UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Luka Kokalj

STORITVENO USMERJENA ARHITEKTURA NA OSNOVI TEHNOLOGIJ JAVA EE 5 IN WS-BPEL

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Denis Trček

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: 01548/2009
Datum: 15.03.2009

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

Kandidat: **LUKA KOKALJ**

Naslov: **STORITVENO USMERJENA ARHITEKTURA NA OSNOVI TEHNOLOGIJ JAVA EE 5 IN WS-BPEL**

SERVICES ORIENTED ARCHITECTURES BASED ON JAVA EE 5 AND WS-BPEL TECHNOLOGIES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Sodobni pristopi pri informatizaciji poslovnih procesov v podjetjih težijo k storitveno usmerjenim arhitekturam, ki temeljijo na spletnih storitvah. Z njihovo pomočjo lahko različne poslovne aplikacije ponudimo kot spletne storitve. S tem je na standardiziran način omogočen dostop do teh aplikacij in njihovih funkcionalnosti ter njihovo združevanje z drugimi aplikacijami v nove storitvene kompozite. Posledično potrebujemo ustrezne rešitve za združevanje omenjenih funkcionalnosti in za definiranje poslovnih procesov. V nalogi tako obdelajte te rešitve za okolje Java EE5 s poudarkom na jeziku WS-BPEL.

Mentor:

prof. dr. Denis Trček

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

ZAHVALA

Zahvaljujem se mentorju prof. dr. Denisu Trčku za njegove zamisli, ideje in nasvete ter strokovno pomoč pri izdelavi diplomskega dela.

Prav tako se zahvaljujem svoji družini, prijateljem in vsem, ki so me v času študija podpirali in razumeli ter s tem omogočili, da sem študij uspešno zaključil s tem diplomskim delom.

Kazalo

Slovar kratic

Java EE	-	Java Platform, Enterprise Edition,
API	-	Application Programming Interface,
BPEL	-	Business Process Execution Language
J2EE	-	Java 2 Platform, Enterprise Edition, po novem Java EE,
EJB	-	Enterprise JavaBeans,
JMS	-	Java Message Service,
XML	-	eXtensible Markup Language,
JNDI	-	Java Naming and Directory Interface,
JPA	-	Java Persistence API,
Java SE	-	Java Platform, Standard Edition,
POJO	-	Plain Old Java Object,
ER diagram	-	Entitetni relacijski diagram,
CRUD	-	Create, Read, Update, Delete,
SQL	-	Structured Query Language,
JTA	-	Java Transaction API,
JAXB	-	Java Architecture for XML Binding,
WSDL	-	Web Services Description Language,
JAX-WS	-	Java API for XML-Based Web Services,
JSF	-	Java Server Faces,
DOM	-	Document Object Model,
SAX	-	Simple API for XML,
W3C	-	World Wide Web Consortium,
JVM	-	Java Virtual Machine,
SOAP	-	Simple Object Access Protocol,
MIME	-	Multipurpose Internet Mail Extension,
HTTP	-	HyperText Transport Protocol,
XSD	-	XML Schema Document,
RPC	-	Remote Procedure Call,
UDDI	-	Universal Description, Discovery and Integration,
SEI	-	Service Endpoint Interface,
SOA	-	Services Oriented Architecture,
COBOL	-	Common Business Oriented Language,
WSCI	-	Web Service Choreography Interface,
WS-CDL	-	Web Services Choreography Description Language,
WSFL	-	Web Services Flow Language,
JBI	-	Java Business Integration,
BPELJ	-	BPEL for Java,
WSIF	-	Web Services Invocation Framework,
DVD	-	Digital Versatile Disk,
JCA	-	Java EE Connector Architecture,
JSP	-	Java Server Pages.

Povzetek

Namen pričujočega diplomskega dela je predstaviti platformo Java EE 5, najpomembnejše tehnologije, ki so v njej zajete, in prikazati uporabnost jezika BPEL. Prikaz temelji na praktičnem primeru dveh podjetij, ki izvajata klasične poslovne procese. Prvo podjetje predstavlja dobavitelja določenih artiklov, drugo pa porabnika njegovih storitev, trgovca.

Informatizacija njunega poslovanja se začne pri podatkovni bazi, kjer se pri upravljanju s podatki za zelo uporabno izkaže prva predstavljena tehnologija JPA. Skupaj z njo je kot novost platforme predstavljena tehnologija notacij. Notacije tesno sodelujejo tudi z razredi EJB, ki skrbijo za vso poslovno logiko aplikacij in so predstavljeni zatem. Aplikacija Dobavitelj šteje med svoje funkcije tudi kreiranje in izvoz zapisov XML, za kar ustrezno poskrbi tehnologija JAXB, ki predstavlja tudi uvod v spletne storitve JAX-WS. Te so predstavljene nekoliko obširneje, skupaj s še nekaterimi ostalimi protokoli in standardi. Nadalje je predstavljen jezik BPEL, ki preko orkestracije v medsebojno sodelovanje poveže več spletnih storitev. Na koncu sta zgrajena še dva preprosta spletna uporabniška vmesnika z namenom predstaviti še eno novost v platformi Java EE, tehnologijo JSF.

Med grajenjem aplikacije in preučevanjem teh različnih tehnologij se izkaže, da sta platforma Java EE 5 in jezik BPEL izvrstna kombinacija za grajenje najrazličnejših modernih rešitev storitveno usmerjene arhitekture.

Ključne besede: Java EE 5, tehnologija, BPEL, aplikacija, spletna storitev, storitveno usmerjena arhitektura.

Abstract

The purpose of this work is to present Java EE 5 platform, the most important technologies that are included with it, and to show the usefulness of the BPEL language. The presentation is based on a practical example of two trading companies which perform standard business processes. The first company represents the supplier of products while the other represents the consumer of its services - the merchant.

Designing the informational support of their businesses starts with database management, where JPA technology proves to be very useful. Annotations as the new feature of the platform are presented next. They closely cooperate with EJB classes which take care of all the applications business logic. Dobavitelj application also implements functions for creating and exporting XML data. These functions are available through JAXB technology, which also introduces the JAX-WS web services. The presentation of the JAX-WS technology is more comprehensive and it also includes some other protocols and standards. BPEL language is presented next. Its purpose is to integrate several web services through orchestration. At the end two simple web user interfaces are built with intention to present yet another novelty in Java EE platform - the JSF technology.

During programming and examining these different technologies, it turns out that Java EE 5 platform and BPEL language are an excellent combination for designing various modern service oriented architecture solutions.

Keywords: Java EE 5, technology, BPEL, application, web service, service oriented architecture.

1 Uvod

Informatizacija poslovnih procesov v podjetjih poteka že od samega nastanka računalnikov. Na začetku so se razvile preproste aplikacije, ki so omogočale podporo vodenju glavne knjige, obračunu plač ipd. Z razvojem strojne opreme so vseskozi napredovale tudi programske rešitve. Razvijati so se začeli sistemi, ki so vsak po sebi ponujali določeno informacijsko podporo poslovnim procesom. To so bili razni pisarniški sistemi, upravljalski in poslovodni sistemi, pa sistemi za podporo odločanju itd. Sam vrh takšnih celovitih rešitev predstavljajo celoviti poslovni informacijski sistemi.

Kljub veliki izbiri omenjenih programskih rešitev pa se številna podjetja iz takšnih ali drugačnih razlogov odločajo za lastne implementacije aplikacij, ki podpirajo njihove specifične poslovne procese. Močna platforma, ki omogoča takšno realizacijo, je prav gotovo Java Enterprise Edition. Ta poleg standardnih specifikacij Java API vsebuje tudi nekatere posebne komponente, ki v navezi z aplikacijskim strežnikom Java EE omogočajo razvoj prenosljivih in razširljivih poslovnih rešitev. Aplikacijski strežnik poskrbi za izvajanje transakcij, varnost, sočasnost in na splošno za upravljanje naloženih komponent, tako da se lahko razvijalec osredotoči na sam poslovni proces in njegovo pripadajočo poslovno logiko.

Z vsakodnevnim povečevanjem tekmovalnosti med podjetji pa poslovne aplikacije ne smejo biti več izolirane. Učinkovito morajo znati sodelovati znotraj podjetja, čedalje bolj pomembno pa je tudi njihovo povezovanje z drugimi podjetji. Integracija različnih aplikacij je bila vedno zapleten problem, zaradi različnih, predvsem tehnoloških razlogov.

Najboljši odgovor na dani problem integracije predstavlja storitveno usmerjena arhitektura *(angl. Services Oriented Architecture ali SOA)*, ki temelji na uporabi spletnih storitev. Poslovne aplikacije lahko izpostavimo kot spletne storitve, ki tako na standardni način ponujajo svojo funkcionalnost različnim odjemalcem. Razvoj spletnih storitev in izpostavljanje njihove funkcionalnosti pa sámo po sebi ni dovolj. Potreben je način za združitev teh funkcionalnosti, način za definiranje poslovnega procesa, ki bo uporabljal in izkoriščal te izpostavljene funkcionalnosti. Na tej točki postane pomemben visokonivojski programski jezik BPEL, ki omogoča kompozicijo spletnih storitev in predstavlja procesno-usmerjeni pristop k storitveno usmerjeni arhitekturi. [1]

2 Java Platform, Enterprise Edition 5

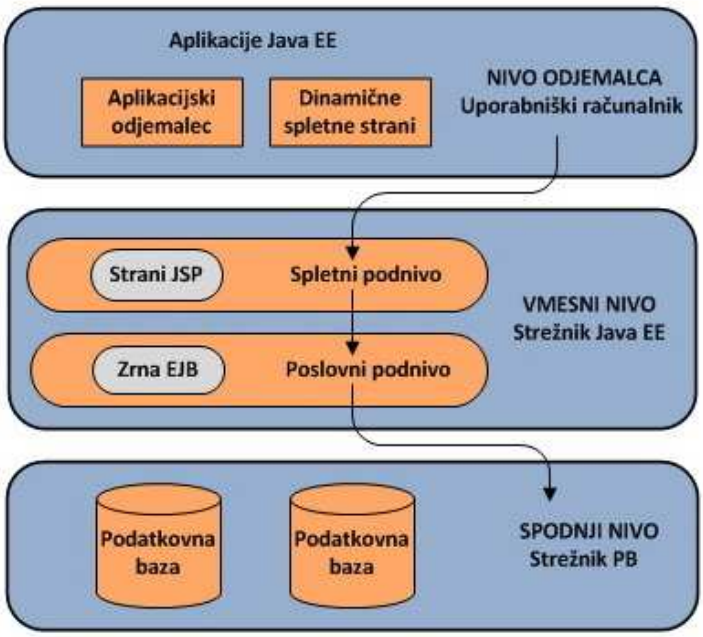
2.1 Pregled

»Z manj dela naredi več«, je vodilo nove platforme Java EE. To je zadnja, najnovejša verzija te vodilne platforme v svetu razvoja poslovnih aplikacij. Njen namen je zmanjšati stroške in kompleksnost razvoja in upravljanja večnivojskih, strežniško usmerjenih aplikacij. Poenostavljen razvoj se odraža v manjšem številu vrstic kode, potrebnih za implementacijo istega problema kot pri prejšnji verziji. K temu pripomore velik nabor privzetih vrednosti, odprava večine deskriptorjev XML ter uporaba notacij. Java EE 5 prinaša boljšo podporo spletnim storitvam, podpira več standardov, povezanih z njimi. Poenostavljen je programski model EJB in prehod od aplikacij k storitvam. Pomembna pridobitev je Java Persistence API, vmesnik, namenjen upravljanju s podatki relacijskih podatkovnih baz, ter nova tehnologija Java Server Faces, ki predstavlja učinkovitejši pristop k načrtovanju spletnih aplikacij. [2]

2.2 Arhitektura

2.2.1 Večnivojski aplikacijski model

Predstavitev, poslovna logika in dostop do podatkov so tipični trije nivoji sodobnih poslovnih aplikacij. Takšne aplikacije so tipično zasnovane tako, da na najvišjem nivoju nudijo predstavitve, na srednjem nivoju zagotavljajo storitve aplikacije ter na spodnjem nivoju dostopajo in upravljajo s podatki. Tem nivojem sledi tudi arhitektura Java EE platforme, ki uporablja porazdeljen, večnivojski aplikacijski model gradnje poslovnih aplikacij. [3, 4]



Slika 1: Večnivojski aplikacijski model

Arhitektura Java EE definira nivo odjemalca, vmesni nivo, ki lahko vsebuje enega ali več podnivojev, in spodnji nivo. Nivo odjemalca podpira celo vrsto različnih odjemalcev, ki lahko tečejo znotraj ali pa zunaj lokalnega omrežja in s tem požarnega zidu. Vmesni nivo preko

spletnega vsebnika *(angl. container)* na spletnem podnivoju odjemalcem zagotavlja storitve in izvršuje poslovno logiko v okviru komponent EJB, ki tečejo na poslovnem podnivoju. Na spodnjem nivoju lahko tečejo različni informacijski sistemi, ki so dosegljivi preko standardnih vmesnikov API. [3]

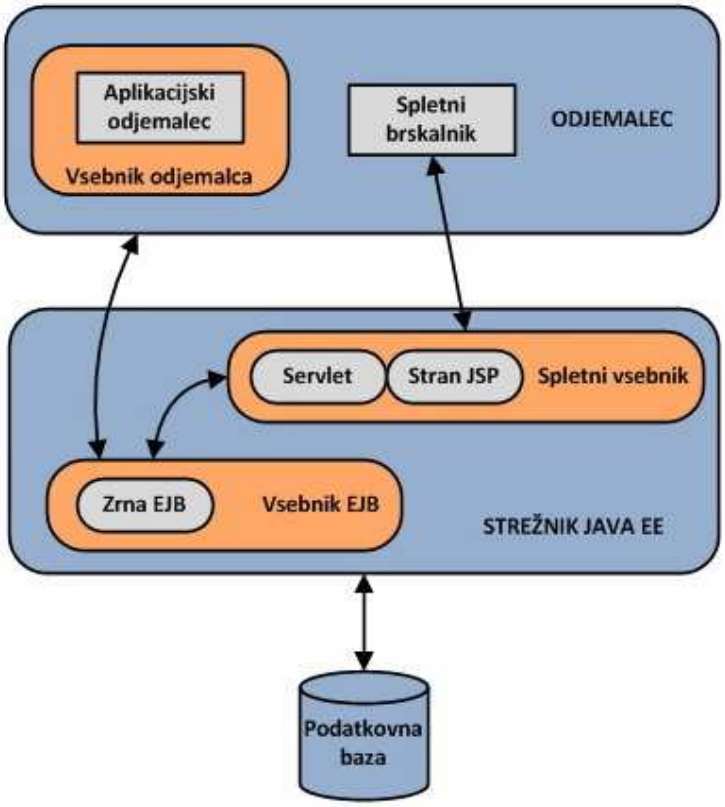
2.2.2 Komponente Java EE

Aplikacije Java EE so zgrajene iz komponent. Le-ta je zaključena celota programske kode, ki je s pripadajočimi razredi in datotekami dodana aplikaciji Java EE, kjer lahko komunicira z drugimi komponentami. Java EE specificira naslednje komponente:

- Komponente EJB so poslovne komponente, ki tečejo na aplikacijskem strežniku Java EE.
- Java Servlet, JSP in JSF so spletne komponente, ki tečejo na spletnem strežniku.
- Aplikacijski odjemalci in apleti so komponente, ki tečejo na odjemalčevem računalniku. [4]

2.2.3 Strežnik Java EE in sistemske storitve

Strežnik Java EE komponentam zagotavlja različne sistemske storitve, kot so varnost, upravljanje transakcij, nadzor sočasnosti, trajnost podatkov ipd. To je pomembna prednost platforme Java EE, saj lastna implementacija teh storitev postane odveč. Vmesniki med sistemskimi storitvami in komponentami, ki te storitve uporabljajo, se imenujejo vsebniki. Ločimo jih več vrst, delijo pa se glede na tip komponente, za katere izvajanje skrbijo. [4]



Slika 2: Vsebniki Java EE

2.3 Tehnologije

2.3.1 Common Annotations for the Java Platform

Uporaba meta podatkov je močan koncept programiranja, ki razvijalcu programske opreme omogoča nastavljanje ali spreminjanje obnašanja aplikacije zunaj dejanjske programske kode. Do sedaj je bilo upravljanje z meta podatki omogočeno preko ločenih datotek, imenovanimi deskriptorji XML. Pravzaprav so bili ti deskriptorji nujno potrebni za delovanje najpreprostejše aplikacije Java EE, pa če je potrebovala meta podatke ali pa ne.

S platformo Java EE 5 se lahko meta podatki vključijo neposredno v programsko kodo, ta proces je znan kot deklarativno oz. parametrsko programiranje. Večino stvari, ki so jih pred tem obvladovali deskriptorji XML, se lahko sedaj enakovredno doseže z uporabo programskih deklaracij – notacij. To sicer ne pomeni, da se sedaj deskriptorji XML ne uporabljajo več, ti so namreč še vedno na mestu. Pravzaprav imajo s stališča strežnika Java EE 5 prednost pred notacijami. Če je nek atribut obnašanja deklariran tako z notacijami kot z deskriptorji, strežnik upošteva zadnje.

Vsekakor pa nova tehnologija notacij predstavlja eno večjih, če ne največjo pridobitev platforme Java EE 5. Poudariti velja tudi, da parametrsko programiranje ni namenjeno le tej platformi, temveč je tudi del Java SE. In zmogljivosti notacij se tukaj še ne končajo. Poleg odprave potrebe po večini deskriptorjev XML notacije prinašajo veliko poenostavitev tudi na drugih področjih. Dovolj je omeniti, da novo tehnologijo notacij uporablja cela vrsta drugih samostojnih tehnologij. Samo nekatere med njimi so EJB 3.0, Java Persistence API, JAX-WS 2.0, itd. Za vse je značilno, da so v primerjavi s prejšnjimi verzijami, če so te obstajale, deležne mnogih poenostavitev in novih zmogljivosti. In da je temu tako, je v veliki meri pripomogla prav tehnologija notacij.

