SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Damir Pintar

MODEL USLUŽNO ORIJENTIRANE ARHITEKTURE ZA STVARNOVREMENSKO SKLADIŠTENJE PODATAKA ZASNOVAN NA METAPODACIMA

DOKTORSKA DISERTACIJA

Zagreb, 2009.

Doktorska disertacija izrađena je na Zavodu za telekomunikacije Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu.

Mentor: Prof.dr.sc.Zoran Skočir, dipl.ing.

FER, Sveučilište u Zagrebu

Doktorska disertacija ima 185 stranica.

Doktorska disertacija broj:

Uvod

Integracija pogonjena modelom (engl. *model-driven integration*) pojam je koji označava automatiziranu ili poluautomatiziranu integraciju heterogenih sustava uz pomoć apstraktnih modela opisanih u nekom normiziranom jeziku (najčešće se radi o jeziku UML ili nekoj njemu srodnoj normi). Osnovna ideja jest egzistencija infrastrukture koja pohranjuje standardizirane opise informacijskih sustava te korištenje predefiniranih sučelja koja bi iz takve infrastrukture mogla prikupljati i obrađivati podatke i na taj način prikupiti dovoljno znanja o svojoj okolini kako bi automatska ili poluautomatska integracija s okolnim aplikacijama i sustavima bila omogućena. Pokretač integracije jest stoga dijeljenje podataka, a da bi se podaci mogli razmjenjivati i dijeliti oni moraju odgovarati zajedničkoj normi, tj. modelu. Taj model pružaju upravo metapodaci.

Metapodaci se definiraju kao podaci koji opisuju druge podatke, najčešće u kontekstu elemenata informacijskih sustava. Isprva su metapodaci korišteni upravo u tu svrhu, tj. za stvaranje dokumentacije vezane uz informacijske svrhe. Usponom i globalnim prihvaćanjem norme UML metapodaci su postali svrhovit alat za modeliranje sustava, tj. stvaranje apstraktne ali precizne slike sustava koja služi kao predložak i pomagalo za njegovu implementaciju. Konačno, metapodaci su prepoznati kao učinkovito sredstvo integracije heterogenih sustava. Nažalost, ova uporaba još nije zaživjela, većinom zbog nedostatka općenito prihvaćene i normizirane specifikacije metapodataka, njihove pohrane i razmjene.

Gledano s tehnološke strane, otvorene tehnologije - poput Usluga weba i XML-a - stvaraju učinkovite temelje za integraciju heterogenih sustava. Jedan od problema s integracijom kroz otvorene tehnologije jest upravo činjenica da još uvijek nema široko prihvaćene metodologije koja stvara uspješnu sinergiju ovih tehnologija s potencijalima integracije uz pomoć normiziranih metapodataka. Implementacije arhitekture SOA (engl. *Service Oriented Architecture* – uslužno orijentirana arhitektura*)* zasnovane na Uslugama weba i XML-u najčešće se rade na najnižoj metarazini, tj. modeliranjem sustava ovisno o konkretnoj poslovnoj okolini i danom problemu, s minimalnim osvrtom na potencijalno dostupne metapodatke koji opisuju dane komponente sustava. No upravo metapodaci najčešće u sebi sadrže informacije ključne za integraciju sustava, pogotovo s obzirom na dizajn komponenti koje bi tu integraciju realizirale.

Jedno od područja koja bi polučila velike pogodnosti od ovakvog pristupa jest skladištenje podataka. Dohvat podataka iz operativnih poslovnih izvora, njihova transformacija i skladištenje predstavljaju klasičan primjer informacijskog sustava sačinjenog od heterogenih komponenti koje treba ujediniti u jedinstvenu i homogenu poslovnu strukturu u svrhu podrške poslovnom odlučivanju. Trend modernih skladišta koji kao imperativ postavlja smanjenje latencije dostave podataka tj. njihovo premještanje iz operativnih izvora u skladište u stvarnom ili skoro stvarnom vremenu upravo je idealan scenarij za korištenje arhitekture SOA. Naravno, ovakva implementacija najčešće zahtjeva potpuni reinženjering skladišnog sustava i znatna ulaganja financijskih i ljudskih resursa. Kako bi se postigao brzi povrat investicije (engl. ROI – *return of investment*), jedno od rješenja mogla bi predstavljati upravo integracija

pogonjena modelom, tj. brzi dizajn infrastrukture SOA uz pomoć modela skladišta opisanom uz pomoć normiziranih metapodataka.

Na tržištu već postoje otvorene norme metapodataka upravo za potrebe skladištenja, a dosad najširu podršku dobila je norma CWM (engl. *Common Warehouse Metamodel*). Nažalost, iako je komplementarna CWM podrška integrirana u širok spektar postojećih skladišnih proizvoda (npr. *Oracle*-ov *Warehouse Builder*), implementacija norme CWM u svrhu integracije još uvijek je minimalna. Unatoč tome, sama činjenica da postoji automatizirana podrška za *produkciju* metapodataka u normi CWM predstavlja bitan korak za realizaciju inverznog procesa - stvaranje programske podrške na osnovu dostupnih ili modeliranih skladišnih metapodataka.

Ovaj rad prikazat će mogućnost korištenja metapodataka u svrhu automatskog ili poluautomatskog stvaranja podrške stvarnovremenskom skladištenju podataka [1]. Razvit će se model metapodataka utemeljen na proširenju norme CWM kako bi se uvrstili koncepti vezani uz SOA arhitekturu i stvarnovremensko skladištenje, te će se prikazati formalizacija modela kroz skup dokumenata normiziranih uz pomoć specifikacije XML Schema. Izrađeni model verificirat će se pomoću modela uporabe tako definiranih metapodataka u sustavu Usluga weba povezanih u poslovni proces, čija će funkcija biti dohvaćanje, elementarna transformacija i pohrana podataka u stvarnovremensku particiju skladišnog sustava, gdje će biti dostupni za daljnje transformacije i analize.

U prvom poglavlju bit će opisani glavni koncepti, karakteristike i prednosti uslužno orijentiranih arhitektura. Drugo poglavlje bavit će se najčešćim tehnologijama i normama koji se povezuju uz uslužno orijentirane arhitekture današnjice. U trećem poglavlju bit će prikazan kratak opis problematike stvarnovremenskog skladištenja te osvrt na potencijalnu ulogu arhitekture SOA u stvaranju sustava za realizaciju stvarnovremenskog skladištenja. Četvrto poglavlje bavi se definicijom i svrhom metamodeliranja te navodi neke češće korištene normizirane modele metapodataka. Peto poglavlje prikazuje rezultat istraživanja kroz stvaranje i formalizaciju novog metamodela za potrebe uslužno orijentiranog sustava za potporu stvarnovremenskom skladištenju. Šesto poglavlje bavi se specifikacijama usluga i poslovnog procesa koji čine okosnicu spomenutog sustava te na konkretnoj implementaciji testira njegovu funkcionalnost. Konačno, u zaključku je dan sažetak primijenjenih koncepata i izvedenog modela te je dan prijedlog nastavka rada na ovoj problematici.

1 Uslužno orijentirana arhitektura

1.1 Definicija pojma uslužno orijentirane arhitekture

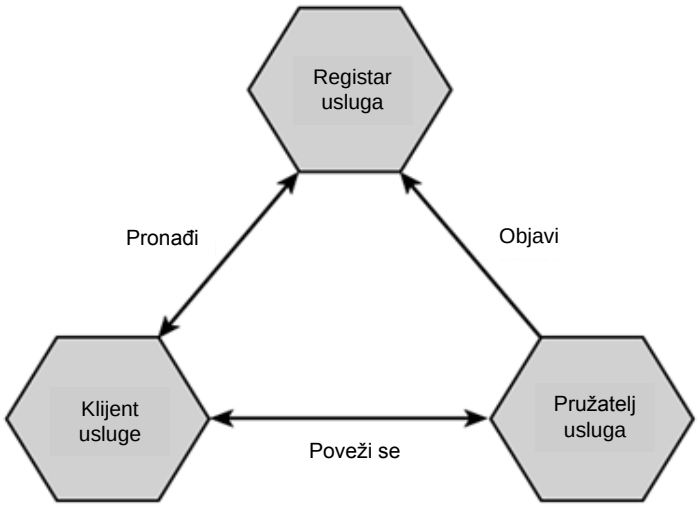
Uslužno orijentirana arhitektura (SOA – engl. *Service Oriented Architecture)* pojam je koji najčešće označava infrastrukturalni model sustava čija je funkcionalnost zasnovana na međusobno interoperabilnim "uslugama", tj. funkcionalnim entitetima koji međudjeluju kroz zajednički definirane norme. Egzaktna definicija ne postoji, pa čak ni koncenzus oko toga da li pojam označava samo paradigmu, model sa svojstvima i smjernicama kojih se pri implementaciji treba pridržavati ili čak strogo definirani skup tehnologija koje zajedno čine okosnicu uslužno orijentirane arhitekture.

Raghu R. Kodali, softverski inženjer iz tvrtke *Oracle* i autor članaka za portal *JavaWorld* smatra da je SOA "[…] e*volucija distribuiranog računarstva nastala na paradigmi zahtjev/odgovor za potrebe sinkronih i asinkronih aplikacija. Poslovna logika aplikacije ili samo individualne funkcije se modulariziraju i objavljuju u obliku usluga za korisnike, tj. klijentske aplikacije. Ključ sustava je svojstvo slabe povezanosti usluga, tj. neovisnost između sučelja usluga i njihove implementacije. Stvaratelji aplikacija mogu izgrađivati aplikacije povezivanjem jedne ili više usluga bez znanja o njihovoj konkretnoj implementaciji.“ [2]*. Ova definicija naglašava nekoliko ključnih karakteristika sustava SOA koje se najčešće spominju u svim definicijama – modularnost, distribuiranost i slabu povezanost.

Jedna od bolje sročenih definicija koja se također često može naći u člancima vezanim uz SOA jest ona koju je dao Malte Poppensieker, profesor Trierskog sveučilišta u Njemačkoj i koja glasi: "*SOA je arhitektura u kojoj autonomne, slabo povezane i široko definirane usluge dobro definiranih sučelja pružaju poslovnu funkcionalnost koja može biti na određeni način otkrivena i kojoj se može pristupiti kroz odgovarajuću infrastrukturu podrške. Ovo omogućuje kako unutrašnju tako i vanjsku integraciju kao i fleksibilnu ponovnu iskoristivost aplikacijske logike kroz kompoziciju i rekompoziciju usluga." [3]*. Ova definicija uz ponovno naglašavanje slabe povezanosti i integracije spominje i nekoliko dodatnih paradigmi koje se najčešće vežu uz pojam uslužno orijentirane arhitekture: svojstvo *otkrivanja* usluga, nužnost *dobro definiranih sučelja* te pojam *ponovne iskoristivosti* koji je okosnica funkcionalnosti uslužno orijentirane arhitekture.

Konačno, za vizualnu reprezentaciju SOA arhitekture često se koristi trokut "klijent-usluga-registar" koji predočuje visoko apstraktnu sliku glavnih značajki ovakvog sustava – objavu, pronalaženje te povezivanje s traženom uslugom. Ovaj trokut prikazuje **Slika 1-1.**

Ovdje se jasno vide tri glavna sudionika SOA arhitekture: pružatelj usluga, njezin klijent te registar usluga kao infrastrukturalna podrška koja omogućuje pretragu usluga te daje konkretne informacije o njihovoj lokaciji i te kako im pristupiti.



Slika 1-1 - Apstraktni pogled na uslužno orijentiranu arhitekturu

Ova vizualna reprezentacija, iako pregledna i intuitivna, nastala je prema modelu Usluga weba a u današnje se vrijeme smatra nedostatnom jer ne uključuje moderne principe uslužno orijentiranih arhitektura (kojima se bavi poglavlje 1.9) niti naglašava arhitekturalni aspekt paradigme.

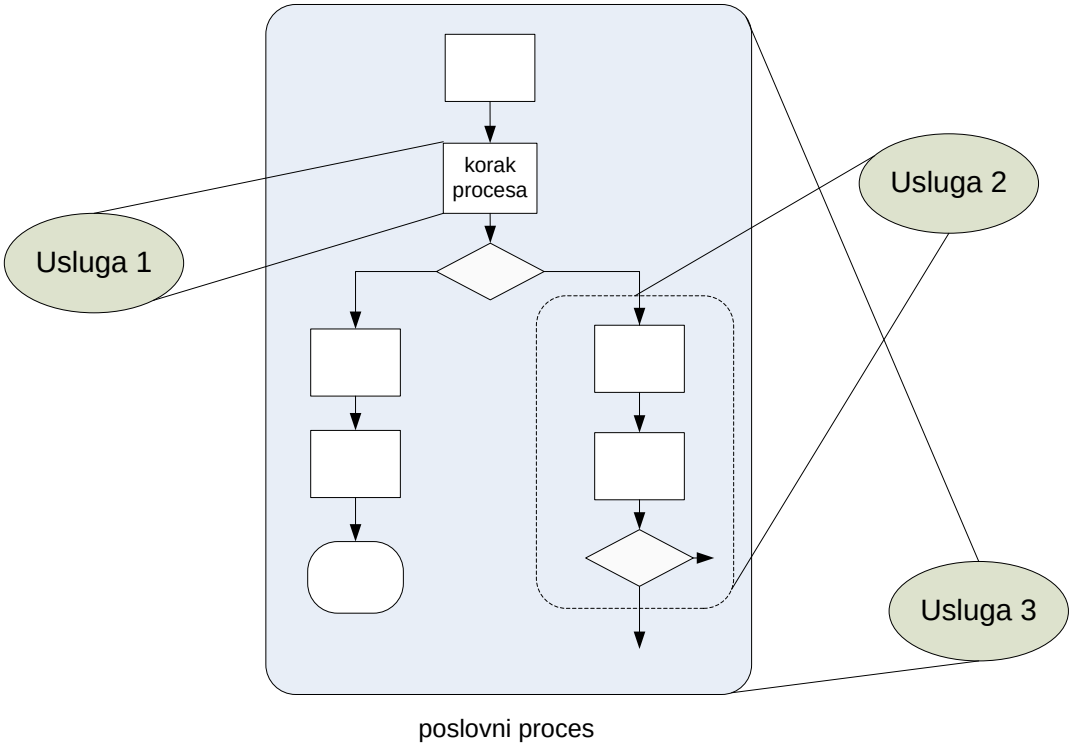
1.2 Temeljna načela uslužno orijentirane arhitekture

Pojam "uslužno orijentiran" u informatičkoj terminologiji postoji već više desetljeća gdje se koristio u različitim kontekstima te za različite svrhe. Temeljna ideja koja prožima sva značenja jest ta da se određena logika koja se koristi za rješenje nekog problema može bolje i lakše razviti, izvesti i upravljati ako se ona rastavi u skup manjih dijelova prema nekom logički opravdanom principu. Svaki taj mali dio logike odnosio bi se na jedan segment rješavanja problema, a svi ti dijelovi bi usprkos samostalnoj egzistenciji bili na određeni način povezani. Nadogradnjom riječju "arhitektura" značenje ovog pojma dobija tehnički smisao – tj. daje se na znanje da se radi o modelu koji je razvijen na prethodno navedenom principu. Logičke jedinice, tj. usluge su u svojoj funkcionalnosti autonomne ali nisu međusobno izolirane; gledano u cjelini one čine jedan organizirani distribuirani sustav široko definirane funkcionalnosti, tj. svojevrsnu arhitekturu [4].

Svaka usluga uokviruje logiku unutar definiranog konteksta. Kontekst može ovisiti o određenom poslovnom zadatku, poslovnom entitetu ili o nekom drugom načinu logičkog grupiranja. Opseg usluge tj. širina njezine funkcionalnosti može biti raznolika; usluga može obavljati jedan mali, strogo definirani dio poslovnog procesa ali može i obavljati široko definirane poslovne zadatke po potrebi i uz pomoć drugih, uže definiranih usluga što za krajnjeg korisnika može biti potpuno transparentno.

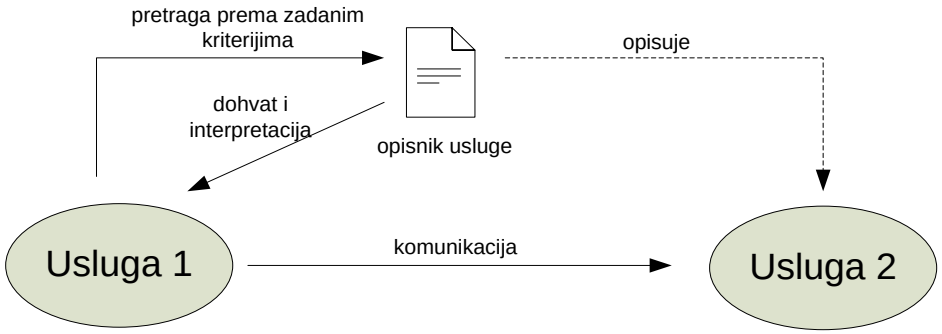
Primjer procesa zasnovanog na uslugama različito definiranih opsega prikazuje **Slika 1-2.** Opseg pojedine usluge definiran je u ovisnosti o kontekstu i ulozi tako definirane usluge unutar poslovnog procesa. Npr.

