhodi in izhodi.

Medtem ko sta prvi dve definiciji opredelili cilje in strukturo toka poslovnega procesa, sta še vedno manjkala dva pomembna elementa: akterji, ki opravljajo delovne aktivnosti, in sodelovalna narava teh akterjev. Po Ouldu [3] poslovni

[1]Metodologija, ki se osredotoča na radikalno spreminjanje in optimizacijo poslovnih procesov z namenom izboljšanja na področju stroškov, kvalitete in storitev.

proces:

- vsebuje namensko dejavnost,
- izvaja ga skupina (ljudi in/ali strojev),
- pogosto prečka funkcijske meje,
- poganja ga zunanji svet.

Tak opis poslovnega procesa vpelje pojem akterjev/vlog in sodelovanja med vpletenimi akterji/vlogami. Torej proces ne izvaja samo posameznik ali oddelek, ampak lahko vključuje več ljudi, strojev in sistemov iz različnih združb, ki sodelujejo z namenom izpolnitve skupnega poslovnega cilja.

2.2 Upravljanje poslovnih procesov - BPM

2.2.1 Splošna definicija

Kljub temu, da obstaja na stotine definicij o BPM, se bomo omejili na tisto, ki jo definira organizacija ABPMP[2] (ang. Association of Business Process Management Proffesionals), ki pravi [14]:

> *"Upravljanje Poslovnih Procesov (oz. BPM) je organiziran in disciplиniran pristop k identifikaciji, načrtovanju, izvajanju, dokumentiranju, spremljanju, nadzorovanju in merjenju tako avtomatiziranih kot neavtomatiziranih poslovnih procesov, zato da bi zagotovili enakomerne, ciljne rezultate, konsistentne s strateškimi cilji združbe."*

2.2.2 Kaj sploh zajema BPM

Kljub interdisciplinarnosti, ki smo jo omenili v uvodu, je zgornja definicija dovolj splošna, da lahko zajame vse discipline in področja, ki se ukvarjajo z upravljanjem poslovnih procesov. Najpomembnejši del definicije je tisti, ki omenja avtomatizirane in neavtomatizirane poslovne procese. Eden izmed glavnih problemov, ki vlada v BPM skupnosti in ki povzroča največ zmede, je namreč ravno medsebojno nerazumevanje med posameznimi skupinami in samosvoje tolmačenje o tem, kaj zajema BPM. Tako skrajno nasprotje lahko

[2]Ne-profitna organizacija, sestavljena iz neodvisnih strokovnjakov, ki si prizadevajo za širitev BPM konceptov in praks.

najdemo pri skupinah, ki zagovarjajo predvsem BPI ali BPR (tudi z metodologijama Six Sigma in Lean[3]) - torej optimizacijo poslovnih procesov s pomočjo omenjenih metod in postopkov in manjšim poudarkom na tehnologiji ali celo brez nje - na drugi strani pa skupine, ki dajejo bistven poudarek tehnologiji (npr. SOA).

Odgovor na to, zakaj prihaja do takih nasprotjih, lahko najdemo v zgodovini razvoja BPM, na katerega je bistveno vplival razvoj informacijske tehnologije. V začetku 90. let, ko se je računalniška tehnologija šele začela pojavljati, so imeli primarno vlogo pri optimizaciji poslovnih procesov menedžerji, organizatorji in za to posebej usposobljeni ljudje. Z vse večjim razmahom informacijske tehnologije in interneta in posledično tehnološke podprtosti poslovanja, je postalo sodelovanje z informatiki/računalniškimi inženirji (oz. IT sektorjem nasploh) nuja. Poudarek je na "sodelovanju", saj je le taka oblika lahko zagotavljala uspešne rezultate. To je predstavljajo velik napredek v primerjavi s klasičnim mišljenjem, ki je najraje ločevalo delo poslovnih ljudi od računalniških inženirjev. Temu vmesnemu obdobju je sledilo današnje obdobje, kjer obliko BPM diktira uporabljena tehnologija oz. izbrani ponudnik programskega paketa - BPMS (ang. Business Process Management Suite).

Kljub temu, da je danes uporaba tehnologije skorajda nujna, torej z izrazom BPM ne predpisujemo nobene konkretne metodologije ali tehnologije. Vsaka združba ali programska hiša, ki nudi BPMS, si lahko načeloma zamisli svojo obliko in obseg BPM-ja. Ne glede na to, pa je, kot omenja splošna definicija, skupni cilj "vsakega" BPM-ja optimizacija procesov in s tem doseganje boljših, enakomernejših poslovnih rezultatov.

Po vsem povedanem lahko povzamemo, da z besedo BPM danes označujemo skupek metod, orodij in tehnologij, s katerimi načrtujemo, sprejemamo, analiziramo in kontroliramo operativne poslovne procese. BPM sloni na procesnem pristopu za izboljšanje učinkovitosti, ki kombinira razne informacijske tehnologije s procesnimi upravljavskimi metodologijami. Je sodelovanje med poslovnimi ljudmi in informatiki, z namenom, da spodbuja učinkovite, prilagajajoče in transparentne poslovne procese. BPM vključuje ljudi, sisteme, funkcije, posle, stranke, dobavitelje in partnerje.

[3]Metodologiji sta razvila Motorola oz. Toyota ob koncu 80. let. Gre za optimizacijo procesov (predvsem) v industriji, ki predpisuje posebne postopke in reorganizacijo za dosego optimalnih rezultatov in zadovoljstvo strank.

2.2.3 Zakaj "upravljati" s poslovnimi procesi

V realnem svetu težko najdemo posameznika, ki bi imel popoln pregled nad poslovnimi procesi v celotni združbi [10]. Znanje o tem, kako potekajo procesi, se ponavadi skriva v glavah zaposlenih in pogosto vsak od njih pozna samo svoj del procesa. Vodstvo pogosto ne motivira zaposlene, da razmišljajo o svojih procesih ali celo o tem, da bi jih lahko izboljšali. Potemtakem procesi ostanejo nespremenjeni in neoptimalni. Vodstvo se tako lahko upravičeno sprašuje:

- ali so naloge in aktivnosti v procesu optimalno organizirane,
- katere naloge in aktivnosti porabijo največ časa,
- kako so naloge in aktivnosti porazdeljene med zaposlenimi,
- kako efektivni so zaposleni.

Če želimo dobiti odgovore na zgornja vprašanja, moramo najprej dodobra razumeti poslovne procese. In prav v tem je bistvo BPM-ja, saj nas sili k razumevanju procesov in njihovi optimizaciji. Prednosti uvedbe upravljanja poslovnih procesov so tako:

- *formaliziranje obstoječih procesov in odkrivanje morebitnih optimizacij* - kot že rečeno, BPM narekuje, da se v procese poglobimo in jih skozi modele formaliziramo (tako lahko pridemo do optimizacij, kot npr. odstranitev nepotrebnih korakov, avtomatizacija ročnih korakov, zamenjava dela procesa z optimalnejšim delovnim tokom itn.),
- *pohitritev in optimalno izvajanje procesov* - BPM programska oprema lahko izvaja procese neprekinjeno in brez zaostankov; tako je časovni zamik med posameznimi aktivnostmi skorajda ničen, poleg tega pa nam oprema omogoča paralelno izvajanje, tako da se lahko procesi izvajajo istočasno in neodvisno drug od drugega,
- *povečanje produktivnosti z manj delovne sile* - primeri kažejo, da pravilna uvedba BPM-ja privede do povečanja produktivnosti, zmanjšanja potrebne administrativne delovne sile, zmanjšanja časa, potrebnega za obdelavo posameznih zahtevkov, in s tem do večjega zadovoljstva strank,
- *Omogočanje ljudem, da rešujejo bolj zapletene probleme* - čeprav BPM pogosto pomeni odstranitev ali zmanjšanje delovne sile, je ena izmed njegovih prednosti tudi v flekibilnejši uporabi ljudi, saj se lahko ti posvečajo zahtevnejšim problemom in jih tako tudi učinkoviteje rešujejo.

- *lažje ustrezanje poslovnim zahtevam in direktivam* - uporaba BPM-ja pomaga podjetjem do poenostavljenih in preglednih procesov, ki ustrezajo raznim regulatorskim zahtevam.

2.2.4 Procesno orientirane aplikacije

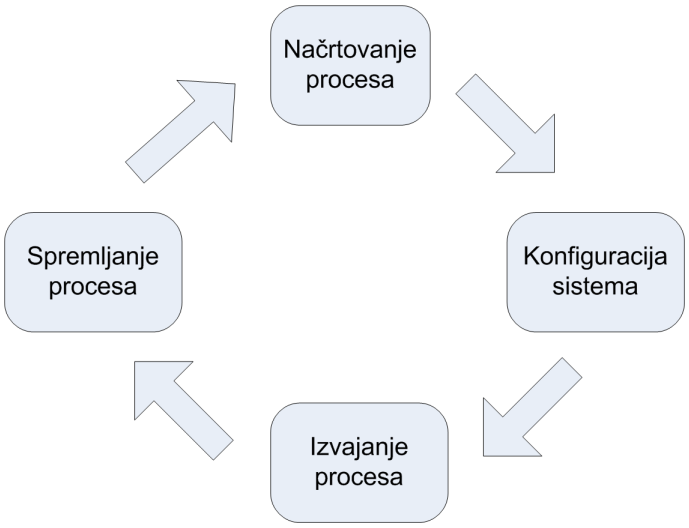
Nenehno pojavljanje pojma BPM nam lahko daje zmotno predstavo, da je BPM rešitev za vse aplikacijske probleme. Konec koncev BPM omogoča poslovnemu analitiku, ki najbolje pozna aplikacijske zahteve, da modelira proces do visokonivojske logike. Problem tega argumenta je v tem, da logika procesa ni v njegovi proceduralni implementaciji, ampak v deklarativnem dinamičnem obnašanju ali v spreminjanju stanja skozi čas. BPM je tako primeren samo za aplikacije, ki imajo lastnosti stanja ali procesa - so procesno orientirane. Za take aplikacije je značilno, da [4]:

- se izvajajo dlje časa - od začetka do konca, proces lahko traja ure, dneve, tedne, mesece in več,
- hranijo stanje v podatkovni bazi - ker se izvajajo dlje časa, se stanje aplikacije hrani v podatkovni bazi, kjer lahko preživi izpade strežnikov, na katerih se izvaja,
- večinoma mirujejo - proces večino časa miruje in čaka na dogodek, ob sprožitvi dogodka pa izvede niz aktivnosti,
- orkestracija sistemskih in človeških komunikacij - proces je odgovoren za upravljanje in koordiniranje komuniciranja med različnimi sistemi in človeškimi akterji.

2.2.5 Življenski cikel BPM

Zaradi raznovrstnosti BPM-ja imamo več pogledov na to, kakšen je njegov osnovni življenjski cikel [6, 4, 5]. Van der Aalst, ki velja za avtoriteto na področju BPM, definira cikel na način, kot ga prikazuje slika 2.1.

1. Načrtovanje procesa - proces se modelira v elektronski obliki in vnese v BPM sistem (BPMS).
2. Konfiguracija sistema - v tem koraku poteka konfiguracija BPMS in infrastrukture na spodnjem sloju (npr. sinhronizacija vlog in organizacijske sheme).



Slika 2.1: Življenski cikel BPM, kot ga definira van der Aalst.

3. Izvajanje procesa - elektronsko modelirane procese se objavi[4] (ang. deploy) na BPMS.

4. Spremljanje procesa - s primernimi orodji za analizo in monitoring se spremlja izvajanje procesov. Pri tem se ugotavlja morebitna ozka grla ali pomanjkljivosti v poslovnih procesih.

Načrtovanje procesa

Načrtovanje procesa je prva stopnja, pri kateri se odločimo, da bomo obstoječi ali nov poslovni proces avtomatizirali. Pogosto srečamo naslednje faze:

- postavitev strategije in planiranje procesa,
- analiza poslovnega procesa,
- načrtovanje in modeliranje procesa.

Pri tem so faze odvisne od pristopa k BPM. Prvi dve fazi narekuje uporabljena metodologija (BPI, BPR, samosvoje prilagojene metode ali tudi brez), medtem ko se pri tretji najpogosteje uporablja vizualno modeliranje procesov. Modeliranje lahko nudijo samostojna orodja [18, 16, 17], ki ne omogočajo nadaljnje

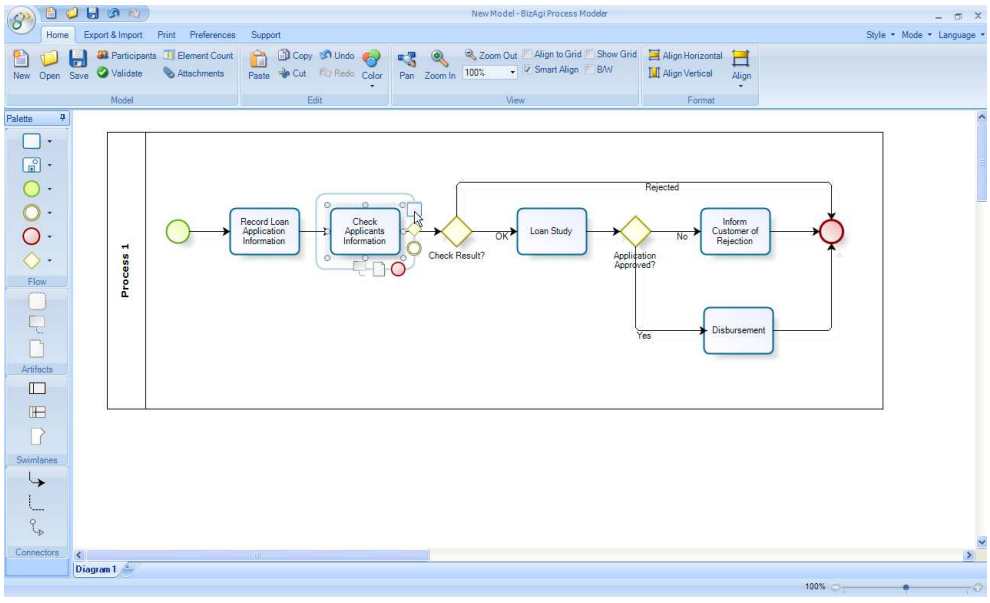
[4]Proces prenosa aplikacije na aplikacijski strežnik, s čimer se aplikacija prične izvajati.

pretvorbe v izvajalni jezik, ali pa so del BPMS (celotnih programskih rešitev) [19, 20, 21].

Skozi modele lahko identificiramo probleme in celo prepoznamo (prej nevidne) optimizacije. Modeliranje je lahko tudi orodje za simuliranje efektivnosti procesov. Prednosti analiziranja in modeliranja so:

- povečanje preglednosti in poznavanja aktivnosti v združbi,
- povečanje možnosti odkrivanja ozkih grl,
- lažja identifikacije področij, ki jih lahko optimiziramo,
- hitrejše izvajanje procesov,
- boljše definiranje vlog in odgovornosti v združbi,
- dobro orodje za revizije/oceno skladnosti ureditve.

Procese lahko modeliramo s pomočjo jezikov, kot so npr. EPC (ang. Event Process Chain), eEPC (ang. Extended Event Process Chain), UML diagrami aktivnosti in zadnja leta BPMN. Zadnji velja kot *de facto* standard na področju modeliranja procesov v BPM.



Slika 2.2: Modeliranje v orodju BizAgi z uporabo notacije BPMN.

Konfiguracija sistema

V tej stopnji se opravi vse nastavitve, ki so potrebne za pravilno delovanje BPMS. To vključuje sam sistem, kot tudi vse ostale notranje in zunanje servise in sisteme, na katerih sloni prej modelirani proces (v primeru SOA npr. storitvene plasti - opravi se implementacija in objava novih spletnih storitev).

V primeru uporabe notacije BPMN, je potrebno opraviti še transformacijo v jezik, ki ga BPMS lahko izvaja - največkrat je to BPEL (ang. Business Process Execution language).

Stopnja je lahko trivialna, v kolikor gre za že vzpostavljen sistem in arhitekturo, kjer se že izvajajo podobni procesi.

Izvajanje procesa

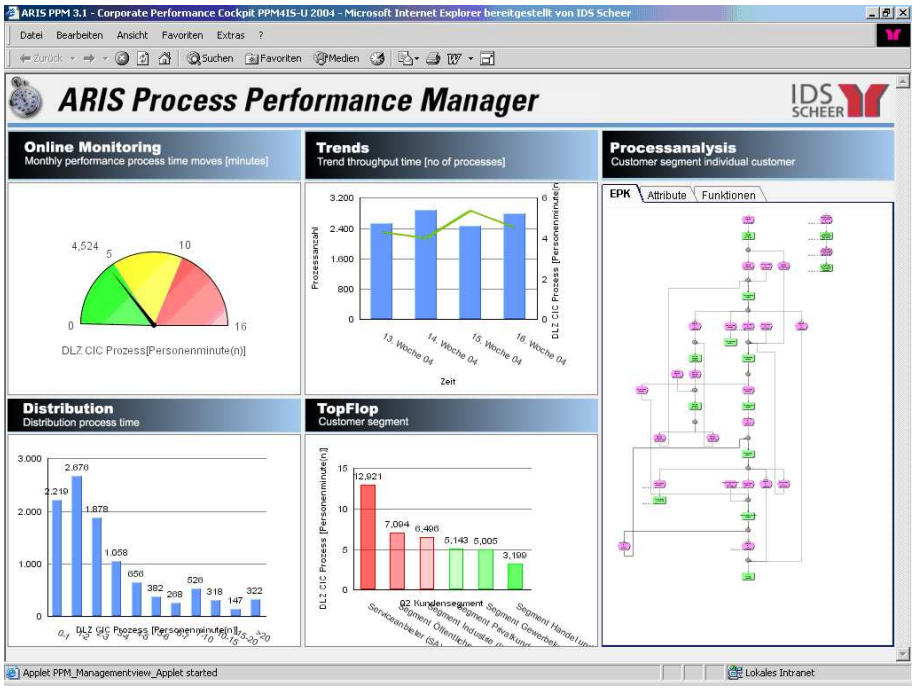
Nov proces se z objavo na BPMS prične izvajati v definiranih korakih. Proces je lahko popolnoma avtomatiziran ali pa zahteva človeško interakcijo. Najpogosteje so procesi mešanica obojega.

Spremljanje procesa

S tem, ko smo avtomatizirali proces, lahko pridobimo podatke o različnih aktivnostih in o tem, kako dolgo se izvajajo. Kvantitativni podatki, ki jih BPMS samodejno beleži, nam omogočajo vpogled v uspešnost aktivnosti in so nam v pomoč pri iskanju ozkih grl in nadaljnji optimizaciji. Orodje oz. vmesnik, ki nam omogoča vizualizacijo podatkov, se imenuje BAM (ang. Business Activity Monitoring) in je običajno del rešitev BPMS. V kolikor je potrebno proces optimizirati ali dopolniti oz. se izkaže, da je napačno zasnovan, se ponovno ponovi prva stopnja življenjskega cikla.

2.2.6 Kako si lahko predstavljamo BPM danes

Če posplošimo, si lahko BPM danes predstavljamo takole: združba uporablja BPM programski paket [19, 20, 21], ki nudi orodja za modeliranje, pretvorbo v izvajalni jezik, konfiguracijo sistema, testiranje, objavo in nadzor procesa. Poslovni analitik modelira proces v notaciji BPMN in ga preda IT inženirju. Ta opravi pretvorbo v izvajalni jezik, običajno BPEL ali v lastniški jezik, ki je specifičen za posamezni BPMS. Ponavadi mora inženir dopolniti proces oz. opis procesa s podatki, ki so potrebni za pravilno izvajanje in komunikacijo s sistemi; definira se tudi ključne parametre za nadzor. Proces se objavi v



Slika 2.3: Primer orodja BAM, ki je dostopno preko spletnega vmesnika.

testnem okolju in s posebnim orodjem simulira izvajanje. V kolikor je testiranje uspešno, se proces objavi v produkcijskem okolju. Poslovni analitik ali menedžer spremlja izvajanje procesa prek orodja BAM. V primeru zaznave ozkega grla ali neoptimiziranih aktivnosti, se proces ponovno modelira in odpravi pomanjkljivosti. BPMS omogočajo zamenjavo obstoječega procesa brez večjih aplikativnih ali sistemskih posegov in prekinitev (takorekoč *"v živo"*).