Standardna podpora parametrskemu programiranju, ki jo prinaša uporaba meta podatkov oz. notacij, označuje nov mejnik celotne platforme Java. Za konec k temu dodaja svoj prispevek tudi dejstvo, da platforma Java omogoča deklariranje lastnih notacij. [7, 8]

2.3.2 Java Persistence API

Java EE 5 je predstavila nov, že omenjen vmesnik Java Persistence API ali krajše JPA. Namenjen je bil kot dodatek komponenti EJB, a vendar se je njegova uporaba razvila čez meje EJB. Neposredno ga lahko uporabljajo spletne aplikacije ali aplikacijski odjemalci, meja uporabnosti pa ni niti platforma Java EE. Uporablja se lahko namreč tudi v klasičnih aplikacijah Java SE.

Glavni namen novega vmesnika je preslikava med Java objekti in relacijskimi podatki, njegove glavne odlike pa so:

- Entitete so klasični objekti Java *(angl. Plain Old Java Object ali POJO)*. V nasprotju s staro tehnologijo EJB in njenimi entitetnimi zrni *(angl. Entity Bean)* sedanji entitetni

objekti niso več komponente. So le klasični objekti Java, kar poenostavi programiranje in razumevanje tehnologije.

- Standardizirana objektno-relacijska preslikava. Ker je to že dolgotrajen problem, je njegova rešitev izjemno dobrodošla. Prejšnja rešitev je sicer vsebovala standardizirane entitetne objekte, ni pa rešila vprašanja preslikave med njimi in relacijskimi podatkovnimi bazami. Sedaj se z uporabo notacij ali deskriptorjev XML natančno določi podrobnosti preslikave, neodvisno od posameznih ponudnikov relacijskih podatkovnih baz. Vmesnik JPA poleg tega definira privzete vrednosti za večino podrobnosti preslikave.
- Podpora dedovanju in polimorfizmu. Ker so entitete klasični objekti Java, lahko entitetni razred podeduje drug entitetni ali ne-entitetni razred in obratno. Poleg tega entitete podpirajo polimorfične asociacije.
- Entitetni upravljalec. Z njegovo pomočjo lahko programer kadarkoli hitro in enostavno izvaja klasične operacije nad entitami. To so operacije kreiranja, branja, spreminjanja in brisanja entitet *(angl. Create, Read, Update, Delete ali CRUD).*
- Podpora klasičnim poizvedovalnim jezikom. Ob uporabi posebnega Java Persistence SQL, ki temelji na starejšem jeziku EJB SQL, se lahko uporablja tudi klasični poizvedovalni jezik, čigar sintaksa je odvisna od uporabljene podatkovne baze.
- Imenske poizvedbe. Imenska poizvedba je statična poizvedba, ki se definira in uporablja s pomočjo notacij. Definira se v entitetnih razredih, uporabi pa se lahko povsod preko entitetnega upravljalca.
- Podpora optimističnemu zaklepanju. Ta tehnika za reševanje problema sočasnih dostopov predvideva, da večina transakcij ni v konfliktu med sabo, zato se največ uporablja v okoljih, kjer je malo pisalnih dostopov. Specifikacija standardizira način pisanja entitetnih objektov, da lahko uporabljajo optimistično zaklepanje ne glede na uporabljenega ponudnika. [5]

Java Persistence API je tehnologija, ki je s sabo prinesla veliko poenostavitev in izboljšanj prejšnjih rešitev, pa tudi vključila povsem nove stvari. V navezi z uporabo notacij predstavlja močno tehnologijo, ki se lahko uporablja v poslovnih Java EE ali pa v klasičnih Java SE aplikacijah.

2.3.3 Enterprise JavaBeans 3.0

Komponenta EJB je opredeljena s segmentom programske kode, ki vsebuje polja in metode, ter pomeni implementacijo posameznega modula poslovne logike. Vsak modul je ločen gradnik aplikacije, uporabljen je lahko posamično ali pa v sodelovanju z drugimi moduli izvaja poslovno logiko na strežniku Java EE.

V tretji različici specifikacije EJB poznamo dve vrsti komponent EJB, to sta sejno zrno *(angl. session bean)* in sporočilno gnano zrno *(angl. message-driven bean)*. Prvo predstavlja začasno komunikacijo z odjemalcem, ki je določena z dolžino seje. Ko se odjemalec preneha izvajati, se uniči tudi zrno in njegovi podatki. Za drugi tip zrna je značilno, da je zmožno

sprejemati in odzivati se na asinhrona sporočila. Navadno so to sporočila Java sporočilne storitve *(angl. Java Message Service ali JMS)*. V prejšnji različici, EJB 2.1, je bilo na voljo še entitetno zrno, ki pa je sedaj zamenjano z entitetami vmesnika Java Persistence API.

Poleg omenjene novosti je EJB 3.0 deležen dodatnih poenostavitev. Prva med njimi je prav gotovo dejstvo, da vsebnik EJB opravlja veliko več nalog, kar znatno razbremeni razvijalca modulov. Druga je uporaba notacij, ki so novost zadnje različice EJB. Posledično se zmanjša število potrebnih implementacij razredov in vmesnikov, vmesnik *javax.ejb.SessionBean* npr. nadomesti notacija *@Session*. Tudi večina deskriptorjev XML postane odveč, saj se razne definicije komponent in odvisnosti enakovredno dosežejo z uporabo notacij. Vmesnik *EJBContext* poenostavi uporabo in pred razvijalcem skrije vse podrobnosti poizvedb JNDI. [4, 5]

Zanimiva je neposredna primerjava med tehnologijo EJB 3.0 in njeno predhodnico, EJB 2.1, če primerjamo implementaciji iste aplikacije. Dejstva, kot so število vrstic kode, število razredov in število uporabljenih deskriptorjev XML, nazorno prikazujejo prednosti novejših tehnologij.

Ime aplikacije	Enota merjenja	EJB 2.1	EJB 3.0	Izboljšanje
RoosterApp	Število razredov	17	7	59%
	Število vrstic programske kode	987	716	27%
	Število XML deskriptorjev	9	2	78%
	Število vrstic XML kode	792	26	97%

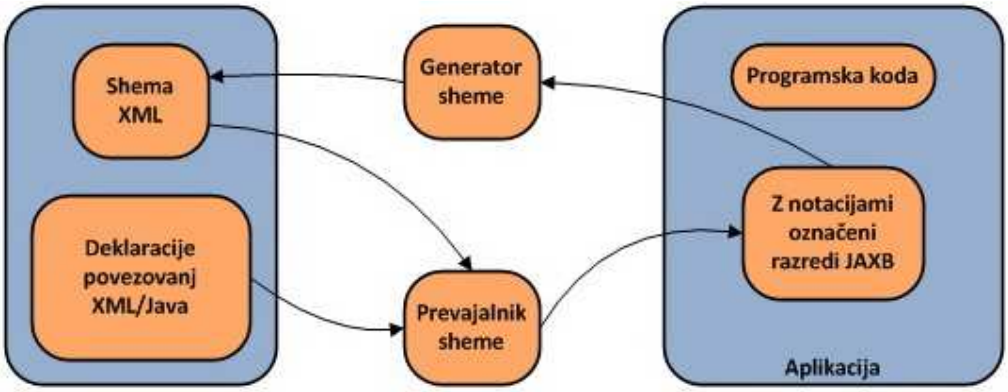
Tabela 1: Povzetek rezultatov primerjav med tehnologijama EJB 2.1 in EJB 3.0

V zgornji tabeli je kot vzorec za primerjavo tehnologij vzeta aplikacija *RoosterApp*, ki je ena izmed uradnih J2EE 1.4 primerov. Rezultati primerjav med obema implemetacijama so povzeti po [6].

2.3.4 Java Architecture for XML Binding 2.0

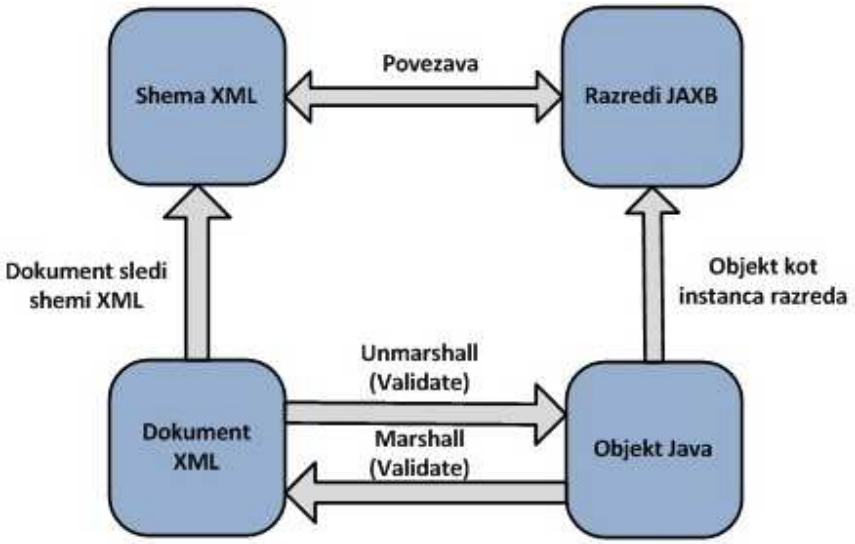
Namen arhitekture JAXB je zagotoviti preprost in hiter način preslikovanja med dokumenti XML in objekti Java. Na nek način jo lahko opišemo kot višjenivojski jezik za preslikovanje, pri katerem programerju ni potrebno skrbeti za samo strukturo datotek XML, ampak le za semantično pomembne informacije. V tem se tudi razlikuje od nekaterih ostalih tehnik za procesiranje XML, kot sta npr. DOM in SAX.

Na spodnji sliki je predstavljena arhitektura JAXB, ki med drugim zajema tudi prevajalnik in generator sheme. Naloga prvega je preslikovanje elementov XML iz pripadajoče sheme XML v ustrezne razrede Java, drugi pa poskrbi za obratno pot, generiranje sheme XML iz razredov Java. Ti morajo biti označeni z ustreznimi notacijami, s pomočjo katerih lahko gradimo povsem poljubne preslikave.



Slika 3: Arhitektura JAXB

Ko je med shemo XML in razredi Java ustvarjena želena preslikava, nam vmesnik API arhitekture JAXB nudi operacije za manipuliranje z vsebino dokumentov XML in programskimi objekti. Operacija *unmarshall* omogoča branje zapisov XML v objekte Java, medtem ko je *marshall* njej obratna operacija. Obstaja tudi operacija za preverjanje ustreznosti *validate*, ki skrbi za preverjanje skladnosti med dokumenti XML in pripadajočo shemo in lahko poteka v obeh smereh. Naštete operacije so prikazane na sliki 4.



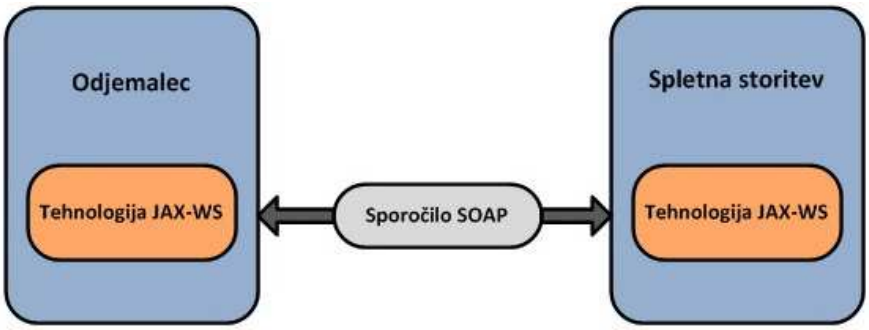
Slika 4: Operacije JAXB

JAXB 2.0, ki spada med spletne tehnologije Java EE 5, uvaja precej novosti glede na svojega predhodnika JAXB 1.1. Najpomembnejša je že omenjena možnost generacije sheme XML iz obstoječih razredov Java, ki je poprej ni bilo. Omogočila jo je uporaba notacij, ki je še ena pridobitev zadnje platforme Java EE. Dve drugi izboljšavi sta še podpora vsem shemam W3C XML in pa znatno zmanjšanje število generiranih razredov pri preslikovanju iz sheme XML. [10]

2.3.5 Java API for XML Web Services 2.0

Java API for XML Web Services ali krajše JAX-WS je tehnologija, ki omogoča gradnjo spletnih storitev in njenih odjemalcev ter temelji na komunikaciji preko jezika XML. Klic operacije

neke spletne storitve je največkrat osnovan na protokolu SOAP. Specifikacija le-tega obsega strukturo sporočila, pravila enkripcije ter standardizirane dogovore o obliki klicev spletnih storitev in njenih odgovorov. Ti klici se po medmrežju prenašajo kot sporočila SOAP preko protokola HTTP.



Slika 5: Komunikacija JAX-WS

Vmesnik API tehnologije JAX-WS odlikuje dejstvo, da pred programerjem skriva večino kompleksnosti sporočil SOAP. Ponuja dva pristopa za kreiranje spletnih storitev. Prvi pristop, od vrha k dnu *(angl. top-down)*, se začne s kreiranjem dokumenta WSDL. JAX-WS vgrajeno orodje zatem iz tega dokumenta avtomatično kreira vse potrebne razrede Java. Med njih sodijo vmesniki, ki navzven ponujajo željene storitve, in pa razredi, ki predstavljajo ogrodje implementacijski logiki. Na programerju je potem le še, da zapolni pripravljene razrede z ustrezno poslovno logiko.

Drugi pristop poteka v nasprotni smeri. Programer sam kreira potrebne razrede in jim doda željene metode, ki bodo kasneje izpostavljene kot operacije spletne storitve. JAX-WS 2.0 za razliko od prejšnje verzije uporablja tehnologijo notacij, ki pri tem pristopu pride zelo prav. Odpadejo spremljevalni deskriptorji XML, potrebna je le pravilna označitev razredov in metod. Zatem na vrsto pride avtomatizem tehnologije JAX-WS, ki generira dokument WSDL in izpostavi spletno storitev.

Implementacija odjemalca spletne storitve je z uporabo tehnologije JAX-WS prav tako poenostavljena. Načrtovalec mora v aplikacijo uvoziti dokument WSDL, nato pa orodje avtomatično kreira ustrezen lokalni objekt, ki predstavlja spletno storitev. Preko tega objekta, imenovanega tudi proksi *(angl. proxy)*, se zatem enostavno kličejo operacije spletne storitve. Programerju ni potrebno kreirati oz. razčlenjevati sporočil SOAP, za to skrbi vmesnik API tehnologije JAX-WS. Navzven je klic operacije oddaljene spletne storitve povsem enak klicu metode lokalnega razreda.

Tehnologija JAX-WS 2.0 prinaša več prednosti pred svojim predhodnikom in ostalimi podobnimi rešitvami. Eno od njih je strojna neodvisnost, ki jo prinaša sam programski jezik Java. Poleg tega ni omejena niti na sam programski jezik, storitev in odjemalec sta namreč lahko implementirana v različnih jezikih. Pred svojim predhodnikom se odlikuje z lažjo implementacijo tako storitev kot odjemalcev. S pomočjo notacij se lahko kot spletna storitev

predstavi vsak klasičen razred POJO. Odpadejo tudi nekateri spremljevalni deskriptorji XML, prednost JAX-WS 2.0 pa je tudi v uporabi že omenjene tehnologije JAXB 2.0. [11, 12]

2.3.6 Java Server Faces 1.2

Tehnologija Java Server Faces predstavlja ogrodje komponentam uporabniškega vmesnika, ki se uporabljajo pri gradnji spletnih aplikacij. Tehnologija obsega vmesnik API, katerega primarna naloga je predstavitev in upravljanje komponent uporabniškega vmesnika. Komponente uporabniškega vmesnika so lahko npr. gumbi ali vnosna polja, ki veljajo za enostavne ali pa razne tabele in menijske vrstice, ki so sestavljene tudi iz več enostavnih komponent.

Programski model omogoča vizualno dodajanje in urejanje komponent na strani, odzivanje na zunanje dogodke preko strežniške aplikacijske kode, povezovanje strežniških podatkov s komponentami uporabniškega vmesnika in shranjevanje stanj komponent med posameznimi sejami. Tehnologija JSF v splošnem prinaša bogata orodja za upravljanje komponent in njihovih stanj, obdelavo podatkov, povezanih z njimi, preverjanje pravilnosti vhodnih podatkov ter upravljanje najrazličnejših dogodkov.

Spletna aplikacija, ki je zgrajena z uporabo tehnologije JSF, vsebuje vrsto določenih gradnikov, med njimi tudi:

- eno ali več strani JSP, čeprav ni omejena le na predstavitveno tehnologijo JSP,
- več podpornih komponent *JavaBean*, ki definirajo lastnosti in funkcije komponent uporabniškega vmesnika,
- konfiguracijsko datoteko, ki skrbi za navigacijo med stranmi in konfiguracijo ostalih komponent in objektov po meri,
- deskriptor *web.xml*, ki vsebuje navodila strežniku, na katerem bo aplikacija tekla.