'Usluga 1' obavlja jedan usko definirani segment poslovnog procesa na najnižoj razini granulacije. 'Usluga 2' također vrši određenu dužnost, ali dio procesa koji ona obavlja je zapravo podproces koji se sastoji od više segmenata. Konačno, 'Usluga 3' je široko definirana usluga koja enkapsulira cijeli proces i koja zapravo apstrahira postojanje 'Usluge 1' i 'Usluge 2' te ih za korisnika 'Usluge 3' može učiniti skrivenim. Korisnik koji sagleda proces iz više razine ne mora biti svjestan Usluga „nižeg sloja“ pošto njega zanima samo poslovni model Usluge koje koristi tj. rezultat obrade informacija koje ona pruža.



Slika 1-2 - Usluge različito definiranog opsega s gledišta poslovnog procesa

Usluge se moraju nalaziti u prethodno definiranim odnosima te često moraju biti svjesne postojanja drugih usluga. Korisnici usluga također moraju imati na raspolaganju mehanizam pomoću kojega mogu dobiti detalje o određenoj usluzi kojoj žele pristupiti i čiju funkcionalnost potrebuju. Saznanje o drugim uslugama provodi se kroz tzv. opise usluga (engl. *service descriptions*). **Slika 1-3** prikazuje osnovni postupak prepoznavanja i komunikacije s drugom uslugom.



Slika 1-3 - Uporaba opisnika usluge

Osnovni oblici opisnika usluga pružaju informacije o imenu i mjestu usluge te nužnim preduvjetima za uspostavu komunikacije. Komunikacija se provodi na principu "slabe povezanosti" – kao što je rečeno u prethodnom poglavlju, sučelje usluge je normizirano i neovisno o samoj implementaciji usluge, što olakšava komunikaciju i pristup samoj usluzi. Normizirano sučelje je ovisno o definiranoj komunikacijskoj infrastrukturi, koja se najčešće provodi pomoću *poruka.*

Poruka se u uslužno orijentiranoj arhitekturi definira kao "nezavisna jedinica komunikacije". To znači da entitet koji pošalje poruku više nema kontrolu nad njom niti nad aktivnostima koje će se nad njom kasnije dogoditi (analogno slanju pismene poruke pomoću poštanskih usluga – korisnik nakon adresiranja i adekvatnog pakiranja svoje pošiljke više nema mogućnost izmjene ili upravljanja što se događa s poslanom pošiljkom, ali ima povjerenje u poštansku infrastrukturu da će pošiljka stići na odabranu adresu u neizmjenjenom obliku). Poput adrese na pošiljci, poruka u sustavu uslužno orijentirane arhitekture mora sadržavati informacije pomoću kojih će se ona moći usmjeravati kroz sustav te dostaviti na odabrano mjesto (i omogućiti primanje odgovora ako je to potrebno).

Principi dizajna usluga i poruka predstavljaju vrlo bitan segment pojma uslužno orijentirane arhitekture. Neki od tih principa su:

- ***slaba povezanost*** – kao što je već rečeno, radi se o uspostavljanju takvog tipa odnosa - između samih usluga ali i usluga i korisnika - koji minimalizira međuovisnost i nužno zahtjeva samo svjesnost o međusobnom postojanju
- ***ugovor o komunikaciji*** – svaka usluga mora se prilagoditi definiranim normama tj. ugovoru o komunikaciji čiji se detalji mogu opisati na određeni način u opisniku usluge
- ***autonomija*** – usluge imaju apsolutnu kontrolu nad poslovnom logikom koju sadržavaju u sebi
- ***apstrakcija*** – sve informacije koje usluga pruža o sebi nalaze se u opisniku usluge; sama implementacija sakrivena je od vanjskog svijeta
- ***ponovna iskoristivost*** – usluga može biti dio većeg broja poslovnih procesa a njezina funkcionalnost ne mora biti prilagođena samo posebno odabranoj prilici već se kao univerzalna može prema potrebi ponovno koristiti
- ***složivost*** – skupovi usluga moraju se moći komponirati u poslovne procese i/ili šire definirane usluge
- ***nepostojanje stanja*** – usluga mora minimalizirati informacije koje su specifične za pojedini slučaj uporabe

- ***mogućnost pronalaženja usluge*** – usluge se stvaraju tako da se mogu opisati preko definiranih mehanizama kako bi se informacije o njima mogle na adekvatan način pronaći te kako bi im se moglo pristupiti

Ono što je ključno naglasiti jest činjenica da je pojam uslužno orijentirane arhitekture tehnološki neovisan – principi koje definira su generički i za izvedbenu funkcionalnost je potrebno odabrati konkretnu implementacijsku platformu. Kao jedna od najraširenijih platformi danas se spominje tehnologija Usluga weba, koja je pobliže objašnjena u poglavlju 2.

Karakteristike, principi i zahtjevi nad uslužno orijentiranim arhitekturama mijenjali su se od njenih početaka do današnjih dana. Naredno poglavlje prikazat će više detalja o povijesti i razvoju uslužno orijentirane arhitekture od njenih začetaka do danas.

1.3 Povijest uslužno orijentirane arhitekture

U 60.-tim godinama prošlog stoljeća razvijen je jezik zvan SGML (engl. *Standard Generalized Markup Language)*. Originalna ideja razvoja ovog jezika bila je stvaranje načina reprezentacije informacije koju će tadašnja računala moći interpretirati te koja će biti relativno postojana s obzirom na izmjenu i evoluciju tehnologije toga doba. Nakon prihvaćanja i normizacije od strane organizacije ISO (engl. *International Organization for Standarization)* jezik je vrlo dobro prihvaćen kao meta-jezik pomoću kojeg se mogu definirati dokumenti zasnovani na oznakama za različite primjene u tadašnjoj informatičkoj tehnologiji.

Navedena ideja "označnog jezika" (engl. *mark-up language*) uvodi anotaciju informacije unutar dokumenta dodatnim podacima koji mogu poslužiti kao metapodaci, upute za način prikaza i sl. Dokument se organizira u stablastu strukturu "oznaka" (engl. *tags*), čija standardna sintaksa uključuje zagrade "<" i ">" za definiciju označnog elementa i kosu crtu "/" koja predstavlja završetak elementa unutar stablaste strukture (**Slika 1-4**)**.**

```
<CITAT izvor="William Shakespeare: Hamlet">
       <ITALIC>Biti ili ne biti, pitanje je sad.</ITALIC>
</CITAT>
```

Slika 1-4 - Primjer jednostavnog SGML-ovskog dokumenta

Najpopularniji nasljednici jezika SGML su jezik HTML (engl. – *HyperText Markup Language* - općeprihvaćena norma za prikaz udaljeno dohvatljivih sadržaja) te jezik XML (engl. *eXtensible Markup Language*). Specifikacija XML-a predstavlja svojevrsni pojednostavljeni podskup SGML-a, dizajniran na način da omogući implementaciju jednostavnijih parsera od onih koji mi podržavali jezik SGML. XML veliku popularnost dostiže sredinom 90.-ih godina prošlog stoljeća kada se Internet polako počinje prepoznavati kao idealna platforma za moderno elektroničko poslovanje. Pomoću XML-a aplikacije imaju šansu poslovnu informaciju uobličiti na platformski neutralan način prilagođen kako ljudskom

razumijevanju (pošto je zasnovan na tekstu a ne binarnom kodiranju) tako i računalnoj obradu (zbog strogo definirane sintakse). Dodatne specifikacije kao što je XML Schema (koja omogućuje definiranje ograničenja nad strukturom XML-ovskih dokumenata i tipovima podataka koje sadržavaju) i XSLT (koja omogućuje transformaciju XML-ovskih dokumenata u drukčije strukturirane XML-ovske dokumente ali i u druge formate) još više pospješuju univerzalno prihvaćanje XML-a. Ono što XML-u u to doba nedostaje je normizirana infrastruktura koja će učinkovito koristiti prednosti koje taj jezik nudi.

2000.-te godine organizacija W3C (engl. *World Wide Web Consortium*) prima zahtjev za ratifikacijom specifikacije nazvane SOAP (engl. *Simple Object Access Protokol* – jednostavni protokol za pristup objektima). Prvotna namjena ove specifikacije bila je normizacija postupka poziva udaljenih metoda (engl. *RPC – Remote Procedure Call*). Glavna ideja je maršaliranje parametara poziva u XML-ovski dokument, transport dokumenta te demaršaliranje na strani primatelja u oblik kojeg primatelj može razumjeti. [1]

Uskoro proizvođači softvera i poslovne tvrtke uočavaju potencijal ove norme koja uz jezik XML te internetske protokole predstavlja kvalitetan izgradbeni blok za razvoj distribuiranih aplikacija za elektroničko poslovanje zasnovanih na Webu. Ovaj koncept nazvan je "Usluge weba" (engl. *Web services*).

Ključni segment Usluga weba bilo je njihova pristupna točka – javno sučelje kojem se pristupa preko internetskih protokola. Zbog toga je jedna od prvih inicijativa bila razvoj jezika za opis toga sučelja, što rezultira specifikacijom WSDL (engl. *Web Services Description Language* – jezik za opis Usluga weba). Iako se Usluge weba ne ograničavaju na protokol SOAP, on je ipak postao jedan od najšire prihvaćenih protokola u konkretnim implementacijama tako da se često i poistovjećuje s pojmom Usluga weba.

Treća komponenta (pored komunikacijskog protokola te normizacije načina opisa usluga) koja je uokvirila specifikaciju Usluga weba jest specifikacija UDDI (*eng. Universal Description, Discovery and Integration)*. Ova specifikacija pružila je način stvaranja registara u kojima se mogu čuvati te taksonomski organizirati opisi Usluga koje mogu ali ne moraju biti locirane unutar granica tvrtke-vlasnika registra. No usprkos činjenici da se registar tipa UDDI smatra punopravnom komponentom specifikacije Usluga weba, sama industrija softvera ga za sada nije pretjerano dobro prihvatila tako da arhitektura SOA danas još uvijek registar usluga smatra uglavnom opcionalnom komponentom sustava.

Što se samih Usluga weba tiče, one su u prvom desetljeću 21. stoljeća odlično prihvaćene kako od strane proizvođača softvera tako i od poslovnih tvrtki: pojedine tvrtke razvile su usluge za različite poslovne primjene čija uporaba se može kupiti ili iznajmiti; rješenja za razmjenu poruka (tzv. *message-oriented middleware* ili MOM) uključila su protokol SOAP u svoja rješenja zajedno s postojećim komunikacijskim protokolima koja su koristila; velike poslovne tvrtke razvile su vlastita prilagođena rješenja zasnovana na Uslugama weba kao alternativu normi EDI (engl. *Electronic Data Interchange*); konačno, Usluge weba uzrokovale su veliku popularnost modela nove arhitekture za realizaciju informacijskih sustava – uslužno orijentirane arhitekture ili SOA.

[1] Norme SOAP, WSDL te Usluge weba bit će dodatno i detaljno objašnjene u poglavlju 2. koje se bavi pregledom najpopularnijih implementacijskih tehnologija za uslužno orijentirane arhitekture.

SOA, kao i svaka druga arhitektura, sa sobom je donijela i određena ograničenja i pravila. Usluge weba i XML su (za sada) glavna implementacijska tehnologija koja omogućuje izvedbu arhitektura SOA ali one kao takve zahtijevaju određene promjene i prilagodbe kako bi se mogle ispravno koristiti unutar granica arhitekture SOA. Neki primjeri ovakvih izmjena su:

- SOA zahtijeva usklađivanje reprezentacije podataka i normi modeliranja usluga kako bi se omogućila intrinzična interoperabilnost
- gotovo sve moderne implementacije SOA oslanjaju se na SOAP kao osnovni komunikacijski protokol; postupak modeliranja poslovnih dokumenata u pripadne XML-ovske dokumente i njihove XSD-vske sheme sada često mora uzimati u obzir i postojanje SOAP-a i njegovih karakteristika
- SOA stavlja naglasak na razmjenu poruka utemeljenu na dokumentima (engl*. document-style messaging*) što se razlikuje od početne primjene Usluga weba koja se uglavnom zasnivala na razmjeni poruka utemeljenoj na udaljenom pozivu metoda (engl*. RPC-style messaging*) – o čemu će više riječi biti u poglavlju 2.7. Prelazak na dokumentnu razmjenu znači izmjenu dizajna opisa usluga, karakteristika sučelja te prelazak na višu razinu zrnatosti uzeto s gledišta funkcionalnosti usluge.
- SOA preferira kontekstualno opširne inteligentne poruke umjesto usko definiranih ciljno usmjerenih poruka kakve prednjače kod poziva udaljenih metoda. Poruka treba imati sposobnost samostalne egzistencije, sadržavati dovoljno informacija kako bi olakšala vlastitu obradu kroz usluge i tako omogućila autonomiju usluga te isključila potrebu za čuvanjem njihovog stanja.
- karakteristike i zahtjevi suvremenih SOA (navedenih na kraju ovog poglavlja) diktiraju proširenje specifikacije Usluga weba kako bi se ti zahtjevi mogli ispuniti; rezultat ove inicijative je nastanak tzv. WS-proširenja (engl*. WS-extensions)* – skupa specifikacija koje Uslugama weba dodaju nove mogućnosti kao npr. mehanizme sigurnosti, razmjene metapodataka i sl.

1.4 SOA i normizacijske organizacije

SOA je utemeljena na normama – normizirana su sučelja, protokol, izvedbena okolina, poruke i sl. Ovo je na neki način jedinstveno u informatičkom svijetu, jer je gotovo svaka prethodna platforma zasnovana na konkretnoj arhitekturi realizirana u okolini određenoj od strane proizvođača softvera; SOA nudi arhitekturu koja nije ovisna o određenom proizvođaču ali svi proizvođači softvera mogu nuditi proizvode zasnovane na njoj koji – što je posebna prednost – se mogu međusobno integrirati.

Opravdano je postaviti pitanje nastanka normi i interesa grupa koje stvaraju norme; činjenica jest da nezavisne grupe kao što su W3C, OASIS i sl. pri razvoju normi aktivno surađuju s proizvođačima softvera (*Microsoft, IBM, Sun*) koji tako imaju vrlo veliki stupanj utjecaja na izgled konačne norme. Gledano s druge strane, norma ima šanse opstanka na tržištu samo ako dobije podršku velikih proizvođača softvera što stvara svojevrsnu povratnu vezu ali i kontradiktorne posljedice – otvorene norme čija je glavna prednost neovisnost o proizvođaču zapravo nastaju i opstaju upravo zahvaljujući velikim proizvođačima.

U sljedećim odlomcima bit će riječi o tijelima koji donose norme te o posljedicama utjecaja proizvođača softvera na iste.

1.4.1 W3C (*World Wide Web Consortium*)

W3C (engl. *World Wide Web Consortium*) je internacionalna organizacija za donošenje normi osnovana 1994. koja se smatra jednim od glavnih čimbenika transformacije *World Wide Web*-a iz infrastrukture povezanih računala u globalni semantički medij za razmjenu informacija. Uspjeh W3C-a započinje izdavanjem specifikacije jezika HTML bez kojeg je Internet današnjice nezamisliv. Kada se krajem prošlog stoljeća počinje uočavati potencijal Interneta kao glavne infrastrukturne tehnologije za omogućavanje modernog elektroničkog poslovanja, W3C donosi ključne norme koje bi takva rješenja podržali: specifikaciju XML te njezine srodne specifikacije, kao što su XML Schema i XSLT. Konačno, W3C je izdao specifikacije ključne za implementaciju Usluga weba – norme SOAP i WSDL.

W3C je poznata kao iznimno stroga i temeljita organizacija čije norme prolaze kroz mnoge revizije i recenzije koje se javno objavljuju kako bi se dobilo što više korisnih povratnih informacija. Temeljitost ima i svoju negativnu stranu – norme na objavu često čekaju i do nekoliko godina.