2.3 Pojmovna in vsebinska zmeda

2.3.1 Upravljanje z delovnimi tokovi - WfM

Ljudje, ki si želijo spoznati področje BPM-a, se največkrat sprašujejo, kakšna je sploh razlika med BPM in delovnim tokom (ang. Workflow) oz. disciplino, ki se ukvarja z njihovim upravljanjem - WfM (ang. Workflow Management). Je BPM samo sodobnejši izraz, ki ga izrabljajo programska podjetja v marketinške namene, ali pa se dejansko ločita po pristopih in uporabljenih tehnologijah in standardih. Zmedo povzroča tudi dejstvo, da današnji BPMS

vključujejo izrazite elemente upravljanja delovnih tokov, t.j. digitalizacija in avtomatizacija poslovnih procesov.

Zgodovinsko gledano je pojem delovni tok nenadno "prešel" v BPM nekje na prelomu tisočletja. Obstajata dva pogleda na razliko med BPM in WfM. Gartner [7] vidi BPM kot upravljavsko disciplino, ki jo WfM podpira kot tehnologija. BPM je procesno orientirana disciplina in ni tehnologija, medtem ko je WfM tehnologija, ki jo najdemo v BPMS in ostalih produktnih kategorijah. Organizacija WfMC[5] (ang. Workflow Management Coalotion) [22] definira WfM kot:

> *"avtomatizacija poslovnih procesov (delna ali v celoti), med katero se dokumenti, informacije ali naloge prenašajo od enega udeleženca do drugega glede na akcije in odločitve, ki so proceduralno določene."*

Podobno Havey [4] pojmuje delovni tok, kot tok dela, ki vključuje izmenjavo in obogatitev informacij. V preteklosti je delovni tok pomenil prenašanje papirjev od človeka do človeka. Tehnologija ni omogočila samo avtomatizacijo procesov, ampak tudi digitalizacijo informacij.

Drugi pogled na razliko med BPM in WfM, ki ga podaja van der Aalst, pravi, da je slednji podmnožica prvega - vendar s pomanjkanjem faze BPM spremljanja in analiziranja (tabela 2.1).

Faza življ. cikla BPM	**WfM**	**BPM**
Načrtovanje procesa	Da	Da
Konfiguracija sistema	Da	Da
Izvajanje procesa	Da	Da
Spremljanje in analiziranje	Skromno	Da

Tabela 2.1: Van der Aalst - razlika med BPM in WfM.

Eden izmed razlogov za zmanjšanje zanimanja za pojem delovni tok je tudi njegova omejenost integracije med podjetji. WfM temelji na ideji centraliziranega pogona, ki koordinira izvajanje znotraj ene domene (podjetja). Z globalizacijo in povečanjem potrebe po enostavni integraciji sistemov in funkcij se je prav ta arhitektura izkazala kot omejitev. Tako se je zanimanje iz sistemov WfM hitro nagnilo v prid BPMS - slednji s pomočjo spletnih storitev in

[5]Organizacija prispeva in skrbi za standarde na področju upravljanja delovnih tokov in procesov (XPDL, Wf-XML)

servisno orientirane arhitekture (SOA) zagotavljajo povezovanje distribuiranih sistemov.

Če želimo videti razliko med BPM in WfM, moramo v kontekst vedno vzeti razvoj tehnologije in razvoj teorij organiziranja skozi zgodovino. Na upravljanje poslovnih procesov moramo gledati kot na logično in postopno razširitev področja upravljanja delovnih tokov.

2.3.2 Servisno orientirana arhitektura - SOA

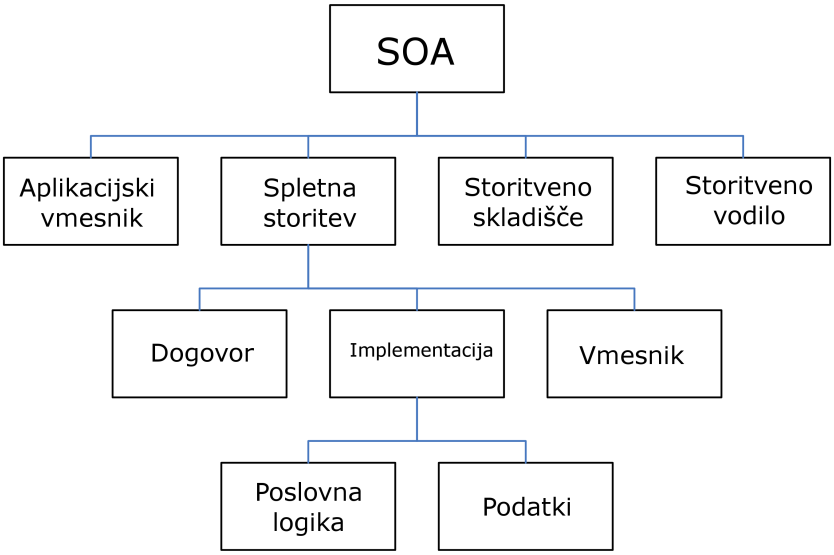
Kratica, ki se danes najpogosteje omenja v povezavi z BPM, je SOA. Kot pri že omenjenem WfM, se ta dva pojma večkrat mešata, enačita ali striktno ločujeta. Najprej si poglejmo, zakaj se je pojavila potreba po SOA oz. katero področje primarno naslavlja.

Današnji tipični informacijski sistem sestoji iz različnih heterogenih aplikacij, ki so bile razvite skozi čas. Največkrat so to:

- samorazvite rešitve,
- prilagojene, vendar zunanje izvedene rešitve,
- komercialne rešitve, kot npr. ERP, CRM, SCM in podobno.

Ti sistemi uporabljajo različno arhitekturo (klient/strežnik, večnivojska arhitektura), različne tehnologije in različne jezike za implementacijo (C++, Java, C#, Visual Basic itn.). Integracija takih sistemov je zelo problematična (tako začetno povezovanje, kot nadaljnja sprememba povezav zaradi spremenjenih zahtev oz. procesov). Tu nastopi SOA, ki zagotavlja medsebojno komunikacijo različnih informacijskih sistemov s pomočjo interoperabilnih servisov. Če poenostavimo, je to arhitektura, ki nudi na najvišjem nivoju servise v obliki spletnih storitev (storitvena plast), pri tem pa je lahko vsak servis implementiran v poljubni tehnologiji. S tem, ko imamo na voljo spletne storitve, ki izvedejo določeno operacijo, lahko servise povezujemo v zaokrožene funkcionalne celote znotraj samega podjetja, tehnologija pa omogoča tudi povezovanje z zunanjimi podjetji ali celo mednacionalnimi podjetji. Pri tem se lahko servisi uporabljajo večkrat v različnih celotah, kar je tudi eno izmed osnovnih načel SOA (ang. service reusability).

Sami servisi in arhitektura pa niso dovolj, da bi samo s tem dosegali višji in boljši nivo integracije in izkoriščenosti potencialov celotnega sistema. Tu nastopi BPMS, ki deluje kot koordinator oz. povezovalec teh servisov. Jurič [10] imenuje to SOA, ki jo vodijo poslovni procesi (ang. Business Process Driven SOA). Gartner [7] pravi, da BPM "organizira *ljudi* za večjo agilnost",



Slika 2.4: Elementi SOA, pri čemer je glavni element spletna storitev.

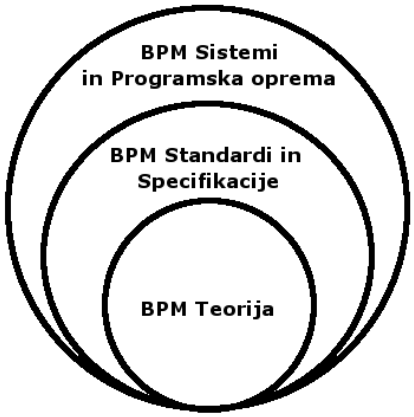
medtem ko SOA "organizira *tehnologijo* za večjo agilnost". Z drugimi besedami: procesi v SOA omogočajo koordinacijo distribuiranih sistemov, ki podpirajo poslovne procese, zato se teh procesov ne sme mešati s poslovnimi procesi.

Sami servisi in arhitektura pa niso dovolj, da bi samo s tem dosegali nek višji in boljši nivo integracije in izkoriščenosti potencialov celotnega sistema. Tu nastopi BPMS, ki deluje kot koordinator oz. povezovalec teh servisov. Jurič [10] imenuje to *SOA, ki jo vodijo poslovni procesi* (ang. Business Process Driven SOA). Gartner [7] pravi, da BPM "organizira *ljudi* za večjo agilnost", medtem ko SOA "organizira *tehnologijo* za večjo agilnost". Z drugimi besedami: procesi v SOA omogočajo koordinacijo distribuiranih sistemov, ki podpirajo poslovne procese, zato se teh procesov ne sme mešati s poslovnimi procesi.

2.3.3 BPM teorija, standardi, jeziki in sistemi

Trenutno dela na BPM standardih okoli 10 skupin. Med njimi je 7 skupin, ki se ukvarja z definicijami modeliranja. Zato ne preseneča, da je BPM področje od devetdesetih let naprej postalo zelo fragmentirano. Zmešnjava je bila tako velika, da je celo teorija postala zmedena glede standardov BPM in standardov BPMS.

Kot prikazuje slika 2.5, BPM standardi in specifikacije slonijo na uveljavljeni BPM teoriji (npr. Pi Calculus in Petri Nets) in so v v končni fazi



Slika 2.5: Relacija med BPM teorijo, standardi in sistemi.

implementirani v programsko opremo. BPM standardi in BPM sistemi so tudi to, kar Gartner [9] imenuje *"Tehnologije, ki omogočajo BPM"* (ang. BPM-Enabling Techonologies).

Heterogenost modeliranja poslovnih procesov je znan problem v BPM [8]. Tabela 2.2 poskuša področje poenostaviti tako, da za vsako tehniko modeliranja ali standard, prikaže področje (BPM, B2B ali SOA), ozadje, uporabo (npr. izvajanje), trenutni status in, ali je že standardiziran.

2.4 Organizacije in njihove vloge pri razvoju standardov BPM

Organizacija W3C (World Wide Web Consortium) je glavno telo, odgovorno za vse pomembnejše standarde tehnologij, ki so značilne za obdobje svetovnega spleta. Tako spadajo pod njihovo okrilje HTTP, CSS, XML in SOAP (najpopularnejši standard na področju spletnih storitev). Kljub temu pa W3C ni vpleten pri standardizaciji BPM tehnologij. Vzroke za to lahko iščemo v tem, da se niso mogli tako hitro prilagajati pritiskom velikih podjetij, ki so nenehno pošiljala na trg nove specifikacije, poleg tega pa velja za vse tehnične komiteje standardni pravilnik o intelektualni lastnini, kar pa ni bilo po godu podjetjem, ki so inovacije ustvarjala.

Z namenom pospešitve splošnega sprejetja in uporabe tehnologij in rešitve zapletov okoli intelektualne lastnine, so velika podjetja ustanovila svojo lastno organizacijo: WS-I (Web Services Interoperability). WS-I ni pripravila nobenih

	BPM/ SOA/ B2B	**Ozadje**	**Teorija/ Grafični/ Izmenjava/ Izvajanje / Diagnoza/ B2B Izmenjava**	**Std.?**	**Trenutni status**
BPDM	BPM	Industrija	Izmenjava	Da	Nedokončan
BPEL	BPM	Industrija	Izvajanje	Da	Široko sprejet
BPML	BPM	Industrija	Izvajanje	Da	Opuščen
BPQL	BPM	Industrija	Diagnoza	Da	Nedokončan
BPRI	BPM	Industrija	Diagnoza	Da	Nedokončan
ebXML BPSS	B2B	Industrija	B2B	Da	Široko sprejet
EDI	B2B	Industrija	B2B	Da	Stabilen
EPC	BPM	Akademsko	Grafični	No	V opuščanju
Petri Net	Vsa	Akademsko	Teorija/ Grafični	/	Široko sprejet
Pi-Calculus	Vsa	Akademsko	Teorija/ Izvajanje	/	Široko sprejet
Rosetta-Net	B2B	Industrija	B2B	Da	Široko Sprejet
UBL	B2B	Industrija	B2B	Da	Stabilen
UML AD	BPM	Industrija	Grafični	Da	Široko Sprejet
WSCI	SOA	Industrija	Izvajanje	Da	Opuščen
WSCL	SOA	Industrija	Izvajanje	Da	Opuščen
WS-CDL	SOA	Industrija	Izvajanje	Da	Široko Sprejet
WSFL	BPM	Industrija	Izvajanje	Ne	Opuščen
XLANG	BPM	Industrija	Izvajanje	Ne	Opuščen
XPDL	BPM	Industrija	Izvajanje/ Izmenjava	Da	Stabilen
YAWL	BPM	Akadamesko	Grafični/ Izmenjava	Ne	Stabilen

Tabela 2.2: BPM standardi, jeziki, notacije, teorije in njihovi statusi

standardov, ampak je zgolj izdala določena priporočila za razvoj interoperabilnih spletnih storitev za številne programske platforme na tem področju.

Kljub temu, da je WS-I uspelo nekoliko racionalizirati fenomen spletnih storitev, so pričela velika podjetja ratificirati lastne standarde. Tako so predvsem IBM, BEA in Microsoft povzročili velik oblak prahu okoli WS-* specifikacij (specifikacij, ki se lotevajo posameznih področij, kot npr. varnosti z WS-Security, transakcij z WS-Transactions, itn.). Podjetja so tekmovala v "standardih" z namenom, da bi prva zagotovila orodja, ki bi jih podpirala. Rešitve, temelječe na "standardih", so postale del prodajne strategije in kasneje de fakto standardi, še posebej, ko sta pri tem skupaj sodelovala IBM in Microsoft.

Vzporedno je skupina ekspertov ustanovila BPMI.org (Business Process Management Initiative), ki je kmalu izdala dve pomembni specifikaciji: BPML jezik za izvajanje poslovnih procesov, ki je bil kasneje opuščen in zamenjan z BPEL (povod in začetno specifikacijo sta podala IBM in Microsoft), in BPMN grafično notacijo, ki omogoča sodelovanje analitikov in IT razvijalcev pri modeliranju in razvoju poslovnih procesov (več o tem v poglavjih 3 in 4).

Poglavje 3

Jezik za modeliranje poslovnih procesov - BPMN

3.1 Predstavitev

BPMN (angl. Business Process Management Notation) je standard na področju modeliranja tokov poslovnih procesov in spletnih storitev. Cilj jezika je ponuditi notacijo, ki je zlahka razumljiva vsem poslovnim uporabnikom. To vključuje tako poslovne analitike, ki izdelajo začetne osnutke procesov, do tehničnih razvijalcev, ki so odgovorni za tehnološko implementacijo, ki bo izvajala te procese in nenazadnje menedžerjem, ki bodo procese upravljali in spremljali. Drugi prav tako pomemben cilj, je zagotoviti, da se XML jezike, ki so zadolženi za izvajanje poslovnih procesov (kot sta npr. BPEL in BPML), lahko vizualno izrazi z enotno notacijo.

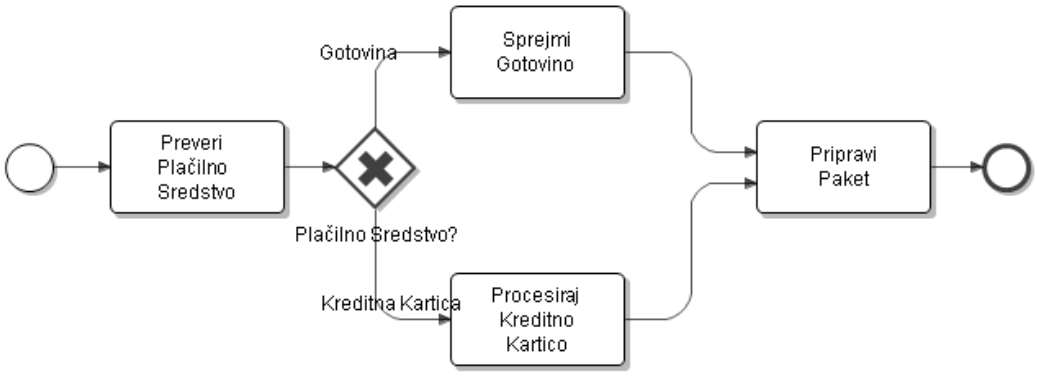
Osnova BPMN je BPD (angl. Business Process Diagram), ki temelji na tehniki diagrama poteka (podobno kot diagrami aktivnosti v UML jeziku), le da je ta prilagojen za izdelavo grafičnih modelov poslovnih procesov. Standard je bil zasnovan na trdnih matematičnih temeljih z uporabo tako imenovanega Pi-Calculusa. Ta se uporablja v računalniški teoretični znanosti kot formalna metodologija za definicijo dinamičnih in mobilnih procesov. Standard je za primerjavo analogen matematičnim temeljem relacijske teorije, ki podpira sisteme za upravljanje relacijskih podatkovnih zbirk (angl. RDBMS). Poslovne procese, ki so modelirani z uporabo BPMN, se lahko torej neposredno upravlja in so tudi pripravljeni na takojšnje izvajanje.

Trenutna verzija BPMN-ja je 1.2, s tem da je že predlagana in v obravnavi verzija 2.0, ki prinaša predvsem izboljšave na področju združljivosti med različnimi orodji in proizvajalci in združitev z BPDM (angl. Business Process

Definition Meta Model) v en konsistenten jezik.

3.2 Elementi BPMN

BPMN je sestavljen iz niza grafičnih elementov. Ti elementi omogočajo enostavno izdelavo preprostih diagramov in so bili izbrani tako, da se medsebojno zlahka razlikujejo in da uporabljajo oblike, ki so že domače načrtovalcem (npr. aktivnosti so predstavljene kot pravokotniki, odločitve pa kot diamanti). Kot že izpostavljeno, je bil eden glavnih vzvodov za razvoj BPMN, izdelati preprost mehanizem za izdelovanje modelov poslovnih procesov, ki bi hkrati lahko podpiral kompleksnost, povezano s poslovnimi procesi. Zato so grafični elementi organizirani v posamezne kategorije, kar zagotavlja bralcu BPD-ja, da zlahka prepozna osnovne elemente in razume diagram. Zaradi potreb po kompleksnosti, so lahko znotraj posameznih osnovnih kategorij elementov dodane še različne variacije in informacije, brez da bi se občutno spremenil osnovni videz in pomen diagrama.



Slika 3.1: Preprost BPMN diagram.

Štiri osnovne kategorije elementov so:

- objekti toka (ang. flow objects),
- povezovalni objekti (ang. connecting object),
- pasovi (ang. swimlanes),
- artefakti (ang. artifacts),

3.2.1 Objekti toka

BPD vsebuje tri osnovne elemente (tabela 3.1):

- **Dogodek** (ang. Event) - prikažemo s krogom in predstavlja nekaj, kar se "zgodi" med izvajanjem poslovnega procesa. Znotraj kroga se lahko nahajajo različni tipi prožilcev ali rezultatov. Rob kroga pove ali je dogodek začetni, vmesni ali končni.
- **Aktivnost** (ang. Activity) - prikažemo s pravokotnikom z zaobljenimi robovi in predstavlja določeno delo, ki ga proces opravlja. Aktivnost je lahko atomska ali ne-atomska (združena). Tipi aktivnosti so lahko: Naloga ali Pod-proces. Pod-proces ločimo po "+" znaku na spodnjem delu pravokotnika.
- **Prehod** (ang. Gateway) - prikažemo z znakom diamanta in ga uporabljamo za nadzor divergence in konvergence toka. Lahko definira klasično odločanje ali pa vejitev, združevanje in pridruževanje. Tip prehoda definirajo različne interne označbe.

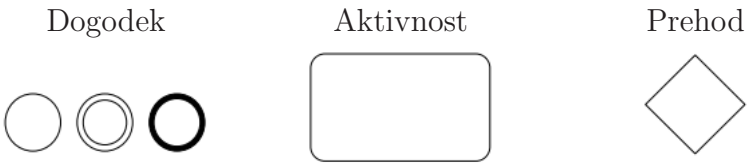


Tabela 3.1: Objekti toka.