Tehnologija Java Server Faces 1.2 je kot prva uradno vključena v platformo Java EE 5, kjer sedaj predstavlja standard za grajenje strežniško usmerjenih uporabniških vmesnikov. [25]

3 Bussiness Process Execution Language

3.1 SOA in BPEL

3.1.1 Spletne storitve kot uvod v SOA

Realizacija storitveno usmerjene arhitekture v konkretno rešitev se začne pri spletnih storitvah. Te predstavljajo funkcionalno zaokrožene arhitekturne gradnike, ki so dostopni preko standardnih internetnih protokolov in so neodvisni glede na različne strojne platforme ali programske jezike. Lahko so nove aplikacije ali pa so zgolj ovite okoli že obstoječih, preverjenih sistemov, ki jim tako omogočijo delovanje preko svetovnega spleta. [18] Vsaka spletna storitev je sestavljena iz dveh delov:

- Implementacija storitve, ki je lahko povsem poljubna. Uporablja se lahko jezik JavaScript, tehnologije EJB, BPEL, SQL ali pa celo stari COBOL. Vse, kar je pomembno, je, da tečejo na platformi, ki je povezana s svetovnim spletom.
- Vmesnik storitve, ki mora biti definiran na standarden način. Upoštevati mora določene dogovore o predstavitvi podatkovnih tipov, operacij, omrežnih protokolov, ki jih storitev uporablja itn.

Rešitev SOA je v grobem zbirka spletnih storitev, ki med sabo komunicirajo. Komunikacija lahko pomeni preprosto prenašanje podatkov ali pa lahko vključuje dve ali več spletnih storitev, ki koordinirajo neko aktivnost. Spodnja slika prikazuje osnovno delovanje storitveno usmerjene arhitekture. Na desni prikazuje porabnika spletne storitve, ki pošilja zahtevo ponudniku spletne storitve, ki je na levi. Ponudnik mu nato vrne odgovor na zahtevo. [19]

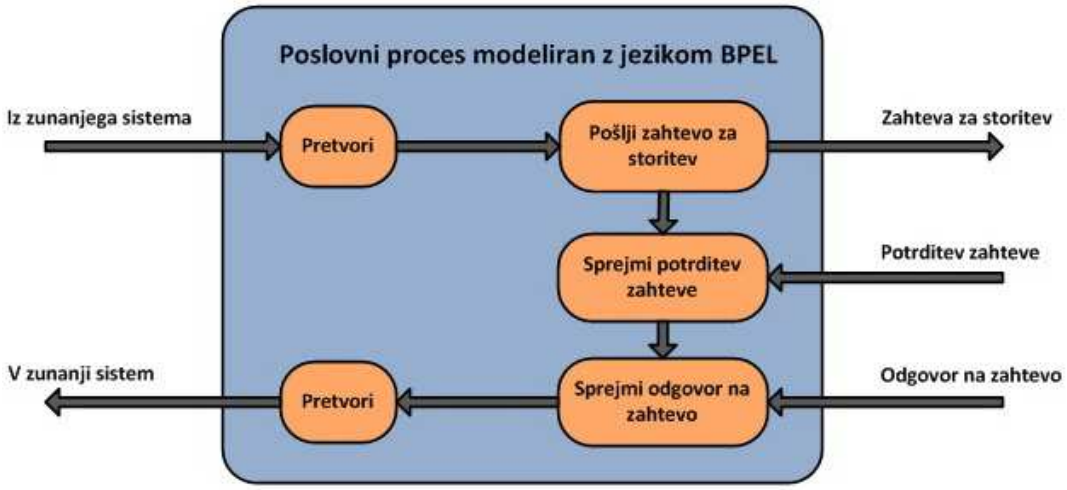
Slika 6: Osnovna arhitektura SOA

Ponudnik in porabnik storitve sta osnovni vlogi arhitekture SOA. Poleg njiju obstajajo še druge vloge, kot so npr. začetni pošiljatelj, končni prejemnik ter posrednik. Obstaja še vloga registra, ki skrbi za podajanje informacij o ponudnikih in njihovih storitev. Primer takšnega registra je UDDI.

Spletne storitve torej predstavljajo uvod v storitveno usmerjeno arhitekturo. Z njihovo pomočjo se doda nova integracijska plast v večnivojski arhitekturni model, ki lahko stoji znotraj ali pa med aplikacijami. Aplikacijska logika se ovije v ohlapno povezano storitev, kar pa je tudi osnovna ideja storitveno usmerjene arhitekture.

3.1.2 Orkestracija in koreografija

Integracija med spletnimi storitvami lahko poteka na dva načina. Pri prvem načinu, orkestraciji, osrednji proces prevzame kontrolo nad spletnimi storitvami in koordinira izvajanje operacij nad njimi. Spletne storitve, ki pri izvajanju sodelujejo, se ne zavedajo druga druge, niti da so del nekega višjega poslovnega procesa. To ve samo osrednji koordinator orkestracije. Orkestracija je centralizirana integracija spletnih storitev, pri kateri je natančno določen vrstni red klicanja spletnih storitev.



Slika 7: Primer orkestracije

Drugi način za izvedbo integracije je koreografija. Ta se po drugi strani ne zanaša na osrednjega koordinatorja, temveč mora vsaka spletna storitev, ki sodeluje v procesu, natančno vedeti, kdaj mora izvršiti določeno operacijo ter tudi s kom mora komunicirati. Koreografija je poskus sodelovanja, ki se osredotoča na izmenjevanje sporočil med spletnimi storitvami. Vsi sodelujoči v koreografiji morajo biti seznanjeni s poslovnim procesom, v katerega so vključeni, poznati morajo operacije, ki jih vršijo, sporočila, ki si jih izmenjujejo, in natančna časovna zaporedja izvedb interakcij.

Slika 8: Primer koreografije

Orkestracija ima pred koreografijo določene prednosti. Prva izmed njih je ta, da vedno natančno vemo, kdo je odgovoren za izvajanje celotnega poslovnega procesa. Nadalje lahko vedno dodamo nove spletne storitve, katerim se ni potrebno zavedati, da bodo del procesa. Prednost pa se pokaže tudi v primeru napake v procesu, saj lažje oblikujemo alternativno rešitev.

Koreografija je sicer podprta z nekaterimi standardnimi pristopi, ki jih določata npr. WSCI (Web Service Choreography Interface) in WS-CDL (Web Services Choreography Description Language), vendar pa zaenkrat še ni dobila prave industrijske podpore, ki bi ji omogočil večjo uporabo v praksi. Po drugi strani pa je orkestracija industrijsko zelo podprta, za njeno implementacijo namreč obstaja vrsta komercialnih orodij. V praksi je BPEL najbolj razširjena različica orkestracije. [1, 20]

3.1.3 Procesno usmerjen pristop

BPEL kot procesno usmerjen pristop k storitveno usmerjeni arhitekturi ponuja enostaven način integracije spletnih storitev v poslovne procese. Je jezik, ki temelji na XML in podpira veliko tehnologij, povezanih s spletnimi storitvami. Močno se naslanja na dokumente WSDL in sporočila SOAP, podpira pa še vrsto drugih standardov, kot so UDDI, WS-Transaction, WS-Security itn.

BPEL je naslednik in združitev jezikov WSFL (Web Services Flow Language) in XLANG. Prvega je načrtoval IBM in je temeljil na usmerjenih grafih. Pri drugem gre za jezik, ki se gradi s strukturnimi bloki in je last Microsoft-a. BPEL kombinira oba pristopa in tako prinaša bogato paleto orodij za gradnjo poslovnih procesov.

Poslovni proces, ki je modeliran z jezikom BPEL (v nadaljevanju proces BPEL), definira natančen vrstni red, v katerem se naj kličejo posamezne spletne storitve. Lahko se kličejo zaporedno ali vzporedno. Z jezikom BPEL lahko specificiramo pogojno obnašanje, tako da je npr. naslednji korak odvisen od odgovora prejšnje poklicane storitve. Poleg tega lahko gradimo zanke, definiramo spremenljivke, korelacijske množice itn. Konstruktov BPEL je več kot dovolj, da lahko definiramo kompleksen poslovni proces.

Prav tako kot spletne storitve je tudi BPEL proces lahko sinhroni ali asinhroni. Sinhroni proces blokira odjemalca, ki proces uporablja, dokler se ta ne zaključi in odjemalcu vrne rezultat. Asinhroni proces ne blokira odjemalca, temveč se za morebitno vrnitev rezultata poslužuje povratnega klica. Navadno se slednji uporablja pri procesih, ki trajajo dlje časa, sinhroni pa pri procesih, ki se izvedejo hitro.

Za svoje odjemalce proces BPEL izgleda tako kot katerakoli druga spletna storitev. Ko definiramo nov proces, v bistvu definiramo novo spletno storitev, ki je kompozicija obstoječih storitev. Vmesnik procesa BPEL, prav tako kot običajna spletna storitev, vsebuje množico končnih točk, preko katerih lahko dostopamo do njegovih operacij. Za dostop do procesa BPEL moramo poklicati njegovo začetno spletno storitev.

BPEL se lahko uporablja znotraj ali pa med organizacijami. Znotraj organizacije je njegova vloga standardizirati integracijo poslovnih aplikacij ter omogočiti integracijo tudi med sistemi, ki so bili poprej izolirani. Med organizacijami pa je njegova naloga vzpostaviti lažje in učinkovitejše povezovanje med poslovnimi partnerji. BPEL je glavna tehnologija v podjetjih, kjer je njihova funkcionalnost že ali pa še bo predstavljena preko spletnih storitev. [1]

3.2 Konstrukti BPEL

3.2.1 Aktivnosti

Vsak proces BPEL je sestavljen iz zaporedja korakov, ki se imenujejo aktivnosti. Le-te delimo na osnovne, ki se uporabljajo za opravljanje enostavnih nalog, ter na sestavljene, ki predstavljajo zaporedje večih osnovnih aktivnosti.

Osnovne aktivnosti:

- <*invoke*> se uporablja za klic partnerske spletne storitve. Klic storitve se izvede preko njene operacije, ki je definirana v pripadajočem dokumentu WSDL. Pokliče se lahko enosmerna operacija ali pa operacija, ki zahteva odgovor.
- <*receive*> čaka na partnersko storitev, da ji vrne odgovor na zahtevo. Med čakanjem na ustrezno sporočilo aktivnost blokira poslovni proces. Ta aktivnost lahko začne poslovni proces.
- <*reply*> omogoča poslovnemu procesu, da pošlje odgovor na zahtevo, ki jo je dobil preko aktivnosti <*receive*>. Uporablja se pri sinhronih operacijah tipa zahteva-odgovor. Nanaša se na isto operacijo partnerske storitve kot aktivnost <*receive*>, ki je sprožila proces.
- <*assign*> se uporablja za dodeljevanje novih vrednosti spremenljivkam. Aktivnost omogoča preslikovanje vrednosti iz ene v drugo spremenljivko ter sestavljanje in računanje različnih matematičnih in logičnih izrazov. Vsebuje lahko eno ali več elementarnih dodeljevanj in izrazov.
- <*throw*> se uporablja za signaliziranje napak, ki se zgodijo znotraj poslovnega procesa.
- <*wait*> blokira poslovni proces za določen čas ali do določenega trenutka.
- <*terminate*> ustavi celoten poslovni proces.

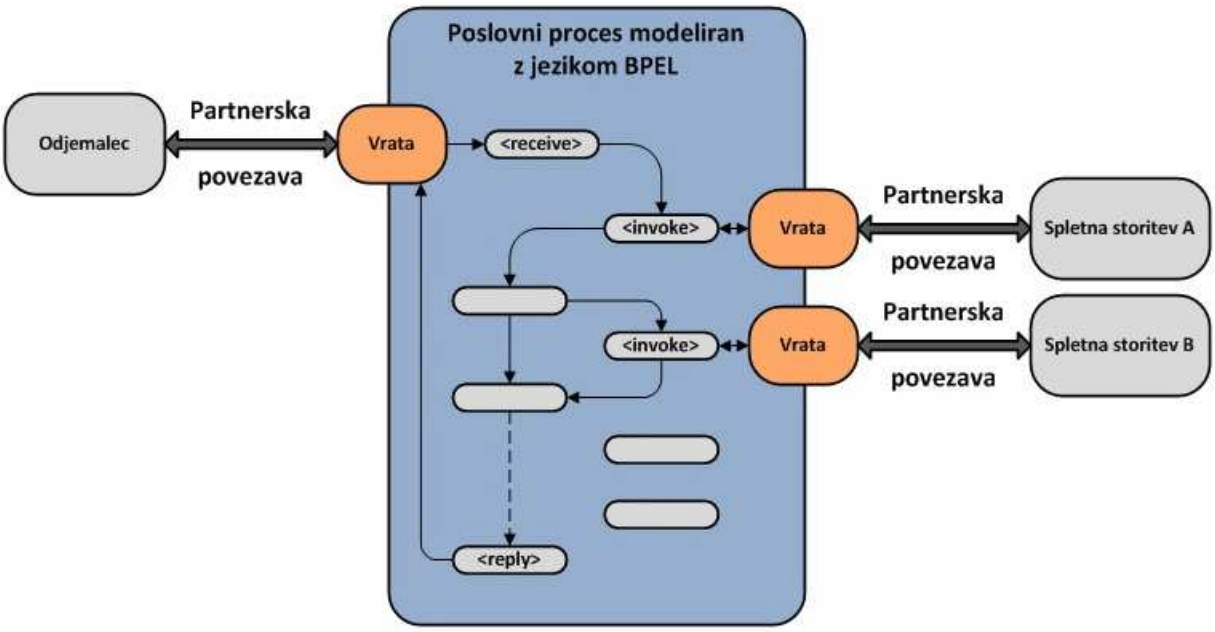
Sestavljene aktivnosti:

- <*sequence*> združuje množico enostavnih aktivnosti, ki se izvajajo po zaporednem vrstnem redu.
- <*flow*> omogoča sočasno izvajanje večih enostavnih aktivnosti. Definiramo lahko dve ali več vzporednih poti, ki se izvajajo vzporedno. Med posameznimi potmi lahko vzpostavimo povezave, ki skrbijo za sinhronizacijo vzporedno izvajajočih se poti.

- <*scope*> je množica enostavnih aktivnosti, ki lahko vsebujejo lastne spremenljivke, korelacijske množice itn. Te so lokalne narave, zato se izven aktivnosti ne vidijo in se ne morejo uporabiti.
- <*pick*> je aktivnost, ki omogoča izbiro med različnimi možnimi potmi v poslovnem procesu. Izbira je odvisna od vrste prejetega sporočila ali pa je pogojena z iztekom določene časovne periode.
- <*switch*> omogoča pogojno obnašanje pri problemih, kjer je malo možnih izbir, te pa so predvidljive.
- <*if*> omogoča pogojno obnašanje pri nepredvidljivih problemih, kjer je pogoj največkrat izražen z logičnim izrazom. Vsebuje lahko več <*else*> podvej.
- <*foreach*>, <*while*>, <*repeat*> so aktivnosti, ki označujejo, da se bo sklop enostavnih aktivnosti ponavljal dlje časa, največkrat do izpolnitve zadanega pogoja. [21]

3.2.2 Partnerske povezave

Proces BPEL z zunanjimi spletnimi storitvami komunicira na dva načina. Pri prvem načinu proces BPEL pokliče neko operacijo zunanje spletne storitve, medtem ko je pri drugem načinu obratno poklican sam proces. Zadnje se lahko zgodi s povratnim klicem spletne storitve ali pa če proces od spletne storitve dobi odgovor na poslano zahtevo. Ne glede na način, se komunikacija izvaja preko t.i. partnerskih povezav. Te označujejo povezavo med procesom BPEL in storitvami, s katerimi komunicira. Vsaka storitev ima v procesu BPEL definirano svojo partnersko povezavo. [1]



Slika 9: Partnerske povezave BPEL

Vsak proces BPEL mora imeti vsaj začetno partnersko povezavo, preko katerega odjemalec prvič pokliče. Navadno ima proces BPEL še vsaj eno partnersko povezavo, preko katere pokliče zunanjo spletno storitev. Glede na naravo procesa BPEL, katerega namen je prav

integracija spletnih storitev, je takih povezav pričakovati še več. Na zgornji sliki je prikazan primer procesa BPEL, ki poleg začetne partnerske povezave vključuje še dve, vsaka pripada svoji spletni storitvi.

Tip partnerske povezave je določen z odnosom med procesom BPEL in spletno storitvijo, s katero komunicira. Element, ki ta odnos ponazarja, se imenuje vloga in je definirana v dokumentu WSDL. Vloga je enojna, če se uporablja sinhrona komunikacija, ker se odgovor na zahtevo vrne z uporabo iste operacije. Če se uporablja asinhrona komunikacija, mora imeti povratni klic na razpolago svojo vlogo. [21]

3.2.3 Korelacijske spremenljivke

Korelacijske spremenljivke v proces BPEL dodajajo zmožnost ohranjanja stanja. Uporabljajo se pri asinhroni komunikaciji med procesom in njegovimi odjemalci. Ne zgodi se redko, da več instanc procesa BPEL čaka na povratne klice asinhronega odjemalca. Korelacijske spremenljivke v tem primeru poskrbijo, da se sporočilo povratnega klica nedvoumno prenese k pravemu primerku procesa BPEL– klicatelju.