1.4.2 OASIS (*Orgranization for Advancement of Structured Information Standards*)

OASIS je također internacionalna organizacija zadužena za norme najviše poznata po preuzimanju WS-BPEL specifikacije, normi ebXML te doprinosu pri nastanku specifikacije UDDI. Organizacija OASIS je također uvelike doprinijela razvoju i proširenju XML-a kao norme, a trenutni veliki projekt je normizacija jedne od najvažnijih komponenti specifikacija proširenja Usluga weba – okvira WS-Security zaduženog za sigurnosne mehanizme.

Dok W3C najčešće preuzima ulogu razvoja jezgrenih normi, OASIS se primarno usredotočuje na proširenje tih normi te njihovu vertikalnu prilagodbu različitim industrijskim granama. Također, norme grupe OASIS imaju kraći rok do objave od W3C normi.

1.4.3 WS-I (*Web Services Interoperability Organization*)

WS-I se ne brine o stvaranju novih normi već o osiguranju da će krajnji cilj otvorene interoperatibilnosti u konačnici biti postignut. Počevši od 2002. godine kada je osnovan, ovaj konzorcij ima podršku od 200 organizacija uključujući neke od najvećih proizvođača softvera koji se bave implementacijama arhitektura SOA.

WS-I je najpoznatija po izdavanju tzv. *Osnovnog Profila* (engl. *Basic Profile*) – dokumenta koji predlaže skup normi koje bi trebali biti korišteni u kompoziciji kako bi se osigurala maksimalna moguća razina interoperabilnosti unutar sustava. Ovaj profil između ostalog sadržava i preporuku točno određenih verzija specifikacija WSDL, SOAP, UDDI, XML i XML Sheme te kao takav ima važnu ulogu kod adaptacija arhitekture SOA na konkretna poslovna rješenja (tvrtke koje provode adaptaciju obično naznače da je njihova adaptacija "suglasna Osnovnom Profilu"). Nedavno je izdano i proširenje Osnovnog Profila nazvano *Osnovni Sigurnosni Profil* (engl. *Basic Security Profile*) koji sadržava – prema mišljenju organizacije WS-I – skup najvažnijih tehnologija vezanih uz sigurnost XML-a i Usluga weba. Uz navedene Profile WS-I nudi niz primjera implementacija i alata koji testiraju suglasnost s Osnovnim Profilima.

WS-I teži jednakosti na tržištu proizvođača softvera zasnovanog na principima SOA i sve tvrtke koje doprinose njihovim prijedlozima su jednakopravne neovisno o svojoj veličini.

1.4.4 Veliki proizvođači softvera i njihov utjecaj na razvoj normi

U razvoju spomenutih normi značajnu ulogu imali su veliki proizvođači softvera kao što su *Sun, IBM, Oracle, Microsoft, Tibco, Hewlett-Packard, Canon, Verisign* itd. Interesi i doprinosi velikih tvrtki za navedene norme dovele su do nekih zanimljivih utjecaja na oblik samih normi i njihov utjecaj, pogotovo uzevši u obzir da usprkos zajedničkom cilju se radi o većinom konkurentskim tvrtkama od kojih svaka ima svoj ulog i interese glede tržišta softvera.

Uloga velikih tvrtki je dvojaka – s jedne strane one surađuju u razvoju normi, dok s druge prihvaćenost njihovih rješenja utječe kako na uspješnost norme tako i na profit same tvrtke. Zbog toga je uravnoteženje i optimizacija normi s politikom tvrtke i zahtjevima tržišta osnovna strateška zadaća svake tvrtke uključene u suradnju. Nije nepoznat slučaj da nekoliko velikih tvrtki osnuje vlastitu koaliciju i tako proširi svoj utjecaj unutar nezavisnih organizacija, iako se takve koalicije najčešće usredotočuju na pojedinu ciljnu specifikaciju od koje sve imaju zajedničku korist (npr. *IBM*, *Microsoft* i *BEA* su zajedničkim snagama izrazito podržavali razvoj i zaključenje više specifikacija vezanih uz tehnologiju proširenja Usluga weba – *WS-extensions*, kako bi njihova najnovija rješenja mogla implementirati neke ključne mehanizme koji trenutno nedostaju softverskim rješenjima zasnovanim na Uslugama weba). Ponekad takve koalicije djeluju kontraproduktivno – npr. *Sun* i *Oracle* su zajedno iznijeli specifikaciju za sigurnu isporuku poruka nazvanu *WS-Reliability*, da bi *Microsoft* i *IBM* gotovo u isto vrijeme izdali *svoju* specifikaciju za identičnu svrhu nazvanu *WS-ReliableMessaging*. Ovakvi primjeri pokazuju da autoritet organizacija za norme još uvijek nije toliko utjecajan da bi zaista mogao garantirati jednaku poziciju svima na tržištu te uistinu otvorene norme, te da će tržišna politika i konkurencija te utjecaj velikih tvrtki u velikoj mjeri utjecati na broj, oblik i prihvaćenost otvorenih normi i interoperabilnost modernih rješenja zasnovanih na arhitekturi SOA.

1.5 Usporedba SOA i ostalih arhitektura

Prvi računalni sustavi nisu koristili pojam "arhitekture", pošto je izvedba rješenja često bila očigledna i jednostavna tako da nije bilo koristi od apstrahiranja modela i definiranja posebnog modela u svrhu formalizacije opisa računalne infrastrukture.

S porastom višerazinskih aplikacija informatički djelatnici uočavaju potrebu upravo za modelima koji bi služili kao predložak za implementaciju informacijskih sustava – predložak koji bi definirao pravila, ograničenja, svojstva i granice koje bi se primjenjivale na sva rješenja koja slijede takav predložak. Tako je nastao pojam *aplikacijske arhitekture*.

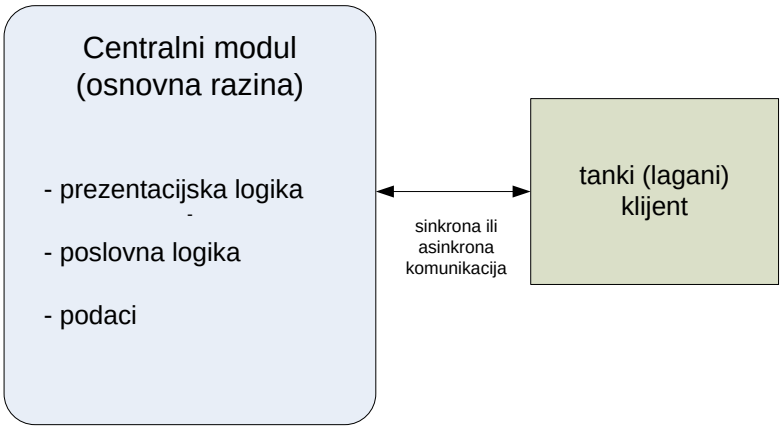
Aplikacijska arhitektura omogućuje pohranu informacija o informacijskim sustavima na različitim razinama – neki odjeli tvrtke zanimaju se samo za visoko apstrahirane i logički organizirane razine, dok drugi odjeli (npr. zaduženi za održavanje i razvoj sustava) imaju potrebu gledati sustav s niže razine tj. znati modele podataka koji se koriste, tokove informacija, sigurnosne mehanizme i sl. Može se postaviti analogija s građevinskim nacrtima nekog objekta – nacrti daju informacije o objektu a količina i detalji sadržani u nacrtu ovise o tipu nacrta te ciljanom korisniku.

Velike tvrtke često posjeduju više različitih informacijskih sustava a time i više aplikacijskih arhitektura. Zbog toga se javlja potreba za organiziranim praćenjem i upravljanjem svih tih sustava, tj. organizacijom aplikacijskih arhitektura u tzv. *poduzetničku arhitekturu (*engl*. enterprise architecture*). Ova arhitektura sagledava različite sustave kao jedinstveni informacijski sustav i može sadržavati globalne sigurnosne postavke, detalje o tehnološkim specifičnostima svake aplikacijske arhitekture itd.

Uslužno orijentirana arhitektura je u isto vrijeme i evolucija pojmova aplikacijske/poduzetničke arhitekture ali i arhitektura čiji opseg može obuhvaćati i jednu i drugu arhitekturu, ovisno o strateškim planovima i implementacijskim zahtjevima. Uslužno orijentirana arhitektura može uključivati cjelokupni informacijski sustav tvrtke pa i sustave njezinih partnera (čak i sustave tvrtki s kojima nije u suradničkom odnosu ako uzmemo u obzir svojstvo otkrivanja usluga), ali isto tako može se odnositi samo na specifični podsustav tvrtke koji je organiziran kao skup međusobno povezanih usluga. Razlika je samo u konceptu i stupnju zrnatosti koje usluge obuhvaćaju.

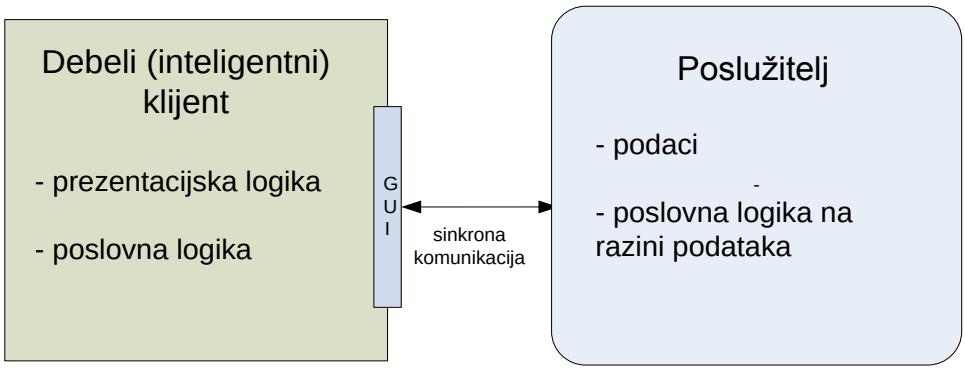
Gledano s funkcionalnog stajališta, arhitektura sustava se često modelira prema tzv. razinama (engl*. tier*). Razina u ovom slučaju opisuje funkcionalni segment sustava izdvojen od ostatka kojem se pristupa preko definiranog sučelja.

Jedna od najranijih takvih arhitektura je tzv. **jednorazinska arhitektura,** gdje su sve funkcije objedinjene u centralnom ili glavnom modulu (engl*. mainframe*). Sliku ovakve arhitekture prikazuje **Slika 1-5.**



Slika 1-5 - Jednorazinska arhitektura

U ovakvom sustavu centralni modul zadužen je za pohranu podataka, njihovo dohvaćanje, provođenje poslovnih procesa nad podacima te prezentaciju podataka klijentima. Korisnik sustavu pristupa preko tzv. "tankog" ili "laganog" klijenta (engl*. thin/dumb client*) koji sadržava minimalnu procesorsku moć i osnovne funkcije spajanja na centralno računalo. Ovakav monolitički pristup prevladavao je u računalnim sustavima sve do 80-tih godina prošlog stoljeća kada započinje uspon **dvorazinske arhitekture** (**Slika 1-6**).



Slika 1-6 - Dvorazinska arhitektura

Dvorazinska arhitektura donosi pojam tzv. "debelog" klijenta (engl. *fat client*) tj. koncept gdje klijent preuzima dio odgovornosti izvedbe poslovne logike i obrade informacija. Centralni poslužitelj je još uvijek prisutan, no on uglavnom ima odgovornosti pohrane podataka te izvođenja poslovne logike vezane uz obrade podataka niže razine; klijent preuzima veći dio aplikacijske poslovne logike. Prezentacija podataka i njihova obrada olakšana je pojavom grafičkih sučelja (engl. *GUI – Graphic User Interface*).

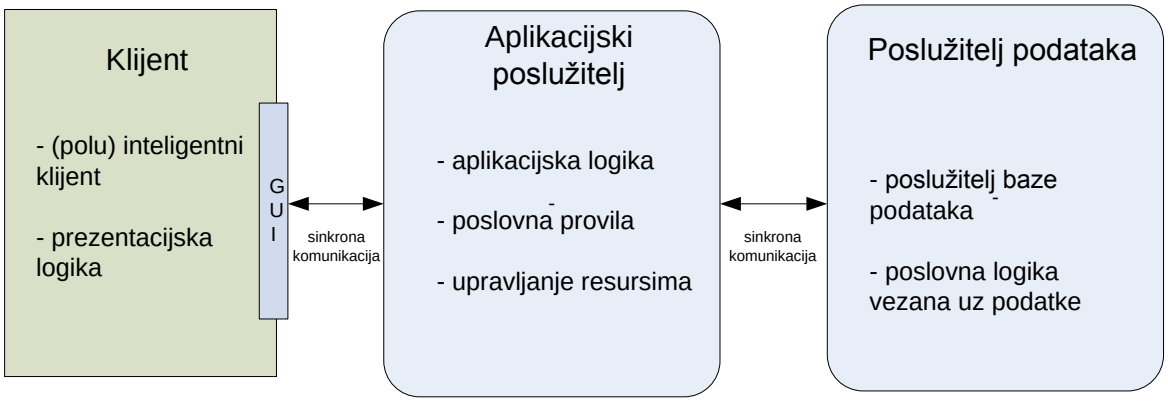
U usporedbi s arhitekturom SOA, ova arhitektura ima izvjesne mane. Velika količina poslovne logike traži i određenu količinu resursa na klijentskoj platformi, dok distribuirana priroda arhitekture SOA omogućuje raspodjelu poslovnog opterećenja kroz poslužitelje koji izvode tražene usluge. Nadalje, klijenti koji se spajaju na istu bazu podataka mogu prouzročiti usko grlo i usporavanje sustava; usluge apstrahiraju pristup resursima i mogu koristiti mehanizme koji ublažavaju ovakve probleme. Poruke u dvorazinskoj arhitekturi su najčešće usko definirane s minimalnim kontekstom integriranim u poruku zbog toga što je primatelj strogo određen. Uslužno orijentirana arhitektura koristi autonomne poruke s velikom količinom kontekstualnih informacija što omogućuje puno veću fleksibilnost i lakšu obradu, pogotovo u asinkronom načinu rada. Dvorazinske arhitekture su često vrlo usko povezane s platformom i aplikacijom koja ih provodi dok otvorene norme omogućuju izvjesnu platformsku neovisnost arhitektura SOA.

S druge strane, arhitektura *klijent-poslužitelj* ima i svoje prednosti, pretežno po pitanju sigurnosti. Održavanje sigurnosti u dvorazinskom sustavu je relativno jednostavno zbog malog broja pristupnih točaka i strogo definiranog platformski zatvorenog sučelja. Uslužno orijentirana arhitektura u svojoj "otvorenosti" donosi i niz sigurnosnih problema koje je potrebno riješiti (čime se npr. bavi jedno od najvažnijih proširenja specifikacije Usluga weba – *WS-Security*). Isto tako, administracija sustava zasnovanom na uslugama zahtijeva puno složenije alate nego što je potrebno za jednostavnije sustave zasnovane na principu *klijent-poslužitelj*.

Krajem 90-tih pojavljuje se novi koncept – **distribuirana arhitektura**. Troškovi i ograničenja arhitekture klijent-poslužitelj doveli su do ideje modularnih aplikacija i sve većeg prelaska na objektno-orijentirana

rješenja. Distribucija aplikacijske logike na više komponenti (od kojih su neke na klijentu, neke na poslužitelju) omogućila je bolju raspodjelu resursa no dovela i dodatne probleme duljeg razvojnog ciklusa i kompleksnije načine upravljanja sustavima.

Klasičan tip distribuirane arhitekture je **trorazinska arhitektura** (**Slika 1-7**).



Slika 1-7 - Trorazinska arhitektura

Kod ovakve arhitekture klijent najčešće preuzima zadatke vezane uz što bolju prezentaciju podataka te omogućavanje izvršavanja poslovnih metoda nad podacima kroz adekvatno interaktivno sučelje. Na srednjem sloju nalazi se tzv. aplikacijski poslužitelj koji je orijentiran na provedbu poslovnih procesa, poslovnu logiku te upravljanje vezama prema resursima. Treći sloj se bavi upravljanjem podacima – pohranom, dohvaćanjem i obradom nižeg sloja i tu se najčešće postavlja zasebni poslužitelj s bazom podataka.

Popularnost Interneta uvela je pojam **distribuirane internetske arhitekture** i izvedbu srednjeg sloja kroz tzv. Web poslužitelj – ovo je omogućilo zamjenu nenormiziranih protokola udaljenog pristupa metodama vrlo dobro prihvaćenim protokolom HTTP.