Dogodki

Med modeliranjem procesov, modeliramo dogodke in prikazujemo, kakšen vpliv imajo ti na tok procesov. Dogodek lahko bodisi sproži proces, lahko se zgodi med izvajanjem procesa ali pa ga konča. BPMN ponuja za vsakega izmed njih posebno notacijo, kot prikazano v tabeli 3.2.

Pri modeliranju bolj zapletenih procesnih tokov, kot je npr. B2B spletna storitev, je potrebno uporabiti bolj zapletene poslovne dogodke, kot so sporočila, časovniki in poslovna pravila. BPMN omogoča, da definiramo tip prožilca za dogodek in ga označimo s pripadajočo ikono (tabela 3.3). Če uporabimo določen tip prožilca na dogodku, potem to postavi določene omejitve na tok

Začetni dogodek	Vmesni dogodek	Končni dogodek
		

Tabela 3.2: Osnovni tipi dogodkov.

procesa, ki ga modeliramo (npr. časovnik ne more končati toka ali pa lahko tok sporočila rišemo samo iz in v dogodke tipa sporočilo). Take tipe pravil, ki so v resnici kar poslovna pravila, ponavadi avtomatično omejujejo že modelirna orodja, ki podpirajo BPMN.

Pomembnejši tipi prožilcev so:

- *Sporočilo (Event)* - začetno sporočilo pride od udeleženca in sproži začetek procesa ali nadaljuje v primeru vmesnega sporočila. V primeru končnega sporočila, se proces konča in sproži izdelavo sporočila.

- *Časovnik (Timer)* - za sprožitev procesa lahko nastavimo določeno uro (npr. vsak Ponedeljek ob 09:00) ali pa ga nadaljujemo v primeru vmesnega časovnika.

- *Pravilo (Rule)* - pravilo se sproži, ko se izpolni določen pogoj (npr. cena delnice se spremeni za več kot 10% od dneva otvoritve)

- *Povezava (Link)* - povezava je mehanizem za povezavo končnega dogodka enega toka procesa na začetni dogodek drugega.

- *Večkratnik (Multiple)* - več različnih prožilcev je možnih pri začetnem večkratniku za sprožitev ali nadaljevanje procesa. Zahtevan je samo en. Atributi dogodka definirajo kateri tipi prožilcev veljajo. Za končni večkratnik je možnih več različnih posledic (npr. pošlje se več različnih sporočil).

- *Izjema (Exception)* - končna izjema sporoči procesnemu pogonu, da ustvari določeno imenovano napako.

- *Nadomestitev (Compensation)* - končna nadomestitev sporoči procesnemu pogonu, da je potrebno proces vrniti v enega izmed prejšnjih stanj.

- *Zaključek (Cancel)* - zaključek pomeni, da se je uporabnik odločil končati proces.

- *Prekinitev (Kill)* - končna prekinitev pomeni, da je prišlo do kritične napake in da se morajo vse aktivnosti v procesu takoj ustaviti.

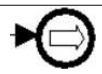
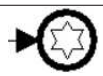
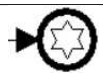
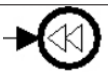
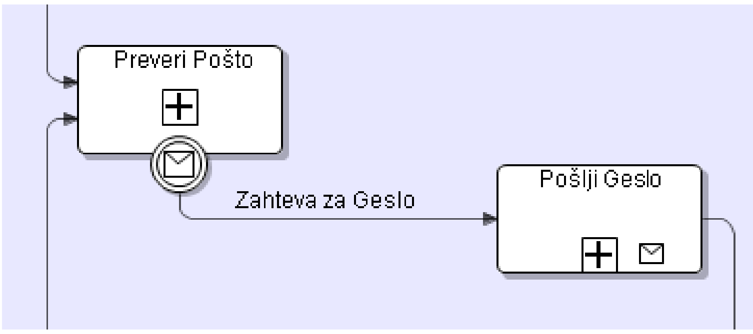
Začetni dogodek	Vmesni dogodek	Končni dogodek
		
začetno sporočilo	vmesno sporočilo	končno sporočilo
		//
začetni časovnik	vmesni časovnik	
		//
začetno pravilo	vmesno pravilo	
		
začetna povezava	vmesna povezava	končna povezava
		
začetni večkratnik	vmesni večkratnik	končni večkratnik
//		
	vmesna izjema	končna izjema
//		
	vmesna nadomestitev	končna nadomestitev
//	//	
		končna zaključitev
//	//	
		končna prekinitev

Tabela 3.3: Oznake tipov prožilcev za dogodke.

Med izvajanjem procesa se velikokrat zgodi, da pride do dogodka, ki povzroči prekinitev izvajanja in sproži nov dogodek in s tem nov proces. Lahko pa se proces zaključi, kar sproži prav tako nov dogodek in s tem nov proces. Take vmesne dogodke modeliramo tako, da postavimo simbol dogodka kar na pripadajoči proces. Na sliki 3.2 lahko vidimo, da se sproži dogodek sporočilo, ko

se zaključi proces *Preveri Pošto*, kar ustvari sporočilo *Zahteva za Geslo*, ki se pošlje procesu *Pošlji Geslo*. Taka notacija poskrbi, da je bralcu hitro jasno, kakšna je povezava med procesoma in kaj se pri tem izvaja.



Slika 3.2: Konec procesa in pričetek procesa.

3.2.2 Povezovalni objekt

Za prikaz vrstnega reda izvajanja toka se uporabljajo povezovalni objekti. Ti objekti so (tabela 3.4):

- Tok sekvence (angl. Sequence flow) - prikažemo z ravno polno črto in puščico. Z njo prikažemo smer v kateri se izvaja tok.
- Tok sporočila (angl. Message flow) - prikažemo s pretrgano črto in prazno puščico. Uporablja se za prikaz potovanja sporočil med dvema udeležencema v procesu (angl. Process Participants). Različna udeleženca sta prikazana z dvema bazenoma (angl. Pools).
- Asociacija (angl. Association) - prikažemo s pretrgano pikčasto puščico in se uporablja za povezovanje podatkov, besedil in ostalih artefaktov z objekti toka. Asociacije prikažejo vhode in izhode aktivnosti.

Tok sekvence	Tok sporočila	Asociacija

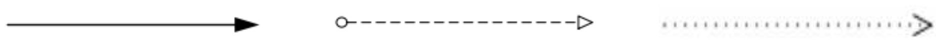


Tabela 3.4: Vrste povezovalnih objektov.

3.2.3 Pasovi

Mnoga orodja za modeliranje procesov uporabljajo koncept pasov (angl. Swimlanes) za organizacijo aktivnosti v ločene vizualne kategorije, z namenom prikazati različne funkcionalne zmogljivosti ali odgovornosti. BPMN podpira dve obliki pasov:

- **Bazen** (angl. Pool) - bazen predstavlja Udeleženca v procesu. Deluje tudi kot grafični kontejner za ločitev aktivnosti od ostalih bazenov, ponavadi v sklopu B2B procesov.

- **Steza** (angl. Lane) - steza nadalje loči bazen in razteza dolžino bazena bodisi vertikalno ali horizontalno. Steze se uporabljajo za organizacijo in kategorizacijo aktivnosti.

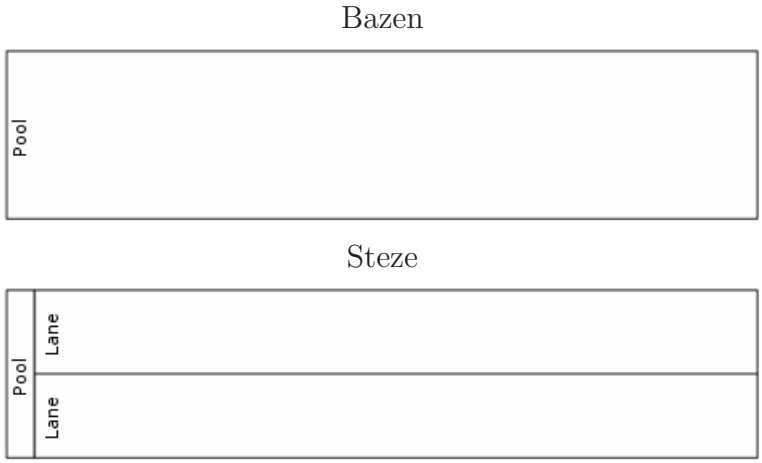
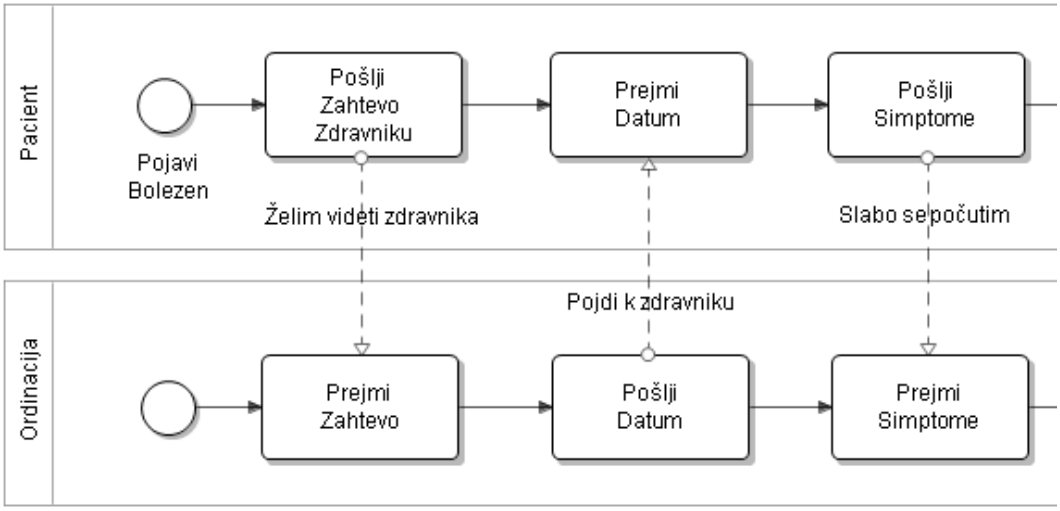


Tabela 3.5: Vrste pasov. Bazen nadalje ločujejo steze.

Ponavadi bazen predstavlja določeno združbo, steza pa posamezen oddelek znotraj te družbe. S tem, ko postavimo aktivnosti znotraj bazenov in stez, določimo *kdo* stori *kaj*, za dogodke določimo *kje* se pojavijo in za prehode *kje* se odločitve zgodijo ali *kdo* se odloči. Aktivnosti znotraj posameznih bazenov se smatrajo kot samozadostni procesi, zato tok sekvence ne more prečkati meje bazena. To lahko stori tok sporočila in s tem tudi prikažemo komunikacijo med dvema udeležencema - bazenoma (slika 3.3).



Slika 3.3: Primer komunikacije med dvema bazenoma.

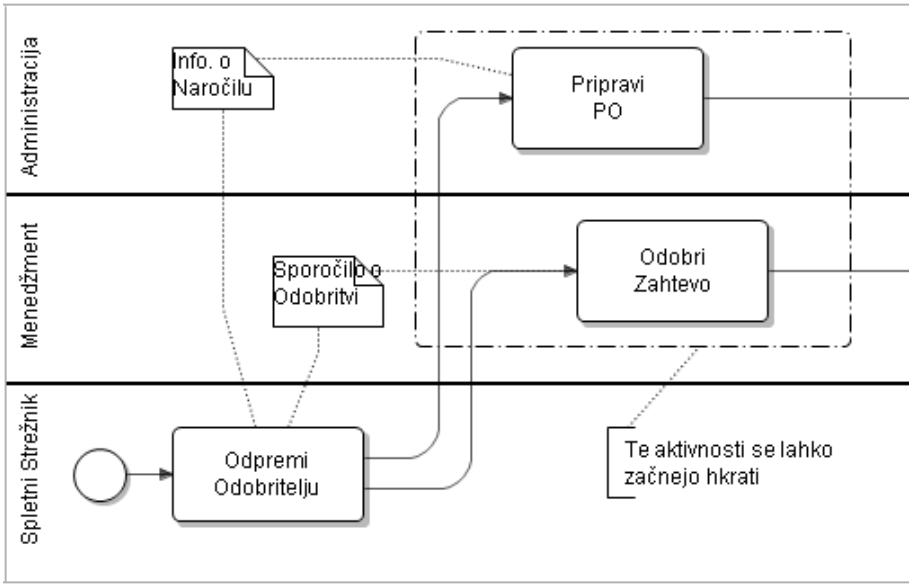
3.2.4 Artefakti

BPMN je zasnovan tako, da dovoli načrtovalcem in modelirnim orodjem določeno mero fleksibilnosti. To omogoči tako, da razširja osnovne notacije in daje možnost dodatnega označevanja, ki je primerna specifični situaciji (npr. vertikalni marketing v zavarovalništvu in bančništvu). Na diagram lahko dodamo neomejeno število artefaktov. BPMN specificira tri tipe artefaktov:

- **Podatkovni objekt** (angl. Data object) - prikažemo kako so podatki zahtevani ali kako se izdelajo v sklopu aktivnosti. Objekti so povezani z aktivnostmi s pomočjo asociacij.

- **Skupina** (angl. Group) - skupino prikažemo kot pravokotnik s prekinjeno črto. Elemente dajemo v skupino z namenom dokumentiranja ali v namen analize, vendar to ne vpliva na sekvenčni tok.

- **Označba** (angl. Annotation) - elemente v diagramu opremimo z označbami in tako podamo dodatne informacije o procesu.



Slika 3.4: Primer uporabe artefaktov v BPMN diagramu.

Načrtovalci lahko ustvarijo svoje tipe artefaktov in s tem podajo podrobnejše informacije o procesih. S tem pogosto prikažejo vhode in izhode aktivnosti v procesu. Ne glede na to se osnovna struktura, ki jo definirajo aktivnosti, prehodi in sekvenčni tok, z dodajanjem artefaktov ne spreminja.

Poglavje 4

Jezik za izvajanje poslovnih procesov - BPEL

4.1 Predstavitev

BPEL (ang. Business Process Execution Language) je jezik, ki omogoča definiranje in izvajanje poslovnih procesov s pomočjo spletnih storitev. Za definiranje procesov se uporablja jezik XML, izvajanje pa opravlja procesni pogon. Poslovni proces, ki za definicijo uporablja BPEL, se razširja preko večih spletnih storitev in v končni fazi tvori popolnoma novo aplikacijo s svojim javnim vmesnikom, ki je na voljo končnim uporabnikom (internim in eksternim). Uporaba BPEL-a nam omogoča [24]:

- paralelno izvajanje aktivnosti,
- koordiniranje asinhronih komunikacij med spletnimi storitvami,
- povezano izmenjavo sporočil med vpletenimi strankami,
- manipulacijo s podatki pri iteracijah med partnerji,
- podporo za poslovne transakcije in aktivnosti, ki se izvajajo dlje časa (ang. long running transactions),
- podporo za konsistentno ravnanje ob napakah.

Proces BPEL je lahko izvajalni ali abstraktni. Izvajalni proces je tisti, ki ga lahko procesni pogon dejansko izvaja, medtem ko je abstraktni lahko le neka definicija protokola ali pa služi kot javno dostopen opis obnašanja

posameznega partnerja. Torej ima abstraktni proces bolj informativno vlogo, ki lahko vsebuje več različnih primerov uporabe in lahko služi kot vzorec za različne procese. Programska koda se od izvajalnega razlikuje v tem, da modelira le aktivnosti, ki sodelujejo v samem delovnem toku in tiste, ki javno komunicirajo z ostalimi partnerji (več o partnerjih v poglavju 4.4). BPEL v datoteki definira abstraktni proces `abstractProcess="yes"`.

4.2 Zgodovina

Potreba po integraciji med poslovnimi sistemi in aplikacijami se je vedno pojavljala in ravno tehnologija spletnih storitev je omogočila, da so aplikacije in heterogeni sistemi postali medsebojno dosegljivi tako znotraj podjetja kot tudi zunanjim partnerjem izven podjetja. Prva faza evolucije spletnih storitev je bila postavitev temeljev za definiranje, objavo in izmenjavo, kar je vključevalo implementacijo internetnih transportnih protokolov (npr. HTTP in SMTP), podatkovnih modelov (temelječih na XML), izmenjavo sporočil (SOAP), definicijo operacij in tipov (WSDL) ter objavo in odkrivanje storitev (UDDI). Nobena od teh ključnih spletnih tehnologij pa ni bila zasnovana tako, da bi nudila mehanizem, ki bi opisoval, kako se lahko posamezne spletne storitve povezujejo tako, da ustvarijo zanesljive poslovne rešitve z ustrezno ravnijo zahtevnosti, ki je potrebna za poslovne procese.

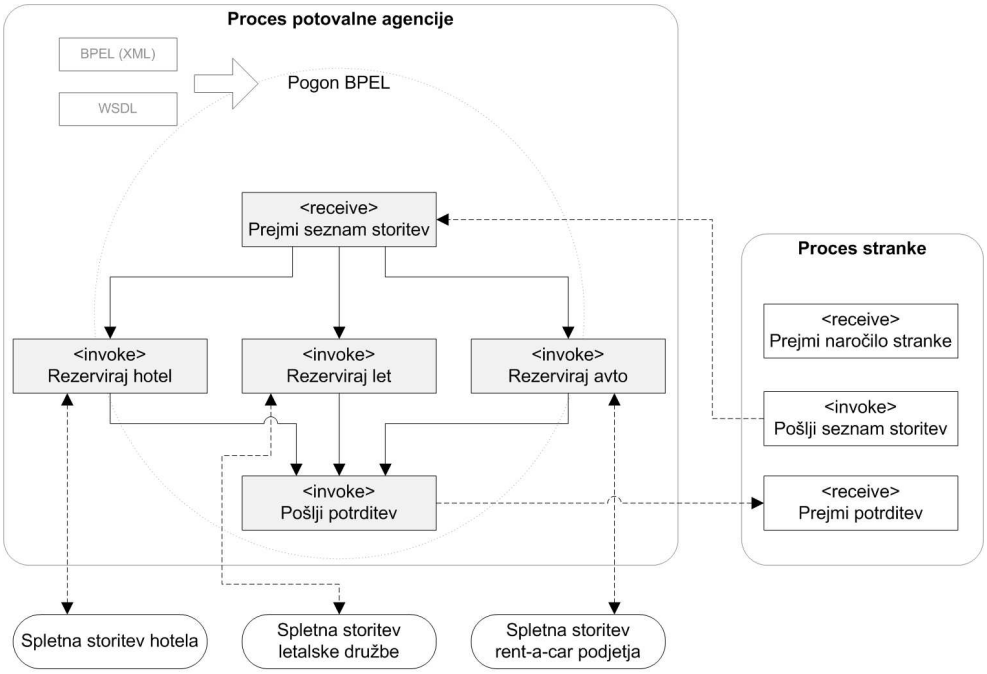
Konec prejšnjega desetletja sta IBM in Microsoft razvijala vsak svoj jezik za upravljanje s spletnimi storitvami (WSFL, XLANG). S povečano popularnostjo in vzponom BPML-a, organizacije BPMI.org in odprtokodnega BPMS gibanja (na čelu z Jbossom in Intalio Inc.), sta se IBM in Microsoft odločila, da združita svoja jezika v nov jezik - BPEL4WS. Aprila 2003 so programski velikani BEA Systems, IBM, Microsoft, SAP in Siebel Systems predložili BPEL4WS 1.1 organizaciji OASIS z namenom standardizacije. Kljub temu, da se je BPEL4WS pojavil že v verziji 1.0 in 1.1, se je septembra 2004 *Web Services BPEL Technical Committee* odločil, da bo svojo specifikacijo preimenoval v WS-BPEL 2.0. Za spremembo imena so se odločili zato, da bi se BPEL ujemal s podobnimi WS- standardi, ki slonijo na uporabi spletnih storitev in da bi poudarili bistvene izboljšave med BPEL4WS 1.1 in prihajajočim WS-BPEL 2.0. Dandanes se pogosto uporablja akronim BPEL za skupek vseh omenjenih tehnologij, razen če razpravljamo specifično o določeni tehnologiji oziroma verziji.