Korelacijske spremenljivke so osnovane na podatkih iz sporočil, ki se prenašajo med odjemalcem in procesom BPEL. Pri sporočilu SOAP se lahko ti podatki nahajajo v glavi ali pa v telesu ovojnice sporočila. Deklaracija koleracijske spremenljivke vsebuje definicijo t.i. korelacijske lastnosti, ki procesu BPEL preko poizvedbe pove, kje natančno se spremenljivka v sporočilu nahaja. Medtem ko je korelacijska spremenljivka del dokumenta BPEL, se korelacijska lastnost vpiše v dokument WSDL spletne storitve, s katero si proces izmenjuje sporočila. [22]

3.3 BPEL in Java

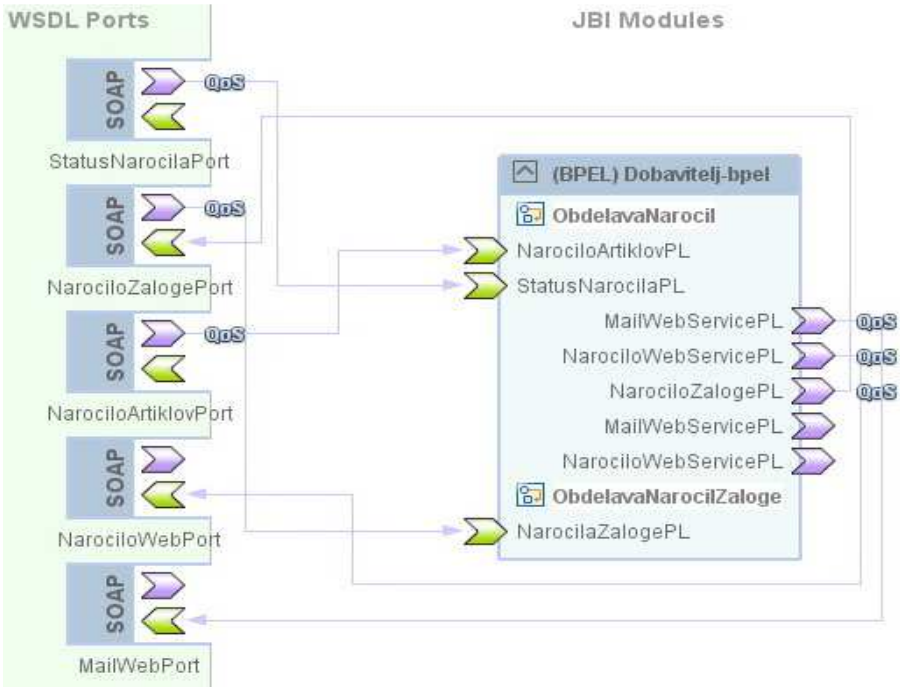
BPEL ni in ne poskuša biti splošno naravnan programski jezik. Je jezik, ki je zasnovan za preprosto in učinkovito opisovanje poslovnih procesov. BPEL torej ni nadomestek Jave, temveč se z njo dopolnjuje. Slednjo uporabljamo kot programski jezik za implementacijo spletnih storitev ter kot platformo, na kateri spletne storitve ter procesi BPEL tečejo.

3.3.1 JBI

Procesi BPEL se izvajajo v okviru komponent JBI (Java Business Integration). JBI je specifikacija, ki v splošnem definira pristop za implementacijo storitveno usmerjene arhitekture v platformi Java. Osnovana je na jeziku XML in tehnologiji spletnih storitev, glavni del arhitekture predstavlja vsebnik JBI. Ta deluje na principu priklapljanja različnih komponent, med katerimi se vzpostavi komunikacija preko izmenjave sporočil, ki tesno sledijo specifikacijam WSDL 2.0. [23]

Spodnja slika prikazuje primer vsebnika JBI, ki ga uporablja aplikacija Dobavitelj in teče na strežniku Java EE. Vanj je kot modul JBI naložen celoten proces BPEL, ki lahko tako komunicira še z morebitnimi drugimi moduli JBI. Iz slike je razvidna komunikacija med

spletnimi storitvami procesa BPEL, ki poteka prek sporočil, definiranih v pripadajočih dokumentih WSDL.



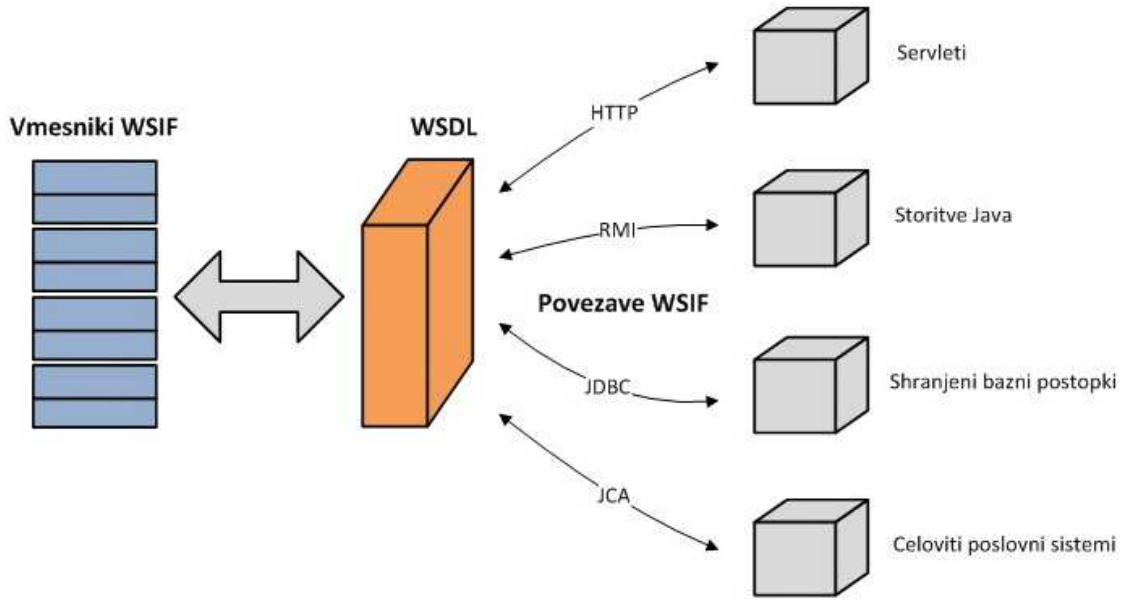
Slika 10: Vsebnik JBI

3.3.2 BPELJ

BPELJ (BPEL for Java) je kombinacija programskih jezikov BPEL in Java, ki omogoča njuno sodelovanje pri gradnji celovitih poslovnih aplikacij. To sodelovanje v splošnem prinaša zmožnost vključevanja programske kode Java v dokumente BPEL. Segmenti programske kode Java, ki se vstavljajo v definicije poslovnih procesov, se lahko uporabijo za različne stvari. Na eni strani ponujajo možnost, da kličemo razrede Java in ostale vire neposredno iz procesa BPEL. Na drugi strani pa izpopolnjujejo sam jezik BPEL in dajejo dodatno moč njegovim konstruktom. S segmenti programske kode Java lahko izboljšamo poslovna pravila, pogoje, zanke, dodatno pa lahko znotraj poslovnega procesa računamo različne izraze, kreiramo dokumente XML in v splošnem izvajamo kodo Java, brez da bi bilo za to potrebno klicati spletno storitev. BPELJ je uradno podprt s strani podjetij IBM in BEA. [1]

3.3.3 WSIF

WSIF (Web Services Invocation Framework) predstavlja drugačen pristop. Ideja je v uporabi jezika BPEL ne le za povezovanje spletnih storitev, temveč storitev in virov vseh vrst. WSIF uporablja sintakso BPEL in lahko pokliče vsako storitev, ki se lahko opiše z dokumentom WSDL, pa tudi če ta ne komunicira prek sporočil SOAP. Bistvo te tehnologije je ravno v opisovanju najrazličnejših virov Java z dokumenti WSDL. S tem pridobimo vmesnik, preko katerega se na enak način kliče tako spletna storitev SOAP, kot npr. razred EJB ali Java. Klic se izvede povsem neodvisno od načina implementacije, lokacije in načina dostopa do omenjenega vira.



Slika 11: Pristop WSIF

Razmejitev med programskim vmesnikom in dejanskimi komunikacijskimi protokoli pomeni dodatno prilagodljivost. Spreminjamo lahko protokole, premikamo storitve na druge lokacije, vse brez potrebnega posega v odjemalčevo programsko kodo. Vse, kar je potrebno storiti, je spremeniti opis storitve v njenem dokumentu WSDL.

Z uporabo pristopa WSIF postane dokument WSDL središče integracijskega modela za dostopanje do storitev, ki tečejo na različnih platformah in uporabljajo različne komunikacijske protokole. Edini pogoj je opis storitve z dokumentom WSDL, ki mora obsegati definicijo uporabljenega načina povezovanja. Tega mora prepoznati tudi odjemalčev model WSIF, kar pomeni, da mora imeti zanj ustreznega ponudnika. Ogrodje WSIF v osnovni različici obsega ponudnike za lokalne protokole Java, EJB, JMS in JCA. [24, 1]

4 Java EE 5 v praksi

4.1 Problemska domena

4.1.1 Trgovec d.o.o.

Trgovec d.o.o. je klasično trgovsko podjetje, ki strankam ponuja različne izdelke. Njegovo delovanje sloni na spletni prodajalni, kar mu omogoča konkurenčne cene. Čeprav ima že sedaj v ponudbi najrazličnejše izdelke, pa so se vodilni v podjetju odločili dodati še enega. Na prodajalne police svoje spletne trgovine bodo uvrstili paleto filmov na medijih DVD. Filme jim bo dobavljalo podjetje Dobavitelj d.o.o., ki predstavlja najboljši odgovor na njihove zahteve.

V pričujočem delu bom uporabil omenjeno podjetje za predstavitev nekaterih tehnologij Java EE 5. Aplikacija Trgovec bo za zagotavljanje trajnosti podatkov uporabljala vmesnik Java Persistence API v navezi s podatkovno bazo MySQL. Preko preprostega spletnega uporabniškega vmesnika bom predstavil uporabnost nove tehnologije Java Server Faces, nazadnje pa bom dodal še odjemalca za komuniciranje s spletno storitvijo podjetja Dobavitelj d.o.o.

4.1.2 Dobavitelj d.o.o.

Podjetje Dobavitelj d.o.o. se poleg ostalih dejavnosti ukvarja tudi z uvažanjem filmov na medijih DVD. S tem artiklom nato oskrbuje druga podjetja in trgovine. Del samega poslovnega procesa je kreiranje zapisov XML, ki vsebujejo različne podatke o artiklih. Te zapise lahko podjetje nato nudi svojim strankam, do katerih lahko dostopajo prek spletne storitve in jih tako uvozijo v svoje aplikacije.

Aplikacija Dobavitelj bo podobno kot Trgovec osnovana na podatkovni bazi MySQL, za trajnost podatkov bo poskrbel Java Persistence API. Izvoz datotek XML bo ponazarjal primer uporabe tehnologije JAXB, vse skupaj pa bo ovito v komponente Enterprise JavaBeans. Prehod iz aplikacijske logike EJB k spletni storitvi bo prikazal prednosti nove tehnologije JAX-WS 2.0, aplikacija Dobavitelj pa bo vključevala tudi proces BPEL, ki bo koordiniral delovanje nekaterih spletnih storitev.

4.2 Podpora poslovnim procesom

4.2.1 JPA

Obe podjetji se za zagotavljanje trajnosti svojih podatkov poslužujeta vmesnika Java Persistence API. Slednji ima, kot je že omenjeno, standardizirano objektno-relacijsko preslikavo, zato je nastavljanje potrebnih atributov glede podatkovne baze povsem enostavno. Vse podatke o potrebnih nastavitvah vsebuje deskriptor XML, imenovan *persistence unit* ali PU.

PU določa ponudnika, t.i. *persistence provider*, ki nudi ogrodje za objektno-relacijsko preslikovanje, ki ga za delovanje potrebuje JPA. Teh ponudnikov ni veliko, komercialni je npr.

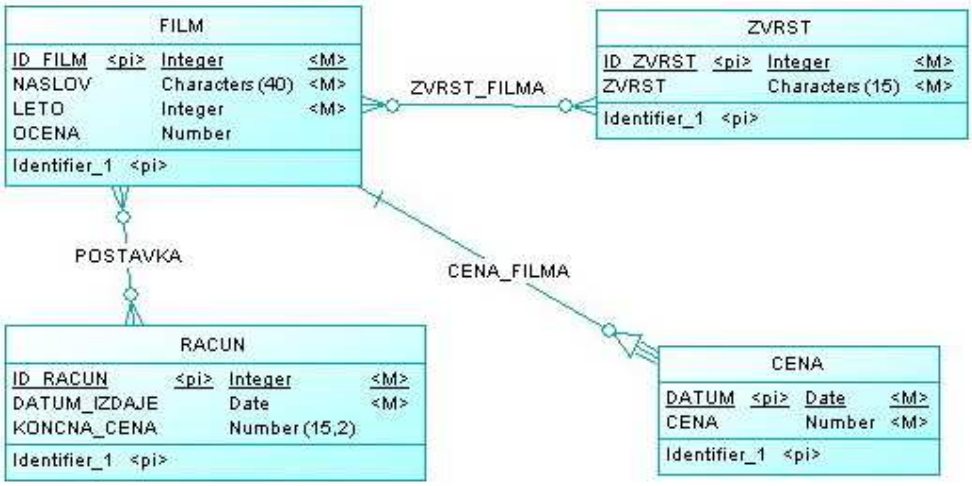
Oracle TopLink, na voljo pa so tudi odprtokodne verzije. Med njimi sta tudi TopLink Essentials in pa novejši EclipseLink, oba izvedena iz komercialne verzije Oracle.

Poleg tega v deskriptorju PU določimo ime podatkovne shrambe, do katere naj dostopa, in ali naj JPA uporablja lastno tehnologijo za upravljanje transakcij, Java Transaction API. Na koncu določimo še množico vseh entitetnih razredov, ki jih prepustimo upravljanju entitetnemu upravljalcu. Ta množica predstavlja podatke, ki so vsebovani znotraj podatkovne shrambe.

Spodaj je podan primer deskriptorja PU, ki ga uporablja Trgovec d.o.o. JPA v tem primeru uporablja Java Transaction API in odprtokodnega ponudnika EclipseLink, ime podatkovne shrambe je jdbc/trgovec, ta implicitno določa dostop do podatkovne baze MySQL, entitetni upravljalec pa razpolaga z vsemi entitetnimi razredi, ki so deklarirani v projektu.

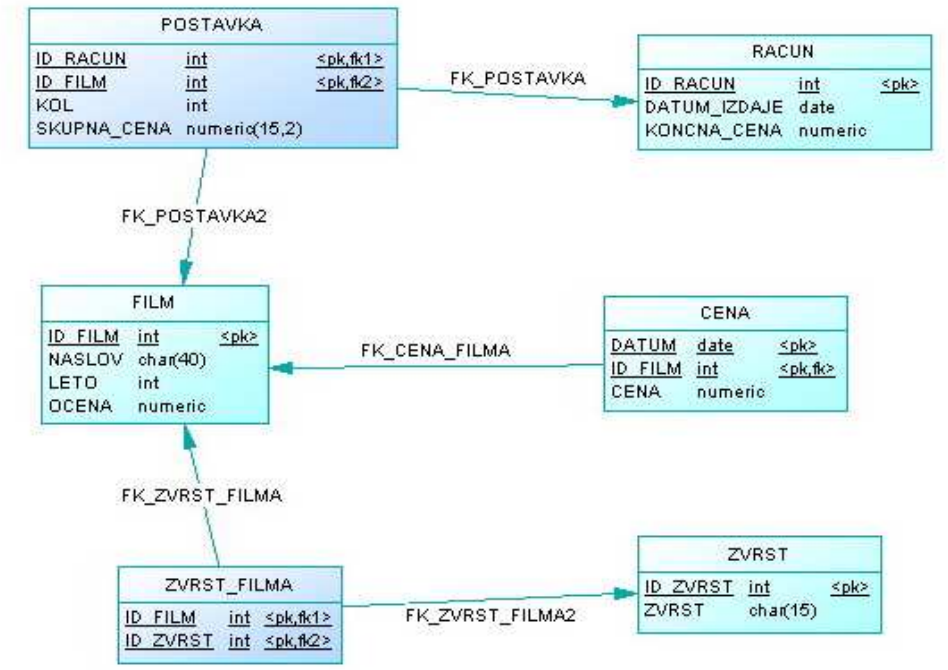
```
<persistence-unit name="Trgovec-ejbPU" transaction-type="JTA">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <jta-data-source>jdbc/trgovec</jta-data-source>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <properties>
  </properties>
</persistence-unit>
```

Vmesniki za dostop do relacijskih podatkov so entitetni razredi, ki predstavljajo preslikavo tabel iz relacijske podatkovne baze. Posamezni objekti teh razredov pa predstavljajo vrstice v tabeli. Preslikava med tabelami in razredi lahko poteka na dva načina. Pri prvem je potrebno najprej napisati razrede, nakar JPA sam poskrbi za preslikavo v ustrezne tabele v podatkovni bazi. Drugi način, ki ga npr. omogoča razvojno okolje NetBeans, pa je, da iz že obstoječih tabel zgeneriramo ogrodje ustreznih entitetnih razredov. Slednji je uporabljen pri aplikaciji Dobavitelj, prvotne tabele so bile generirane preko entitetno-relacijskega diagrama. Spodaj je podan konceptualni model ER diagrama.



Slika 12: Konceptualni model ER diagrama aplikacije Dobavitelj

Vrednosti atributov v vseh entitetah so atomarne, kar pomeni, da ne vsebujejo ponavljajočih skupin atributov. Vsi atributi v posameznih entitetah so odvisni od celotnega primarnega ključa, delne funkcionalne odvisnosti atributov ne obstajajo. Prav tako v posameznih entitetah ne obstajajo tranzitivne odvisnosti od ključa, vsi atributi so odvisni le od primarnega ključa. Vsi primarni ključi so izbrani tako, da enolično določajo vse atribute posamezne entitete, ki je tako v celoti določena s celotnim primarnim ključem. S tem so vse podatkovne strukture v normalizirani obliki in pripravljene na uporabo v relacijski podatkovni bazi. Z normalizacijo zagotovimo, da kasneje pri dodajanju, brisanju in spreminjanju zapisov fizične podatkovne baze ne prihaja do anomalij. Logični model ER diagrama, ki prikazuje zadnji nivo pred kreiranjem fizične podatkovne baze in upošteva vsa pravila normalizacije, je podan spodaj.