Iz gledišta fizičke implementacije, distribuirana internetska arhitektura vrlo je bliska uslužno orijentiranoj arhitekturi – poslovna logika nalazi se na poslužiteljskim komponentama kojima se pristupa preko normiziranih protokola. Ključna razlika je u tipu povezanosti poslužiteljskih komponenti – kod tradicionalnih internetskih arhitektura poslužiteljske komponente predstavljaju usko povezani i relativno zatvoreni sustav, čija je funkcionalnost monolitna i definirana svrhom poslovne aplikacije čije zahtjeve poslužuje. Uslužno orijentirana arhitektura je otvoreni sustav čija je svaka komponenta u izvjesnoj mjeri autonomna a njezina svrha definirana je funkcijom koju obavlja, bez potrebe za znanjem o procesu ili sustavu unutar kojeg se ta funkcionalnost koristi. Isto tako, komunikacija između poslužiteljskih komponenti provodi se na već spomenutom principu razmjene dokumenata temeljenom na normiziranim protokolima, za razliku od principa pozivanja metoda i prilagođenih protokola koje koriste tradicionalne poslužiteljske aplikacije.

1.6 Usporedba uslužno-orijentiranih i objektno-orijentiranih principa

Koncepti "uslužne orijentiranosti" i "objektne orijentiranosti" imaju dosta zajedničkih točaka; štoviše, objektno-orijentirano programiranje se uglavnom i koristi za izgradnju aplikacijske logike unutar Usluga weba, jedne od glavnih implementacijskih tehnologija uslužno orijentirane arhitekture. No objektno-orijentirana metodologija ima i izvjesne različitosti od uslužno orijentiranih principa. Neke od njih nabrojane su u nastavku.

I objektno-orijentirani i uslužno-orijentirani principi se temelje na izgradnji ponovno iskoristivih modularnih komponenti, no objektna orijentiranost ipak rezultira jače povezanim logičkim jedinicama od "slabo" povezanih usluga. Usluga se često može vrlo lako sagledati izdvojeno – ona ima svoju funkciju te svoje ulazno/izlazne parametre. Iako je ona glavni izgradbeni blok uslužno orijentirane arhitekture njena samostalnost i autonomnost je neosporna. Objekt se gotovo nikad ne gleda samostalno, njegova logička povezanost s objektima unutar njegove izvedbene okoline je često jedna od ključnih karakteristika njegove funkcije i uloge unutar sustava.

Uslužna orijentiranost preporučuje stvaranje usluga široke zrnatosti, tj. široko definirane funkcionalnosti; objekti najčešće funkcioniraju na finijim razinama zrnatosti s uže definiranim metodama. Sagledanjem usluga uže definiranih opsega često dolazimo do razina na kojima se nalaze konkretni objekti koji implementiraju same usluge, dok su kod usluga šire definiranih opsega konkretni objekti često nevidljivi jer se sustav sagleda s više apstrakcijske razine.

Objektno-orijentirani principi nalažu logičku integraciju podataka i metoda za njihovu obradu. Uslužno orijentirani principi s druge strane razgraničuju poslovnu logiku sadržanu u uslugama i podatke - često autonomne i inteligentne informacijske jedinice koje sadržavaju zasebne informacije za pomoć pri svojoj obradi.

Konačno, objektno-orijentirani principi često nalažu čuvaju stanje objekata, tj. funkcionalnost objekata vodi do toga da oni "posjeduju stanje" (engl*. stateful objects*). Objekt zadržava svoje "stanje" dok god to ima smisla za poslovni proces u kojem sudjeluje što ponekad zahtijeva i pronalaženje mehanizama za trajnu pohranu stanja jer bez podataka o stanju objekt gubi svoj smisao. Za usluge se često preporučuje da ne čuvaju stanje jer se konkretne informacije ne zadržavaju u uslugama, već u poslovnim dokumentima koji se razmjenjuju u obliku poruka. Usluga obrađuje podatke, ali ih ne čuva; funkcionalnost usluge očituje se kroz analizu dokumenata koje je ta usluga vratila kao izlazne parametre. Sama usluga se potom vraća u početno "stanje" - nakon obrade ona više nema memorije o prethodnim obradama.

Dakle, objektno orijentirani i uslužno orijentirani principi imaju dodirnih točaka, ali i dovoljno različitosti kako bi omogućili smislenost njihove diferencijacije te različitih scenarija uporabe.

1.7 Modeli usluga u uslužno orijentiranoj arhitekturi

Kod uslužno orijentiranih arhitektura često ima smisla uvesti neki princip klasifikacije i kategorizacije usluga, tj. definiranja tipa usluga ovisno o nekim njezinim svojstvima. Kategorizacija se npr. može provesti prema prepoznatoj ulozi koju usluga preuzima u procesu ili prema funkcionalnosti koju obavlja (tzv. "model" usluge). U ovom poglavlju bit će dan primjer ova dva tipa kategorizacije.

Usluga se često teško može trajno klasificirati kroz standardne pojmove kao što su "klijent" ili "poslužitelj", jer se kroz izvedbu poslovnog procesa uloga usluge često mijenja – ona može biti inicijator procesa, dio komunikacijskog kanala, krajnja točka komunikacije itd. No usprkos tome moguće je definirati neke temeljne uloge koje usluga privremeno preuzima tokom izvođenja poslovnog zadatka i koje pomažu pri prepoznavanju funkcionalnosti usluge i njezinog mjesta u poslovnom procesu.

U nastavku slijede neke od važnijih temeljnih uloga koje usluge privremeno (ili – rjeđe – trajno) preuzimaju.

1) *Pružatelj usluge* (engl. *service provider*)

Ovu ulogu usluga preuzima pod sljedećim uvjetima:

- usluga je objavila svoju funkcionalnost pomoću nekog normiziranog opisa koji je korisnicima dostupan na određeni način i koji definira informacije o svojstvima i ponašanju usluge
- usluga je pozvana od strane vanjskog izvora (potražitelja usluge)

U ovom slučaju usluga ima identičnu ulogu poslužiteljskog sloja kao u klasičnoj dvorazinskoj arhitekturi *klijent-poslužitelj*. Često se uloga definira i kao *agent pružatelja usluge* kako bi se uvela distinkcija između same usluge i tvrtke koja "nudi" dotičnu uslugu.

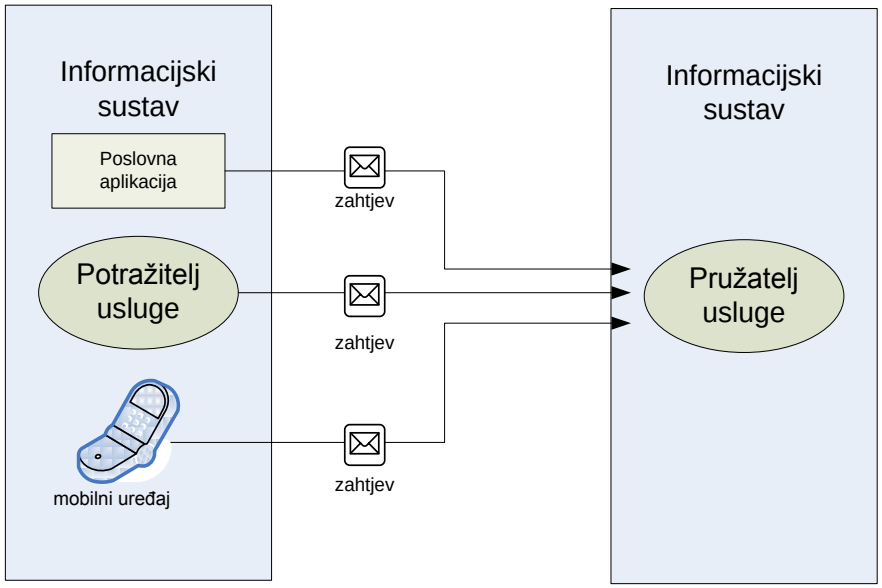
2) *Potražitelj usluge* (engl. *service requestor)*

Ovu ulogu usluga preuzima pod sljedećim uvjetima:

- usluga na određeni način saznaje ili otkriva informacije o usluzi koju želi pozvati
- usluga šalje poruku pružatelju usluge

Potražitelj je prirodni komplement pružatelju usluge i analogni koncept klijenta iz dvorazinske klijent-poslužitelj arhitekture. Isto tako, pojam potražitelja se često proširuje frazom *agent potražitelja usluge* kako bi se semantički odvojio od korisnika ili tvrtke koja na određeni način inicira zahtjev ili poslovni proces koji dovodi do potražnje određene usluge.

Važno je napomenuti da potražitelj usluge ne mora biti druga usluga; to može biti poslovna aplikacija, aplikacija mobilnog uređaja i sl. (**Slika 1-8**).



Slika 1-8 - Potražitelji i pružatelj usluge

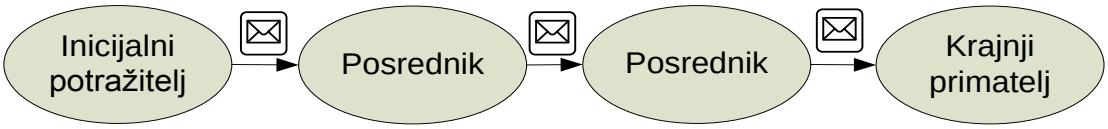
3) Posrednik

Poruka poslana od strane potražitelja namijenjena pružatelju usluge ne mora do njega stići direktnim putem – komunikacija u sustavu SOA često uključuje različite putove kojima poruka može prolaziti te posrednike koji mogu ali ne moraju na neki način izmjenjivati poruku na njezinom putu do odredišta.

Posrednik je usluga koja prima poruku ali nije njezin krajnji primatelj, već ju nakon obrade prosljeđuje sljedećoj komponenti u komunikacijskom toku. Postoje dva tipa posrednika: *pasivni posrednik* čija je uloga uglavnom usmjeravanje poruke na njezinu sljedeću lokaciju. Ona može analizirati i eventualno mijenjati zaglavlje poruke ako to komunikacijska infrastruktura i politika preusmjeravanja zahtijeva, ali sama poruka i pogotovo njezin sadržaj se ne mijenjaju. *Aktivni posrednik* obavlja identičnu funkciju kao i pasivni, ali prije prosljeđivanja poruke on je analizira i eventualno na određeni način mijenja. Aktivni posrednik najčešće vrši izmjene u zaglavlju pa čak i potpuno briše određene retke zaglavlja ako to poslovni proces nalaže.

4) *Inicijalni pošiljatelj i krajnji primatelj*

Ove uloge su zapravo specijalni slučajevi potražitelja i pružatelja usluge, no često se moraju posebno identificirati jer u semantičkom smislu predstavljaju početak i završetak poslovnog procesa. Inicijalni pošiljatelj odgovoran je za iniciranje poslovnog procesa, dok krajnji primatelj predstavlja završnu točku i predviđeni kraj provedbe procesa (**Slika 1-9**). Važno je napomenuti kako nema razloga da inicijalni pošiljatelj i krajnji primatelj ne budu ista usluga, ili da proces ima više krajnjih točaka ovisno o konkretnoj provedbi.



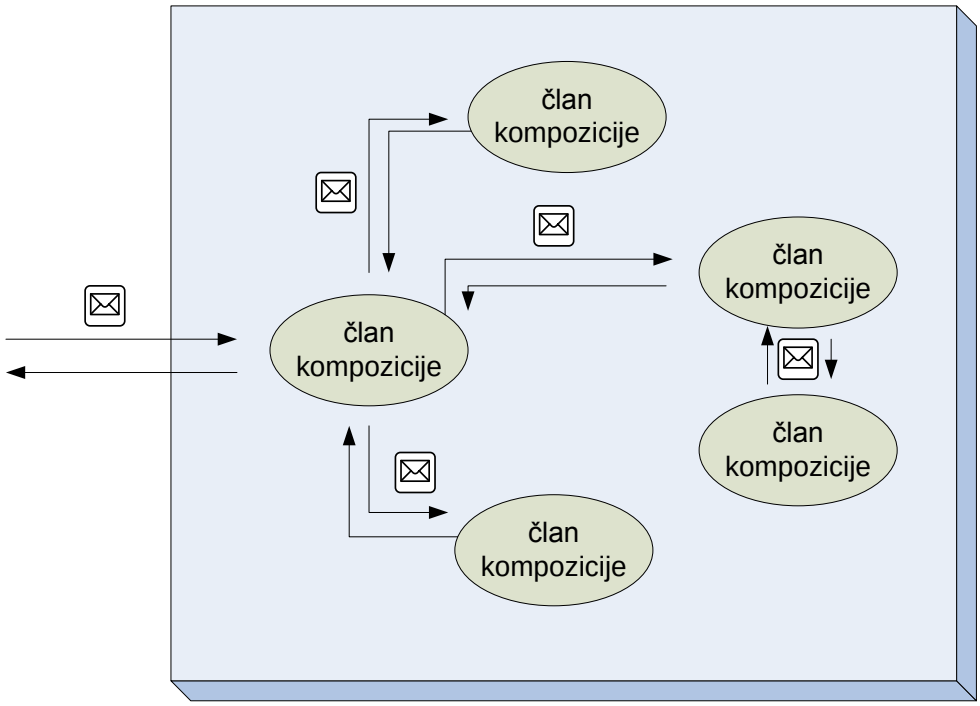
Slika 1-9 - Tok poslovnog procesa

5) Član kompozicije usluga

Pojam *kompozicije usluga* označava skup usluga koje su grupirane u određenu logičku jedinicu. Poslovni zadaci često zahtijevaju sudjelovanje više usluga u obradi poslovnih podataka vezanih uz konkretni zahtjev tako da svaka pojedina usluga preuzima jedan smisleni dio posla. Usluga tako preuzima ulogu *člana kompozicije usluga.*

U tipičnom slučaju svaka usluga mora biti dizajnirana na takav način da omogući usluzi sudjelovanje u kompoziciji čak i ako njezina inicijalna izvedba nema u vidu takav tip sudjelovanja. Svojstvo kompozitivnosti jedno je od inherentnih svojstava uslužno orijentirane arhitekture. Danas zbog toga već postoje i vrlo dobro prihvaćena rješenja za organizaciju usluga u ulančane poslovne procesa kao što je npr. specifikacija WS-BPEL (engl. *Web Services – Business Process Execution Language*) izdana od strane grupe OASIS.

Slika 1-10 prikazuje kompoziciju usluga sačinjenu od pet članova kompozicije.



Slika 1-10 - Kompozicija usluga

Navedene uloge nisu usko povezane sa samom funkcionalnosti koju usluga pruža svojim korisnicima – one su samo generički opisi stanja u kojima se usluga nalazi unutar nekog konteksta. Kao što je već rečeno, klasifikacije usluga prema načinu na koji se one koriste nazivaju se *modelima usluga.* U nastavku su kratko opisani neki od osnovnih modela.

1) *Poslovne usluge*

Poslovna usluga je osnovni izgradbeni blok unutar uslužno orijentirane arhitekture. Ona predstavlja točno određeni skup poslovne logike omeđen dobro definiranim funkcionalnim granicama. Potpuno je autonomna ali i predviđena za korištenje u sklopu s drugim uslugama.

Poslovne usluge uključuju:

- imenovanu reprezentaciju poslovne logike
- reprezentaciju poslovnog entiteta ili imenovanog skupa poslovnih informacija
- definirane poslovne procese
- članove kompozicija usluga

2) *Pomoćne usluge (engl. utility services)*

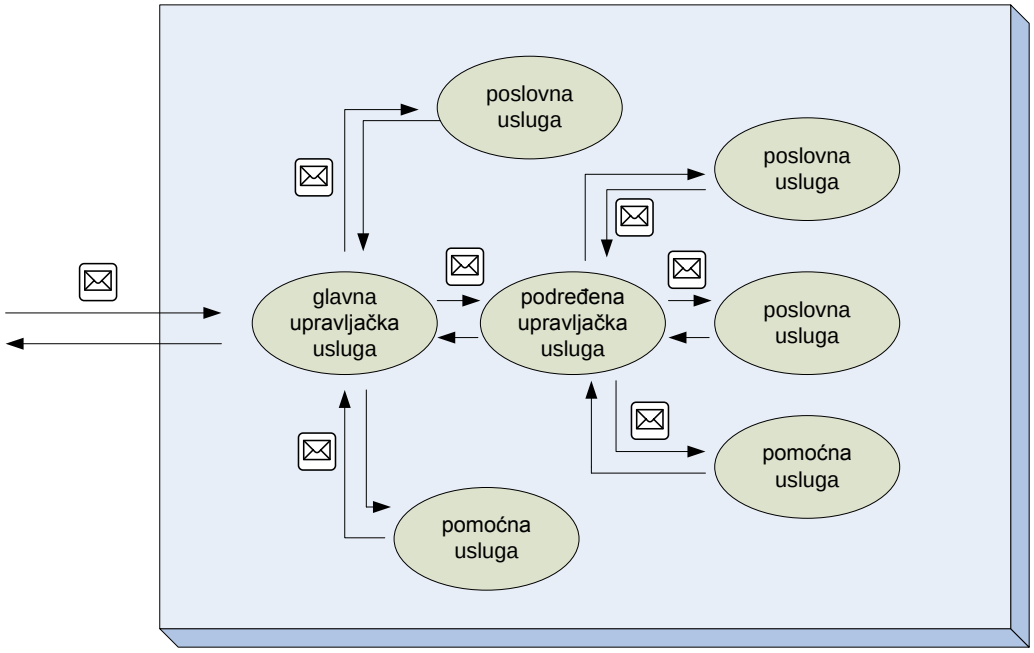
Pomoćni model uključuje generičke usluge koje se mogu ponovno iskorištavati za različite primjene. Ovakve usluge:

- nisu vezane za određenu usku primjenu ili aplikaciju
- najčešće služe kao posrednici
- potpomažu intrinzičnu interoperabilnost same arhitekture
- imaju najveći stupanj autonomije

3) *Upravljačke usluge*

Kompozicija usluga predstavlja skup nezavisnih usluga od kojih svaka doprinosi izvedbi sveukupnog poslovnog zadatka. Organizacija i upravljanje ovih usluga je sama po sebi zadatak čije izvršenje može biti funkcija posebne usluge; ovakva *upravljačka usluga* preuzima ulogu upravljača procesom i ima zadaću omogućiti ostalim uslugama izvršavanje svoje uloge u sklopu procesa.