4.3 Osnovna struktura procesa

BPEL definicija procesa sestoji iz dveh tipov datotek:

- datoteke WSDL (Web Services Definition Language), ki definira vmesnike spletnih storitev - tipe povezav med partnerji (partner link types), lastnosti (properties), tipe vrat in operacije (port types and operations) in del, ki se tiče samega procesa - storitve, ki jih proces sam definira in storitve, ki jih proces kliče.

- datotek BPEL, zapisanih v XML jeziku, ki vsebujejo definicijo procesa, vključno z glavnimi aktivnostmi (activities), povezavami partnerjev (partner links), korelacijskimi sklopi (correlation sets), spremenljivkami (variables) in podporniki za kompenzacije, napake in dogodke (compensation, fault, event handlers).



Slika 4.1: BPEL struktura na primeru potovalne agencije

Omenjene datoteke tvorijo poslovni delovni tok, ki deluje kot vmesnik z zunanjimi partnerji in pri tem za komunikacijo uporablja spletne storitve. Tako so glavni koraki v poslovnem procesu, ki definirajo tok, kar točke storitev, ki

jih predstavljajo aktivnosti `receive, pick, invoke` in `reply`. Najbolj togo pravilo programiranja v BPEL-u določa, da se proces lahko začne le, če ga kliče druga spletna storitev. Osnovna struktura datoteke BPEL je sledeča:

```
<process name="MojProces" ... >

  <partnerLinks>
    <!-- Deklaracije povezav med partnerji -->
  </partnerLinks>

  <variables>
    <!-- Deklaracije spremenljivk -->
  </variables>

  <sequence>
    <!-- Deklaracija glavnega procesa -->
  </sequence>

</process>
```

BPEL proces lahko vsebuje samo eno glavno aktivnost, ki lahko nadalje vsebuje poljubno število pod-aktivnosti na različnih hierarhičnih nivojih. Najbolj preprost način za pričetek procesa je uporaba konstrukta `sequence`, ki ima kot prvo aktivnost `receive` in atribut `createInstance="yes"`, kot v naslednjem primeru:

```
<sequence>
  <receive ... createInstance="yes" ... > ... </receive>
  <!-- ostale aktivnosti -->
</sequence>
```

Proces se prične, ko se sproži `receive`, ostale aktivnosti se izvedejo v zaporedju. Proces se konča, ko se zaključi zadnja aktivnost.
Drugačen pristop je uporaba `receive` znotraj konstrukta `flow`. Pri tem `receive` ne sme imeti nobenih vhodnih povezav:

```
<flow>
  <receive ... createInstance="yes" ... > ... </receive>
  <!-- ostale aktivnosti -->
</flow>
```

Tudi ta proces se prične, ko se sproži `receive`, vendar se ostale aktivnosti izvajajo paralelno oziroma v primeru uporabe povezav v zaporedju. Proces se zaključi, ko tok združi vse njegove aktivnosti.

4.4 Komunikacija med partnerji

V BPEL-u procesi komunicirajo med seboj kot poslovni partnerji (business partners). Relacije med partnerji navedemo deklarativno v definiciji procesnih povezav, lahko pa tudi znotraj samega toka procesa.

4.4.1 Tipi povezav med partnerji

V datoteki WSDL `partnerLinkType` povezuje tip vrat spletne storitve z vlogo partnerja. Tip povezave ima ime in eno izmed dveh vlog, kateri imata nadalje vsaka svoje ime in tip vrat. Povezava z dvema vlogama predstavlja relacijo, v kateri si partnerja, kot sta prodajalec in kupec, izmenjata servisne klice:

```
<partnerLinkType name="KupecProdajalec">
  <role name="kupec">
    <portType name="kupecPT"/>
  </role>
  <role name="prodajalec">
    <portType name="prodajalecPT/>
  </role>
</partnerLinkType>
```

Tip povezave med partnerji z eno vlogo je primeren, ko procesu ni potrebno poznati svojega klicatelja:

```
<partnerLinkType name="Streznik">
  <role name="streznik">
    <portType name="streznikPT"/>
  </role>
</partnerLinkType>
```

4.4.2 Povezave med partnerji

Povezave med partnerji (partner links) definirajo, katere vloge, ki so definirane v datoteki WSDL, izvaja proces in katere se izvajajo v okviru partnerjev:

```
<partnerLink name="KupecProdajalecPovezava"
    partnerLinkType="KupecProdajalec"
        myRole="kupec" partnerRole="prodajalec"/>
```

V kolikor proces kliče *streznik* storitev iz zgornjega primera, se za povezavo definira vlogo partnerja na sam strežnik:

```
<partnerLink name="StreznikPovezava" partnerLinkType="Streznik"
    partnerRole="streznik"/>
```

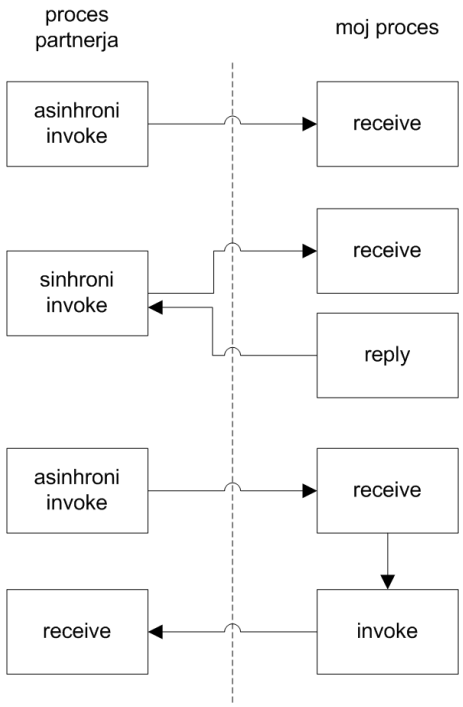
V večini primerov so povezave med partnerji definirane statično, vendar BPEL omogoča tudi dinamične povezave, kjer se informacije o povezavah izmenjajo med izvajanjem.

4.4.3 Interakcije med partnerji

BPEL procesni tok vsebuje aktivnosti `receive, reply, invoke` in `pick`, ki implementirajo dejansko komunikacijo med partnerji. `Receive` in `pick` predstavljajo povezave, ki jih implementira proces in ki so klicane s strani partnerja (npr. kupec). `Reply` in `invoke` pa predstavljajo servise partnerja, ki jih proces kliče (npr. prodajalec, strežnik).

Tabela 4.1 prikazuje možne tipe interakcij s partnerji. Beseda `my` se uporablja za ločevanje procesa od partnerjev.

Iz slike 4.2 je razvidno, da ima vsaka interakcija (iz stališča trenutnega procesa) svojo ustrezno interakcijo na strani partnerja.



Slika 4.2: Interakcije med partnerji.

Vzorec	**Vloga**	**Opis**
Asinh. `receive`	MyRole = receive WS	Partner kliče moj servis, kar sproži logiko v mojem procesu. Takoj po klicu partner nadaljuje s svojim procesom.
Sinh. `receive-reply`	MyRole = receive WS PartnerRole = reply WS	Partner kliče moj servis, kar sproži logiko v mojem procesu. Proces partnerja se ustavi. Sčasoma moj proces vrne nazaj odgovor partnerju in ta nadaljuje svoj proces.
Asinh. `receive-invoke`	MyRole = receive WS PartnerRole = invoke WS	Partner kliče moj servis, kar sproži logiko v mojem procesu. Takoj po klicu partner nadaljuje s svojim procesom. Sčasoma moj proces vrne nazaj odgovor partnerju z ukazom `invoke`.
Sinh. `invoke`	PartnerRole = invoke WS	Moj proces kliče partnerjev servis z ukazom `invoke` in čaka na odgovor.
Asinh. `invoke`	PartnerRole = invoke WS	Moj proces kliče partnerjev servis z ukazom `invoke`. Proces se nadaljuje.
Asinh. `invoke-receive`	MyRole = receive WS PartnerRole = invoke WS	Moj proces kliče partnerjev servis z ukazom `invoke`. Proces se nadaljuje. Sčasoma partner pokliče moj proces in sproži `receive`.

Tabela 4.1: Vzorci interakcij med partnerji.

Invoke

Aktivnost `invoke` lahko kliče spletno storitev partnerja sinhrono kot tudi asinhrono. Spletna storitev partnerja je identificirana s pomočjo tipa povezave partnerja in tipa vrat v datoteki WSDL. Sinhrona spletna storitev zahteva, da so podane vhodne in izhodne spremenljivke. V primeru, da servis sproži napako, lahko `invoke` aktivnost definira eno ali več podpornikov za napake ali podpornik za kompenzacijo.
Primer sinhronega klica spletne storitve:

```
<invoke partnerLink="mojPartner" portType="servis"
   operation="sinhronaZahteva" inputVariable="zahteva"
      outputVariable="odgovor"/>
```

Receive

Medtem ko `invoke` omogoča procesu, da kliče partnerjevo spletno storitev, `receive` definira spletno storitev, ki jo lahko partner kliče. Kot `invoke`, tudi `receive` uporablja tip povezave partnerja in tip vrat v datoteki WSDL za identifikacijo storitve. Argumenti, ki jih pošlje partner, se vežejo na posamezno spremenljivko. Naslednji primer implementira storitev *zacetnaZahteva*, katere vhod se veže na spremenljivko *zahteva*. Atribut `createInstance` pove, da se bo ob klicu te storitve sprožila nova instanca procesa.

```
<receive partnerLink="mojPartner" portType="servis"
   operation="zacetnaZahteva" variable="zahteva"
      createInstance="yes"/>
```

Naslednji primer prikazuje storitev *drugaZahteva*, ki ne sproži nove instance, ampak čaka na dogodek znotraj že obstoječe instance:

```
<receive partnerLink="mojPartner" portType="servis"
   operation="drugaZahteva" variable="zahteva"
      createInstance="no"/>
```

Reply

Aktivnost `reply` sproži klic k partnerju kot odgovor na predhodni sinhroni klic na `receive`. Med klicema lahko proces izvaja poljubno procesno logiko. Kot prikazuje naslednji primer, se `reply` ujema z atributi `partnerLink`, `portType` in `operation`; izhod je definiran v spremenljivki `variable`:

```
<reply parnerLink="odjemalec" portType="c:RacunOdprtPT"
  operation="posljiPodatke"  variable="zahteva"/>
```

```
<!-- izvajaj vmesno logiko -->

<reply parnerLink="odjemalec" portType="c:RacunOdprtPT"
   operation="posljiPodatke" variable="potrditvenaStevilka"/>
```

Pick

Aktivnost `pick` čaka na dogodke in nato izvede aktivnosti vezane na ta dogodek. Kot `receive` in `pick` ima atribut `createInstance`, ki definira novo instanco procesa ob sprožitvi dogodka. Opcijsko lahko definira tudi časovnik `onAlarm`, ki izvede specifično aktivnost, v kolikor se v določenem časovnem okvirju ne izvede noben od definiranih dogodkov. Seznam dogodkov je tipa `onMessage`; dogodek je definiran kot spletna storitev, podobno kot aktivnost `receive`. V naslednjem primeru `pick` sproži nov proces, če se sproži dogodek *ev1* ali *ev2*. Pogoj `onAlarm` se sproži, če se noben od dogodkov ne sproži v 3 dneh in 10 urah[1].

```
<pick createInstance="yes">
  <onMessage partnerLink="p1" portType="pt"
        operation="ev1" variable="v">
    <sequence> ... </sequence>
  </onMesage>
  <onMessage partnerLink="p1" portType="pt"
        operation="ev2" variable="v">
    <sequence> ... </sequence>
  </onMesage>

  <onAlarm for="P3DT10H">
    <sequence> ... </sequence>
  </onAlarm>
</pick>
```

4.5 Elementi BPEL

4.5.1 Spremenljivke

Večina procesov mora med samim izvajanjem operirati z določenimi aplikacijskimi podatki. Podatki se inicializirajo ob pričetku procesa in se med procesom berejo in spreminjajo. BPEL proces lahko definira več spremenljivk, ki jih pošlje spletnim storitvam kot vhodne ali izhodne parametre in jih po

[1] Sintaksa v XPath verziji 2.0 je P3DT10H.

potrebi priredi ostalim spremenljivkam v drugem procesu. Procesna spremenljivka mora imeti unikatno ime znotraj svojega obsega (`scope`) in je lahko sporočilnega WSDL tipa, element XML sheme ali eden izmed primitivnih tipov.

Spremenljivka se lahko inicializira na naslednje načine:

- vezana na vhod začetne aktivnosti, kot npr. `receive`, `pick` in `eventHandler`,
- vezana na izhod sinhronega `invoke`,
- priredi se vrednost z aktivnostjo `assign`.

Aktivnost `assign` je definirana kot kopija podatkov iz izvora v ciljno spremenljivko. Izvor je lahko osnovna primitivna vrednost, izraz, delna ali celotna vrednost spremenljivke drugega procesa ali del spremenljivke trenutnega procesa. Naslednji primeri prikazujejo prirejanje vrednosti za vsak tip:

```
<!-- iz primitivne vrednosti -->
<variable name="x" type ="xsd:int"/>
<assign>
  <copy>
    <from>1</from>
    <to variable="x"/>
  </copy>
</assign>

<!-- iz izraza -->
<variable name="x" type ="xsd:int"/>
<assign>
  <copy>
    <from expression="bpws:getVariableData('x') + 1"/>
    <to variable="x"/>
  </copy>
</assign>

<!-- celotno kopiranje -->
<variable name="x" type ="xsd:int"/>
<variable name="y" type ="xsd:int"/>
<assign>
  <copy>
    <from variable="y"/>
    <to variable="x"/>
  </copy>
</assign>
```

```
<!-- parcialno kopiranje -->
<!-- v datoteki WSDL
<message name="person">
  <part name="name" type="xsd:string"/>
  <part name="address" type="xsd:string"/>
</message>
-->

<variable name="person" messageType="person"/>
<variable name="personAddress" type ="xsd:string"/>
<assign>
  <copy>
    <from>y</from>
    <to variable="x"/>
  </copy>
</assign>
```

S pomočjo vgrajene BPEL funkcije `bpws:getVariableData` lahko pridobimo vrednost spremenljivke. V primeru elementa XSD lahko funkcija uporabi XPath izraz za pridobitev posameznega toka podatkov iz XML dokumenta; v primeru WSDL sporočilnega tipa, lahko funkcija pridobi podatek iz kateregakoli dela sporočila.

4.5.2 Podpora za izjeme in Kompenzacije

Obseg `scope` je del procesne kode, ki definira svoje lastne aktivnosti, spremenljivke, podpornike za napake, kompenzacije in dogodke. Obseg je lokaliziran kontekst za izvajanje: spremenljivke so vidne samo znotraj njegovih meja in podporniki se navezujejo samo na tok, ki ga le-ta definira. Obsegi so hierarhični - znotraj obsega je lahko definiranih poljubno število gnezdenih obsegov, pri tem je tudi sam proces obseg na prvem nivoju.

Kompenzacije

Kompenzacija je vrsta transakcije, ki povrne (izniči) spremembe predhodno končane transakcije. V mnogih spletnih aplikacijah se spremembe na enem ali več sistemih zgodijo znotraj lokalne ali distribuirane transakcije in se ne shranijo dokler transakcija ni zaključena. Za povrnitev v prvotno stanje, aplikacija preprosto vrne transakcijo nazaj (ang. rollback). Vendar pri poslovnih procesih nastane problem, saj pogosto trajajo tako dolgo, da držanje odprte transakcije ni mogoče. V primeru, da mora biti predhodni korak negiran, proces sproži kompenzacijo. Naslednji primer prikazuje podpornik, ki sproži

spletno storitev *povrni*. Ta naj bi izničil spremembe, ki jih naredi storitev *posodobi*, in obvestil partnerja, da preneha z izvajanjem določene aktivnosti.

```
<scope name="s">
  <compensationHandler>
    <invoke operation="povrni" ... />
  </compensationHandler>

  <invoke operation="posodobi" ... />
</scope>
```

Podpornik za napake

Kompenzacija pomeni izničenje končanega obsega, podpornik za napake pa pomeni procesiranje napake, ki se zgodi znotraj obsega, ki je trenutno aktiven. Ko pride do izjeme (bodisi implicitno od pogona BPEL bodisi eksplicitno od `throw` aktivnosti), tok preskoči na tisti podpornik, ki je definiran za tak tip napake. V kolikor tak tip ne obstaja, se napaka hierarhično prenese na višji nivo in tako naprej. V primeru, da se napaka ne obdela, se proces zaključi. Podpornik za napake je definiran kot niz `catch` struktur:

```
<scope name="s1">
  <faultHandlers>
    <catch faultName="x:neveljavenRacun">
      ...
    </catch>
    <catch faultName="x:ukinjenRacun">
      ...
    </catch>
    <catchAll>
      ...
    </catch>
</scope>
```

Vsaka BPEL napaka ima svoje unikatno ime. Nekatere so standardne napake, ki so dokumentirane v BPEL specifikaciji, kot npr. `uninitializedVariable`, ki jo sproži procesni pogon, v kolikor proces želi brati neinicializiran del spremenljivke. Druge so specifične glede na aplikacijo. Strukture so definirane za obe vrsti napak, struktura `catchAll` pa ujame vse napake, ki niso bile ujete v predhodnih strukturah. Vsaka struktura definira aktivnosti, ki naj se izvedejo v primeru napake. Podpornik lahko ujame napako in omogoči, da se proces nadaljuje ali pa ponovno vrže isto ali drugačno napako in propagira napako na višji obseg. Aktivnost `throw` sproži napako in preusmeri tok na podpornik za napake, ki je definiran za trenutni obseg.

```
<switch name="zahteva">
  <case name="preverjanje" ... > ... </case>
  <case name="prihranki" ... >
    <switch name="preceveriRacun">
      <case name="odprt" ... > ... </case>
      <case name="zaprt" ... >
        <throw faultName="ukinjenRacun"/>
      </case>
    </switch>
  </case>
  <otherwise>
    <throw faultName="neveljavenRacun"/>
  </otherwise>
</switch>
```

4.5.3 Usmerjanje toka

BPEL aktivnosti za razdružitev in združitev sta `switch` in `flow`. `Switch` je običajna programska struktura za pogojno izvajanje, medtem ko `flow` omogoča paralelno izvajanje s pomočjo varovanih povezav (ang. guarded links).

Switch

Aktivnost `switch` je ekskluzivna-OR struktura, ki je sestavljena iz ene ali več `case` struktur. Izvede se tista aktivnost, katere prvi pogoj se izvede na `true`. Opcijsko je lahko definiran tudi `otherwise`, ki lahko vsebuje aktivnost, vendar brez pogoja; ta aktivnost se bo izvedla samo, če noben od predhodnih pogojev ni pravilen. Naslednji primer prikazuje aktivnost `switch` z dvema pogojema in blokom `otherwise`, ki se izvede v kolikor spremenljivka i ne vsebuje vrednosti 1 ali 2:

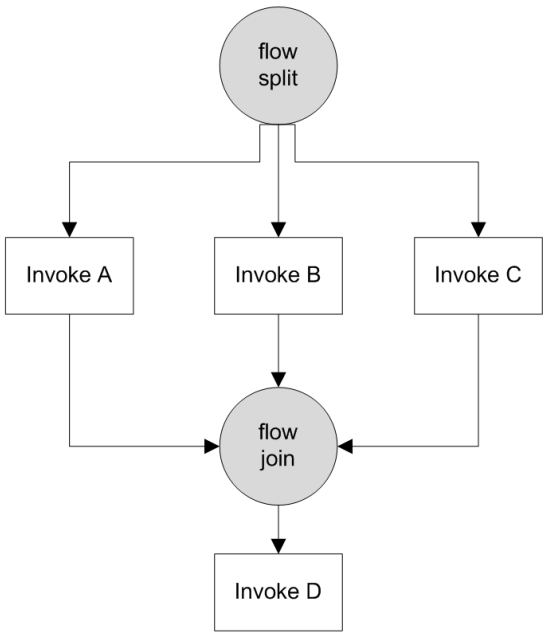
```
<switch>
  <case condition="bpws:getVariableData('i')=1">
    <flow> ... </flow>
  </case>
  <case condition="bpws:getVariableData('i')=2">
    <sequence>
      <assign ... />
      <switch> ... </switch>
    </sequence>
  </case>
```

```
    <otherwise>
      <invoke ... />
    </otherwise>
  </switch>
```

Flow

Aktivnost `flow` (najbrž najbolj zanimiva in samosvoja struktura v BPEL jeziku) modelira paralelno izvajanje in sinhronizacijo. Obnašanje strukture je odvisno od tega, kako je nastavljena oziroma od primera uporabe.