Slika 13: Logični model ER diagrama aplikacije Dobavitelj

Podlago za entitetni razred predstavlja klasični razred, ta pa je razširjen z določenimi notacijami. Vsak entitetni razred mora vsebovati notaciji *@Entity* in *@Id*. Prva pove, da gre za entitetni razred in se nahaja pred deklaracijo razreda, druga pa določa primarni ključ entitete in se nahaja neposredno pred deklaracijo ustrezne spremenljivke. Notacijo *@Id* v primeru sestavljenega primarnega ključa zamenja *@EmbeddedId.* Čeprav imajo ostale nastavitve določene privzete vrednosti, jih je velikokrat potrebno nastaviti ročno. Z notacijo *@Table* tako določimo katalog in shemo podatkovne shrambe ter ime tabele, če se ta razlikuje od imena razreda. Podobno lahko z *@Column* določimo tudi imena posameznih stolpcev tabele, nastavimo pa lahko še kopico drugih podrobnosti, ki zadevajo posamezen stolpec, kot npr., ali je polje lahko prazno ali se vanj lahko dodaja vrednosti itn.

4.2.1.1 Entitetni upravljalec

Preko entitetnega upravljalca izvajamo vse dostope do fizične podatkovne baze. Njegove naloge so kreiranje in brisanje entitet ter izvajanje poizvedb nad njimi. Omogoča nam tudi iskanje določene entitete s pomočjo primarnega ključa.

Na voljo sta dva tipa entitetnega upravljalca. Prvi tip uporabljamo v aplikacijah Java EE, zanj namreč skrbi strežnik Java EE oz. pripadajoči vsebnik. Ta poskrbi, da je entitetni upravljalec na voljo vsem komponentam aplikacije znotraj ene same transakcije JTA. Do njega preprosto dostopamo kjerkoli znotraj aplikacije preko notacije *@PersistenceContext.*

```
@PersistenceContext
private EntityManager em;
```

Za kreiranje in delovanje entitetnega upravljalca drugega tipa skrbi sama aplikacija in posledično programer. V prvi vrsti moramo poskrbeti, da se vsi dostopi do fizične podatkovne baze vršijo v okviru transakcij. Če je potrebno izvajati dostope iz več komponent znotraj ene same transakcije, moramo posamezno instanco entitetnega upravljalca iz ene komponente po referenci podati drugi komponenti. Dobra lastnost tega tipa pa je, da ga lahko uporabljamo tudi v klasičnih aplikacijah Java SE. [9]

```
@PersistenceUnit
EntityManagerFactory emf;
EntityManager em = emf.createEntityManager();
```

Tako aplikacija Trgovec kot tudi Dobavitelj uporablja prvo različico entitetnega upravljalca. Spodaj je prikazan tipičen primer shranjevanja novega računa v bazo podatkov. Ker gre za prvi tip, je uporaba preprosta, potrebno je samo generirati nov primerek entitetnega razreda Racun ter ga kot parameter posredovati metodi *persist* entitetnega upravljalca:

```
Racun racun = new Racun(new Date());
em.persist(racun);
```

4.2.1.2 Imenske poizvedbe

V entitetne razrede pišemo tudi statične imenske poizvedbe, ki se kasneje izvajajo preko entitetnega upravljalca. Notacija je *@NamedQuery*, njena parametra pa sta ime poizvedbe, na katerega se sklicujemo pri izvajanju poizvedbe, ter sintaksa same poizvedbe.

```
@NamedQuery(name = "Zvrst.findByZvrst",
            query = "SELECT z FROM Zvrst z WHERE z.zvrst = :zvrst")
```

Zgornji primer imenske poizvedbe uporablja sintakso EJB-ju lastnega poizvedovalnega jezika, opazimo pa lahko tudi, da uporablja imenski parameter. Tega v poizvedbi označimo z dvopičjem, ki mu sledi unikatno poimenovanje samega parametra, na katerega se kasneje sklicujemo.

Imenske poizvedbe prožimo preko ustrezne metode entitetnega upravljalca, kateri kot atribut podamo ime poizvedbe ter imena in vrednosti morebitnih parametrov. Spodnji primer s pomočjo entitetnega upravljalca *em* uporablja navedeno imensko poizvedbo.

```
Zvrst zvrst = (Zvrst) em.createNamedQuery("Zvrst.findByZvrst").
        setParameter("zvrst", "Comedy").getSingleResult();
```

4.2.1.3 Sestavljen primarni ključ

Primarni ključ entitete je lahko enostaven ali sestavljen. V primeru sestavljenega moramo definirati poseben razred, ki združuje vse atribute primarnega ključa. Objekt tega razreda tako predstavlja primarni ključ entitete, ki ga potrebujemo v entitetnem razredu.

Entitetni tip Cena aplikacije Dobavitelj uporablja primarni ključ, ki je sestavljen iz datuma in primarnega ključa entitetnega tipa Film. Razred CenaPK, ki definira ta sestavljeni ključ, je predstavljen spodaj:

```
@Embeddable
public class CenaPK implements Serializable {

    @Column(name = "DATUM", nullable = false)
    @Temporal(TemporalType.DATE)
    private Date datum;

    @Column(name = "ID_FILM", nullable = false)
    private int idFilm;

    public CenaPK() {
    }

    public CenaPK(Date datum, int idFilm) {
        this.datum = datum;
        this.idFilm = idFilm;
    }
}
```

Vidimo, da je namesto notacije, ki označuje entitetni razred, uporabljena notacija *@Embeddable*. Ta označuje razred, čigar objekti se bodo uporabili v prvotnem entitetnem razredu kot primarni ključ. Razen te razlike razreda delujeta enako in uporabljata iste notacije, kot so *@Column, @Temporal, @Basic* itd. Spodaj je prikazan del entitetnega razreda Cena, ki uporablja razred CenaPK za generiranje primarnega ključa entitete. Spremenljivka, ki označuje primarni ključ, je namesto z *@Id* označena z notacijo *@EmbeddedId.*

```
@EmbeddedId
protected CenaPK cenaPK;

public Cena(Date datum, int idFilm, double cena) {
    this.cenaPK = new CenaPK(datum, idFilm);
    this.cena = cena;
}
```

4.2.1.4 Avtomatsko generiranje primarnih ključev

Vmesnik Java Persistence API ponuja možnost uporabe avtomatskega generiranja primarnih ključev. To pomeni, da entitetam ni potrebno dodeljevati primarnih ključev znotraj aplikacije, temveč se to opravilo prepusti JPA. Notacija *@GeneratedValue* v entitetnem razredu pove, da se bodo uporabljali generirani primarni ključi.

Eden od parametrov omenjene notacije je *GenerationType*, ki določa način generiranja primarnih ključev. Na voljo je več načinov, vsi pa so odvisni od mehanizmov fizične podatkovne baze. Entitetni razred Racun npr. uporablja avtomatsko generiranje primarnih ključev s pomočjo posebne tabele v podatkovni bazi. Ta vsebuje dva stolpca, prvi določa primarni ključ, za katerega generiramo vrednosti, drugi pa samo generirano vrednost.

GEN_KEY	GEN_VALUE
idFilm	100024
idRacun	13
idZvrst	2009

Tabela 2: Tabela za generiranje primarnih ključev *ID_GEN*

V entitetni razred je poleg *@GeneratedValue* potrebno dodati še notacijo *@TableGenerator*, ki opredeljuje omenjeno tabelo. Kot parametre ji podamo ime tabele, imena stolpcev, ki opredeljujeta primarni ključ ter naziv ključa. Opcijsko lahko definiramo tudi začetno vrednost ter stopnjo povečevanja primarnega ključa. Spodaj je primer za primarni ključ idRacun.

```
@Id
@TableGenerator(name = "racunGen", table = "ID_GEN", pkColumnName = "GEN_KEY",
valueColumnName = "GEN_VALUE", pkColumnValue = "idRacun", allocationSize = 1)
@GeneratedValue(strategy = GenerationType.TABLE, generator = "racunGen")
@Column(name = "ID_RACUN", nullable = false)
private Integer idRacun;
```

4.2.1.5 Povezava 1:m

Iz ER diagrama aplikacije Dobavitelj vidimo, da lahko podjetje hrani zgodovino cen za posamezni film. To mu omogoča entitetni tip Cena, čigar primarni ključ je sestavljen iz datuma, ko določena cena stopi v veljavo, in iz primarnega ključa idFilm, ki je v tem primeru tudi tuji ključ. Z drugimi besedami to pomeni, da ima lahko en film določenih več cen.

Za realizacijo preko entitetnih razredov to pomeni, da moramo v razred Film dodati spremenljivko, preko katere bomo dostopali do seznama cen določenega filma. Spremenljivka mora biti označena z notacijo *@OneToMany.* Njeni parametri med drugim določajo ime spremenljivke v entitetnem razredu Cena, s katero je povezana, ter obliko kaskadnih operacij. Drugi konec povezave, torej entitetni tip, ki vsebuje tuji ključ, je lahko določen preko atributa notacije ali pa preko samega tipa spremenljivke, kot je prikazano spodaj.

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "film")
private List<Cena> cenaCollection;
```

Na drugem koncu povezave entitetni razred Cena vsebuje že omenjeno spremenljivko film, ki je opremljena z obratno notacijo *@ManyToOne.* Poleg nje stoji še notacija *@JoinColumn*, ki opredeljuje atribute, povezane s tujim ključem. To je med drugim ime stolpca, ki vsebuje tuji ključ, in ime stolpca v entitetnem tipu Film, na katerega se tuji ključ nanaša.

```
@JoinColumn(name = "ID_FILM", referencedColumnName = "ID_FILM",
            nullable = false, insertable = false, updatable = false)
@ManyToOne(optional = false)
private Film film;
```

4.2.1.6 Povezava m:n

Povezava m:n asociira na vmesno tabelo, ki vsebuje primarna ključa obeh povezanih tabel. Java Persistence API omogoča, da nam za to tabelo ni potrebno definirati entitetnega razreda. Vsi dostopi do tabel na obeh koncih povezave se izvajajo neposredno, prek spremenljivk, označenih z notacijo *@ManyToMany.*

Aplikacija Dobavitelj uporablja takšno povezavo med tabelama Film in Zvrst. Film ima lahko več zvrsti, ena zvrst pa je lahko dodeljena več filmom. Spodnji izsek je iz entitetnega razreda Film:

```
@ManyToMany(mappedBy = "filmCollection")
private List<Zvrst> zvrstCollection;
```

Entitetni razred Zvrst na drugi strani povezave pa poleg omenjene notacije vsebuje še *@JoinTable*, ki opredeljuje parametre, povezane z vmesno tabelo. S parametri je določeno ime vmesne tabele ter oba potrebna tuja ključa.

```
@JoinTable(name = "zvrst_filma",
    joinColumns = {@JoinColumn(name = "ID_ZVRST",
           referencedColumnName = "ID_ZVRST", nullable = false)},
    inverseJoinColumns = {@JoinColumn(name = "ID_FILM",
           referencedColumnName = "ID_FILM", nullable = false)})
@ManyToMany
private List<Film> filmCollection;
```

Do seznama zvrsti določenega filma dostopamo prek spremenljivke zvrstCollection entitetnega razreda Film. Obratno, do seznama filmov, ki jih opredeljuje določena zvrst, dostopamo preko spremenljivke filmCollection entitetnega razreda Zvrst. Primer dodajanja zvrsti določenemu filmu, kjer se implicitno dodajajo zapisi v vmesno tabelo, kaže spodnji izsek programske kode:

```
Film film = vrniFilm(idFilm);
Zvrst zvrst = vrniZvrst(idZvrst);
film.addZvrst(zvrst);
zvrst.addFilm(film);
```

4.2.2 EJB 3.0

Dobra lastnost komponent EJB je njihova prenosljivost in zmožnost ponovne uporabe. Komponento, ki vsebuje neko poslovno logiko, lahko poganjajo različni strežniki Java EE, uporabljajo pa jo lahko različni aplikacijski odjemalci. V nekem odjemalcu tako ni potrebno skrbeti za implementacijo raznih poslovnih pravil, temveč se namesto tega le pokliče ustrezna komponenta EJB. To lastnost izkorišča tudi modul EJB aplikacije Dobavitelj, ki vsebuje več komponent, vse pa vsebujejo določene implementacije poslovne logike in pravil.

Tipičen primer uporabe komponente EJB je dostopanje do podatkovne baze in upravljanje z njenimi podatki. Aplikacija Dobavitelj v ta namen uporablja komponento *DatabaseBean.* Ta preko že omenjene tehnologije JPA dostopa do podatkovne baze MySQL in implementira vsa potrebna poslovna pravila za pravilno upravljanje s podatki. Za pravilno izvajanje transakcij skrbi strežnik Java EE.

Naslednja komponenta EJB, ki je del aplikacije Dobavitelj, se imenuje *JAXBBean*. Ta vsebuje poslovno logiko, ki njenim odjemalcem omogoča izvoz podatkov o artiklih v obliki dokumentov XML. Nazadnje pa so tu še tri komponente EJB, ki navzven nudijo specifične storitve, kot so pregled in dodajanje zaloge, pošiljanje e-pošte, naročilo artiklov z izdelavo računov in predračunov ipd. Ti so po strukturi aplikacije višjeležeče, saj v implementaciji programske kode uporabljajo spodnjo komponento EJB za dostop do podatkovne baze, navzven pa nudijo povsem definirane storitve.

Vsi našteti razredi, ki uporabljajo mehanizme EJB, so označeni z *@Stateless* notacijo. Ta pove, da je razred v bistvu komponenta tipa sejno zrno, poleg tega pa ga označi še z nekaterimi drugimi lastnostmi. Ena izmed njih je ta, da zrno ne ohranja stanja med različnimi klici odjemalcev, druga pa npr. ta, da lahko v nasprotju z ostalimi tipi zrn, razred zlahka preobrazimo v spletno storitev. Ta lastnost se bo kasneje obrestovala pri zgoraj omenjenih komponentah, ki ponujajo končne storitve.

Do metod in s tem poslovne logike komponent omenjenega tipa odjemalci lahko dostopajo le prek vmesnika. Le-ta definira, kako odjemalec vidi določen zrno, njegove metode ter zahtevane vhodne in izhodne argumente. Na ta način vmesnik ščiti odjemalca pred kompleksnostjo komponente EJB, po drugi strani pa omogoča enostavno spreminjanje programske kode, ne da bi to vplivalo na odjemalca. Vmesniki se delijo na lokalne in oddaljene. Komponenta, ki implementira lokalni vmesnik, je dosegljiva le odjemalcem na istem JVM (Java Virtual Machine), medtem, ko pri drugem tipu to ni omejitev. Posledično je oddaljen vmesnik dosti bolj prilagodljiv različnim odjemalcem, to so lahko spletne komponente, aplikacijski odjemalci ali pa druge komponente EJB. Poleg tega pa je za

odjemalca lokacija komponente EJB povsem transparentna. Tip vmesnika označimo z notacijama *@Local* oz. *@Remote.* Vse komponente aplikacije Dobavitelj uporabljajo slednji tip vmesnika. Dostop do njih je enak ne glede na tip odjemalca, izvede pa se preko notacije *@EJB*, v kar priča spodnji izsek programske kode:

```
@EJB
private static DatabaseRemote database;
@EJB
private static JAXBRemote jaxb;
```

4.2.3 JAXB 2.0

Tehnologija JAXB 2.0 je tesno povezana s spletnimi storitvami JAX-WS 2.0. Uporablja se pri kreiranju spletnih storitev iz poslovnih razredov, kjer se samodejno zgenerira shema XML, ki jo potrebuje dokument WSDL spletne storitve. Uporablja se tudi pri kreiranju odjemalcev spletnih storitev, pri čemer se uporabi obraten pristop, iz podane sheme XML spletne storitve se generirajo ustrezni razredi Java. Tehnologija pa je po drugi strani tudi povsem na voljo programerjem, da jo uporabljajo v lastne namene. Aplikacija Dobavitelj tako npr. uporablja JAXB 2.0 za izvažanje podatkov o filmih v obliki datotek XML.

Najprej je potrebno določiti obliko in sestavo bodočih zapisov XML. To najlažje storimo s kreiranjem sheme XML. V njej definiramo vse elemente, ki se bodo nahajali v izvoženih datotekah, ter njihove tipe. Spodnja shema vsebuje en kompleksen element, ki predstavlja en zapis XML, vsebuje pa vse podatke poljubnega artikla, v tem primeru filma.