Upravljačke usluge podržavaju principe mogućnosti slaganja usluga u kompoziciju, ponovne iskoristivosti te autonomnosti usluga. Isto tako upravljačke usluge se mogu organizirati u hijerarhijsku strukturu pa tako i same biti dio kompozicije usluga. **Slika 1-11** prikazuje kompoziciju usluga čiji su članovi deklarirani kao poslovne, pomoćne ili upravljačke usluge.



Slika 1-11 - Kompozicija upravljačkih, poslovnih i pomoćnih usluga

1.8 Neposredne prednosti i mane uslužno orijentiranih arhitektura

Opravdano je postaviti pitanje konkretne prednosti koje arhitekture SOA nude modernim informacijskim sustavima, poglavito onima usmjerenim na elektroničko poslovanje.

Jedna od ključnih prednosti je svojstvo intrinzične interoperabilnosti i lake integracije sustava. Jedan od vodećih problema poslovnih sustava današnjice je nepostojanje zajedničkih normi i mnoštvo zatvorenih poslovnih aplikacija koje se ne mogu na jednostavan način povezivati i međusobno komunicirati te čija je funkcionalnost ograničena na pristup kroz strogo definirana prilagođena korisnička sučelja. Jednostavna mogućnost integracije različitih komponenti sustava i adaptacija otvorenih normi mogu troškove integracije svesti na minimum te umnogome olakšati kako izvedbu poslovnih procesa unutar tvrtke, tako i provedbu poslovanja između tvrtki.

Ponovna iskoristivost komponenti opravdava ulaganje u informacijske resurse ovakvog tipa jer postoji potencijal njihovog budućeg korištenja bez potrebe za nabavkom ili izgradnjom novih komponenti slične ili iste funkcionalnosti.

Proširivost sustava omogućuje laku nadogradnju sustava prema novim poslovnim zahtjevima i potrebama bez potrebe za opsežnim reinžinjeringom sustava. Npr. tzv. WS-proširenja omogućuju nadogradnju postojećeg sustava SOA zasnovanom na Uslugama weba mehanizmima sigurnosti, pouzdanosti i sl.

Oslonac modernih SOA sustava na tehnologiju XML omogućuje tvrtkama lakši prelazak na XML-ovsku reprezentaciju poslovnih podataka te kao direktni rezultat toga i lakšu konsolidaciju poslovnih informacija. XML-ovski dokumenti te pripadne XML-ovske sheme omogućuju lako upravljanje, pohranu i

analizu poslovnih podataka. Do sada je prelazak na XML-ovsku reprezentaciju bilo teže opravdati bez infrastrukture koja bi takvu reprezentaciju mogla učinkovito koristiti, no postojanje SOA arhitektura koja nju najčešće zahtijeva kao preduvjet uvelike opravdava uvođenje takvog modela.

Troškovi povezani uz skalabilnosti informacijske infrastrukture su uvelike smanjeni, pošto se ulaže u jedinstvenu komunikacijsku tehnologiju zasnovanu na otvorenim normama. Konačno, slaba povezanost omogućuje tvrtkama izbor proizvođača i implementacijske tehnologije prema vlastitim potrebama, zahtjevima i mogućnostima.

Unatoč navedenim prednostima, prilagodba na principe SOA može dovesti i do negativnih posljedica, pogotovo ako se ista uvodi bez potrebnih predznanja i procjena svih mogućih utjecaja na postojeći poslovni sustav.

Česta greška kod adaptacije na sustav SOA je pristup od-dna-prema-gore (engl. *bottom-up approach*) gdje se implementacija arhitekture SOA svodi samo na proširenje postojećih aplikacija pomoću sučelja Usluga weba dok se tipovi podataka koje one koriste direktno preslikavaju u XML-ovske sheme i navode kao parametri metoda usluge. Rezultat ovoga može biti skup usluga koje unatoč korištenju istih normi nema konzistentno definirane tipove podataka za razmjenu te je s time integracija otežana a ulaganje u sustav SOA neopravdano.

Nadalje, sustav SOA nužno sa sobom donosi i određeni utjecaj na performanse sustava. Mapiranje podataka na XML-ovsku reprezentaciju, provjera valjanosti i obrada XML-ovskih dokumenata često mogu negativno utjecati na performanse cijelog sustava. Zbog toga se arhitektura SOA treba uvoditi postupno i evolucijski, uočavajući potencijalne probleme i uska grla koja najviše utječu na smanjenje performansi i lošu učinkovitost sustava.

Usluge weba predstavljaju jedno od najraširenijih implementacijskih tehnologija SOA arhitekture, ali one u izvornoj inačici ne sadržavaju sigurnosne mehanizme. Otvaranje poslovnih aplikacija kroz sučelja Usluga weba može nenamjerno stvoriti sigurnosne rupe unutar sustava te kompromitirati osjetljive informacije unutar poslovnog sustava i omogućiti neovlašteni pristup informacijskim resursima tvrtke. Zbog toga je ključno unaprijed ocijeniti sigurnosne zahtjeve nad sustavom i pronaći adekvatno rješenje njihove implementacije.

1.9 Karakteristike modernih SOA sustava

Popularnost uslužno orijentiranih arhitektura i industrijski trendovi prvog desetljeća 21. stoljeća doveli su do evolucije temeljnog modela SOA i pojave skupa karakteristika koje sve više postaju važni preduvjeti adaptacije uslužno orijentirane arhitekture u konkretan informacijski sustav. Ovi moderni principi često se mogu pronaći pod skupnim imenom "suvremenog modela SOA".

Suvremeni model SOA zadržava postojeća svojstva uslužno orijentiranih arhitektura ali ih nadopunjuje brojnim dodatnim svojstvima koje su nastale kao rezultat zahtjeva modernih poslovnih informacijskih sustava. Navest će se neka od važnijih svojstava te kratki opisi obrazloženja njihovog nastanka i učinka na uslužno orijentirane sustave.

Mehanizam kvalitete usluge

Nove tehnologije elektroničkog poslovanja nužno uključuju način uspostave razine kvalitete usluge (engl. *QoS – Quality of Service*). Zahtjevi nad kvalitetom usluge mogu biti:

- sigurnosni mehanizmi, npr. zaštita pristupa sadržaju poruke ili određenoj usluzi
- pouzdanost, tj. sigurnost da će poruka stići na odredište ili da će greška biti objavljena na adekvatan način
- performanse sustava u skladu s očekivanjima, tj. informacijska infrastruktura koja ne smije inhibirati provođenje poslovnih zadataka u traženim vremenskim okvirima
- transakcijske mogućnosti tj. zaštita integriteta skupa poslovnih zadataka koji se moraju ili uspješno obaviti ili potpuno obustaviti

Pošto se često događaju scenariji gdje se usluga koja se koristi nalazi *izvan* poslovnog sustava, moraju postojati mehanizmi garancije da će rezultat korištenja te usluge biti na adekvatnoj razini koju zahtjeva poslovni informacijski sustav – potražitelj usluge. Drugim riječima, nije dovoljna implementacija tražene funkcionalnosti, već i garancija da će usluga zahtijevani posao napraviti na pravovremen, robustan i kvalitetan način.

Autonomija temeljne razine uslužno orijentirane arhitekture

Sustav uslužno orijentirane arhitekture mora dozvoljavati autonomiju kako usluga tako i poruka koje se razmjenjuju unutar sustava. Usluge imaju apsolutnu kontrolu nad poslovnom logikom koju implementiraju a poruke sadržavaju dovoljnu količinu informacijske "inteligencije" kako bi u ograničenom smislu mogle upravljati načinom svoje obrade.

Autonomnost je potrebno proširiti i na više razine apstrakcije; npr. aplikacija koja se sastoji od većeg broja autonomnih usluga također može egzistirati kao autonomni entitet koji ima slobodu upravljanja nad svojim komponentama, konkretno uslugama "niže" razine.

Otvorene norme

Integracija kroz distribuirane usluge te daljnja evolucija sustava moguća je samo uz dobro odabranu infrastrukturu zasnovanu na normama. Globalno prihvaćene otvorene norme predstavljaju ključnu karakteristiku modernih uslužno orijentiranih sustava – pomoću njih usluge imaju već unaprijed definiran "ugovor" o komunikaciji (protokoli, tipovi podataka i sl.) što uvelike olakšava uspostavu razmjene poruka bez potrebe za prethodnim usuglašavanjem međuovisnosti normi. Isto tako, otvorenost normi onemogućuje isključivost dobavljača (engl. *vendor lock-in*) gdje odabir sustava zasnovanom na prilagođenim normama određenog proizvođača softvera onemogućuje njegovu integraciju s informacijskim resursima koji nisu razvijeni od istog izvora.

Otvorene norme današnjice (SOAP, WSDL, XML, XML Schema) omogućuju programerima usredotočivanje na implementaciju usluge te jednostavno "uključenje" u uslužno orijentirani sustav dodavanjem normiziranog sučelja, bez potrebe za *a priori* znanjem o samom sustavu tj. njegovim pojedinim komponentama.

Fleksibilnost odabira proizvođača

Kao što je rečeno u prethodnoj točki, otvorene norme i slaba povezanost usluga onemogućuju isključivost dobavljača, što korisnicima i programerima usluga daje slobodu odabira platformi i softverskih proizvoda prema vlastitim kriterijima i potrebama bez ograničenja na proizvođače softvera preostalih komponenti poslovnog sustava. Rezultat ovoga je raznovrsnije i kvalitetnije tržište softvera te mogućnost jednostavne integracije konkurentskih sustava bez potrebe za izgradnjom usko definiranih prilagodbenih proizvoda.

Važnost aspekta otkrivanja usluga

Klasični model uslužno orijentirane arhitekture neizostavno uključuje registar usluga koji služi kao svojevrsni "katalog" ili "žute stranice" s opisima usluga, njihovih lokacija te metoda koje podržavaju. Iako bitna komponenta modela, prve implementacije (poglavito kroz tehnologije Usluga weba) taj su element smatrali opcionalnim te se integracija kroz usluge najčešće provodila kroz *a priori* znanje o drugim uslugama, ne kroz pronalaženje istih uz pomoć registara.

Suvremeni principi SOA naglašavaju važnost registara usluga kod implementacije razgranatih modela uslužno orijentiranih arhitektura. Potpuna integracija upravo zahtijeva mogućnost dinamičkog pronalaska usluge prema traženim zahtjevima, no u trenutnim implementacijama ovaj koncept se još uvijek pokazuje prezahtjevan za realizaciju funkcionalnih poslovnih sustava zasnovanih na modelu SOA.

Intrinzično svojstvo interoperabilnosti

Intrinzična interoperabilnost znači da se komponenta može integrirati u sustav bez potrebe za ulaganjem truda da se ta mogućnost integracije omogući. Prateći otvorene norme i principe uslužno orijentiranih arhitektura komponenta će imati inherentnu karakteristiku lake integracije sa sličnim komponentama neovisno o činjenici da li zahtjevi za integracijom ili sustav unutar kojeg bi se trebala integrirati uopće postoje.

Implicirana ponovna iskoristivost

Jedna od značajnih karakteristika uslužno orijentirane okoline je ponovna iskoristivost njezinih komponenti, čak i ako same komponente nisu razvijene s idejom ponovne iskoristivosti. Funkcionalnost neke usluge često se može iskoristiti u nekom drugom poslovnom procesu pored onog kojem ona matično pripada, a ugradnja iste u takav proces je jednostavno omogućena izvedbom samoga sustava. Nadalje, kompozicijom usluga mogu se graditi nove usluge šire funkcionalne definicije koje se također mogu ponovno koristiti na razini poslovnog sustava tvrtke.

Podrazumijevana proširivost sustava

Elementi uslužno orijentiranih sustava svoju funkcionalnost otvaraju pomoću normiziranih sučelja. Sučelja koncipirana pomoću principa slabe povezanosti komponenti tj. svojevrsna neovisnost sučelja o implementaciji omogućuje transparentnu proširivost sustava bez potrebama za izmjenom samih sučelja. Nadalje, apstrakcija usluga kroz "usluge višeg sloja" također omogućuje rekompoziciju i proširenje usluga nižeg sloja dok samo sučelje više razine može ostati potpuno neizmijenjeno.

Različite razine apstrakcije

Kao što je već rečeno, usluge se mogu ukomponirati u usluge višeg sloja i šire funkcionalne definicije. Logičkom organizacijom sustava na ovakvom principu mogu se stvoriti različite hijerarhijske razine apstrakcije koje omogućuju fleksibilnu definiciju pristupnih točki sustava prema poslovnim zahtjevima i potrebama tvrtke.

Podrška modernom dinamičkom poslovanju

Poslovanje današnjice je dinamično i sklono brzim promjenama. Dosadašnji informacijski sustavi često su se smatrali inertnima, tj. nedovoljno fleksibilnima za praćenje stalnih promjena. Sustav SOA sa svojim sposobnostima proširivosti, slabe povezanosti i lake integrabilnosti puno više odgovara konceptu modernog poslovanja od klasičnih sustava usko definirane funkcionalnosti i inherentne zatvorenosti unutar vlastite izvedbene okoline.

Uslužno orijentirana arhitektura kao glavni izgradbeni blok za sustave elektroničkog poslovanja

SOA predstavlja otvoreni sustav koji nije ograničen na samostalnu egzistenciju već nudi niz mogućnosti proširenja i integracije prema ostalim sustavima, kako unutar tvrtke koja ga koristi tako i na globalnoj razini. Zbog toga se SOA često smatra prvim korakom prema realizaciji tzv. uslužno orijentiranog poduzetništva (engl. *SOE – Service Oriented Enterprise*) gdje se cjelokupno poslovanje gleda kao logičku, ali i fizičku strukturu niza poslovnih procesa sastavljenih kompozicijom različitih usluga.

Kao što je već rečeno, SOA može implementirati različite razine apstrakcije tako da u krajnjem slučaju i sama SOA aplikacija može predstavljati jednu uslugu unutar poslovnog sustava, tj. uslugu unutar SOE okoline.

Suvremena SOA je evolucija dosadašnjih informacijskih sustava

Dosadašnji informacijski sustavi evoluirali su od centraliziranih modela, preko klijent/poslužitelj paradigme i distribuiranih arhitektura do orijentiranosti uslugama. SOA arhitektura umnogome koristi isprobane elemente dosadašnjih modela uvodeći principe otvorenosti, distribucije aplikacijske logike i ponovne iskoristivosti svojih komponenti, što se može smatrati važnim evolucijskim korakom u razvoju informacijskih sustava. No važno je napomenuti da proces evolucije nije završen te da će SOA principi i implementacijske platforme vjerojatno proći kroz mnoge promjene i prilagodbe do njezinog prihvaćanja kao zrele i univerzalno iskoristive arhitekture.

Uzevši sva navedena svojstva u obzir, jedinstvena definicija suvremene SOA koja koristi ove principe može biti:

"*Suvremena uslužno orijentirana arhitektura predstavlja otvorenu, dinamičku, proširivu, modularnu arhitekturu sastavljenu od autonomnih, raznovrsnih, interoperabilnih usluga s mogućnostima upravljanja kvalitetom usluge, pronalaženja te ponovne iskoristivosti, izvedenu uz pomoć otvorenih normi i tehnologija." [4]*.