Paralelna razdružitev in združitev. Slika 4.3 prikazuje paralelno klicanje spletnih storitev partnerjevih povezav A, B in C (razdružitev - ang. split). Vrstni red klicanja je nepredvidljiv. Aktivnost `flow` čaka, da se vse aktivnosti zaključijo (združitev - ang. join). Klicanje povezave D se izvede šele, ko se vsi trije predhodni klici zaključijo.



Slika 4.3: Paralelna razdružitev in združitev v BPEL.

Programska koda, ki tak scenarij implementira:

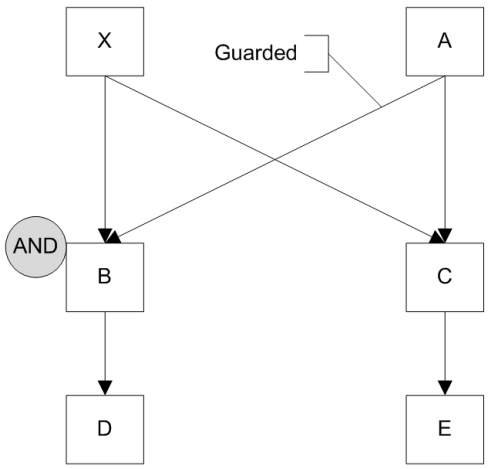
```
<flow>
  <invoke partnerLink="A" ... />
```

```
    <invoke partnerLink="B" ... />
    <invoke partnerLink="C" ... />
  </flow>
  <invoke partnerLink="D" ... />
```

Povezave in sinhronizacijske odvisnosti. Mehanizem `flow` ponuja več funkcij, ki omogočajo modeliranje situacije, ko aktivnost ne more začeti dokler se ena ali več drugih aktivnosti ne zaključi. `Flow` lahko definira množico povezav, kjer vsaka povezava prične v izvoru (`source`) in konča v ciljni (`target`) aktivnosti. Tukaj srečamo dve posebnosti:

- Kot cilj več povezav, je lahko izvor tudi sama aktivnost. Izvorna aktivnost lahko na svojih izhodnih povezavah definira pogoj na prehodu; v kolikor ga ne, se privzeto nastavi na *true*.
- Ciljna aktivnost lahko določa združitveni pogoj, ki se prevede glede na pogoje na vsaki izmed vhodnih povezav; v kolikor ni določen, privzeto upošteva pogojni OR, ki se prevede v *true*, če je vsaj ena povezava *true*. Ciljna aktivnost se izvede samo v primeru, da se združitveni pogoj prevede v *true*.

Slika 4.4 prikazuje primer, kjer A vsebuje povezavo do B in C, X do B in C, B do D in C vsebuje povezavo do E.



Slika 4.4: Sinhronizacija v BPEL.

Potek prikazan na sliki 4.4 je sledeč: X in A se začneta izvajati paralelno. Aktivnosti B in C morata čakati, da se X in A zaključita, saj pravilo `flow`

definira, da se morajo vse vhodne povezave končati. C ne definira pogoja za združitev, zato se izvede privzeti OR, t.j. C se izvede, če imata X ali A (ali oba) *true* vrednost na svojih povezavah. Kljub temu, da C potrebuje samo eno *true* povezavo, mora čakati, da se obe predhodni aktivnosti zaključita. Aktivnost B določa eksplicitni združitveni pogoj, kar pomeni, da morata obe povezavi (X in A) biti *true*. Povezava iz A v B vsebuje ekspliciten pogoj na prehodu (povezava je varovana - ang. guarded), medtem ko je povezava iz X v B implicitno *true*. Na koncu aktivnost D čaka, da se izvede B; prav tako E čaka na C.
Programska koda, ki tak scenarij implementira:

```
<flow>
  <links>
    <link name ="AB"/>
    <link name ="AC"/>
    <link name ="XB"/>
    <link name ="XC"/>
    <link name ="BD"/>
    <link name ="CE"/>
  </links>

  <invoke partnerLink="A" ... >
    <source linkName="AB" transitionCondition=" ... "/>
    <source linkName="AC"/>
  </invoke>

  <invoke partnerLink="X" ... >
    <source linkName="XB"/>
    <source linkName="XC"/>
  </invoke>

  <invoke partnerLink="B" ...
      joinCondition="bpws:getLinkStatus('XB') and
          bpws:getLinkStatus('AB')">
    <source linkName="BD"/>
    <source linkName="AB"/>
    <source linkName="XB"/>
  </invoke>

  <invoke partnerLink="C" ... >
    <source linkName="CE"/>
    <source linkName="AC"/>
    <source linkName="X"/>
  </invoke>
```

```
    <invoke partnerLink="D" ... >
      <source linkName="BD"/>
    </invoke>

    <invoke partnerLink="E" ... >
      <source linkName="CE"/>
    </invoke>
  </flow>
```

4.6 Transakcije

K temu, da je težko zagotavljati transakcije v poslovnih procesih, pripomore več faktorjev. Prvič, proces, ki se izvaja dlje časa (ang. long-running process), se ne more izvesti v eni ACID[2] transakciji, zato je potrebno njegovo funkcijo razbiti na več manjših transakcij. Drugič, specifikacija standardov za distribuirane transakcije preko več partnerskih spletnih storitev je še v izvedbi. Posledično, če partner sproži BPEL proces, le-ta ne more sodelovati s partnerjem v primeru napake in obnoviti prvotno stanje (vsaj ne z uporabo standardnega protokola). BPEL strategija za spopadanje s prvim problemom so kompenzacije (opisane predhodno v podpoglavju 4.5.2). Kompenzacije ponujajo strukturiran način za izničenje sprememb v sklopu manjših aktivnosti, ki so se že zaključile. Operacije so izključno lokalne. Drugi problem ostaja še odprt. Specifikacija BPEL uporablja pojem LRT (ang. Long Running Transaction), ko se navezuje na uporabo hierarhičnega gnezdenja podpornikov za kompenzacije. LRT ni specifična funkcija jezika, ampak je bolj skupek postopkov, ki opisujejo, kako uporabiti kompenzacije za razveljavitev sprememb, ki so se zgodile v predvidoma daljšem časovnem obdobju. Ker je tak način zagotavljanja "transakcij" programersko zelo zahteven (in specifičen glede na aplikacijo), različna podjetja nudijo svoje pristope k temu problemu, ki pa niso standardizirani (npr. Oracle BPEL Process Manager [26]).

4.7 Objava, izvajanje in testiranje

Za izvajanje samega BPEL procesa potrebujemo deskriptor procesa. To je datoteka, ki je specifična vsakemu BPEL strežniku in jo BPEL standard ne pokriva. Datoteka je tudi edini del implementacija, ki mora biti prilagojen, v kolikor želimo proces izvajati na drugem BPEL strežniku. Deskriptor največkrat vsebuje osnovne informacije o procesu: ime in lokacijo BPEL da-

[2]ang. kratica za Atomicity, Consistency, Isolation, Durability

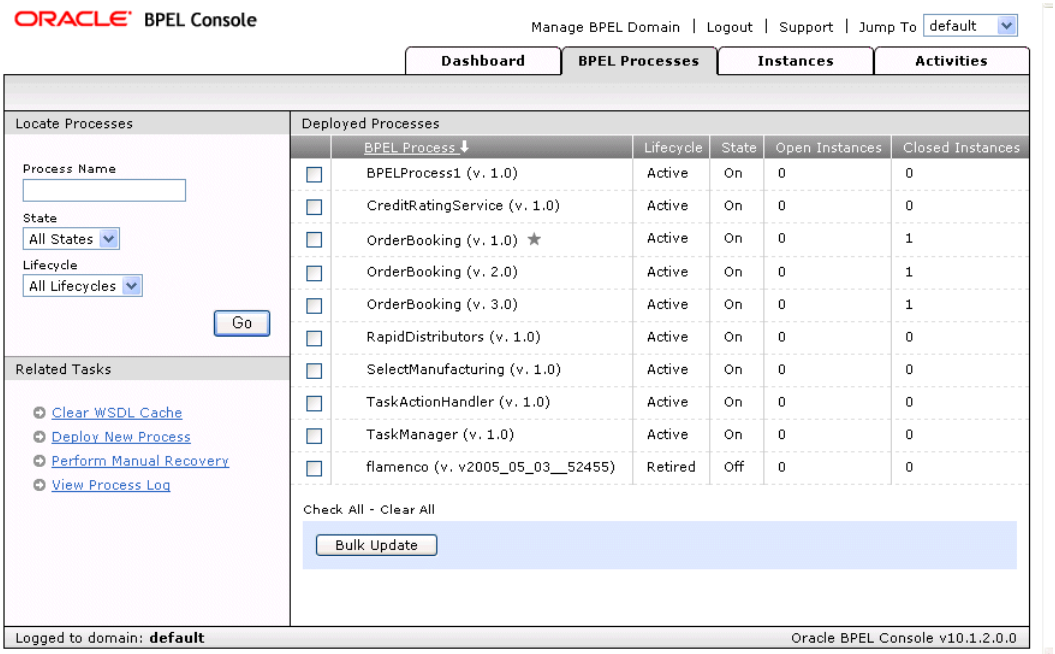
toteke, ime procesa (ID), WSDL lokacijo vseh partnerjevih spletnih storitev in dodatno konfiguracijo. Primer deskriptorja za Oracle BPEL strežnik:

```
<BPELSuitcase>
  <BPELProcess src="Travel.bpel" id="TravelProcessCh4">
    <partnerLinkBindings>

      <partnerLinkBinding name="client">
        <property name="wsdlLocation">
          Travel.wsdl
        </property>
      </partnerLinkBinding>

      <partnerLinkBinding name="employeeTravelStatus">
        <property name="wsdlLocation">
  http://localhost:97/orabpel/Employee?wsdl
        </property>
      </partnerLinkBinding>

      <partnerLinkBinding name="AmericanAirlines">
        <property name="wsdlLocation">
  http://localhost:97/orabpel/AmericanAirline/AmericanAirline?wsdl
        </property>
      </partnerLinkBinding>

      <partnerLinkBinding name="DeltaAirlines">
        <property name="wsdlLocation">
  http://localhost:97/orabpel/DeltaAirline/DeltaAirline?wsdl
        </property>
      </partnerLinkBinding>

    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>
```

Ko imamo vse potrebne datoteke, lahko proces tudi objavimo (največkrat to pomeni kar kopiranje datotek v ustrezno mapo na strežniku) in izvajamo. Postopek se razlikuje glede na uporabljen BPEL strežnik. Sodobna orodja, kot so Oracle BPEL Console ali jBPM Web Console, omogočajo, da izvajamo, testiramo, nadzorujemo in spreminjamo procese kar prek spletnega vmesnika.

4.8 Transformacija BPMN v BPEL

Pretvorba iz BPMN v BPEL poteka prek transformacije tipov objektov, uporabe atributov in lastnosti BPD-ja in njegovih objektov v funkcijsko sorodne BPEL

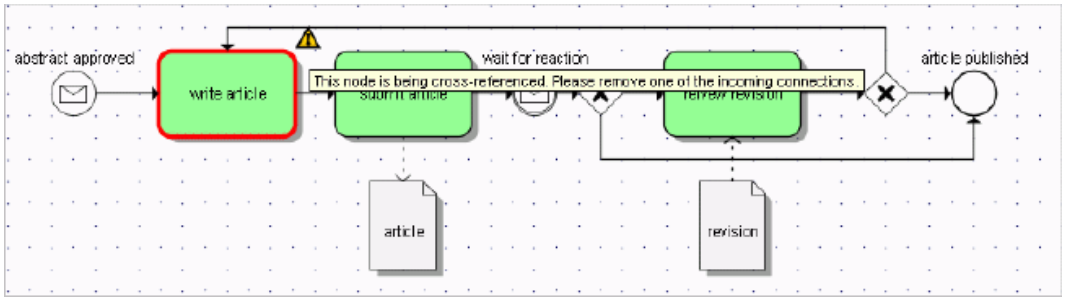


Slika 4.5: Upravljanje procesov v Oracle BPEL konzoli.

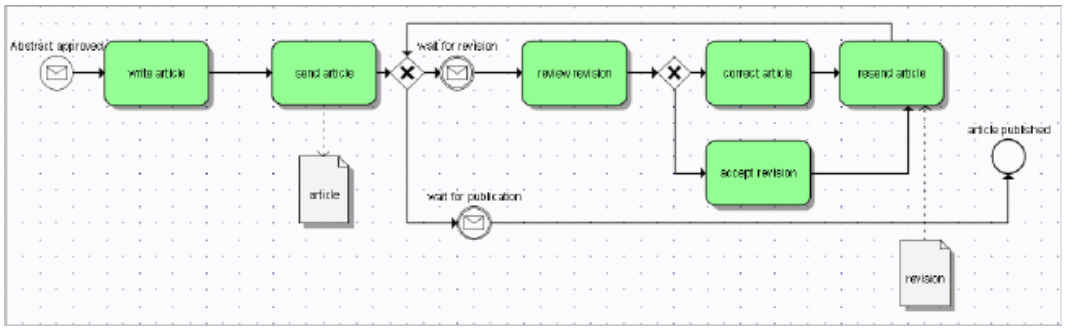
objekte. Vsak objekt v BPD-ju, kot je proces, dogodek ali prehod, ima atribute, ki jih lahko uporabnik določi in ki se lahko uporabijo pri transformaciji v BPEL. Tako bo začetni dogodek tipa *message* postal `receive` struktura z atributom `createInstance="true"` in končni dogodek tipa *message* se bo pretvoril v `reply` ali `invoke`.

Kljub temu, da BPMN specifikacija ponuja določene atribute in funkcije, ki omogočajo poslovnemu analitiku, da doda v proces potrebne podatke za implementacijo, je trenutna podpora še daleč od idealne [10]. K temu pripomore zaenkrat še skopa uporaba BPMN-ja med poslovnimi uporabniki, bolj ključno pa je pomanjkanje podpore BPEL-a pri upravljanju človeškega dela in podprocesov. To je prisililo proizvajalce, da so v svojih orodjih razvili določene razširitve in dodatke, ki ležijo nad BPMN in BPEL specifikacijami in podpirajo te pomembne funkcije. Največkrat se uporabnik tega niti ne zaveda. Problem nastane, ko BPMN proces razvijemo v določenem orodju (oziroma ga takega že dobimo) in ga želimo drugje pretvoriti v BPEL in uporabiti.

Nekaterim zapletom in kompleksnostim se lahko izognemo, če v BPMN-ju modeliramo bolj strukturirane procese in se izogibamo arbitrarnim zankam, ki se težko pretvorijo v BPEL. Kljub temu to v praksi pomeni, da bi morali poslovni analitiki razmišljati kot programerji, zato da bi izdelovali poslovne



Slika 4.6: BPMN proces, ki je popolnoma veljaven, vendar ga orodje ne zna transformirati v BPEL.



Slika 4.7: Popravljen proces se zdaj lahko izvozi v BPEL.

procese, ki bi bili tehnično dovolj konsistentni za izvoz v BPEL. To pa ni v skladu z načeli BPM-ja in postavlja dodatne ovire pri vpeljavi in osvajanju tehnologij BPM.

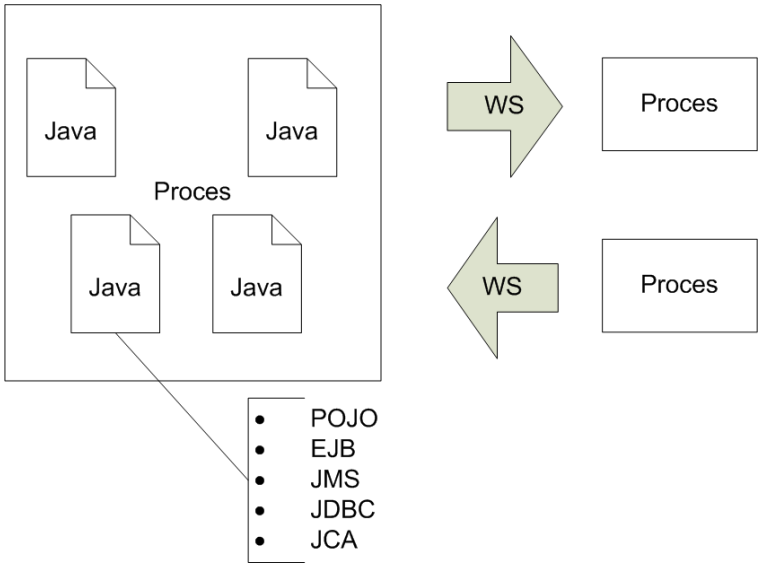
4.9 Dodatne razširitve BPEL-a

4.9.1 BPELJ

"Čisti" BPEL, ki smo ga pravkar obravnavali, zagovarja "programiranje na veliko" (ang. programming in the large) oziroma definiranje glavnih aktivnosti v čisti XML formi. Vendar ima lahko proces, ki naj bi se dejansko izvajal, nešteto manjših opravil, ki jih lahko veliko bolje rešimo s programsko kodo, kot pa z aktivnostmi v procesu. Funkcije, kot npr. procesiranje vhoda, grajenje izhoda, komuniciranje s "hišnimi" sistemi, odločanje in nenazadnje preprosto izračunavanje datumov, lahko vodijo proces, vendar so pogosto prezapletene, da bi bile del toka procesa. "Programiranje na malo" (ang. programming in

the small) je lahko ključno za razvijanje realnih procesov, vendar ga težko uresničimo z uporabo čistega BPEL-a. Največ kar lahko BPEL ponudi, je spletna storitev in pogosto se omenjene probleme rešuje tako, da se razvije posebno spletno storitev, ki kompleksno logiko implementira in ki jo kasneje kličemo iz samega procesa. Tak pristop deluje, vendar ni najbolj primeren. Najbolj problematična je storitvena (performančna) učinkovitost dodatnih spletnih storitev, saj proces potrebuje hiter lokalni dostop do logičnih komponent. Nenazadnje je problem tudi dodatni razvoj in kompleksnost procesa, ki s tem nastane (delno lahko to rešuje dobra obstoječa SOA arhitektura).

Zaradi omenjenih razlogov, sta leta 2004 IBM in BEA (zdaj pod okriljem Oracla) predstavila javnosti Java BPEL razširitev, imenovano BPELJ (ang. BPEL for Java). BPELJ proces, kot prikazan na sliki 4.8, lahko vsebuje vgrajene koščke Java kode, kot tudi klice Java POJO[3] objektov, EJB-jev[4] in ostalih Java komponent.



Slika 4.8: BPEL Java razširitev - BPELJ.

BPELJ primer:

```
<process name="Zavarovanje" ... >

  <partnerLinks>
    ...
```

[3]ang. kratica za Plain Old Java Object
[4]ang. kratica za Enterprise Java Beans

```
    <partnerLink name="ObdelavaZahtevka"
      partnerLinkType="bpelj:zavarovanjeProcesorEJB">
  </partnerLink>

  <variables>
    <variable name="input" messageType="tns:ZahtevekSporocilo"/>
    <variable name="zahtevekOK"
        messageType="bpelj:java.lang.Boolean"/>
  </variables>

  <sequence name="glavna">
    <receive name="prejmiZahtevek" createInstance="true" .../>

    <invoke name="obdelajZahtevek" partnerLink="ObdelavaZahtevka"
          operation="obdelaj">
      <input part="input" variable="input"/>
      <output variable="zahtevekOK"/>
    </invoke>

    ...
  </sequence>
</process
```

4.9.2 BPEL4People

Človeške interakcije v poslovnih procesih segajo od preprostih, kot je npr. ročna odobritev, do kompleksnih, kjer uporabniki vnašajo podatke v sistem. Primer preproste odobritve je lahko zahtevek za novo uporabniško ime. Nadzornik ali sistemski administrator ali oba, morata odobriti zahtevek, preden se lahko uporabniško ime ustvari in dodeli. Primer zapletenega procesa je lahko zahteva, da uporabniki vnašajo podatke ali spreminjajo stanja procesov t.j. sprožitev, zaustavitev ali reševanje napak, ki preprečujejo, da bi se proces nadaljeval. Očitno je, da lahko človeške interakcije hitro povečajo kompleksnost poslovnih procesov.