```
<xs:complexType name="filmResponse">
  <xs:sequence>
    <xs:element name="id" type="xs:int"/>
    <xs:element name="naslov" type="xs:string" minOccurs="0"/>
    <xs:element name="leto" type="xs:int"/>
    <xs:element name="ocena" type="xs:double"/>
    <xs:element name="zvrst" minOccurs="0" maxOccurs="unbounded">
      <xs:simpleType>
        <xs:list itemType="xs:string"/>
      </xs:simpleType>
    </xs:element>
    <xs:element name="trenutnaCena" type="xs:double"/>
  </xs:sequence>
</xs:complexType>
```

Nadalje je potrebno preslikati shemo v programske razrede, kjer nam pride prav prevajalnik sheme, ki je del arhitekture JAXB. Po končanem postopku imamo na voljo po en razred Java za vsak v shemi definiran kompleksen element, v tem primeru je to en sam razred. Ta vsebuje klasične deklaracije spremenljivk in metod, ki ustrezajo elementom sheme. Poleg tega vsebuje tudi nekatere notacije, ki dajo razredu dodatno vrednost. Pomembna je notacija *@XmlRootElement*, ki se nahaja pred deklaracijo razreda in pove, da se bodo objekti tega razreda uporabljali kot elementi v dokumentih XML. Preko notacij lahko vplivamo tudi na obliko predstavitve zapisov, nastavimo lahko npr. vrstni red elementov ali pa način

predstavitve elementov, ki vsebujejo več vrednosti, torej seznamov. Prva se nastavi preko parametra v notaciji *@XmlType*, druga pa z notacijo *@XmlList*, ki pomeni, da se bodo vrednosti seznama izpisovale v vrsti, namesto vsaka v svoji vrstici. Spodaj je prikazan začetni del razreda, ki ga uporablja aplikacija Dobavitelj.

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "filmResponse", propOrder = {
    "id","naslov","leto","ocena","zvrst","trenutnaCena"})
public class FilmResponse {
    protected int id;
    protected String naslov;
    protected int leto;
    protected double ocena;
    @XmlList
    protected List<String> zvrst;
    protected double trenutnaCena;
```

Ko je razred pripravljen, lahko s pomočjo operacije vmesnika API JAXB tehnologije izvažamo objekte razreda kot elemente v dokumente XML. Operacija se imenuje *marshall*, do nje pa dostopamo preko abstraktnega razreda *JAXBContext*, ki deluje kot vstopna točka do programskega vmesnika API.

```
File output = new File("/root/Diploma/XMLDoc/film_" + idFilm + ".xml");
try {
    JAXBContext jaxbCtx = JAXBContext.
            newInstance(filmSchema.getClass().getPackage().getName());
    Marshaller marshaller = jaxbCtx.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    marshaller.marshal(filmSchema, output);
} catch (Exception e) {
    e.printStackTrace();
    throw new DatabaseException(e.getMessage());
}
```

Zgornji izsek programske kode za vsak objekt *filmSchema* kreira datoteko XML z lastnostmi, ki odgovarjajo začetni shemi XML in pripadajočemu razredu vključno z morebitnimi naknadno uporabljenimi notacijami. Končni dokument XML pa izgleda takole:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:filmResponse xmlns:ns2="http://service.dobavitelj/">
    <id>1001</id>
    <naslov>Shrek</naslov>
    <leto>2003</leto>
    <ocena>7.8</ocena>
    <zvrst>Comedy, Animation</zvrst>
    <trenutnaCena>10.99</trenutnaCena>
</ns2:filmResponse>
```

4.3 Usmerjenost k strankam

4.3.1 JSF 1.2

Eden izmed možnih načinov interakcije s strankami je preko spletnih aplikacij. Tehnologija JavaServer Faces omogoča razvoj uporabniških vmesnikov, ki vsebujejo vse standardne komponente, kot so vnosna polja, tabele, gumbi itd. Aplikacija Dobavitelj uporablja tak uporabniški vmesnik za dodajanje in urejanje zapisov o artiklih ter preko potrjevanja novih zalog za komunikacijo s procesom BPEL. Vmesnik je preprost, njegov namen pa je predstaviti nekaj osnovnih komponent in prikazati uporabnost tehnologije, ki je z različico 1.2 prvič dodana k platformi Java EE 5.

Slika 14: Uporabniški vmesnik JSF aplikacije Dobavitelj

Spletni uporabniški vmesnik aplikacije Dobavitelj poleg klasičnih komponent, kot so gumbi in vnosna polja, vključuje še nekatere druge, bolj specifično usmerjene komponente. Uporabnik preko prvega vnosnega polja vpiše identifikacijsko številko filma, katerega želi urejati. Nato lahko preko ostalih polj poljubno spreminja podatke trenutnega filma. S pritiskom na gumb se spremembe uveljavijo v podatkovni bazi. V primeru, da uporabnik želi dodati nov film, je postopek enak. Uporabniški vmesnik sam zazna, kdaj gre za urejanje podatkov že obstoječega filma in kdaj za dodajanje novega. Uporabnik lahko poleg tega še dodaja nove zvrsti ter ureja cenike, ki vstopijo v veljavo na določen dan. Vsa funkcionalnost glede urejanja in dodajanja zapisov v podatkovno bazo je opravljena preko obstoječih razredov EJB, povezava z procesom BPEL pa se vzpostavi preko spletne storitve.

```
@EJB
private DatabaseRemote databaseBean;

@WebServiceRef(wsdlLocation =
               "WEB-INF/wsdl/client/NarociloZaloge/NarociloZaloge.wsdl")
private NarociloZalogeService narociloZalogeService;
```

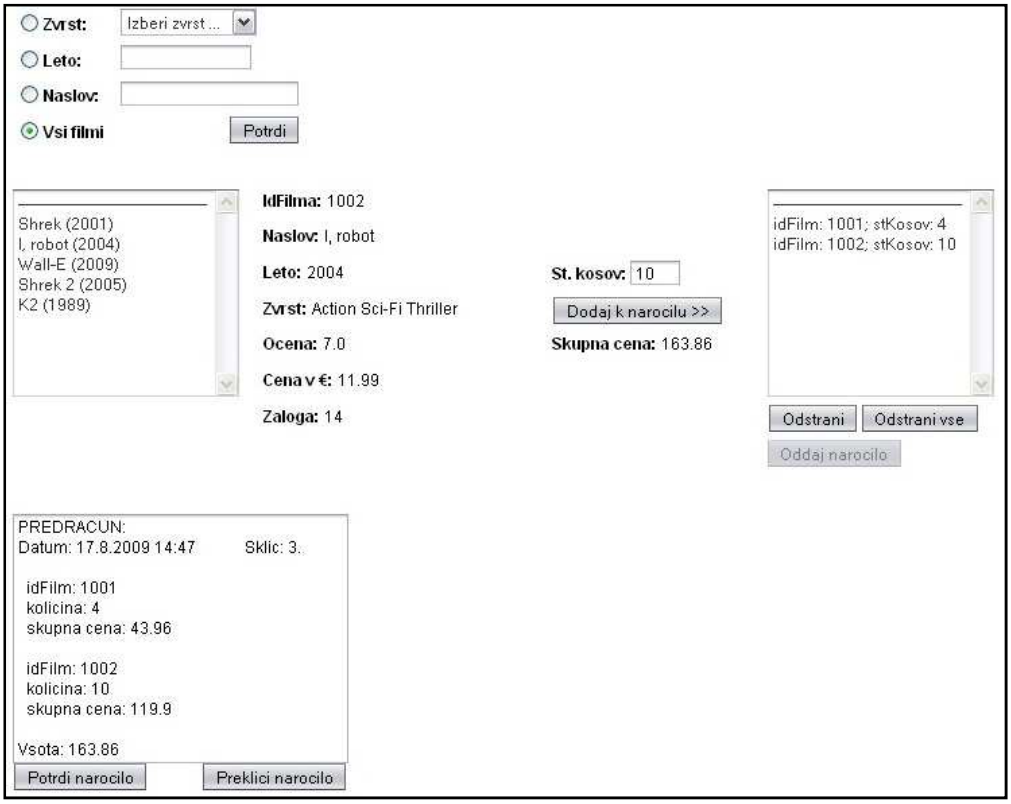
Spletni uporabniški vmesnik JSF uporablja tudi aplikacija Trgovec. Uporablja ga za komuniciranje s spletnimi storitvami, ki jih ponuja Dobavitelj. Te storitve zadevajo pridobivanje informacij o filmih, izvajanje ter potrjevanje oz. preklicevanje naročil.

```
@WebServiceRef(wsdlLocation =
               "WEB-INF/wsdl/client/StatusNarocila/StatusNarocila.wsdl")
private StatusNarocilaService statusService;

@WebServiceRef(wsdlLocation =
               "WEB-INF/wsdl/client/NarociloArtiklov/NarociloArtiklov.wsdl")
private NarociloArtiklovService narociloService;

@WebServiceRef(wsdlLocation =
   "WEB-INF/wsdl/client/FilmWeb/localhost_8080/FilmWebService/FilmWeb.wsdl")
private FilmWebService filmService;
```

Preko uporabniškega vmesnika uporabnik pregleduje seznam filmov, nakar izbere enega ali več in jih doda k naročilu. Izbira lahko med prikazom po zvrsteh, letnici ali pa vpiše željeni naslov filma. Vmesnik sproti naredi izračun končne cene glede na ceno posameznega filma in število kosov, ki jih je uporabnik dodal naročilu. Ko uporabnik naročilo odda, se vmesnik poveže s procesom BPEL aplikacije Dobavitelj, ki nadalje procesira naročilo. Glede na odgovor s strani dobavitelja, ki je navadno v obliki predračuna, nato uporabnik s pritiskom na gumb potrdi ali pa prekliče svoje naročilo. V primeru potrditve predračuna trgovec prejme vse podatke o naročenih filmih, ki jih lahko neposredno uporabi v svoji aplikaciji.



Slika 15: Uporabniški vmesnik JSF aplikacije Trgovec

Ta spletna aplikacija za razliko od prejšnje ohranja stanje znotraj posameznih sej. Med sejo se morajo shraniti podatki o izbranih artiklih, ki jih je uporabnik dodal k naročilu, shraniti se mora tudi skupna cena, ki se računa med dodajanjem oz. brisanjem artiklov iz naročila, ter številka naročila, ki jo je potrebno poslati skupaj s potrdilom oz. preklicem naročila. Za ohranjanje seje JSF tehnologija uporablja poseben razred, znotraj katerega implementiramo vse spremenljivke, za katere želimo ohraniti stanja. Spodnji izsek programske kode prikazuje deklaracijo razreda z omenjenimi spremenljivkami.

```
public class SessionBean1 extends AbstractSessionBean implements Serializable {

    private List<Option> izbraniFilmi = new ArrayList<Option>();
    private double skupnaCena = 0;
    private int stNarocila;
```

Te spremenljivke se nato naslavljajo iz glavnega Java razreda, ki vsebuje vso poslovno logiko spletnega uporabniškega vmesnika. Branje spremenljivke, ki vsebuje skupno ceno za izbrane artikle, prikazuje spodnji izsek programske kode.

```
cenaTextField.setText(getSessionBean1().getSkupnaCena());
```

5 Storitveno usmerjena arhitektura

5.1 Od aplikacij k storitvam

5.1.1 Shema XML

Shema XML podaja predlogo naboru dokumentov XML, tipično z oblikovanjem omejitev njihove strukture in vsebine. Eden od jezikov, s katerimi oblikujemo te omejitve, je jezik XSD, katerega med drugimi uporablja tudi razvojno okolje Netbeans. Jezik XSD je predlagala organizacija W3C v letu 2001 pod imenom W3C XML Schema. Da bi se izognili dvoumnosti, se je predlagalo krajše poimenovanje XSD, predvsem zato, ker se ta kratica tipično uporablja kot predpona imenskega prostora XML in zato, ker imajo datoteke XML sheme istoimensko končnico. [13]

Shema XML ločuje med definicijami in deklaracijami spremenljivk oz. njihovih tipov. Običajno se najprej definirajo novi, enostavni ali sestavljeni tipi. Nato pa z deklaracijo določimo, kateri elementi in atributi se bodo uporabljali v izpeljanih dokumentih XML. Določimo jim imena in poprej definirane tipe. Spodnji primer prikazuje definicijo sestavljenega tipa *narociloZaloge* in deklaracijo elementa, ki ta tip uporablja.

```
<xsd:complexType name="narociloZaloge">
    <xsd:sequence>
        <xsd:element name="idFilm" type="xsd:int"/>
        <xsd:element name="minStKosov" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="NarociloZaloge" type="tns:narociloZaloge"/>
```

Poleg definicij enostavnih in sestavljenih tipov in deklaracij elementov lahko v shemo XML dodajamo še različne omejitve. To so npr. vrstni red in število elementov, ki se bodo pojavljali v dokumentih, določimo lahko ali je element lahko prazen ali pa zanj nastavimo privzeto vrednost. Končna shema XML mora vsebovati en ciljni imenski prostor in poljubno število imenskih prostorov ostalih virov.

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://service.dobavitelj/"
xmlns:tns="http://service.dobavitelj/"
```

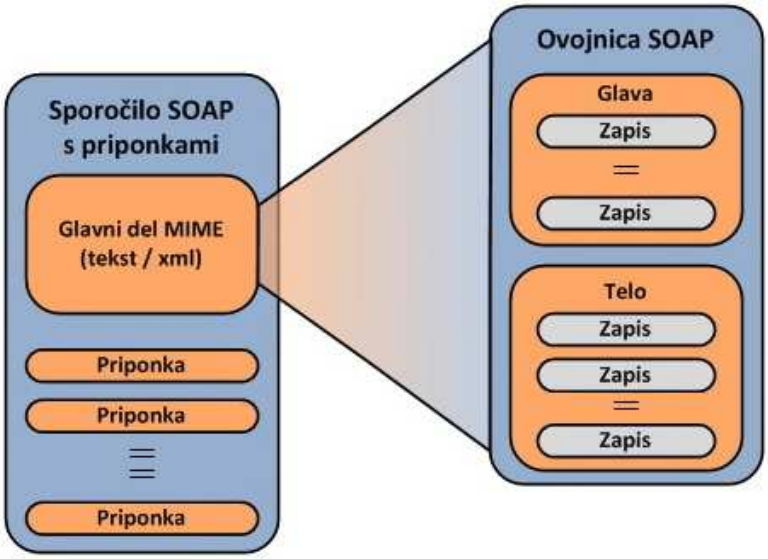
Shema XML je pomemben gradnik storitveno usmerjene arhitekture. Najdemo jo v ozadju vsake spletne storitve, kjer skrbi za podatkovne tipe parametrov vseh njenih operacij. Pravzaprav se lahko gradnja spletne storitve začne prav z oblikovanjem sheme XML. Največkrat to pride v poštev, če hočemo neko spletno storitev razviti povsem od začetka. Najprej torej načrtujemo shemo XML, ki jo nadalje vključimo v dokument WSDL, ki specificira določene višjenivojske parametre storitve. Če uporabljamo orodje JAX-WS, lahko potem preko dokumenta WSDL zlahka kreiramo ogrodja razredov, ki bodo izvajala poslovno logiko storitve, in nazadnje implementiramo še to. S pomočjo tehnologije JAX-WS lahko spletne

storitve gradimo tudi v obratni smeri, torej s preoblikovanjem že obstoječih poslovnih razredov. V tem primeru je načrtovalcu prihranjeno podrobno poznavanje shem XML, vsekakor pa le-te nekje znotraj aplikacije obstajajo.

5.1.2 SOAP

SOAP je protokol, ki definira način izmenjave strukturiranih informacij v porazdeljenem okolju. Grajen je na osnovi jezika XML, deluje pa v navezi s protokoli iz aplikacijske plasti, največkrat sta to HTTP ali RPC. Osnovan je tako, da deluje neodvisno od vrste programskega jezika in drugih podrobnosti implementacije spletnih storitev.

Protokol ima standarden format sporočila, ta vsebuje ovojnico in poljubno število morebitnih priponk. Le-te omogočajo, da se preko sporočila SOAP prenašajo tudi podatki, ki niso v obliki XML. Ovojnica, ki je korenski element sporočila, vsebuje glavo in telo sporočila. Glava je neobvezna, vsebuje pa lahko razne napotke prejemnikom, kot so informacije o transakcijah ter varnosti. Telo je obvezen del dokumenta in vsebuje podatke XML ali pa metode RPC in parametre, odvisno, ali gre za dokumentni ali pa za tip RPC sporočila SOAP.



Slika 16: Struktura sporočila SOAP

Pri prvem tipu sporočila SOAP, katerega uporabljajo vse spletne storitve aplikacije Dobavitelj, se prenašajo dokumenti XML. Za ustrezno interpretacijo podatkov znotraj teh dokumentov mora biti definirana shema XML. Spodaj je podana zahteva SOAP, ki se tvori ob klicu operacije *vrniFilm* spletne storitve *FilmWeb*.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:vrniFilm xmlns:ns2="http://service.dobavitelj/">
            <idFilma>1001</idFilma>
        </ns2:vrniFilm>
    </S:Body>
</S:Envelope>
```

Ovojnica ima definiran imenski prostor, glava sporočila je prazna, telo pa vsebuje ime operacije ciljne spletne storitve ter vrednost parametra, po katerih ta sprašuje. V odgovoru, ki ga tvori spletna storitev, so zapisane izhodne vrednosti klicane operacije, v formatu, ki ga predpisuje pripadajoča shema XML. Spodaj je podan odgovor SOAP na zgornjo zahtevo.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:vrniFilmResponse xmlns:ns2="http://service.dobavitelj/">
            <return>
                <id>1001</id>
                <naslov>Shrek</naslov>
                <leto>2001</leto>
                <ocena>8.0</ocena>
                <zvrst>Comedy Animation</zvrst>
                <trenutnaCena>10.99</trenutnaCena>
            </return>
        </ns2:vrniFilmResponse>
    </S:Body>
</S:Envelope>
```

5.1.3 WSDL

WSDL je tip jezika XML, ki se uporablja za opisovanje spletnih storitev. Te največkrat temeljijo na komunikaciji preko sporočil SOAP ter uporabljajo sheme XML za predstavitev lastnih podatkovnih tipov. Klasično se dokument WSDL objavi na spletu ali v registru UDDI, kjer je na voljo širšemu krogu uporabnikov. Uporabnik, ki želi dostopati do določene spletne storitve, prebere opis storitve WSDL in iz njega razloči, katere operacije ta storitev ponuja. Zatem kreira odjemalca spletne storitve, s katerim lahko do njenih operacij dostopa preko sporočil SOAP.