2 Implementacijske tehnologije uslužno orijentirane arhitekture

Uslužno orijentirana arhitektura i tehnologija Usluga weba često se međusobno izjednačavaju iako se zapravo radi o dva različita koncepta. Principi uslužno orijentiranih arhitektura su stariji od Usluga weba čak i kada se gledaju tehnološke implementacije – npr. tehnologija CORBA (*Common Object Request Broker Architecture)* te *Microsoft*-ov DCOM (*Distributed Component Object Model*) umnogome odgovaraju osnovnim principima uslužno orijentiranih arhitektura jer nude mogućnost izgradnje slabo povezanih usluga s normiziranim sučeljima. Ove tehnologije se doduše ne spominju često kao implementacijske tehnologije arhitekture SOA iz nekoliko razloga – DCOM nije bio platformski neutralan već usko povezan uz proizvode tvrtke *Microsoft,* dok je CORBA bila implementirana od većeg broja proizvođača softvera ali konačna rješenja nisu bila međusobno kompatibilna.

XML, SOAP te konačno tehnologija Usluga weba prve su tehnologije koje su gotovo bez iznimke podržavale sve zahtjeve uslužno orijentiranih arhitektura. Ono što je ključno jest činjenica da su početkom 21. stoljeća veliki proizvođači softvera prihvatili tu tehnologiju te počeli razvijati rješenja koju su – za razliku od tehnologije CORBA – bila kompatibilna te se mogla lako integrirati u jedinstveni informacijski sustav. Može se reći da je od 2002. do 2007. godine pojam SOA i pojam Usluga weba u gotovo svim aspektima bio gotovo jednoznačan.

Krajem prvog desetljeća 21. stoljeća neki stručnjaci smatraju da pojam uslužno orijentirane arhitekture polako nadrasta tehnologiju Usluga weba [5]. Zahtjevi modernog poslovnog informatičkog tržišta rastu u opsegu i kompleksnosti što tehnologija ne može pratiti. Primjer toga su proširenja tehnologije Usluga weba, tzv. *WS-Extensions*, o kojima će nešto riječi biti dano i u ovom poglavlju. Ova proširenja za sada još nisu dio norme Usluga weba (tj. *Osnovnog profila* spomenutog u poglavlju 1.4.3) te su predmet stalne rasprave i preuzimanja ovlasti od strane velikih proizvođača softvera koji su čak u nekoliko navrata izdali paralelne specifikacije iste predviđene funkcionalnosti.

Zbog toga se za iduće desetljeće predviđa sve jače razdvajanje pojma SOA i pojma Usluga weba tj. prelazak pojma SOA na više razine apstrakcije dok Usluge weba postaju samo jedna od nizu izgradbenih blokova uslužno orijentiranih arhitektura. Također, očekuje se šira primjena Usluga weba pošto one kao tehnologija ne moraju nužno funkcionirati samo kao dio sustava SOA.

Usprkos tome, trenutno Usluge weba i dalje slove kao glavna i ključna implementacijska tehnologija za izgradnju uslužno orijentiranih arhitektura te će se takvima i smatrati u ostatku ovoga rada. U ovom poglavlju će se stoga pobliže opisati pojam Usluga weba tj. tehnološki detalji komponenata koje taj pojam uključuje.

2.1 Usluge weba

Usluge weba su zapravo *tehnološka infrastruktura,* tj. skup međusobno povezanih tehnologija čija sinteza omogućuje traženu funkcionalnost. Postoji više definicija Usluga weba, a jedna od mogućih je sljedeća [6]:

Usluga weba je dio poslovne logike smještene na Internetu i dostupne kroz normizirane Internetske protokole kao što su HTTP ili SMTP. Njene glavne karakteristike su:

- *zasnovana je na XML-u*
- *slabo je povezana s klijentom*
- *može biti sinkronog i asinkronog tipa*
- *podržava udaljene pozive procedura (engl. RPC – Remote Procedure Calls)*
- *podržava razmjenu dokumenata*

Kada se govori o Uslugama weba često se govori o "Platformi Usluga weba". Taj pojam označava okolinu unutar koje Usluge weba egzistiraju i izvršavaju svoju funkciju; ona sadrži jednu ili više Usluga weba, jedan ili više poslužitelj poruka protokola SOAP te opcionalno jedan ili više poslužitelja registra UDDI uz ostale infrastrukturalne elemente kao što su transakcijske ili sigurnosne usluge. Isto tako, "Pružatelj Usluga weba" (engl. *Web services provider*) je softverski proizvod srednje razine u trorazinskoj arhitekturi – najčešće aplikacijski poslužitelj - koji predstavlja platformu Usluga weba i nudi svu ili barem osnovnu funkcionalnu podršku sustavu Usluga weba tj. uslužno orijentiranom sustavu.

Usluge weba mogu predstavljati normizirano sučelje prema bilo kojem funkcionalnom modulu poslovnog sustava, koji takvim postupkom postaje "omogućen Uslugom weba" (engl. *Web service – enabled*). Usluga weba je tako samo "omotač" koji time omogućuje integraciju drugih aplikacija u poslovni sustav, funkcionalnost samog modula je još uvijek nepromijenjena i ovisna o konkretnoj implementaciji. "Ispod" samih Usluga weba mogu se nalaziti različite tehnologije – npr. JSP (engl. *Java Server Pages)* za jednostavno procesiranje zahtjeva, EJB (engl. *Enterprise Java Beans*) kao podrška sustavu pristupa bazi, JMS (engl. *Java Messaging System*) za asinkrono slanje poruka i sl.

Primarne tehnologije izvedbe Usluga weba koje su prepoznate kao međunarodne norme su:

1. XML – tehnologija normiziranog zapisa informacija u dokumente zasnovane na oznakama
2. SOAP (engl. *Simple Object Access Protocol*)– protokol koji pruža normiziranu strukturu slaganja XML-ovskih dokumenata u pakete koji se potom koriste raznim protokolima nižeg sloja, najčešće HTTP ali i drugih kao što su SMTP ili FTP.
3. WSDL (engl. *Web Service Description Language*) – tehnologija koja omogućuje normizirani opis sučelja Usluge weba
4. UDDI (engl. *Universal Description, Discovery and Integration*) – registar za objavu i pretragu postojećih Usluga weba.

Nastavak ovog poglavlja bavi se detaljima vezanim uz ove tehnologije.

2.2 Jezik XML

Povijest jezika XML već je ukratko dana u poglavlju 1.3. Nastao kao podskup specifikacije SGML, jezik XML se nakon svoje normizacije 1998. godine ubrzo nametnuo kao *de facto* norma kada se radi o platformski neovisnom načinu zapisa podataka zasnovanih na tekstualnim znakovima. Upravo platformska neovisnost ovog jezika predstavljala je glavni temelj integracije raznorodnih sustava te se često smatra da je upravo XML zacrtao put tehnologiji Usluga weba.

2.2.1 Oblikovanje XML-ovskog dokumenta

Tipičan XML-ovski dokument je zapravo običan tekstualni dokument koji sadrži određeni broj *oznaka.* Oznake su nizovi znakova omeđeni uglatim zagradama "<" i ">". Postoje tzv. "početne" (engl. *start*) i "završne" oznake (engl. *tags*) koje se razlikuju po tome što završna oznaka nakon zagrade ima kosu crtu npr. "*</oznaka>*".

Prostor između para početnih i konačnih oznaka s istovjetnim nizom znakova unutar njih naziva se *elementom* XML-ovskog dokumenta. Ime elementa odgovara nizu znakova unutar oznaka. Postoji nekoliko sintaksnih pravila koja se moraju poštivati da bi XML-ovski dokument bio "dobro oblikovan" (engl. *well-formed*):

- elementi koji nisu prazni *moraju* posjedovati početnu i završnu oznaku koje definiraju *opseg* elementa
- prazni elementi označavaju se oznakom koja sadrži kosu crtu *ispred druge zagrade* npr. "*<prazni_element/>*"
- element može sadržavati atribute, koji se navode unutar oznake pomoću para *ime_atributa* i *vrijednost_atributa* u obliku *ime_atributa="vrijednost_atributa"*. Mogu se koristiti jednostruki i dvostruki navodnici, bitno je samo da par otvorenih i zatvorenih navodnika bude istovjetan
- elementi se mogu gnijezditi (tj. unutar elementa može se definirati drugi element) ali se ne smiju preklapati

Slika 2-1 prikazuje primjer jednostavnog dobro oblikovanog XML-ovskog dokumenta.

```
<osoba spol='M'>
      <ime>Mate</ime>
      <prezime>Perić</prezime>
      <godina>43</godina>
      <posao radni_staz="23"/>
</osoba>
```

Slika 2-1 - Jednostavni XML-ovski dokument

Popularnost XML-a leži u tome što se informacija oblikuje na način koji omogućuje čitljivost ljudskom biću ali i laku interpretaciju od strane raznorodnih programskih aplikacija. No pored toga veliku važnost u prihvaćanju XML-a imale su prateće tehnologije, poglavito tehnologije DTD (engl. *Document Type Definition* – definicija tipa dokumenta) i XML Scheme.

2.2.2 DTD i XML Schema

Tehnologije DTD i XML Schema uvele su pojam valjanosti XML-ovskog dokumenta (engl. *valid document*). Valjani dokument je dokument koji odgovara zadanoj predefiniranoj strukturi tj. ograničenjima koje ona nameće. Ova ograničenja zapravo definiraju tip dokumenta te dodatno olakšavaju razmjenu informacija pošto aplikacija može prepoznati da li primljena poruka odgovara očekivanom "tipu".

Prva specifikacija koja je omogućila ograničavanje tipa dokumenta je DTD. **Slika 2-2** prikazuje DTD-ovski dokument koji može poslužiti za definiranje tipa prikazanog XML-ovskog dokumenta**.**

```
<!ELEMENT godina (#PCDATA)>

<!ELEMENT ime (#PCDATA)>

<!ELEMENT osoba (ime, prezime, godina, posao)>

<!ATTLIST osoba

      spol CDATA #REQUIRED

>

<!ELEMENT posao EMPTY>

<!ATTLIST posao

      radni_staz CDATA #REQUIRED>

<!ELEMENT prezime (#PCDATA)>
```

Slika 2-2 - Primjer DTD-ovskog dokumenta

Specifikacija DTD bila je dobro prihvaćena no odmah su uočeni neki njeni bitni nedostaci. Najveći nedostatak bio je taj što nije odgovarala XML-ovskoj normi - tipičan DTD-ovski dokument sam po sebi nije bio dobro oblikovani XML-ovski dokument. Uz to, postojalo je mnogo drugih primjedbi: nemogućnost definicije tipa sadržaja elementa, nepostojanje podrške za objektno orijentirane principe kao što je npr. nasljeđivanje, nemogućnost definicije imenskih područja itd. Sve je to dovelo do toga da nakon usvajanja specifikacije XML Scheme specifikacija DTD se gotovo uopće ne koristi te se smatra zastarjelom i isplativom za korištenje samo u slučajevima kada je ključno održati kompatibilnost sa starijim verzijama poslovnih aplikacija.

XML Schema ispravila je većinu nedostataka specifikacije DTD – dodana je podrška za imenska područja, implementirani su određeni objektno orijentirani principi te je omogućena definicija sadržaja elementa. No najveća prednost XML Scheme nad DTD-om bila je ta što je dokument XML Scheme bio dobro oblikovani i valjani XML-ovski dokument (valjani zato što i sama XML Schema ima vlastitu shemu koja definira strukturu dokumenta-sheme). **Slika 2-3** prikazuje shemu danog XML-ovskog dokumenta**.**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

      <xs:element name="godina" type="xs:byte"/>
      <xs:element name="ime" type="xs:string"/>
      <xs:element name="osoba">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ime"/>
                        <xs:element ref="prezime"/>
                        <xs:element ref="godina"/>
                        <xs:element ref="posao"/>
                  </xs:sequence>
                  <xs:attribute name="spol" use="required">
                        <xs:simpleType>
                              <xs:restriction base="xs:string">
                                    <xs:enumeration value="M"/>
                                    <xs:enumeration value="Z"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:attribute>
            </xs:complexType>
      </xs:element>

      <xs:element name="posao">
            <xs:complexType>
                  <xs:attribute name="radni_staz" use="required">
                        <xs:simpleType>
                              <xs:restriction base="xs:byte">
                                    <xs:enumeration value="23"/>
                              </xs:restriction>
                        </xs:simpleType>
                  </xs:attribute>
            </xs:complexType>
      </xs:element>
      <xs:element name="prezime" type="xs:string"/>
</xs:schema>
```

Slika 2-3 - Primjer sheme XML-ovskog dokumenta

Iako danas nije jedini, XML Schema je najšire prihvaćeniji način definicije tipa dokumenta.

Na slici se može uočiti i jedan od češće spominjanih nedostataka XML Scheme, što je njezina složenost. Direktan rezultat dodatnih mogućnosti jest povećana količina tekstualne informacije koju sadržava jedan XSD-ovski dokument (XSD – engl. *XML Schema Document*). Važno je napomenuti da schema dokumenta ne mora biti jednoznačna tj. tip istovjetnog XML-ovskog dokumenta može se definirati na više načina od kojih neki mogu sadržavati i manju količinu ograničenja i definicija. Usprkos tome, XSD-ovski dokumenti najčešće su tekstualno vrlo sadržajni i opsežni.

2.2.3 XSLT, XPath, XQuery

Postoje još brojne prateće specifikacije koje pružaju dodatne mogućnosti kod korištenja norme XML.

Specifikacija XSLT (engl. *eXtensible Style Sheet Transformation*) omogućuje transformaciju XML-ovskih dokumenata u druge XML-ovske dokumente ili neke druge tipove dokumenata (npr. HTML za prikaz unutar Web-preglednika). To je omogućeno stvaranjem tzv. predložaka (engl. *template*) koji se naslanjaju na određeni tip dokumenta te pomoću uzorkovanja elemenata iz njega slažu "drugi" dokument. Predlošci su također XML-ovski dokumenti što je i glavna prednost specifikacije XSLT. Ova specifikacija je dosta podobna za prevođenje tipova XML-ovskih dokumenata te prezentaciju informacije no postoje i primjedbe na njezinu složenost i nečitljivost.

Specifikacija XPath je zapravo jezik koji omogućuje stvaranje upita čiji je rezultat izvođenja jedan ili više čvorova određenog XML-ovskog dokumenta. XPath definira tzv. izraze čija interpretacija dovodi do određenog segmenta XML-ovskog dokumenta koji se može potom dalje obrađivati (što je upravo način na kojem funkcionira spomenuta specifikacija XSLT). Pored sintakse za definiranje izraza XPath nudi i niz pogodnih funkcija za manipulaciju XML-ovskim elementima.

XQuery je svojevrsna generalizacija specifikacije XPath s identičnom sintaksnom strukturom. XQuery specifikaciji dodaje konstruktore za stvaranje elemenata i atributa, mogućnost definiranja vlastitih funkcija te nešto snažniji sustav definiranja tipova elemenata.

Ove tri navedene specifikacije nisu jedine koje su usko povezane s normom XML, no one se – uz XML Schemu - najčešće smatraju obaveznim i nerazdruživim proširenjem osnovne specifikacije XML. Sinergija navedenih normi dovela je do toga da je XML preuzeo vodeću ulogu među normama za platformski neutralni opis podataka te da poslovne aplikacije današnjice u velikoj mjeri koriste XML i srodne norme za upravljanje, transformaciju i pohranu podataka. Konačno, Usluge weba koriste XML kao temeljnu tehnologiju kako za enkapsulaciju podataka, tako i za stvaranje omotnica poruka te opisnika usluga kroz protokole SOAP i normu WSDL, o čemu će biti riječi u narednim poglavljima.

2.3 Protokol SOAP

SOAP je protokol dizajniran 1998. uz podršku tvrtke *Microsoft,* a zamišljen kao protokol za razmjenu strukturiranih informacija između dva udaljena programska entiteta. Sama kratica SOAP nastala je od konstrukta "jednostavan protokol za pristup objektima" (engl. ***S**imple **O**bject **A**ccess **P**rotocol)*[2]. "Jednostavan" (engl. *simple*) je uglavnom označavalo da protokol ne uključuje neke dodatne funkcije kao što su sigurnost, pouzdanost, usmjeravanje te pravila uspostave veze. Ostatak kratice ticao se "pristupa objektima" (engl. *object access*) pošto je inicijalna primjena protokola SOAP bila ograničena na poziv objekata tipa COM (engl. *Component Object Model,* norma tvrtke *Microsoft*) dostupnih kroz Internet.

Protokol SOAP ubrzo je nadrastao inicijalnu svrhu te dobio podršku kako ostalih proizvođača softvera tako i konzorcija W3C koji je i preuzeo brigu o normizaciji i daljnjem vođenju razvoja protokola.

SOAP je zapravo transportni protokol zasnovan na XML-u koji naliježe na niže slojeve transportnih protokola kao što je poznati HTTP-TCP/IP složaj. SOAP komunikacija nema ugrađeno pamćenje, tako da pošiljatelj i primatelj ne moraju održavati sesiju kako bi komunicirali. Stanje sesije - ako se pokaže nužnim - može biti pohranjeno unutar same poruke. Normiziranim protokolom SOAP mogu komunicirati dva entiteta dok god zadovoljavaju sljedeće uvjete:

- mogu slati i primati poruke preko mreže korištenjem protokola HTTP ili SMTP
- imaju razumijevanje pravila MIME (eng. *Multipurpose Internet Mail Extensions*) te mogu koristiti ista pravila za konstrukciju i dekonstrukciju binarnih privitaka
- mogu interpretirati i procesirati XML-ovske dokumente
- mogu izvršiti akciju određenog tipa ako je ona navedena u SOAP dokumentu

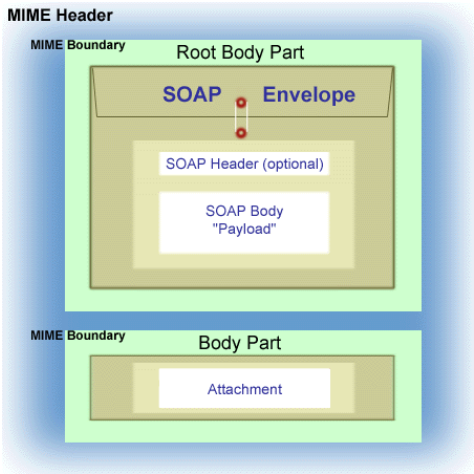
Sama specifikacija protokola SOAP opisuje četiri glavne komponente:

- opis konvencija oblikovanja SOAP poruka u smislu enkapsulacije podataka i opisa usmjeravanja u obliku SOAP "koverte" (engl. *envelope*)
- povezivanje s transportnim protokolom
- pravila kodiranja poruke
- mehanizam udaljenog poziva procedura.

Temeljni dio protokola SOAP upravo je definicija normiziranog oblika poruke koja je sposobna prenositi podatke vezane uz udaljeni pristup metodama ili dokumentno orijentirane podatke, što pogoduje kako sinkronim tako i asinkronim modelima razmjene poruka.

Strukturu SOAP poruke prikazuje **Slika 2-4.** (slika prikazuje i opcionalni privitak tipa MIME, normizirani način slanja privitaka unutar poruka zasnovanih na tekstu).

[2] Od verzije 1.2 izdane 2003. ova kratica službeno se više ne upotrebljava jer se smatra da netočno opisuje svrhu protokola tako da danas SOAP predstavlja službeno ime, a ne akronim. U novijoj literaturi može se naći i "novo" objašnjenje kratice kao "protokol za uslužno orijentirane arhitekture" (*engl. Service-Oriented Architecture Protocol).*



Slika 2-4 - Struktura poruke protokola SOAP

SOAP poruka sastoji se od tzv. omotnice (*SOAP envelope*) u kojoj se nalaze dva glavna dijela – zaglavlje (*header*) i tijelo (*body)*. Norma SOAP ne daje strogu definiciju sadržaja koji će se stavljati u te dijelove. Zaglavlje principijelno služi za informacije vezane uz samu komunikaciju, dok tijelo treba sadržavati podatke namijenjene primatelju poruke. Unutar tijela može se definirati i element pogreške (*fault)* u kojeg se stavljaju informacije vezane za slučaj greške u komunikaciji. **Slika 2-5** prikazuje izgled tipične SOAP poruke (bez poslovnih informacija u zaglavlju i tijelu poruke).