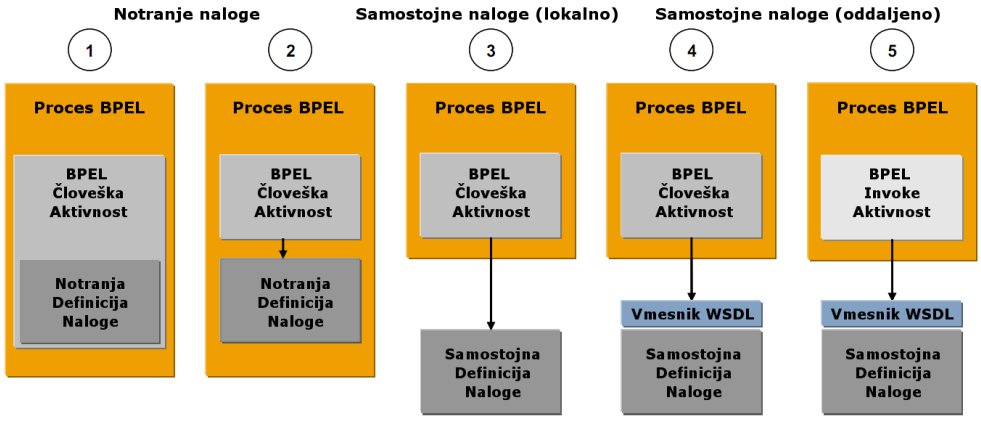
Večina BPM sistemov veže naloge na vloge in vloge na uporabnike. Izvajanje nalog se običajno prične s pomočjo servisa za upravljanje nalog, ki nudi spletni vmesnik, prek katerega lahko uproabniki spremljajo in izvajajo naloge, ki so jim dodeljene. Trenutno ni standarda, ki bi predpisoval, kako naj se izvajajo človeške interakcije med različnimi BPM sistemi.

Jezik BPEL določa obnašanje poslovnih procesov dokler so aktivnosti spletne storitve. Kljub širokemu sprejetju spletnih storitev v distribuiranih poslovnih aplikacijah, predstavlja pomanjkanje podpore za človeške interakcije veliko vrzel za mnogo realnih poslovnih procesov. Tako sta SAP in IBM leta 2005

predlagala standard imenovan BPEL4People, ki naj bi to vrzel zapolnil. Dodatna razširitev BPEL-a podpira orkestracijo človeških vlog in aktivnosti. Trenutno je BPEL4People pri organizaciji OASIS in čaka na odobritev standarda.

BPEL4People prinaša dve pomembni razširitvi: človeško aktivnost (ang. people activity) in človeško povezavo (ang. people link). Človeška aktivnost je osnovna aktivnost, ki jo izvaja človek in ni implementirana s pomočjo programske opreme. Človeška povezava povezuje nalogo s človeško vlogo, ki se kasneje med izvajanjem prevede v posamezne uporabnike in skupine. Izvajalca (ang. actor) človeške aktivnosti definira človeška povezava. Implementacije človeških aktivnosti so definirane kot naloge - nevidne enote dela, ki jih izvaja človek. Opisi nalog so lahko izvedeni tako, da se npr. enovrstični povzetek pojavi v seznamu nalog, podrobnejši opis pa se prikaže, če uporabnik izbere nalogo. Naloge imajo lahko tudi prioriteto, rok izvedbe in atribute za prikaz uporabniškega vmesnika.

Procesi in naloge se lahko prepletajo na različne načine. Naloge lahko procesi uporabljajo bodisi kot *notranje naloge* bodisi kot *samostojne naloge*. Slika 4.9 prikazuje različne modele interakcij med nalogami in procesi.



Slika 4.9: BPEL4People modeli interakcij med nalogami in procesi.

Modela 1 in 2 prikazujeta primere, pri katerih človeške aktivnosti uporabljajo naloge kot notranje naloge. Notranja naloga je lahko del človeške aktivnosti. V tem primeru je uporaba naloge omejena na človeško aktivnost, ki nalogo zaobjema. V drugem primeru je lahko naloga definirana kot osnovni konstrukt BPEL procesa. Tako nalogo lahko uporablja več različnih človeških aktivnosti, kar je pomembno z vidika ponovne uporabe. Notranje naloge imajo dostop do konteksta procesa, v katerem se izvajajo (npr. spremenljivk). BPEL

procesi, ki uporabljajo naloge na tak način, so prenosljivi med različnimi izvajalnimi pogoni, ki implementirajo BPEL4People. Primer 3 prikazuje samostojno nalogo znotraj istega okolja, ki ne definira spletnega vmesnika za spletno storitev. Taka uporaba je podobna modelu 2 z izjemo, da je definicija naloge neodvisna od procesa. S tem naloga nima neposrednega dostopa do konteksta procesa. Prav tako naloga nima vmesnika WSDL, torej je klic naloge specifičen glede na implementacijo. Model 4 je tudi primer notranje naloge, vendar se tokrat uporablja vmesnik WSDL. Pri taki uporabi se spremembe v stanju procesa prenašajo med procesi in nalogami. BPEL procesi, ki uporabljajo naloge na tak način, so prenosljivi med različnimi izvajalnimi pogoni, ki implementirajo BPEL4People. Taka samostojna naloga se lahko uporablja tudi neodvisno od poslovnega procesa, vendar morajo ostali iniciatorji nalog podpirati koordinacijski protokol. Model 5 je najbolj generičen primer. BPEL proces uporablja BPEL `invoke` aktivnost za klic spletne storitve, ki implementira samostojno nalogo. Z vidika procesa BPEL je klic naloge popolnoma transparenten. Povedano drugače - proces BPEL se ne zaveda, da je spletna storitev implementirana kot človeška naloga. V tem primeru lahko uporabljajo nalogo tudi pogoni BPEL, ki ne podpirajo BPEL4People. Oddaljena samostojna naloga se lahko uporabi neodvisno od poslovnih procesov in jo lahko sproži katerikoli klient za spletne storitve.

Med izvajanjem aktivnost pošlje podatke na vhod naloge in prejme podatke na izhod naloge. Naslednji primer prikazuje BPEL4People nalogo za izbiro delavca meseca:

```
<extensionActivity>
  <b4p:peopleActivity name="izvoliDelavcaMeseca"
    inputVariable="kandidati" outputVariable="glas">
    <htd:task name="votingTask">
      <htd:interface operation="oddajGlas"
                portType="el:oddajanjeGlasovaPT"/>
      <htd:peopleAssignments>
        <htd:potentialOwners>
          <htd:from>\$volilci/uporabniki/uporabnik[i]</htd:from>
        </htd:potentialOwners>
      </htd:peopleAssignments>
      <htd:presentationElements/>
    </htd:task>
  ...
  </b4p:peopleActivity>
</extensionActivity>
```

BPEL4People aktivnost `peopleActivity` vsebuje ime in definira vhode in izhode naloge. Kako je naloga implementirana ni domena BPEL-a - v tem

primeru je lahko to spletna stran ali pa se pošlje elektronsko sporočilo vsem uporabnikom, ki lahko volijo.

Poglavje 5

Praktični del - jBPM

5.1 Uvod

Namen praktičnega dela je prikazati implementacijo, ki vsebuje osnovne koncepte BPM. Med različnimi BPMS smo se odločili za jBoss jBPM. Kljub temu, da je na voljo veliko bolj uveljavljenih proizvajalcev, smo izbrali jBPM, ker lahko enostavno BPM rešitev zgradimo hitro in iz samega prosto dostopnega programja. jBPM ne cilja na "čiste" BPM rešitve, kjer bi nam zapletena orodja omogočala direktno transformacijo iz modeliranega poslovnega procesa v izvajalnega in poganjanje z minimalno dodano programsko kodo. Nasprotno, jBPM nudi procesni pogon in ogrodje, ki se zlahka zlije z današnjimi sodobnimi programskimi Java ogrodji (npr. JBoss Seam, Spring, Struts ...) in omogoča programerju, da v svoj novi ali obstoječi projekt vključi podporo za poslovne procese.

Primer, ki smo ga implementirali, prikazuje uporabo jBPM kot podporo za delovne tokove. V okviru implementacije bomo prikazali tudi modeliranje v orodju jPDL Process Desginer, integracijo s programskim ogrodjem Seam in spremljanje procesa s pomočjo jPBM konzole.

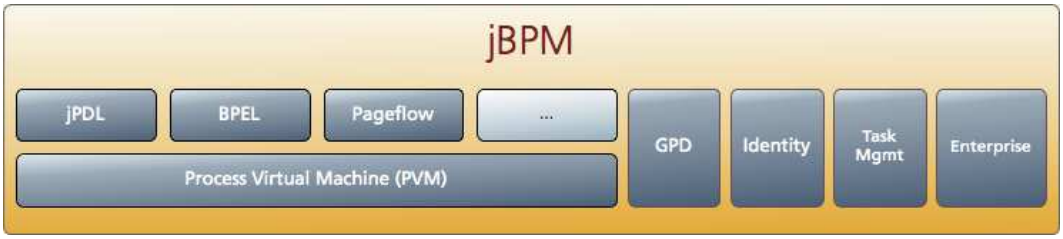
5.2 Zgradba in funkcionalnosti jBPM

jBPM je platforma in ogrodje, ki nudi podporo izvajalnim procesnim jezikom in orkestracijo poslovnih procesov. Osnovan je na uveljavljenih J2EE tehnologijah in projektih, pri čemer prednjačijo tisti, ki spadajo pod okrilje JBossa[1].

[1]JBoss je eno izmed vodilnih podjetij na področju razvoja odprto-kodne vmesne programske opreme (ang. middleware). Leta 2006 ga je kupilo podjetje Red Hat.

Njegov osrednji del je pogon PVM (ang. Process Virtual Machine). Ta je zgrajen tako, da omogoča modularno dodajanje podpore za procesne jezike. Trenutno PVM podpira procesne jezike jPDL, BPEL in Pageflow. Vsak izmed omenjenih jezikov cilja na določene funkcionalnosti in okolja:

- jPDL (ang. Process Definition Language) - jezik, ki je bil prvotno razvit prav za jBPM in ki omogoča popolno interakcijo z jBPM aplikacijskim programskim vmesnikom. Naslanja se neposredno na Java tehnologijo in omogoča napredno upravljanje z nalogami. Najbolj primeren je za modeliranje in izvajanje delovnih tokov, ki slonijo na človeški interakciji.
- BPEL - kot že omenjeno, sloni standard BPEL na spletnih storitvah. Tako je tudi najbolj primeren za izvajanje avtomatiziranih poslovnih procesov, ki ne potrebujejo veliko človeške interakcije. Določeno prednost predstavlja tudi možnost prenosljivosti med različnimi procesnimi pogoni.
- Pageflow - jezik, ki je bil razvit v sklopu programskega ogrodja Seam. Namenjen je za modeliranje in orkestracijo navigacije v spletnih aplikacijah.



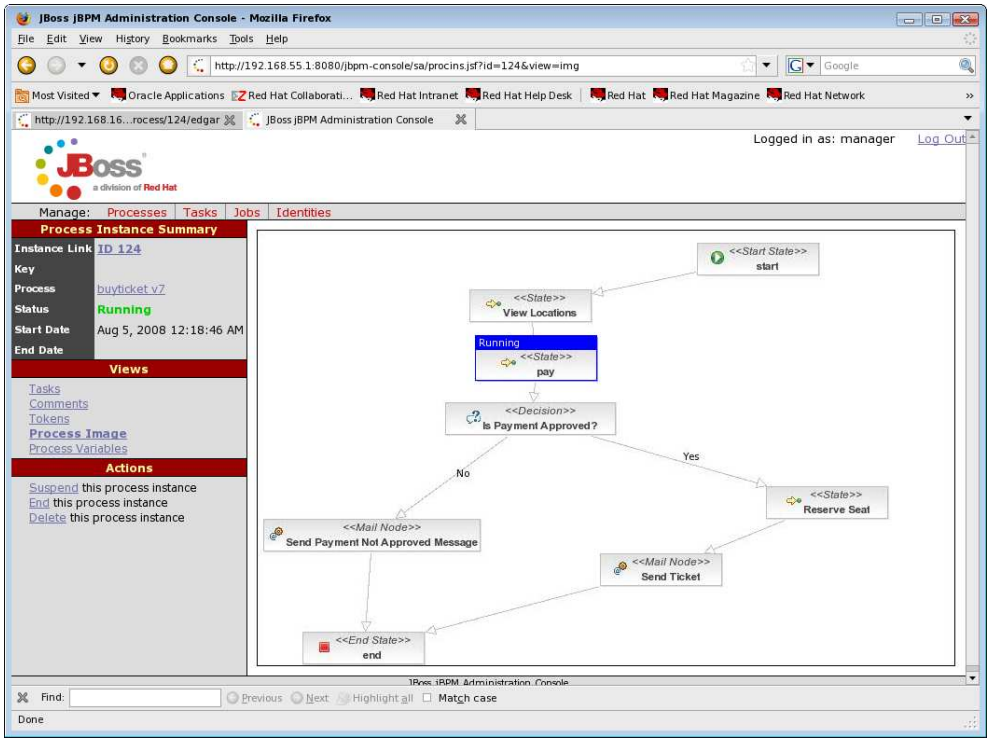
Slika 5.1: Zgradba jBPM in uporabljene tehnologije.

Kot večina procesnih pogonov tudi jBPM hrani definicije poslovnih procesov in njihova stanja v podatkovni bazi. To je tudi obvezen predpogoj, saj se poslovni procesi izvajajo dlje časa in mora njihovo izvajanje preživeti tudi ponovne zagone procesnega pogona. Poleg samega hranjenja jBPM omogoča tudi številčenje procesov. V primeru, da se pojavi potreba po popravku procesa ali nove funkcionalnosti, jBPM zažene proces pod novo verzijo in omogoča, da se starejše instance procesov izvedejo do konca.

Poleg samega izvajanja jBPM nudi tudi konzolo prek spletnega vmesnika ("jBPM Console"), v kateri imamo pregled nad objavljenimi procesi in njihovimi stanji. V sklopu konzole lahko definiramo več uporabnikov in

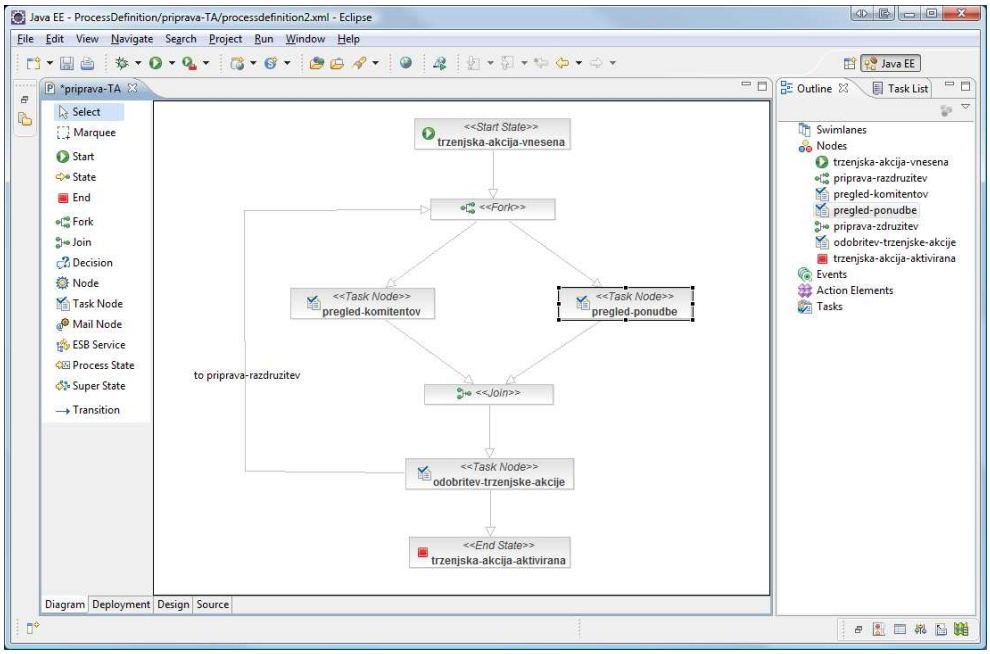
uporabniških skupin, od katerih ima vsaka svojo vlogo. Tako bo npr. skrbnik procesov imel pravice nad izvajanjem procesov (objava, ustavitev, brisanje ...), poslovni uporabnik pa pregled nad obnašanjem in uspešnostjo procesov.

Poleti je izšel popolnoma prenovljeni jBPM 4.0, ki prinaša čisto novi, izboljšani in optimiziran procesni pogon in množico novih funkcionalnosti. Poleg prenovljene in bogatejše konzole je zdaj dodana tudi možnost modeliranja v jeziku BPMN. Za naš primer smo uporabili verzijo 3.4, ki je trenutno veliko bolj stabilna in jo Seam tudi uradno podpira.



Slika 5.2: Spremljanje procesa v jBPM konzoli.

Znotraj paketa jBPM dobimo tudi vtičnik za popularno ogrodje Eclipse. Vtičnik spremeni Eclipse v jPDL Process Designer, ki nam omogoča grafično modeliranje v jeziku jPDL in direktno objavo na aplikacijski strežnik oz. procesni pogon.



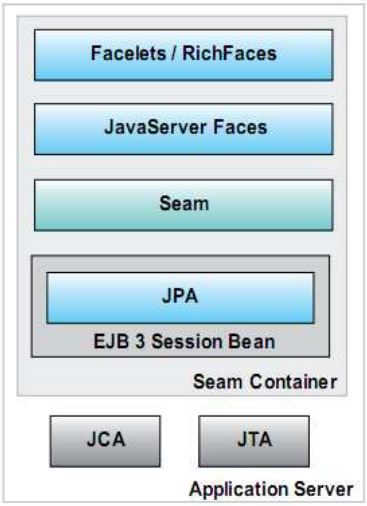
Slika 5.3: Orodje jPDL Process Designer omogoča grafično modeliranje jPDL.

5.3 Uporabljene tehnologije

5.3.1 JBoss Seam

Ker sloni jBPM na programskem jeziku Java, smo si za programsko ogrodje izbrali JBoss Seam. Ta je v zadnjih letih postal zelo priljubljen med Java spletnimi razvijalci, saj prinaša veliko funkcionalnosti, ki omogočajo hitro pisanje učinkovite kode in razvoj naprednih web 2.0 spletnih aplikacij. Seam gradi na tehnologijah Java EE in standardih JSF, EJB3, JPA in poleg standardnih funkcionalnosti dodaja množico svojih. Ključni novosti, ki ju prinaša Seam, sta bijektiranje (ang. bijection) in možnost, da komponenta živi za obdobje konversacije (ang. conversation scoped component).

Bijektiranje je poenostavljeno dvosmeren DI (ang. Dependancy Injection). DI omogoča, da lahko zelo enostavno in brez nepotrebne kode, v svojo komponento prikličemo neko drugo instanco komponente, ki trenutno živi v programskem prostoru (ang. context). Seam poleg priklica in dostopa do komponente omogoča, da lahko komponento (lahko je to tudi objekt ali spremenljivka) izpostavimo nazaj v prostor in se tako nanjo sklicujemo v drugi komponenti (najpogosteje v JSF kodi, ki skrbi za vizualizacijo). Če želimo



Slika 5.4: Seam in povezane tehnologije.

uporabljati bijektiranje, moramo komponento (lahko je to SLSB[2], SFSB[3] ali navaden POJO) označiti s Seam anotacijo `@Name` in določiti ime.