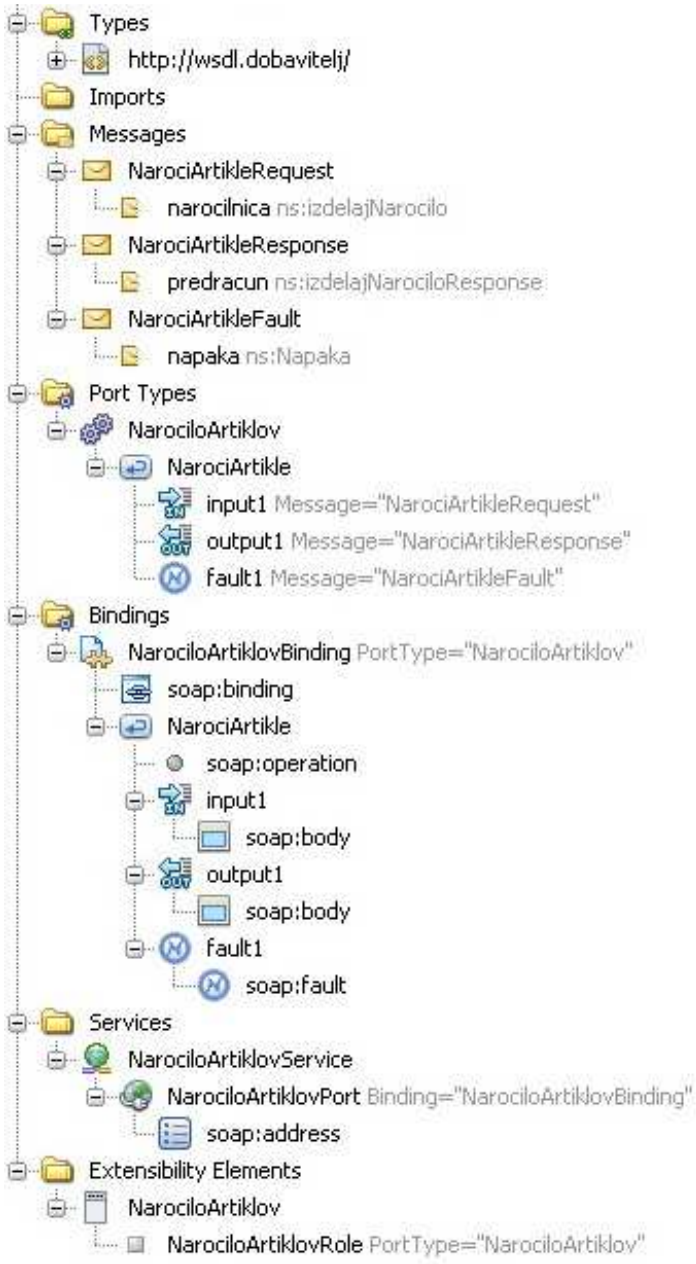
Spletna storitev je opisana z množico komunikacijskih končnih točk, ki so sposobna medsebojnega izmenjevanja sporočil. Vsaka končna točka ali *port* je sestavljena iz dveh delov. Prvi je množica abstraktnih definicij operacij in sporočil, drugi pa je nastavitev konkretnih omrežnih protokolov in naslovov končnih točk. Dela sta ločena z namenom ponovne uporabe istih operacij spletne storitve preko različnih omrežnih protokolov.

Opis spletne storitve WSDL vsebuje več elementov. Ti elementi so v jeziku XML strukturirani v dokument WSDL. Glavni elementi so:

- Podatkovni tipi, s katerimi so opisana sporočila, ki si jih izmenjujejo končne točke. Navadno so predstavljeni z shemo XML, ki je uvožena v dokument WSDL.
- Sporočila, ki se preko operacij prenašajo med končnimi točkami. Lahko so zahtevki, odgovori ali napake. Vsako sporočilo sestoji iz enega ali več logičnih delov, pri čemer vsakemu delu pripada določen podatkovni tip.
- Tipi končnih točk ali *Port Types*, ki predstavljajo skupek operacij, ki jih nudi spletna storitev. Vsaka operacija je določena z enim ali več sporočili, ki so lahko vhodna ali izhodna, lahko pa predstavljajo napako.

- Povezavo, ki določa konkreten protokol in format podatkov, v katerem se prenašajo sporočila.
- Storitev, ki opredeljuje množico končnih točk in njihove naslove. [15, 16, 17]

Spodnja slika prikazuje strukturo tipičnega dokumenta WSDL, za katero se v ozadju skriva XML programska koda.



Slika 17: Struktura dokumenta WSDL

Zgornji dokument WSDL opisuje spletno storitev naročanja artiklov. Vsebuje operacijo NarociArtikle, ki je določena s tremi sporočili. Prvo sporočilo predstavlja zahtevo, s katero odjemalec pokliče spletno storitev. V tem primeru je to naročilnica. Drugo sporočilo je odgovor na to zahtevo, ki se kaže v obliki predračuna. Tretje sporočilo je napaka, ki se odjemalcu posreduje ob vsaki nepravilnosti v izvajanju spletne storitve. Pod vejo povezav

(angl. Bindings) je definiran protokol SOAP, preko katerega se prenašajo sporočila v dokumentnem načinu. Prav tako je za vsako sporočilo podana struktura ovojnice, ki v tem primeru vsebuje samo telo sporočila. Pod vejo storitev *(angl. Services)* sta navedeni imeni vrat *(angl. Port)* in povezave, v okviru katerih storitev deluje. Tu je podan tudi naslov SOAP, preko katerega lahko dostopamo do storitve. Zadnja veja drevesnega prikaza dokumenta WSDL je namenjena elementom, ki ne ustrezajo ostalim kategorijam. Poleg vlog *(angl. Roles)*, ki jih uporabimo pri partnerskih povezavah procesa BPEL, sodijo v to kategorijo še korelacijski elementi in poizvedbe, preko katerih se ti določajo. Zadnjega elementa omenjen dokument WSDL ne opisuje.

5.1.4 JAX-WS 2.0

Kot je že omenjeno, tehnologija JAX-WS prinaša veliko poenostavitev pri razvoju spletnih storitev. Nekatere od njih uporablja tudi aplikacija Dobavitelj. Nekatere spletne storitve, ki jih aplikacija nudi svojim odjemalcem, so zgrajene s pomočjo pristopa od dna proti vrhu *(angl. bottom-up).* To je brez dvoma zelo poenostavljen pristop, saj dobesedno omogoča prehod iz navadnih poslovnih razredov k spletnim storitvam, katere lahko uporabljajo tudi odjemalci zunaj organizacije.

FilmWeb in *NarociloWeb* sta storitvi, ki sta se prvotno uporabljali le znotraj podjetja. Prva vsebuje operacije za pridobivanje informacij o filmih, kot so iskanje filmov po različnih atributih, vračanje podatkov o izbranih filmih itd. Druga omogoča kreiranje naročil in predračunov, izstavljanje računov ter preverjanje in dodajanje artiklov na zalogo. Sprva sta bili realizirani kot poslovna EJB razreda, z razširitvijo poslovanja organizacije prek spleta pa je bilo potrebno njuno preoblikovanje v spletni storitvi.

Da bi se razred EJB obnašal kot končna točka spletne storitve, je potrebna le notacija *@WebService.* Postavimo jo pred definicijo razreda ter ji nastavimo atribute, kot je npr. ime spletne storitve. Preostali atributi največkrat niso potrebni, saj imajo prednastavljene privzete vrednosti. Prav tako tudi ni potrebno generirati vmesnika končne točke SEI (Service Endpoint Interface), saj je z razredom JAX-WS implicitno določen. Zatem z notacijo *@WebMethod* označimo vse metode, za katere želimo, da jih bodo lahko uporabljali oddaljeni odjemalci. Tudi tej notaciji lahko poljubno dodamo nekatere atribute, ki pa niso obvezni. Spodaj je prikazano ogrodje spletne storitve z operacijo preverjanja zaloge, opremljeno s potrebnimi notacijami.

```
@WebService()
@Stateless()
public class NarociloWeb {

    @WebMethod(operationName = "preveriZalogo")
    public int preveriZalogo(@WebParam(name = "idFilma") int idFilma) {

    }
}
```

Na koncu le še izpostavimo spletno storitev aplikacijskemu strežniku in že lahko do nje dostopajo oddaljeni odjemalci. Tako shema XML kot tudi dokument WSDL se zgenerirata avtomatično. Pri shemi XML se za imena elementov in njihovih kompleksnih podatkovnih tipov uporabijo kar imena metod, v omenjenem primeru je to ime *preveriZalogo*. Imena elementov znotraj podatkovnih tipov so povzeta po imenih argumentov pripadajoče metode.

```
<xs:element name="preveriZalogo" type="tns:preveriZalogo"/>

<xs:complexType name="preveriZalogo">
  <xs:sequence>
    <xs:element name="idFilma" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
```

Pri avtomatično generiranem dokumentu WSDL je podobno. V tem primeru gre za dvosmerno operacijo, ki na zahtevo pošlje odgovor, zato sta definirana dva tipa sporočil. Imena sporočil sta povzeta po operacijah, elementa znotraj njiju pa se navezujeta na uvoženo shemo XML.

```
<message name="preveriZalogo">
  <part name="parameters" element="tns:preveriZalogo"/>
</message>
<message name="preveriZalogoResponse">
  <part name="parameters" element="tns:preveriZalogoResponse"/>
</message>
```

Načrtovanje odjemalcev spletnih storitev JAX-WS se začne z uvozom dokumenta WSDL. Ko se ta uvozi v aplikacijo, tehnologija JAXB poskrbi za generiranje vseh potrebnih razredov, ki bodo predstavljali spletno storitev. Nekateri se generirajo glede na elemente sheme XML, drugi predstavljajo samo spletno storitev in njene operacije. Načrtovalec odjemalca nato preko teh razredov kliče operacije spletne storitve ter upravlja s pridobljenimi podatki. Notacija, s katero se naslovi lokalni objekt, ki predstavlja spletno storitev, je *@WebServiceRef*. Pripeti ji moramo atribut, ki predstavlja relativno pot do dokumenta WSDL.

```
@WebServiceRef(wsdlLocation = "WEB-INF/wsdl/client/FilmWeb/" +
                              "localhost_8080/FilmWebService/FilmWeb.wsdl")
private FilmWebService filmService;
```

Preko tega objekta se nato kličejo operacije spletne storitve na enak način, kot bi se klicale lokalne metode.

5.2 Integracija spletnih storitev

5.2.1 Ideja

S preoblikovanjem nekaterih aplikacij v spletne storitve lahko sedaj podjetje Dobavitelj d.o.o deluje tudi preko svetovnega spleta. Odjemalci lahko uporabljajo različne operacije spletnih storitev za pregledovanje zaloge, pridobivanje informacij o artiklih, kreiranje računov, predračunov ipd. Na določeni točki razvoja pa tudi to ni več dovolj. Podjetje želi poenostaviti naročanje artiklov, ki naj bo čimbolj enostavno in avtomatizirano, poleg tega pa je pomembno, da se pri načrtovanju novega pristopa uporabi kar največ že obstoječih modulov in tehnologij. Rešitev je osrednji poslovni proces, ki bo koordiniral izvajanje večih spletnih storitev za izvajanje naročil, obveščanje strank in naročanje novih zalog artiklov. Poslovni proces bo v celoti implementiran s tehnologijo WS-BPEL 2.0.

5.2.2 Obdelava naročil

Poslovni proces, podan v BPEL, ki skrbi za obdelavo naročil artiklov, ima eno vhodno partnersko povezavo. Preko nje zunanji odjemalci oddajajo svoja naročila, do nje pa dostopajo na enak način, kot bi dostopali do operacije klasične spletne storitve. Vhodna sporočila so klasične naročilnice, ki vsebujejo identifikacijske številke artiklov in pripadajoča števila željenih kosov.

Vsakič, ko stranka pošlje naročilnico, se aktivira proces BPEL. Ta za vsak naročen artikel preko druge partnerske povezave najprej preveri zalogo. Če je ta premajhna, se o tem obvesti naročnika, nato pa se pošlje zahtevek za naročilo novih zalog artikla, kar aktivira drug proces BPEL. V primeru, da artikla z določeno identifikacijsko številko v podatkovni bazi ni, se o tem prav tako obvesti naročnika. Obveščanje poteka preko elektronske pošte, in sicer s pomočjo spletne storitve, ki je dosegljiva samostojno, ali pa jo, kot v tem primeru, prek partnerske povezave uporablja proces BPEL.

Ko je preverjanje zaloge končano, poslovni proces izvrši kreiranje predračuna. V primeru, če naročniku ni mogoče dostaviti nobenega artikla, se mu kot odgovor na zahtevo pošlje napako z ustrezno vsebino. V tem primeru se proces BPEL obdelave naročil zanj zaključi. V nasprotnem primeru pa se naročniku pošlje predračun, ki vsebuje določeno korelacijsko vsebino. Glede na to korelacijsko vsebino se kasneje enoumno poveže naslednji klic naročnika.

Naročnik ima po prejetju predračuna na voljo izbiro potrditi ali zavrniti naročilo. V tem primeru gre za asinhrono komunikacijo s procesom BPEL, saj po prejetju predračuna odjemalec naročnika ni zaklenjen in lahko nadaljuje z drugim delom. Prav zato se tu uporabljajo korelacijske spremenljivke, ki pravilno povežejo povratni klic naročnika z njegovim poprejšnjim naročilom artiklov. Korelacijska spremenljivka je v tem primeru številka predračuna, katero mora naročnik navesti ob povratnem klicu.

Ko se naročnik odzove, proces BPEL poskrbi za pravilno nadaljnje izvajanje. V primeru, da se odzove pritrdilno, se najprej generira račun, kar pomeni, da se v podatkovni bazi izvedejo ustrezne spremembe količine zalog. Na tej točki se lahko zgodi, da se je medtem ko je proces čakal na povratni klic že zgodil nek nakup s strani drugega naročnika. Če je ta nakup obsegal tudi katerega od artiklov, ki bi jih želel trenutni naročnik, potem se lahko zgodi, da je zaloga dotičnega artikla zopet premajhna. V tem primeru se na končnem računu ta artikel ne pojavi. V najslabšem primeru, če so bili v vmesnem času prodani vsi želeni artikli, pa se naročniku pošlje napaka z ustreznim sporočilom. V vsakem primeru se proces BPEL tukaj zaključi.

Po drugi strani pa se lahko naročnik na predračun odzove tudi negativno. V primeru, da hoče naročilo preklicati, preprosto pošlje ustrezno povratno sporočilo, ki mora prav tako vsebovati korelacijsko spremenljivko. Proces BPEL zatem poskrbi za uradni preklic naročila in naročnika o tem tudi obvesti po elektronski pošti.

Proces BPEL je pred neskončnim čakanjem na povratne klice naročnikov zavarovan s časovnikom. Če od poslanega predračuna mine določen čas, ne da bi naročnik potrdil ali zavrnil naročilo, se to avtomatsko prekliče, o čemer je naročnik obveščen po elektronski pošti.

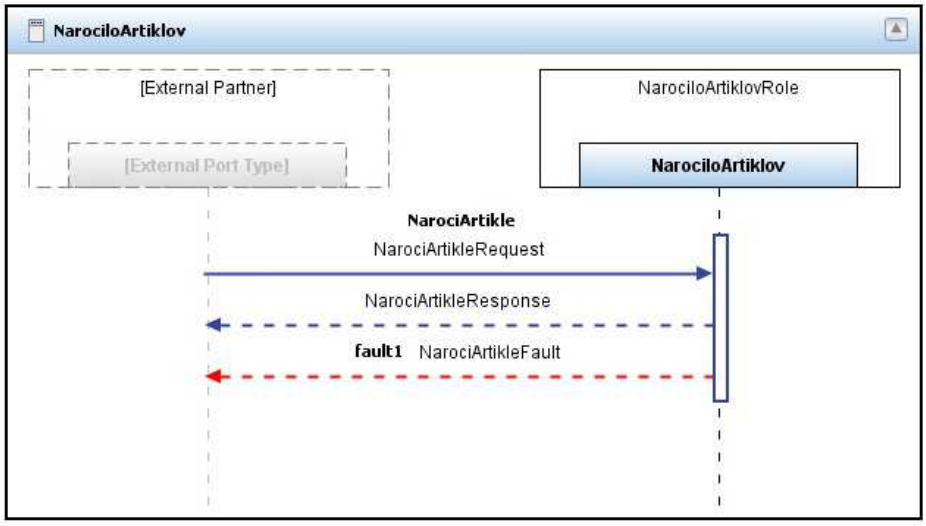
5.2.3 Obdelava naročil zaloge

Proces BPEL za obdelavo naročil zaloge se aktivira preko glavnega procesa BPEL za obdelavo naročil artiklov. Aktivira se v primeru, da je za izpolnitev strankinih zahtev na zalogi premajhno število določenih artiklov. V tem primeru se v proces pošlje sporočilo, ki vsebuje identifikacijsko številko artikla in minimalno številno kosov, ki jih je potrebno naročiti. S prejetjem tega sporočila proces BPEL o potrebnem naročilu novih zalog artiklov najprej obvesti odgovorne v podjetju. Zatem po elektronski pošti o premajhnem številu artiklov obvesti tudi naročnika. Proces BPEL se zatem ustavi in čaka na ustrezno sporočilo s strani podjetja.

Ko naročene zaloge prispejo v skladišče, nabavni oddelek pošlje sporočilo procesu BPEL. To sporočilo mora biti korelirano, vsebovati pa mora tudi število dobavljenih artiklov. Ko sporočilo prispe, se poveže s čakajočim zahtevkom po novih zalogah, proces BPEL pa tako nadaljuje z izvajanjem. Preko partnerske povezave se poveže s podatkovno bazo in posodobi število prispelih artiklov. Zatem pa še preko elektronske pošte kontaktira stranko, ki ji ni bilo mogoče v celoti dobaviti naročenega artikla. Sporoči ji, da je artikel ponovno na zalogi v dovolj velikih količinah za ponovno naročilo.