```
<?xml version="1.0"?>

<soap:Envelope xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

        <soap:Header>
                ...
        </soap:Header>

        <soap:Body>
                ...

                <soap:Fault>
                ...
                </soap:Fault>

        </soap:Body>

</soap:Envelope>
```

Slika 2-5 – Klasični oblik poruke tipa SOAP

SOAP ne mora služiti samo za prijenos tekstualnog sadržaja. Proširenje norme nazvano "*SOAP with attachments*" (engl. SOAP s privitcima) omogućuje slanje gotovo bilo kakvog tipa sadržaja korištenjem protokola SOAP. Sam SOAP neovisan je o platformi koja implementira sustav za razmjenu poruka, programskom jeziku koji interpretira poruka te sadržaju kojeg poruka prenosi. Iako je HTTP danas podrazumijevani "temeljni" transportni protokol za poruke protokola SOAP, moguće je koristiti i druge protokole; HTTP je najpopularniji većinom zbog činjenice da se radi o široko prihvaćenom protokolu, da lako prolazi kroz vatrozid i usmjerivače te da je uz pomoć njega lako uspostaviti udaljenu komunikaciju.

2.4 Jezik WSDL

WSDL je specifikacija koja definira XML-ovsku sintaksu za izgradnju opisa Usluge weba kao niza krajnjih točaka (engl. *endpoints*), tj. adresa koje služe za razmjenu poruka kroz proceduralno- ili dokumentno-orijentirane principe (objašnjene u poglavlju 2.7). WSDL se često smatra svojevrsnim receptom za automatsku uspostavu komunikacije između dvije aplikacije.

Za razliku od drugih jezika za opis sučelja (kao što su CORBA IDL[3] ili Microsoft IDL) WSDL pruža mehanizme proširivosti koji omogućuju:

- opis krajnjih točaka i poruka koji se kroz njih razmjenjuju bez obzira na konkretni oblik poruke ili mrežni protokol koji se koristi za razmjenu
- tretiranje poruka kao apstraktnih opisa podataka koji se razmjenjuju
- tretiranje tipova ulaza kao apstraktnih skupova operacija koje Usluge weba izvršavaju

Pojednostavljeno rečeno, WSDL opisuje što konkretna usluga radi, gdje se nalazi i kako pozvati njezine metode. WSDL sadrži zasebne definicije i terminologiju za definiranje Usluge weba, komunikacijske krajnje točke gdje se ta usluga nalazi, dozvoljeni format ulaznih i izlaznih poruka te apstraktni način deklariranja veze s konkretnim protokolom i oblikom podataka.

Većina elemenata koje WSDL opisuje su apstraktni – Usluga weba se mora prilagoditi ograničenjima unutar WSDL-ovskog dokumenta no još uvijek ima slobodu izbora pojedinosti implementacije. WSDL također pruža mogućnost definicije veze s protokolom te pruža već gotova proširenja za SOAP, HTTP i MIME.

Pošto je WSDL apstraktni opis sučelja Usluge weba, način primjene WSDL dokumenta je dvojak; moguće je iz WSDL dokumenta automatski generirati predložak implementacijskog koda, ali i obrnuto – za gotovu Uslugu weba moguće je automatski generirati WSDL dokument.

2.4.1 Struktura WSDL-ovskog dokumenta

Slika 2-6 prikazuje izgled klasičnog WSDL-ovskog dokumenta, dok u nastavku slijedi kratko objašnjenje važnijih elemenata.

[3] engl. *Interface Description Language* – jezik za opisivanje sučelja.

```
<?xml version="1.0"?>
<definitions … >
        <types>… </types>
        <message name="…"> … </message>
        <portType name="…">
                <operation name="…"> … </operation>
        </portType>
        <binding name="…" type="…"> … </binding>
        <service name="…">… </service>