Druga omenjena novost omogoča, da lahko poljubno nadzorujemo življenjsko dobo komponente. To storimo tako, da komponento označimo z anotacijo `@Scope(ScopeType.CONVERSATION)`. S tem pridobimo možnost, da vsi objekti v naši komponenti in njihova stanja preživijo skozi več spletnih strani; predvsem je tu dobrodošla možnost razširjenega konteksta za delo z entitetami (ang. extended persistence context).

```
1  @Name("mojaPrvaKomponenta")   //ime Seam komponente
2  @Scope(ScopeType.CONVERSATION)    //tip obsega komponente
3  public class MojaPrvaKomponenta {
4    @In
5    @Out(required=false)
6    private MojaDrugaKomponenta mojaDrugaKomponenta;
7
8    //geterji in seterji
9    ...
10
11   @Begin
12   public void pricniKonversacijo { ... }
13
14   @End
15   public void zakljuciKonversacijo { ... }
16 }
```

[2]ang. kratica za Stateless Session Bean
[3]ang. kratica za Stateful Session Bean

5.3.2 JavaServer Faces

Kot omenjeno, je Seam povezovalec različnih tehnologij. Za prikazovanje spletnih strani smo tako uporabili standard JSF, kateremu smo dodali RichFaces komponente, s katerimi lahko obogatimo statične strani z AJAX (ang. Asynchronous JavaScript and XML) funkcionalnostmi. Naslednji primer prikazuje JSF kodo za prikaz tabele z možnostjo sortiranja stolpca, iskanja po njegovi vsebini in dodatno komponento za premikanje po straneh zapisov (pred nekaj leti bi potrebovali za podobno funkcionalnost nekaj sto vrstic HTML in JavaScript kode). JSF za deklaracijo uporablja jezik XHTML. Ob prikazu se JSF programska koda transformira v HTML in JavaScript.

```
<rich:dataTable value="#{taKomitenti}" var="_taKomitent" rows="10"
             id="tabelaKomitentiNe" reRender="ds">
   <f:facet name="header">
      <h:outputText value="Seznam komitentov v Trženjski akciji"/>
   </f:facet>
   <rich:column sortBy="#{_taKomitent.ime}" sortOrder="ASCENDING"
              filterBy="#{_taKomitent.ime}" filterEvent="onkeyup">
      <f:facet name="header">
         <h:outputText value="Ime"/>
      </f:facet>
   <h:outputText value="#{_taKomitent.ime}"/>
   </rich:column>

   <f:facet name="footer">
      <rich:datascroller id="ds" renderIfSinglePage="false"/>
   </f:facet>
</rich:dataTable>
```

5.3.3 Java Persistence API

Standard EJB 3.0 je prinesel tudi nov standard za delo z relacijskimi podatki imenovan JPA. Ta nam omogoča predstavitev relacijskih podatkov v podatkovni bazi kot Java objekte (imenovane entitete) v aplikaciji. JPA sestavljajo sam programski vmesnik za delo z entitetami, jezik za poizvedovanje JPQL (ang. Java Persistence Query Language) in objektno/relacijski metapodatki. Ker je JPA v osnovi samo programski vmesnik, moramo za delovanje izbrati še implementacijo teh vmesnikov - v našem primeru smo uporabili Hibernate. Spodnji primer prikazuje deklaracijo entitete, ki predstavlja tabelo v podatkovni bazi (opazimo lahko relacijo na drugo entiteto/tabelo v obliki anotacije `@ManyToMany`). Prikazan je še primer poizvedbe v jeziku JPQL.

```java
@Entity
@Table(name = "KOMITENT")
public class Komitent implements Serializable {
   private Long id;
   private String ime;
   ...
   private List<TrzenjskaAkcija> trzenjskeAkcije =
         new ArrayList<TrzenjskaAkcija>();

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   @Column(name = "ID_KOMITENT")
   public Long getId() {
       return id;
   }

   public void setId(Long id) {
       this.id = id;
   }

   @Column(name = "IME", nullable = false)
   public String getIme() {
       return ime;
   }

   public void setIme(String ime) {
       this.ime = ime;
   }

   ... //ostale deklaracije polj

   @ManyToMany(fetch = FetchType.LAZY)
   public List<TrzenjskaAkcija> getTrzenjskeAkcije() {
       return trzenjskeAkcije;
   }
   public void setTrzenjskeAkcije(List<TrzenjskaAkcija> trzenjskeAkcije) {
       this.trzenjskeAkcije = trzenjskeAkcije;
   }
}
```

```java
  private List<Komitent> komitentiSeznam;
   ...
   public void naloziTrzenjskeAkcije() {
    komitentiSeznam = entityManager
      .createQuery("select k from Komitent k order by k.ime")
      .getResultList();
    }
```

Izvorna koda 1: Prikaz definicije JPA entitete in poizvedbe v jeziku JPQL

Ostale tehnologije

V projektu so bile uporabljene še naslednje tehnologije:

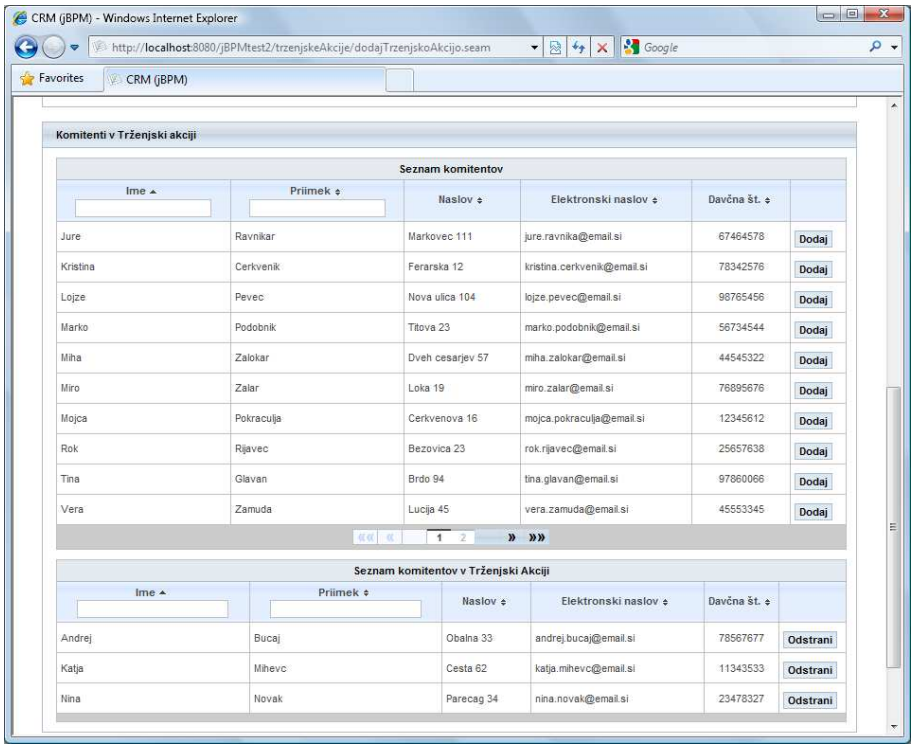
- aplikacijski strežnik Jboss Application Server (4.2.3GA),
- podatkovna baza MySQL (5.1).

5.4 Implementacija trženjskih akcij

Za prikaz delovanja jBPM smo si izbrali pripravo trženjskih akcij v mali banki. Želeli smo prikazati praktičen primer implementacije delovnega toka s pomočjo jBPM-a, zato smo podatkovni model poenostavili (realen primer bi zahteval veliko bolj kompleksen podatkovni model in upoštevanje množice poslovnih pravil).

5.4.1 Opis problemske domene

Trženjska akcija je odločitev banke, da bo pričela tržiti nek ugoden produkt, ki je namenjen določenemu segmentu komitentov. Trženjska akcija traja določeno obdobje, v tem času pa je interes banke, da vse vključene komitente obvesti o novi akciji. V naši mali banki so se odločili, da bodo avtomatizirali pripravo trženjske akcije. Do sedaj je priprava potekala ročno - uslužbenec je pripravil seznam komitentov v Excel datoteki, ki jo je nato sistem uporabil za samodejno pošiljanje po elektronski pošti, skupaj z datoteko, ki je vsebovala vsebino ponudbe. Uslužbenci imajo že na voljo aplikacijo CRM, ki skrbi za vodenje kontaktov s komitenti, tako da bi sedaj dodali še funkcionalnost priprave in vodenja trženjskih akcij. Nova funkcionalnost naj bi vključevala vnos nove trženjske akcije, vnos vsebine ponudbe in izbiro komitentov. Preden se trženjska akcija aktivira in prične pošiljanje, mora akcijo najprej pregledati predstavnik komerciale (pregled vnešene ponudbe) in predstavnik nadzornikov (pregled komitentov). Šele po obeh opravljenih pregledih gre akcija v pregled vodji, ki mora akcijo odobriti. V primeru odobritve se prične pošiljanje ponudb in izdelava aktivnosti, ki bodo služile za vodenje kontaktov. V primeru, da vodja zavrne trženjsko akcijo, se le-ta prenese v ponovni pregled komerciali in nadzornikom.



Slika 5.5: Izbira komitentov, ki so vključeni v trženjsko akcijo.

5.4.2 Podatkovni model

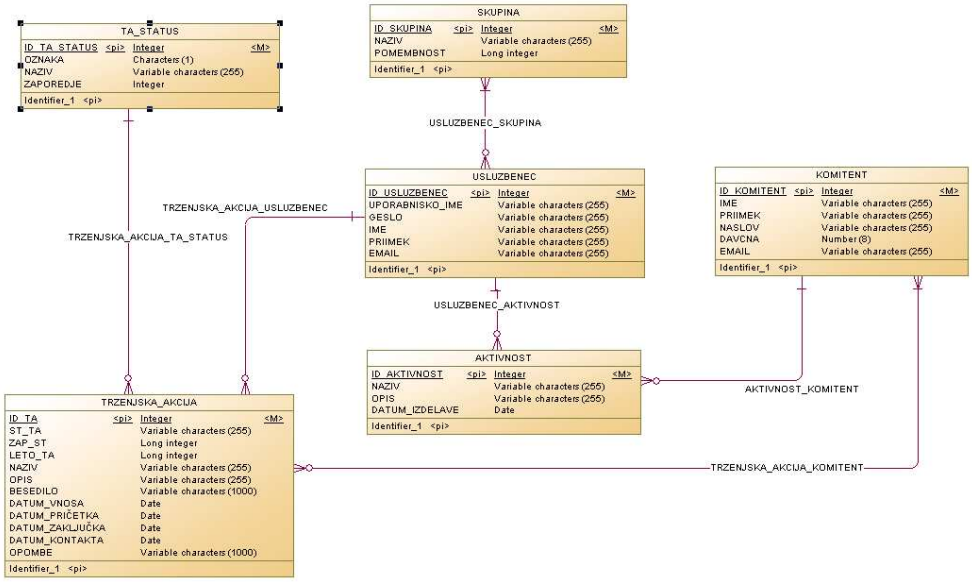
Zaradi enostavnosti smo predpostavili, da v banki delajo samo s fizičnimi osebami. Slika 5.6 prikazuje konceptualni model za trženjske akcije.

Pri tem je tabela:

- TA_STATUS - tabela statusov trženjske akcije; možne vrednosti so v pripravi, v odobritvi, aktivirana, zaključena,
- SKUPINA - tabela uporabniških skupin; v našem primeru so to Skrbniki, Nadzorniki, Komerciala, Vodja OE, Administratorji

5.4.3 Modeliranje poslovnega procesa

Na sliki 5.3 smo že prikazali modeliran proces trženjskih akcij v orodju JPDL Process Designer. Potek procesa je iz grafa zelo enostavno razbrati. Vsako `task` vozlišče predstavlja nalogo, ki jo bomo v aplikaciji implementirali, vozlišča pa povezujejo puščice, ki kažejo smer in zaporedje izvajanja. Vsakemu



Slika 5.6: Konceptualni model za trženjske akcije.

vozlišču lahko določimo tudi posameznika ali skupino, ki mora nalogo opraviti. Vrednosti so lahko fiksne ali pa jih dinamično določimo ob izvajanju.

Naš proces zahteva, da se pregled nadzornikov in komerciale zgodi paralelno. Šele ko vsaka skupina potrdi pregled, gre lahko trženjska akcija v odobritev. Paralelnost zagotovimo z uporabo vozlišča `fork`, ki loči izvajanje procesa, in vozlišča `join`, ki izvajanje ponovno združi.

Spodaj je proces jPDL prikazan še v jeziku XML kot ga ob objavi prebere procesni pogon jBPM.

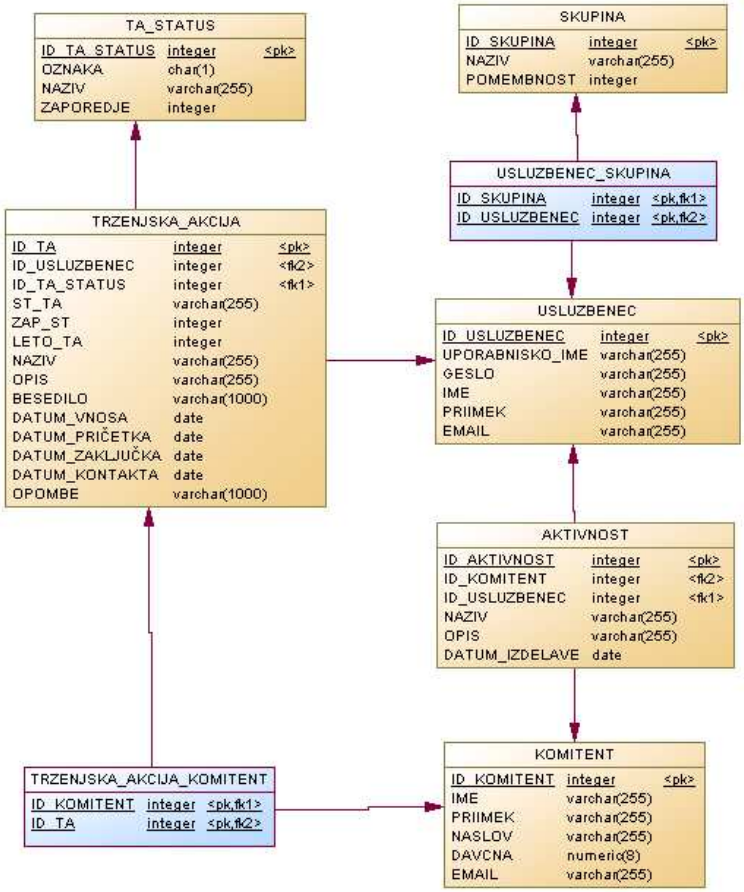
5.4.4 Priprava na objavo procesa

Namestitveni program jBPM-a poskrbi za namestitev potrebnih knjižnic na aplikacijski strežnik. Poskrbeti moramo še za uvoz tabel, ki jih jBPM potrebuje za shranjevanje podatkov. V našem primeru smo se odločili, da bomo jBPM tabele ločili od tistih, ki jih uporablja aplikacija. Tabel je veliko in jih ne želimo mešati z aplikacijskimi. Upravljanje z več podatkovnimi bazami in hkratno sodelovanje znotraj istih transakcij si lahko zagotovimo z XA podatkovnimi viri[4] (ang. XA Data Sources).

[4]Standard XA omogoča, da so lahko različni viri (kot npr. podatkovne baze, strežniki, sporočilne vrste) dosegljivi znotraj ene same transakcije, s čimer lahko zagotovimo ACID

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2" name="priprava-TA">

  <start-state name="trzenjska-akcija-vnesena">
    <transition to="priprava-razdruzitev"/>
  </start-state>

  <fork name="priprava-razdruzitev">
    <transition name="pregled-komitentov" to="pregled-komitentov"/>
    <transition name="ponudba-pregled" to="pregled-ponudbe"/>
  </fork>

  <task-node name="pregled-komitentov">
    <task name="pregled-komitentov" description="Pregled komitentov za
        Trženjsko akcijo: #{novaTA.stTA}">
      <assignment pooled-actors="Nadzorniki"/>
    </task>
    <transition name="zdruzitev-priprave" to="priprava-zdruzitev"/>
  </task-node>

  <task-node name="pregled-ponudbe">
    <task name="pregled-ponudbe" description="Pregled ponudbe za Trženjsko
         akcijo: #{novaTA.stTA}">
      <assignment pooled-actors="Komerciala"/>
    </task>
    <transition name="zdruzitev-priprave" to="priprava-zdruzitev"/>
  </task-node>

  <join name="priprava-zdruzitev">
    <transition name="trzenjska-akcija-odobritev" to="odobritev-trzenjske-
        akcije"/>
  </join>

  <task-node name="odobritev-trzenjske-akcije">
    <task name="odobritev-trzenjske-akcije" description="Odobritev
        Trženjske akcije: #{novaTA.stTA}">
      <assignment pooled-actors="Vodje OE"/>
    </task>
    <transition name="odobri" to="trzenjska-akcija-aktivirana"/>
    <transition name="zavrni" to="priprava-razdruzitev"/>
  </task-node>

  <end-state name="trzenjska-akcija-aktivirana"/>

</process-definition>
```

Izvorna koda 2: jPDL proces zapisan v jeziku XML.



Slika 5.7: Fizični model za trženjske akcije.

Proces smo predhodno že skopirali na aplikacijski strežnik s pomočjo orodja jPDL Process Designer. Ta pripravi tudi sliko procesa, ki jo lahko po objavi gledamo preko jBPM konzole in na njej spremljamo potek določene instance. V komponentnem deksriptorju moramo samo še definirati, kje se nahaja definicija našega procesa (datoteka *components.xml*):

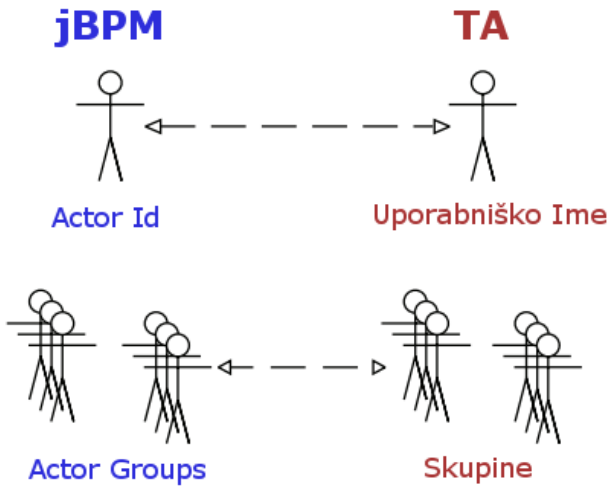
```
<bpm:jbpm>
    <bpm:process-definitions>
        <value>priprava-TA.jpdl.xml</value>
    </bpm:process-definitions>
</bpm:jbpm>
```

lastnosti med različnimi aplikacijami.

Tako bo Seam ob dvigu aplikacije poskrbel za vnos procesa v podatkovno bazo (objavo) in zagon potrebnih jBPM komponent.

5.4.5 Avtentikacija in dodelitev vlog

Ena izmed osnovnih funkcionalnosti jBPM-a je dodeljevanje nalog glede na definicijo procesa. Če želi pogon vedeti, komu mora nalogo dodeliti, moramo najprej povezati vloge, ki veljajo za našo domeno, z vlogami, ki jih pozna jBPM. Trenutnega uporabnika v jBPM-u imenujemo `Actor`, skupine katerim pripada pa `Actor Groups`.



Slika 5.8: Povezava med uporabniki iz trženjskih akcij in uporabniškim modelom jBPM.

Povezavo lahko opravimo med procesom prijave uporabnika. Spodnja koda prikazuje metodo, ki to povezavo opravi. Postopek je enostaven in vključuje najprej preverjanje uporabniškega imena v podatkovni bazi, nastavitev uporabnika `Actor` in branje uporabnikovih skupin v seznam `Actor groups`. Dostop do jBPM uporabniškega konteksta si zagotovimo z injektiranjem istoimenske Seam komponente (`@In private Actor actor;`)

```
1   ...
2   @Transactional
3   public boolean prijava() {
4     String upoIme = identity.getCredentials().getUsername();
5     String geslo = identity.getCredentials().getPassword()
6     Usluzbenec usluzbenec = (Usluzbenec) entityManager
7       .createQuery("select u from Usluzbenec u where u.uporabniskoIme = :
            uporabniskoIme")
8       .setParameter("uporabniskoIme", upoIme)
9       .getSingleResult();
10
11    if(!validatePassword(geslo,usluzbenec)) {
12      return false;
13    }
14
15    actor.setId(upoIme);
16    if (usluzbenec.getSkupine() != null) {
17      for (Skupina skupina : usluzbenec.getSkupine()) {
18        actor.getGroupActorIds().add(skupina.getNaziv());
19      }
20    }
21    return true;
22  }
23  ...
```

5.4.6 Sprožitev procesa

Proces želimo sprožiti, ko uslužbenec shrani trženjsko akcijo. Pričetek procesa sprožimo s Seam anotacijo `@CreateProcess` nad metodo, ki bo opravila proces shranitve. Kot parameter dodamo še ime definicije procesa.