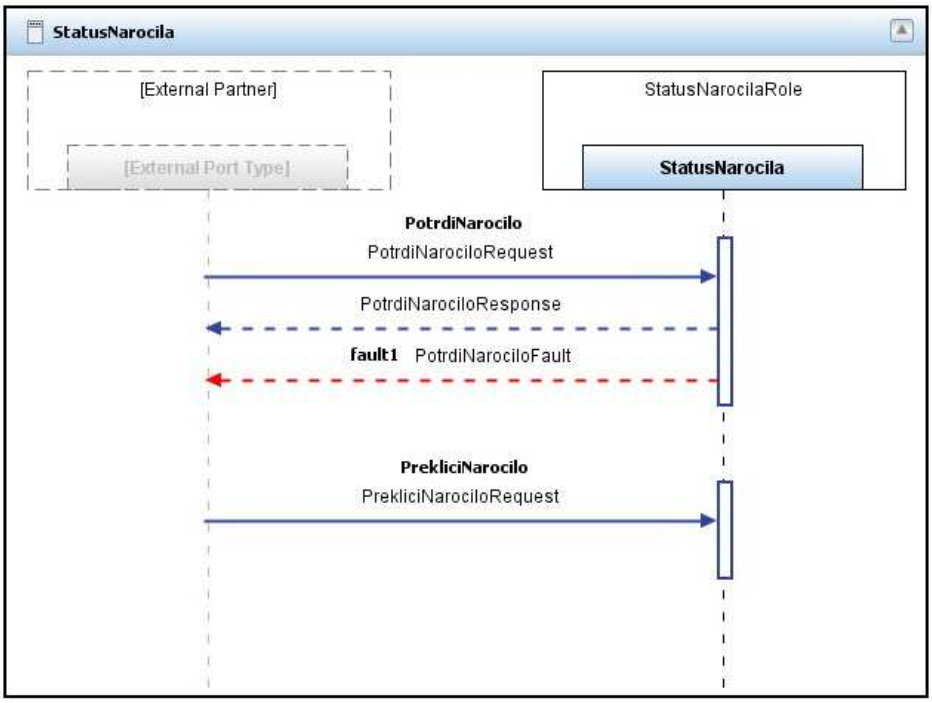
5.2.4 Partnerske povezave

Prva partnerska povezava predstavlja vstopno točko v proces BPEL. Namenjena je izključno komunikaciji z naročnikom. Ta preko nje v proces pošlje naročilnico, nakar mu proces po določenem zaporedju korakov vrne predračun ali pa napako, če v procesu pride do kakršnekoli nepravilnosti.



Slika 18: Partnerska povezava NarociloArtiklov

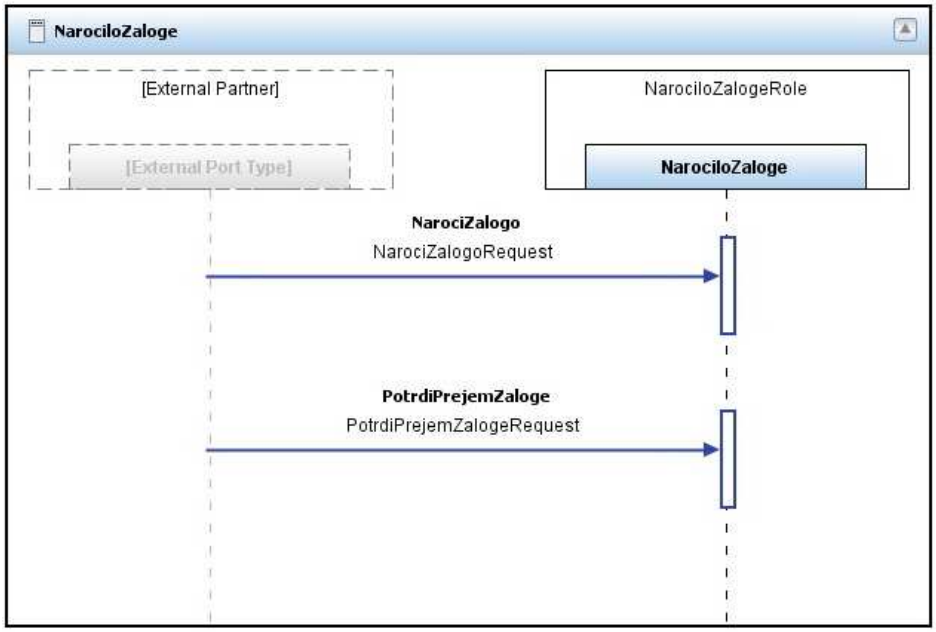
Druga partnerska povezava je namenjena asinhroni komunikaciji z naročnikom. Ta lahko preko nje v določenem roku potrdi ali pa zavrne naročilo. Skladno z njegovim odgovorom mu proces pošlje končni račun, napako ali pa ga obvesti o preklicu naročila.



Slika 19: Partnerska povezava StatusNarocila

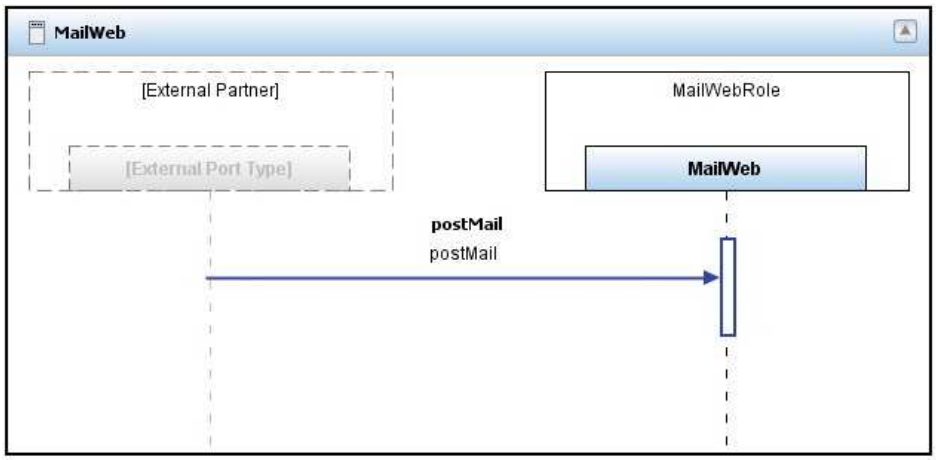
Naslednja partnerska povezava je namenjena komunikaciji med obema procesoma BPEL. Prek nje se poda zahteva po naročilu novih zalog ustreznih artiklov. Zahteva, ki je določena z identifikacijsko številko artikla ter z najmanjšim potrebnim številom novih kosov, aktivira drug proces BPEL, ki poskrbi za obveščanje naročnika o na novo prispelih zalogah. Ko se nova zaloga dobavi in se ta vpiše v podatkovno bazo, se preko iste partnerske povezave pošlje

asinhrono sporočilo, ki da poslovnemu procesu za obdelavo naročil zaloge zeleno luč za obvestitev naročnika o novo prispelih količinah.



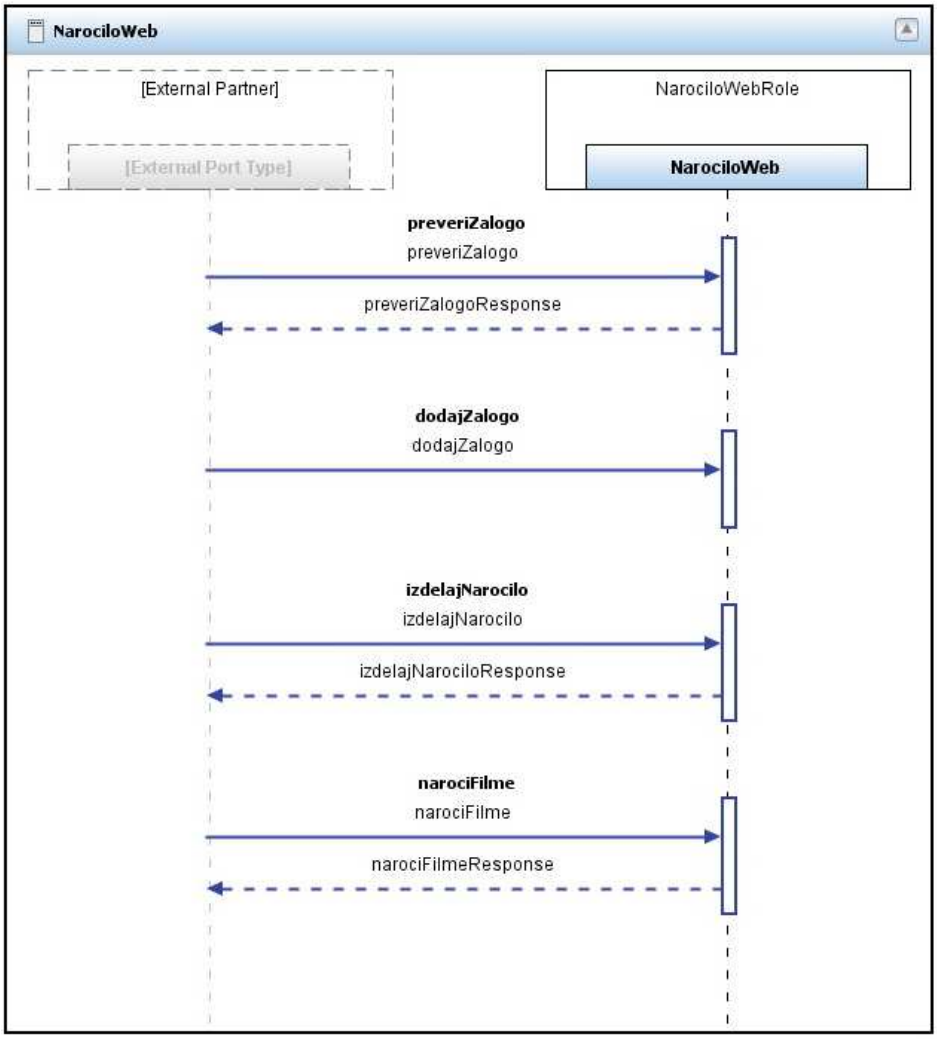
Slika 20: Partnerska povezava NarociloZaloge

Partnerska povezava, ki jo procesa BPEL uporabljata za obveščanje naročnika, predstavlja preprosto spletno storitev z operacijo pošiljanja elektronske pošte. Dostopna je tudi kot samostojna storitev, ki se uporablja znotraj organizacije Dobavitelj d.o.o.



Slika 21: Partnerska povezava MailWeb

Zadnja partnerska povezava predstavlja spletno storitev, ki vsebuje vse že poprej implementirane operacije, kot npr. preverjanje zaloge, kreiranje računov ipd. Vse te operacije so bile najprej na voljo uslužbencem znotraj podjetja, s preoblikovanjem aplikacij v spletne storitve so postale dosegljive odjemalcem prek spleta, sedaj pa jih BPEL povezuje v celovito, integrirano poslovno rešitev.



Slika 22: Partnerska povezava NarociloWeb

6 Zaključek

V diplomskem delu sem predstavil primer realizacije storitveno usmerjene arhitekture s pomočjo nabora tehnologij platforme Java EE 5 in orkestracijskega jezika BPEL. V uvodnih poglavjih sem izbral ustrezne tehnologije in jih opisal s teoretičnega vidika. Prav tako sem predstavil koncept SOA in procesno usmerjen vidik razvoja, ki ga prinaša BPEL. Kasneje sem opisane tehnologije uporabil pri praktičnem primeru, ki združuje prednosti in zmogljivosti vseh opisanih tehnologij. Izkazalo se je, da lahko z ustrezno uporabo vsaka prinese svoj delež h gradnji najrazličnejših rešitev storitveno usmerjene arhitekture. Dodana vrednost končne rešitve se znatno poveča z vpletenostjo jezika BPEL, ki je razvit za definiranje in opisovanje poslovnih procesov in se brez večjih omejitev lahko uporablja znotraj enega ali pa povezuje storitve večih podjetij.

Nadalje lahko več pozornosti posvetimo tehnologijama BPELJ in WSIF, ki temeljita na ideji jezika BPEL in sta v tem delu opisani le na kratko. Obe predvidevata še tesnejše sodelovanje med programskim jezikom Java in BPEL. Če se usmerimo v podrobnejše preučevanje slednjega jezika, je razpravam odprtih še več vprašanj. Na prvem mestu je tu seveda vprašanje varnosti. V povezavi s spletnimi storitvami obstaja vrsta WS-* specifikacij, ki pripomorejo k varnosti, raziščemo lahko, kako se te prenesejo v procese, implementirane z jezikom BPEL. Poleg tega nas lahko zanima prihodnost jezika BPEL in njegova uporaba v praksi. Različica BPEL 2.0 je prinesla nekaj dopolnitev k prvotni izvedbi jezika, v prihodnosti pa lahko še pričakujemo določene dopolnitve in izboljšave. Govorimo lahko o razširitvi komunikacije procesov BPEL prek meja spletnih storitev, o kateri v okviru jezika Java že govori WSIF, nadalje o vključevanju človeških interakcij, ki ga predvideva specifikacija BPEL4People, o izboljšanju prenosljivosti itn.

V okviru diplomske naloge sem se ukvarjal tudi z odprtokodnim operacijskim sistemom Puppy Linux, ki temelji na jedru Linux. Posebnost tega operacijskega sistema je v njegovi zmožnosti zagona s kateregakoli prenosnega medija. Celoten sistem lahko na izbiro uporabnika teče iz glavnega pomnilnika, kar ga postavlja med najhitrejše na trgu. Svoje delo lahko uporabnik na koncu shrani na katerikoli pomnilni medij ali pa ga s priloženim orodjem doda k že obstoječemu zagonskemu delu operacijskega sistema. Puppy Linux je zaradi naštetih prednosti dobra izbira pri manjših ali večjih opravilih in je vreden omembe kljub temu, da ne sovpada s samo tematiko diplomske naloge.

V omenjenem operacijskem sistemu je pripravljena celotna programska rešitev diplomske naloge. Na prenosnem mediju je nameščena podatkovna baza, strežnik Java EE in celotno razvojno okolje NetBeans. Poleg tega je v namen predstavitve nastavljen tudi odjemalec e-pošte. Uporabnik si lahko programsko rešitev ogleda neposredno prek tega prenosnega medija, ki je obenem tudi zagonski disk. V uporabnikov računalniški sistem se pri tem ne posega, kar je le še ena dobra lastnost operacijskega sistema Puppy Linux.

7 Priloge

7.1 Seznam slik

7.2 Seznam tabel

8 Viri in literatura

[1] M. Jurič, (2005), BPEL and Java. Dostopno na http://www.theserverside.com/tt/articles/article.tss?l=BPELJava

[2] Sun Microsystems, Inc. Java EE at Glance. Dostopno na http://java.sun.com/javaee/

[3] Sun Microsystems, Inc. (2002), Designing Enterprise Applications with the J2EE™ Platform, Second edition. Dostopno na http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/introduction/introduction3.html

[4] Sun Microsystems, Inc. (2008), Distributed Multitiered Applications. Dostopno na http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html

[5] J. Stearns, R. Chinnici, Sahoo (2006), Update: An introduction to Java EE 5 platform. Dostopno na http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/

[6] R. R. Kodali (2005), The Simplicity of EJB 3.0. Dostopno na http://java.sys-con.com/node/117755

[7] D. Rubio (2006), The Benefits of Java EE 5. Dostopno na http://www.theserverside.com/tt/articles/article.tss?l=BenefitsJavaEE5

[8] Sun Microsystems, Inc. (2005), Annotations. Dostopno na http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

[9] Sun Microsystems, Inc. (2008), Managing Entities. Dostopno na http://java.sun.com/javaee/5/docs/tutorial/doc/bnbqw.html

[10] Sun Microsystems, Inc. (2008) Binding between XML Schema and Java classes. Dostopno na http://java.sun.com/javaee/5/docs/tutorial/doc/bnazf.html

[11] Sun Microsystems, Inc. (2008), Building web services with JAX-WS. Dostopno na http://java.sun.com/javaee/5/docs/tutorial/doc/bnayl.html

[12] S. Shin (2007), JAX-WS Basics. Dostopno na http://www.javapassion.com/webservices/jaxwsbasics.pdf

[13] XML Schema (W3C). Dostopno na http://en.wikipedia.org/wiki/XML_Schema_(W3C)

[14] S. Shin (2007), SOAP 1.2 (Simple Object Access Protocol). Dostopno na http://www.javapassion.com/webservices/SOAPBasics_speakernoted.pdf

[15] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana (2001), Web Service Definition Language (WSDL). Dostopno na http://www.w3.org/TR/wsdl

[16] Web Services Description Language. Dostopno na http://en.wikipedia.org/wiki/Web_Services_Description_Language

[17] S. Shin (2007), WSDL Basics. Dostopno na http://www.javapassion.com/webservices/WSDLBasics.pdf

[18] Service-Oriented Architecture. Dostopno na http://en.wikipedia.org/wiki/Service-oriented_architecture#Web_services_approach

[19] Barry & Associates, Inc. Service-Oriented Architecture (SOA) Definition. Dostopno na http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html

[20] S. Shin (2007), BPEL: Building Standards-Based Business Processes with Web Services. Dostopno na http://www.javapassion.com/soaprogramming/BPELOverview.pdf

[21] B. May, I. Filippova (2007), Developer Guide to the BPEL Designer – Using the Palette Elements. Dostopno na http://www.netbeans.org/kb/60/soa/bpel-guide-palette.html

[22] M. Kondratyev, A. Stashkova (2007), Using Correlation Sets, Properties and Property Aliases in BPEL. Dostopno na http://www.netbeans.org/kb/60/soa/correlation.html

[23] F. Sommers, (2005), Service – Oriented Java Business Integration. Dostopno na http://www.artima.com/lejava/articles/jbi.html

[24] The Apache Software Foundation (2006), Welcome to WSIF: Web Services Invocation Framework. Dostopno na http://ws.apache.org/wsif/

[25] Sun Microsystems, Inc. (2008), JavaServer Faces Tehnology. Dostopno na http://java.sun.com/javaee/5/docs/tutorial/doc/bnaph.html