```
1   @Out(scope = ScopeType.BUSINESS_PROCESS, required = false)
2   private String stTrzenjskeAkcije;
3
4   @Out(scope = ScopeType.BUSINESS_PROCESS, required = false)
5   private String pripravilTrzenjskoAkcijo;
6   ...
7   @CreateProcess(definition = "priprava-TA")
8   public String dodajTA() {
9
10    if (!preveriNovaTA())
11      return null;
12
13    novaTA.setKomitenti(komitentiTA_da);
14    entityManager.persist(novaTA);
15
16    stTrzenjskeAkcije = novaTA.getStTA();
17    pripravilTrzenjskoAkcijo = novaTA.getVnesel().getPolnoIme();
18
19    facesMessages.add("Trženjska akcija št: " + novaTA.getStTA() + " je
          bila uspešno dodana.");
20    return "/trzenjskeAkcije/seznamTrzenjskihAkcij.xhtml";
21  }
22  ...
```

Opazimo lahko, da smo s pomočjo bijektiranja, v Seam kontekst vnesli spremenljivki `stTrzenjskeAkcije` in `pripravilTrzenjskoAkcijo`. Z obsegom `BUSINESS_PROCESS` smo povedali Seam-u, da shrani spremenljivki tudi v jBPM kontekst in jih s tem dodeli trenutnemu procesu. Tako lahko kadarkoli znotraj istega procesa dostopamo do navedenih spremenljivk (spremenljivke in vrednosti se hranijo v jBPM podatkovni bazi). jBPM nam omogoča hranjenje vseh osnovnih tipov, nekaj dodatnega truda pa je potrebnega za hranjenje entitet aplikacije. V našem primeru zadostuje, da shranimo številko trženjske akcije, s katero lahko kadarkoli naredimo poizvedbo v podatkovno bazo in pridobimo celotno entiteto.

Proces se je sedaj iz začetne pozicije premaknil na vozlišče `join` in naprej na vozlišči `pregled-komitentov` in `pregled-ponudbe`. Glede na definicijo so lastniki prve naloge *Nadzorniki*, medtem ko je druga dodeljena *Komerciali*. Namesto imena skupine bi lahko definirali uporabniško ime, s čimer bi se naloga dodelila določenemu uporabniku. V našem primeru si zagotovimo, da lahko nalogo prevzame kdorkoli iz pripadajoče skupine.

5.4.7 Izvajanje nalog

Seam omogoča dostop do uporabnikovih nalog preko posebnih registriranih globalnih spremenljivk. Najpomembnejši sta:

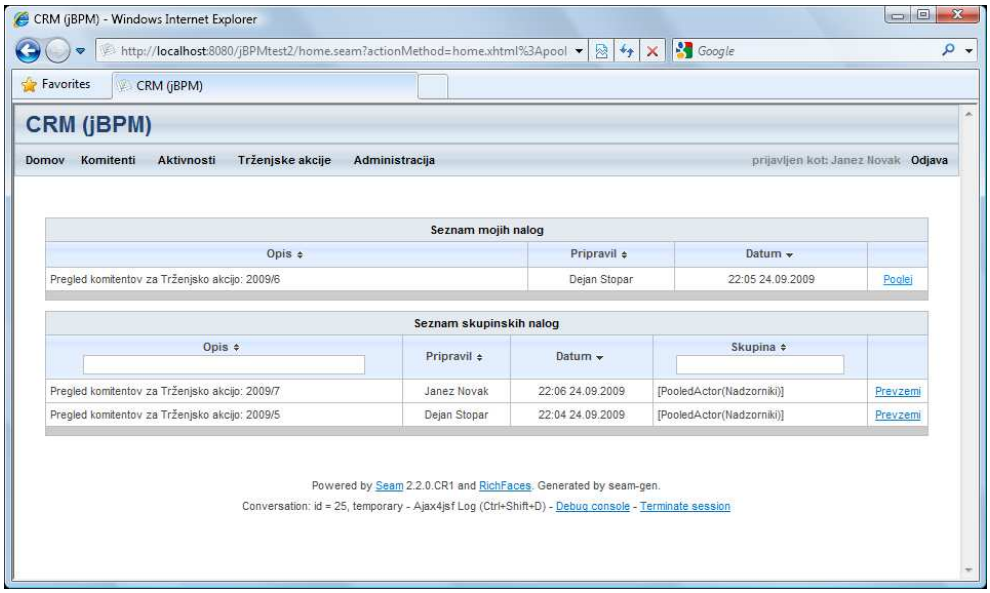
- *taskInstanceList* - hrani seznam vseh nalog, ki so trenutno dodeljene trenutnemu uporabniku;

- *pooledTaskInstanceList* - hrani seznam vseh nalog, ki so trenutno dodeljene katerikoli skupini, katere član je trenutni uporabnik.

Naloge, ki so dodeljene skupini, mora uporabnik najprej prevzeti, šele nato lahko operira z njimi. Prevzem naloge opravimo z metodo `assignToCurrentActor`, ki je dosegljiva nad vsakim zapisom naloge v tabeli s seznamom. Ob prevzemu se naloga prenese iz seznama `pooledTaskInstanceList` v `taskInstanceList`:

```
...
<rich:column>
  <s:link action="#{pooledTask.assignToCurrentActor}"
      taskInstance="#{_naloga}" value="Prevzemi"/>
</rich:column>
...
```

Pričetek same naloge sprožimo s Seam anotacijo `@BeginTask`, ki jo dodamo nad metodo, ki bo sprožila določeno logiko. Podobno sprožimo tudi konec naloge, in sicer z anotacijo `@EndTask`, ki je lahko nad isto metodo ali poljubno drugo. Dejanski pričetek oz. zaključek naloge Seam izvede ob uspešno izvedeni metodi. Logika, ki jo izvedemo med pričetkom in zaključkom naloge, nima nobene povezave z jBPM in se lahko izvaja neodvisno in poljubno dolgo. jBPM bo vedno zaključil predhodno pričeto nalogo in prešel na naslednje vozlišče.



Slika 5.9: Pregled nalog uporabnika in skupin.

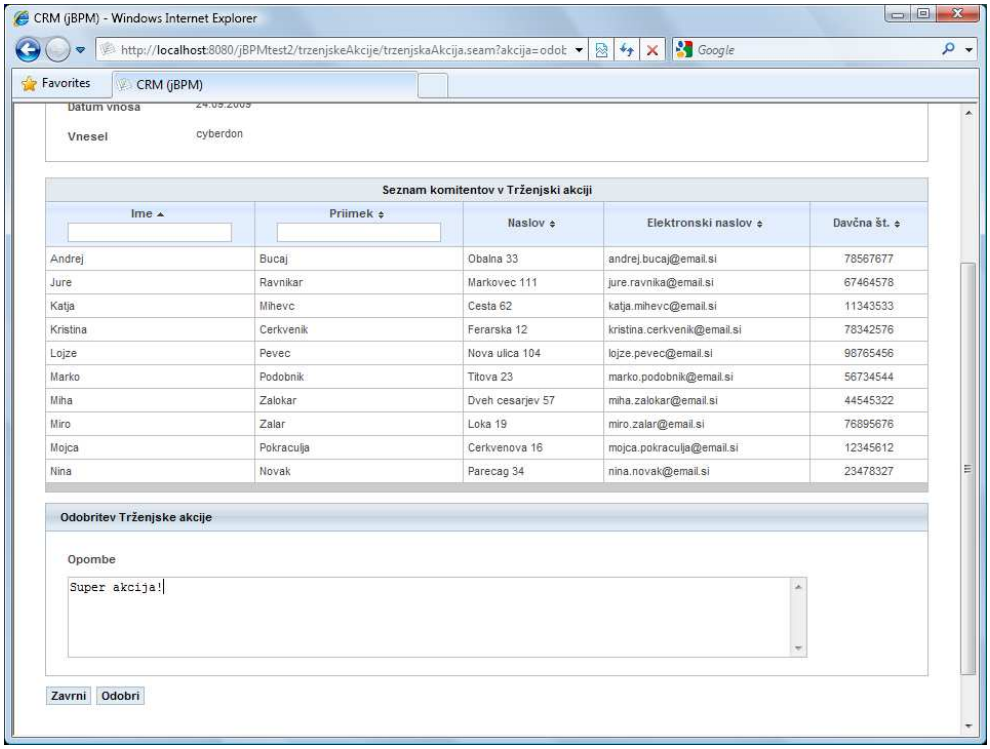
Spodnja koda prikazuje metodo, ki jo kličemo, ko uporabnik izbere nalogo iz seznama njegovih nalog. Klic metode sproži izvajanje trenutne naloge (anotacija `@BeginTask`). S pomočjo injektiranja spremenljivke `stTrzenjskeAkcije` iz konteksta jBPM, lahko naložimo trženjsko akcijo na katero se naloga veže. Metoda vrne ime naloge, ki jo uporabimo za navigacijo na ustrezno stran z ustreznimi parametri. Pravila navigacije definiramo v posebni Seam datoteki *pages.xml*. V kolikor gre za pregled podatkov (pregled ponudbe, pregled komitentov), uporabnika preusmerimo na urejevanje trženjske akcije, kjer lahko podatke ureja in shrani. V primeru odobritve, bo uporabnik preusmerjen na standarden pregled trženjske akcije, kjer mu prikažemo tudi posebno okno za odobritev.

```
...
@In
private TaskInstance taskInstance;  //dostop do trenutne naloge

@In(scope = ScopeType.BUSINESS_PROCESS, required = false)
private String stTrzenjskeAkcije;   //st. akcije v procesu

@BeginTask
public String pricniNalogo() throws Exception {

  naloziTA();     //nalozi akcijo iz konteksta ali podatkovne baze

  return taskInstance.getName();  //vrni ime trenutne naloge
}
...
```

```
<page view-id="/home.xhtml">
  <navigation from-action="#{pripravaTrzenjskeAkcije.pricniNalogo}">
    <rule if-outcome="pregled-komitentov">
      <redirect view-id="/trzenjskeAkcije/urediTrzenjskoAkcijo.xhtml"/>
    </rule>
    <rule if-outcome="pregled-ponudbe">
      <redirect view-id="/trzenjskeAkcije/urediTrzenjskoAkcijo.xhtml"/>
    </rule>
    <rule if-outcome="odobritev-trzenjske-akcije">
      <redirect view-id="/trzenjskeAkcije/trzenjskaAkcija.xhtml">
        <param name="idTA" value="#{novaTA.id}"/>
        <param name="akcija" value="odobritev"/>
      </redirect>
    </rule>
  </navigation>
</page>
```

jBPM bo v primeru opravljenega pregleda, premaknil proces na `join` vozlišče. Šele, ko se bosta končala oba pregleda, bo proces prešel na končno vozlišče `odobritev-trženjske-akcije` in člani skupine *Vodje OE* bodo obveščeni o novi nalogi. Uporabnik bo nato lahko prevzel nalogo in jo pričel izvajati. Spodnja koda prikazuje metodo, ki se izvede, ko uporabnik odobri trženjsko akcijo. V njej prestavimo status na *odobrena* in shranimo novo stanje. Uporabnika vrnemo na prvo stran in izpišemo sporočilo o uspešno aktivirani trženjski akciji. Premik procesa po ustrezni poti zagotovimo z dodatnim parametrom na anotaciji `@EndTask(transition = "odobri")`. V primeru zavrnitve, kličemo metodo z anotacijo `@EndTask(transition = "zavrni")`.

```
1   ...
2   @EndTask(transition = "odobri")
3   public String odobritev() {
4     TaStatus novStatus = (TaStatus) entityManager
5         .createNamedQuery("taStatusIsciPoOznaki")
6         .setParameter("oznaka", TaStatus.TaStatusOznaka.ODOBRENA)
7         .getSingleResult();
8     izbranaTA.setTaStatus(novStatus);
9
10    entityManager.persist(izbranaTA);
11
12    facesMessages.add("Trženjska akcija je bila odobrena...");
13    return "/home.xhtml";
14  }
15  ...
```



Slika 5.10: Odobritev trženjske akcije.

Z odobritvijo se proces priprave trženjske akcije zaključi. Sedaj trženjska akcija samo še čaka na datum pričetka, s čimer se njen status spremeni na *aktivna* in prične se obveščanje izbranih komitentov po elektronski pošti in izdelava aktivnosti.

5.5 Ugotovitve

Dandanes je večina programerjev navajena standardnih načinov razmišljanja, ki jih pogojuje razvoj CRUD[5] aplikacij. Tako razmišljanje pogosto teži k temu, da spregledamo možnost uporabe poslovnega procesa (ali delovnega toka) v aplikaciji. Z vpeljavo upravljanja procesa se pogosto izognemo posebnim stolpcem v tabelah, ki služijo shranjevanju stanja o zapisih. Tako se izognemo dodajanju prehodnih informacij (podatki, ki niso relevantni zunaj poslovnega procesa) v našo podatkovno shemo. V nasprotnem primeru se lahko ob pojavu novih zahtev stanje in pregled nad podatki v podatkovni bazi in sami aplikaciji samo poslabšuje.

Na implementiranem primeru smo lahko videli, da je vpeljava delovnega toka v Java projekt s pomočjo Seam-a in jBPM-a relativno enostavna. Poudariti je potrebno, da jBPM nudi obilico drugih možnosti, ki tukaj niso bile opisane (avtomatizacija procesov s pomočjo časovnikov, uporaba sporočilnih sistemov, dinamično računanje in odločanje ...). Integracija s Seam-om je malce omejena, tako da je potrebno za naprednejše funkcije dostopati direktno do jBPM programskega vmesnika, kar pa se bo najverjetneje izboljšalo s prihodnjimi verzijami Seam-a.

Mogoče bi lahko ugovarjali, da jBPM ni pravi predstavnik področja BPM, ampak le malce naprednejše ogrodje, ki nudi procesni pogon in podporo za izvajanje delovnih tokov. V prid takemu razmišljanju govori tudi dejstvo, da ne moremo izvajati naprednih BAM analiz in poročil v sklopu nudene konzole. Vseeno moramo upoštevati, da je jBPM šele prišlek na tem področju in da se bo v prihodnje izboljševal, kar kaže tudi nova verzija 4.0.

[5] ang. Create, Read, Update, Delete - Dodaj, Beri, Posodobi, Briši

Poglavje 6

Zaključek

Podjetja, ki prodajajo svoje rešitve in oglašujejo kratici BPM in SOA, srečamo že na vsakem koraku. Večja podjetja so bila že prisiljena take metode in tehnologije spoznati oziroma implementirati, vendar je poznavanje področja med ostalimi uporabniki (tako poslovnimi kot IT uporabniki) slabo. Razlogov za nezainteresiranost med IT uporabniki je več: okorelost implementacij, uporaba nestandardnih tehnologij in izredno visoke cene storitev oziroma rešitev. Poleg tega bi bile tudi nujne spremembe v razmišljanju in spremembe organizacijske strukture, ki jih take metodologije predvidevajo. Še pred kratkim je prevladovalo mnenje, da je to le marketinški trik, ki ne prinaša nobenih izboljšav podjetju, kvečjemu nasprotno.

Vendar pa se je področje v zadnjih letih močno okrepilo. K temu je prispevala ustanovitev organizacij, s pomočjo katerih so bili sprejeti določeni pomembni standardi kot sta BPMN in BPEL. Poleg programskih velikanov so se pojavila tudi podjetja, ki nudijo prosto dostopne rešitve (npr. JBoss, Intalio). Na internetu so se organizirale določene skupnosti, kjer se razpravlja o BPM-ju in povezanih tehnologijah. Glede na poročanja [27] je BPMS trg v letu 2006 dosegel približno za 1.7 milijarde dolarjev prihodkov in s tem pridobil lastnosti "mainstream" trga programske opreme, t.j. stabilna tehnologija, konsolidirani ponudniki in hitra adaptacija uporabnikov. BPMS trg je tudi drugi najhitreje rastoči tržni segment na področju vmesne programske opreme. Nesporno je torej, da se bo področje v prihodnje samo še širilo in postajalo vse bolj pomemben del vsakega podjetja, ki želi uspešno delovati na trgu.

Literatura

[1] M. Hammer, J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*, Harper Business, 1993.

[2] T. H. Davenport, *Process innovation : reengineering work through information technology*, Mass.: Harvard Business School Press, 1993.

[3] M. A. Ould, *Business Process: Modelling and Analysis for Re-engineering and Improvement*, Baffins Lane, 1995.

[4] M. Havey, *Essential Business Process Modelling*, O'Reilly, 2005.

[5] M. Hammer, J. Champy, *What is reengineering?*, InformationWEEK, 1992.

[6] W. M. P. van der Aalst, A. H. M. ter Hofstede *Business Process Management: A Survey*, Mednarodna konferenca, BPM 2003, Eindhoven, Nizozemska, 2003.

[7] J. B. Hill, M. Pezzini, Y. V. Natis, *Findings: Confusion Remains Regarding BPM Terminologies*, Gartner Research, 2008.

[8] J. Mendling, G. Neumann, *A Comparison of XML Interchange Formats for Business Process Modelling*, Workflow Handbook 2005, 2005.

[9] J. B. Hill, M. Kerremans, T. Bell, *Cool Vendors in Business Process Management*, Gartner Research, 2007.

[10] M. Juric, K. Pant, *Business Process Driven SOA Using BPMN and BPEL*, Packt Publishing, 2005, pogl. 5.

[11] (2007) Formal Foundations of Business Process Management (Systems). Dostopno na: http://is.tm.tue.nl/staff/wvdaalst/publications/p226.pdf

[12] W.M.P van der Aalst, K.M van Hee, *Workflow Management: Models, Methods, and Systems*, MIT press, Campbridge, 2002.

[13] (2006) What is BPM Anyway? Dostopno na: http://www.bpminstitute.org/articles/article/article/what-is-bpm-anyway.html

[14] (2006) Defining the discipline and practice of BPM Dostopno na: http://www.abpmp.org/displaycommon.cfm?an=1subarticlenbr=28

[15] (2009) Intalio—Designer Dostopno na: http://www.intalio.com/products/bpm/community-edition/designer/

[16] (2009) ILOG JViews BPMN Modeler Dostopno na: http://www.ilog.com/products/jviews/diagrammer/bpmnmodeler/

[17] (2009) Oryx Editor Dostopno na: http://bpt.hpi.uni-potsdam.de/Oryx/

[18] (2009) Microsoft Visio Dostopno na: http://office.microsoft.com/en-gb/visio/default.aspx

[19] (2009) Microsoft BizTalk Dostopno na: http://www.microsoft.com/biztalk/en/us/bpm.aspx

[20] (2009) IBM BPM Suite Dostopno na: http://www-01.ibm.com/software/info/bpm/offerings.html

[21] (2009) Oracle BPM Suite Dostopno na: http://www.oracle.com/technologies/bpm/bpm-suite.html

[22] (2009) WfMC Dostopno na: http://www.wfmc.org/

[23] (2009) Business Process Management Systems (BPMS). Dostopno na: http://www.abpmp.org/displaycommon.cfm?an=1subarticlenbr=28

[24] (2004) What is BPEL and why is it so important to my business? Dostopno na: http://www.softcare.com/whitepapers/wp_whatis_bpel.php

[25] (2004) BPELJ Enough is enough. Dostopno na: http://www.fairdene.com/bpelj/BPELJ-Enough-Is-Enough.pdf

[26] (2008) Transaction Processing in BPEL Processes. Dostopno na: http://blogs.oracle.com/reynolds/2008/09/transaction_processing_in_bpel.html

[27] B. Hill, M. Cantara, E. Deitert, *Magic Quadrant for Business Process Management Suites*, Gartner Research, 2007.