</definitions>
```

Slika 2-6 - Izgled klasičnog WSDL-ovskog dokumenta

Element <definitions> služi kao spremnik globalnih deklaracija imenskih područja koja se koriste u ostatku dokumenta. Ovaj element najčešće je korijenski element WSDL-ovskog dokumenta i polazišna točka za aplikacije koje prikupljaju informacije o traženoj Usluzi weba.

Opcionalni element <import> omogućuje „ugradnju“ drugih WSDL-ovskih dokumenata u „glavni“ dokument čime se postiže izvjestan stupanj modularnosti i omogućuje ponovna iskoristivost.

Element <types> je spremnik za definicije tipova podataka koji se koriste u elementu <messages>. Za tipove podataka najčešće se koriste tipovi definirani u XSD-ovskoj shemi ali su mogući i drugi pristupi definiranja tipova. U općenitom slučaju sadržaj ovog elementa identičan je sadržaju klasičnog XSD-ovskog dokumenta s identičnim konstruktima i sintaksom. Element <types> moguće je ugraditi kroz element <import> tako da on ne mora biti fizički prisutan u dokumentu.

Element <message> služi za modeliranje podataka koji se razmjenjuju kroz Uslugu weba. Elementi unutar njega referenciraju tipove koji su definirani unutar <types> odlomka ili se opišu pomoću normiziranih konstrukata XSD-ovske sheme. Svaka poruka se sastoji od jednog ili više dijelova, označenih elementom <part>.

Element <portType> specificira podskup operacija za krajnju točku Usluge weba. U određenom smislu ovaj element je jedinstveni identifikator grupi akcija koje se mogu izvršiti na jednoj završnoj točki.

Element <operation> predstavlja jednu operaciju, tj. apstraktnu definiciju akcije koju podržava Usluga weba. Operacija se identificira preko njezinog imena te opisuje ulazne i izlazne poruke – za to se koriste elementi <name>, <input> i <output>. Operacije mogu biti inicirane od klijenta ili poslužitelja, mogu biti jednosmjerne ili dvosmjerne, a sve se to kontrolira pomoću redoslijeda i opcionalnog korištenja ulazno/izlaznih poruka.

Tipovi operacija su:

- zahtjev-odgovor
- potražnja odgovora
- jednosmjerna komunikacija
- obavještavanje

Prva dva tipa (koja predstavljaju dvosmjernu komunikaciju) mogu sadržavati i element `<fault>` koji se vraća drugom sudioniku komunikacije u slučaju da dođe do bilo kakvog nepredviđenog rezultata tokom provođenja operacije.

Ime svake operacije mora biti jedinstveno unutar opsega elementa `<portType>`. Imena ulaznih i izlaznih poruka moraju biti također jedinstvena unutar opsega elementa `<portType>`, nije dovoljno samo unutar elementa `<operation>`.

Unutar elementa `<binding>` nalaze se specifikacije konkretnog protokola i formata podataka za element `<portType>`. Ovdje se mogu koristiti normizirana proširenja veze kao što su HTTP, SOAP ili MIME a mogu se izvesti i vlastita prilagođena proširenja ako za to postoji potreba.

Konačno, element `<service>` tipično se pojavljuje na kraju WSDL-ovskog dokumenta i pokazuje gdje se točno nalazi Usluga weba tj. pomoću kojeg URL-a se ona može dozvati. Dokument ne mora sadržavati ovaj element – u slučaju da WSDL predstavlja samo konkretni opis usluge koja još nema konkretnu implementaciju.

2.4.2 Uporaba, prednosti i nedostaci WSDL-ovskih dokumenata

Kao što je rečeno, WSDL-ovski dokument je zapravo recept koji aplikaciji opisuje način na koji će kontaktirati određenu Uslugu weba. WSDL-ovski dokument će poslovnoj aplikaciji dati sve nužne detalje – lokaciju usluge, protokol kojom joj se može pristupiti, imena operacija koje usluga podržava te izgled ulaznih i izlaznih parametara. Pošto je opis normiziran prilagodba Usluzi weba može se provoditi automatski, bez potrebe za ljudskim uplitanjem.

Scenarij uporabe uključuje pronalazak WSDL-ovskog dokumenta – bilo pretraživanjem registra ili primitkom preko nekih drugih kanala – analizu istog te stvaranje klijentskog sučelja koje će pristupiti traženoj usluzi. Važno je napomenuti da WSDL-ovski dokument nije nužan za pristupanje usluzi – poslovna aplikacija može na drugi način saznati detalje o usluzi i pristupiti joj. Ovaj scenarij tipičan je za uslužno orijentirane sustave implementirane unutar tvrtki gdje Usluge weba predstavljaju module poslovnog sustava koji nisu predviđeni za pristup izvana.

Najčešće spominjani nedostatak WSDL-ovskog dokumenta je nepostojanje semantičkog opisa usluge – konkretna poslovna funkcija koju usluga izvršava, u kojim scenarijima je ona predviđena za uporabu te dodatne informacije o brzini, kvaliteti usluge i sl. WSDL-ovski dokument opisuje sučelje, sve ostale informacije moraju se saznati iz dodatnih izvora, kao što je npr. UDDI registar objašnjen u idućem poglavlju.

2.5 Registar UDDI

UDDI je kratica za "univerzalni opis, otkrivanje i integraciju" (engl. *Universal Description, Discovery and Integration*) i označava normiziranu metodu objave i opisa informacija o Uslugama weba. UDDI se usredotočava na problematiku *pronalaženja* Usluga weba.

U konceptualnom smislu, poslovni subjekt koristi UDDI registar kao svojevrstan oblik "poslovnog imenika". Informacije koje se zapisuju u registar nisu ograničene na Usluge weba, tip podataka je puno širi. Poslovni subjekt može registrirati tri tipa informacija u UDDI registar:

- **bijele stranice** – opisuju osnovne kontaktne informacije tvrtke te pružaju metodu pronalaska Usluga weba preko identifikatora tvrtke
- **žute stranice –** opisuju Usluge weba pomoću otprije definiranih taksonomija što omogućuje pronalazak usluge prema biranoj kategoriji
- **zelene stranice –** pružaju tehničke informacije i podržane funkcije Usluge weba. Ove informacije bave se tehničkim detaljima lokacije i pristupa Usluzi weba.

2.5.1 Elementi specifikacije UDDI

Sama specifikacija UDDI (trenutna verzija 2.0) je zapravo skup specifikacija koje definiraju različite aspekte sustava za podršku navedenog registra. Neki od njih su:

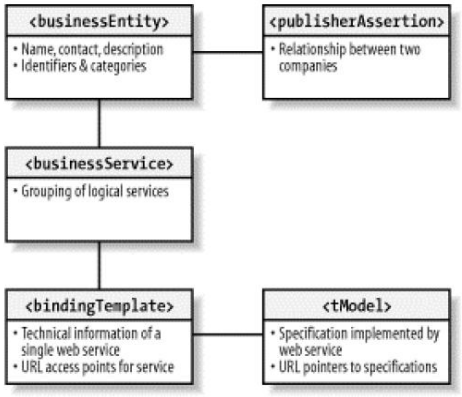
- **replikacija –** opisuje procese replikacije podataka te sučelja koje operator registra mora podržavati kako bi omogućio replikaciju podataka između fizičkih lokacija integriranih UDDI registara.
- **operatori –** opisuje ponašanje i parametre koje UDDI čvorovi moraju podržavati.
- **programersko sučelje –** skup funkcija koje svi UDDI registri podržavaju kako bi njihovi klijenti mogli postavljati upite o uslugama čiji opisi se čuvaju u registrima te za objavu informacija o vlastitim uslugama.
- **podatkovne strukture –** pet osnovnih podatkovnih struktura te njihovi međuodnosi (**Slika 2-7**).

Programer koji želi programski pristupiti UDDI registru ima sljedeće opcije na raspolaganju:

- **SOAP aplikacijsko sučelje zasnovano na programskom jeziku Java –** najkompliciraniji izbor pošto programer mora sam programski konstruirati XML poruku koja odgovara UDDI shemi te ju ugraditi u SOAP omotnicu i poslati na odgovarajuću adresu
- **vlastito klijentsko sučelje UDDI registra –** programer može stvoriti vlastito sučelje prema UDDI specifikaciji a može se i koristiti postojećim softverskim proizvodima za tu svrhu
- **JAXR –** specifikacija koja pruža normizirani način programskog pristupa registrima zasnovanim na XML-u, posebno ističući UDDI i ebXML registar.

UDDI definira dva glavna sučelja, sučelje za upit (engl. *Inquiry API*) i sučelje za objavu (engl. *Publishing API*). Ona koriste iste tehnike ali druge XML dokumente, podatkovne strukture i pristupne točke. Svaka operacija čitanja koristi sučelje upita i u pravilu ne zahtijeva autentikaciju. Objava uključuje stvaranje, pohranu i ažuriranje informacija u registru. Sve te funkcije zahtijevaju autentikaciju.

Ključan element specifikacije UDDI su njezine **podatkovne strukture**, tj. model podataka korišten za pohranu poslovnih informacija te informacijama o Uslugama weba koje tvrtka nudi.



Slika 2-7 - Podatkovne strukture registra UDDI

Element **"businessEntity"** opisuje osnovne poslovne informacije, kao što su ime tvrtke, kontaktne informacije i sl. Moguće je definirati različite veze između poslovnih entiteta kao npr. partnerstvo dviju tvrtki.

Struktura "**publisherAssertion"** služi za definiciju odnosa između dva poslovna entiteta. Veza između dva entiteta postaje javna kada obje tvrtke objave identičnu instancu ovog elementa.

Struktura **"businessEntity"** sadrži jednu ili više podatkovnih struktura **"businessService"**. Ovaj element predstavlja jednu logičku klasifikaciju usluga i sadrži skup usluga koje nudi tvrtka, tj. poslovni entitet.

Nadalje, opis usluge sadržava jednu ili više struktura "**bindingTemplate**" koje se bave tehničkim detaljima usluge i koje mogu sadržavati specifikaciju same usluge koja se pohranjuje u element **"tModel"**.

Svaka ova struktura ima jedinstveni identifikator, ili UUID (univerzalno jedinstveni identifikator – engl. *Universally Unique Identifier*). On se najčešće gradi prema normi ISO/IEC 11578:1996 koja preporučuje izgradnju UUID-a pomoću heksadecimalnog niza znakova koji nastaje konkatenacijom trenutnog vremena, hardverske adrese, IP adrese te nasumično odabranog broja.

2.5.2 Kategorizacija

Jedna od glavnih značajki UDDI registra jest i kategorizacija Usluga weba prema definiranim kriterijima – npr. industriji, tipu proizvoda ili geografskoj lokaciji. Kao i kod svake kategorizacije opravdano je postaviti pitanje njezine zrnatosti – široko definirane kategorije mogu na upit vraćati preveliku količinu rezultata, dok su detaljne kategorije izvedbeno kompleksne i često zbog loše definicije mogu uzrokovati nemogućnost pronalaska željenog rezultata.

Zbog svega toga trenutno nije realistično očekivati pojavu softvera koji će potpuno samostalno dinamički pronalaziti i koristiti Usluge weba kao izgradbene blokove poslovnih procesa; realniji scenarij ipak uključuje poslovne stručnjake koji će ručno pregledavati kategorije usluga i prema željenim kriterijima odabirati one koji odgovaraju njihovim trenutnim poslovnim potrebama. Isto tako, umjesto globalnih UDDI registara izglednija je upotreba privatnih registara s ograničenim skupom usluga koji će omogućiti lakšu B2B (engl. *Business-to-Business*) integraciju s poslovnim partnerima.

Postoje neke normizirane sheme kategorizacije koje se često koriste u konkretnim softverskim proizvodima:

- **NAICS –** (engl*. North American Industry Classification System)* – sadrži stotine industrijskih klasifikacija i nudi dosta detaljne kategorije
- **UNSPSC –** ***(***engl*. Universal Standard Products and Services Classification*) - sustav klasifikacije proizvoda i usluga namjenjen globalnoj uporabi
- **ISO 3166 –** norma za zemljopisne regije

2.5.3 UDDI i WSDL-ovski dokumenti

Kao što je rečeno u prethodnom poglavlju, WSDL-ovski dokumenti se koriste za opis sučelja Usluge weba. Dokument *<tModel>* je dio UDDI-ja koji pruža metapodatke o Usluzi weba i pokazivače na njezinu implementaciju. WSDL dokumenti i UDDI dokumenti su povezani kroz nekoliko načina:

- za svaki WSDL dokument trebao bi se stvoriti po jedan *<tModel>* dokument. Pošto *<tModel>* opisuje apstraktni tip usluge WSDL ne bi trebao sadržavati elemente *<service>* i *<port>.*
- element *<bindingTemplate>* se stvara za svaki jedinstveni URL pristupne točke koju koristi Usluga weba. Svaki *<bindingTemplate>* referencija jedan ili više elemenata *<tModel>* koji sadrže WSDL dokument za ovu pristupnu točku.
- za svaku Uslugu weba postoji jedan element *<businessService>.* Taj dokument sadrži po jedan element *<bindingTemplate>* za svaku pristupnu točku Usluge weba.

2.5.4 Prednosti i nedostaci UDDI registara

Specifikacija UDDI se smatra neizostavnim dijelom tzv. "trojstva" Usluga weba još od njihovih najranijih dana. Pored potražitelja i pružatelja usluge katalog koji potražitelju pomaže pri pronalasku usluge te objašnjava kako joj pristupiti smatra se ključnim dijelom implementacije uslužno orijentiranih principa. UDDI je vrlo detaljno specificiran i ima jasno definirana sučelja što omogućuje laku implementaciju klijenata koji bi se priključivali na registar, pretraživali ga, dohvaćali postojeće podatke te pohranjivali nove. Osim grupe OASIS koja se službeno brine o normi on je isprva imao i snažnu podršku velikih proizvođača softvera kao što su *IBM, Microsoft* i *SAP.*

U praksi, UDDI registri nisu zaživjeli. Kao što je rečeno u članku [7], najširа uporaba Usluga weba današnjice je za potrebe integracije poslovnih aplikacija - EAI (engl*. Enterprise Application Integration*). Za ovu svrhu dovoljno je poznavati WSDL-ovski opis usluge koji je najčešće već pohranjen interno unutar poslovnog sustava tvrtke. Izrada UDDI registra za ove svrhe ne može se troškovno opravdati.

Potencijal UDDI registara leži u globalnom objavljivanju Usluga weba te dinamičkom pronalaženju i kontaktiranju otprije nepoznatih usluga. No ovaj scenarij još nije zaživio u poslovnom i informatičkom svijetu zbog niza problema – pitanje sigurnosti, kvalitete usluge, načina naplate, iscrpnih semantičkih opisa i sl. Još jedan udarac budućnosti UDDI registara zadale su tvrtke *IBM, Microsoft* i *SAP* koje su 2006. ugasile svoje javne UDDI registre. Konačno, tvrtka *Microsoft* je izdala još jedno od WS-proširenja (o kojem će više riječi biti u poglavlju 2.8) pod nazivom WS-Inspection koje se u mnogočemu preklapa s funkcijama UDDI registra.

UDDI registar se i dalje smatra ključnim segmentom Usluga weba no njegova budućnost na toj poziciji je trenutno vrlo upitna i umnogome ovisi o daljnjoj evoluciji uporabe Usluga weba.

U sljedećem poglavlju obradit će se jezik koji omogućuje povezivanje Usluga weba u organizirani poslovni proces, poglavito pomoću koordinirane razmjene poruka između njih.

2.6 WS-BPEL

Jedan od velikih problema Usluga weba jest činjenica da svaka usluga djeluje samostalno. Iako je – prema principima uslužno orijentiranih arhitektura – lako stvoriti kompoziciju usluga ili delegirati odgovornost na tzv. upravljačke usluge koje će kontrolirati tok podataka kroz niz usluga, sama specifikacija Usluga weba nigdje ne definira normiziran način slaganja usluga u egzaktno specificirane poslovne procese. Iako je moguće stvoriti vlastita učinkovita rješenja organizacije i provođenja poslovnih procesa zasnovanih na uslužno orijentiranoj arhitekturi [8] te implementirati vlastite alate za tu svrhu [9], ipak se osjeća potreba za normama koje bi taj proces strogo definirale i omogućile lakši dizajn i izvedbu takvih procesa. Odgovornost za izvršavanje ove zadaće preuzeli su veliki komercijalni proizvođači softvera te kasnije organizacija *OASIS* sa svojim specifikacijama BPEL4WS, odnosno WS-BPEL (kako je kasnije preimenovana).

WS-BPEL (engl. *Web Services – Business Process Execution Language*) – definira se kao specifikacija koja definira jezik za izgradnju poslovnih procesa zasnovanih na Uslugama weba. Ona je nasljednik specifikacije BPEL4WS (engl. *Business Process Execution Language for Web Services)* te se zbog toga često uz WS-BPEL dodaje i oznaka verzije 2.0 (posljednja službena verzija specifikacije BPEL4WS bila je 1.1).

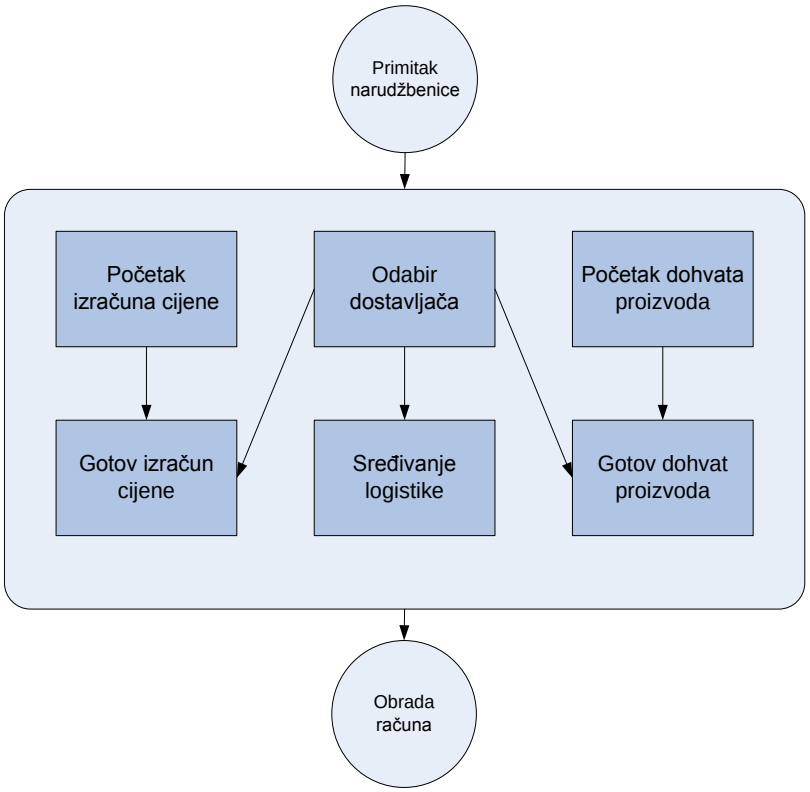
WS-BPEL (tj. njezina prva verzija BPEL4WS) nastala je 2002. godine kao rezultat zajedničkog rada tvrtki *IBM*, *Microsoft* i *BEA*, a inspiraciju su predstavljale prethodne varijacije kao što su *IBM*-ov WSFL (engl.*Web Services Flow Language*) i *Microsoft*-ova specifikacija XLANG. U razvoj jezika priključile su se i neke druge softverske tvrtke kao što su *SAP* i *Siebel Systems*. Specifikacija je u 2007. predana organizaciji OASIS kako bi postala službena, otvorena norma za izgradnju poslovnih procesa povezivanjem Usluga weba.

U nastavku će biti dani detalji o tehnološkoj strukturi te uporabi specifikacije WS-BPEL za stvaranje poslovnih procesa.

2.6.1 Struktura BPEL-ovskog procesa

WS-BPEL omogućuje definiranje poslovnog procesa kroz tzv. orkestraciju usluga u jedan povezani protok informacija.

BPEL-ovski proces zapravo je spremnik unutar kojeg se mogu deklarirati odnosi između vanjskih partnera, podaci za obradu, rukovatelji za razne primjene te aktivnosti koje se moraju izvršiti. Proces se identificira imenom i imenskim područjem što se definira kao atributi glavnog elementa u XML-ovskom dokumentu koji predstavlja normizirani opis samog procesa. **Slika 2-8** prikazuje primjer poslovnog procesa koji se može implementirati pomoću Usluga weba i specifikacije WS-BPEL.



Slika 2-8 - Primjer jednostavnog poslovnog procesa

WS-BPEL koristi već postojeće mehanizme specifikacije Usluge weba – poglavito normu XML te normu za opis Usluga weba WSDL. Glavna ideja jest ta da se stvori jedna upravljačka usluga – tzv. poslužitelj BPEL procesa – koja na osnovi danog XML-ovskog predloška ima odgovornost provođenja poslovnog procesa. Upravljačka usluga naziva se BPEL-ovskim procesorom te ima svoj vlastiti WSDL-ovski opis te je i sama po svim karakteristikama jedna klasična Usluga weba. Često se oko ovakve usluge izgrađuje zasebni poslužiteljski sustav s dodatnim komponentama te upravljačkom konzolom kako bi se stvorila kvalitetna platforma za upravljanje poslovnim procesima. Jedna od takvih platformi je npr. *Apache ODE*.

Jezgra BPEL-ovskog procesa je njegov XML-ovski opis. **Slika 2-9** daje primjer XML-ovskog opisnika BPEL-ovskog procesa (koji je najčešće tekstualna datoteka s ekstenzijom *.bpel*).

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

<partnerLinks>
      <partnerLink name="..."
            partnerLinkType="..."
            myRole="..."/>
      ...
</partnerLinks>

<variables>
      <variable name="..." messageType="..."/>
      ...
</variables>

<sequence name="...">
      <receive name="receiveOrder"../>
      <assign>
            <copy>
                  <from variable="..." part="...">
                  <query>
                        ...
                  </query>
                  </from>
                  <to variable="...">/>
            </copy>
      </assign>
      <invoke name="checkPayment"../>
      <invoke name="shippingService"../>
      <reply name="sendConfirmation"../>
</sequence>

<faultHandler>
      <catch faultName="..."
            faultVariable="...">
      ...
      </catch>
      <catchAll>...</catchAll>
</faultHandler>
```

Slika 2-9 - Primjer BPEL-ovskog procesa u XML-ovskom obliku

Proces može biti izvršiv (engl. *executable*) ili apstraktni (engl. *abstract*) što se može vidjeti prema definiranom imenskom području. Apstraktni procesi opisuju ponašanje procesa bez pokrivanja svih detalja njegovog izvršavanja. Izvršivi procesi opisuju kompletno ponašanje procesa – vanjski dio koji je vidljiv entitetima koji surađuju s procesom te unutarnji koji omogućuje izvršenje samog procesa.

Element `<process>` je korijenski element BPEL procesa. Postoji i element `<scope>` koji omogućuje modularizaciju procesa kroz lokalne podprocese.

BPEL stvara poslovni proces povezivanjem s vanjskim Uslugama weba, tzv. "partnerskim vezama" (engl. *PartnerLink).* Te veze su instance tipova ulaza definiranih u pripadnim WSDL-ovskim dokumentima. Jedan partner može imati više partnerskih veza. Veze se definiraju odmah ispod elementa `<process>` ili unutar elemenata `<scope>` ako ne moraju biti globalno vidljive. "Partner" opisan partnerskom vezom može biti klijent koji je odgovoran za pozivanje i pokretanje procesa (tj. potražitelj usluge) a može biti i usluga koja se mora pozvati od strane procesa (pružatelj usluge).

Proces uključuje i *varijable* koje se deklariraju globalno ili lokalno. Varijable služe kao elementi čuvanja stanja procesa a mogu sadržavati jednostavne informacije ali i čitave XML-ovske dokumente definirane određenom XSD shemom. Tip podataka unutar varijable je predefiniran pomoću jednog od tri atributa: `messageType`, `element` ili `type`. U klasičnom slučaju varijable tipizirane kroz atribut "`messageType`" definiraju se za svaku ulaznu i izlaznu poruku procesa. Nadalje, kao i partnerske veze, varijable se mogu definirati pod elementom `<process>` ali i unutar vlastitog opsega.

Glavni izgradbeni blokovi BPEL-ovskog procesa su aktivnosti. Aktivnosti mogu biti strukturirane (složene) ili jednostavne, koje obavljaju određenu funkciju i ne mogu sadržavati druge aktivnosti. Jednostavne aktivnosti su: primanje (engl. *receive*), odgovor (engl. *reply*) i poziv (engl. *invoke*). Primanje je aktivnost koja uključuje prihvaćanje poruke od vanjskog partnera. Ova aktivnost može ali ne mora imati pripadnu aktivnost odgovora. Ono što mora imati je ime preko kojeg se identificira i varijablu (ili više njih) u koju se sprema primljena poruka. Aktivnost može također definirati da li je potrebno stvoriti novu instancu procesa ili obraditi poruku pomoću postojeće instance. Dvostrana komunikacija uključuje i odgovor partneru koji je postavio upit. Odgovor može sadržavati normalne podatke ili podatke o pogrešci tj. podatke koji se šalju kao odgovor kada zahtjev ne završi na očekivani način, što se onda označava atributom *faultName* koji identificira ime WSDL pogreške do koje je došlo.

Jedna od karakteristika poslovnih procesa je i da se određene aktivnosti često moraju izvoditi paralelno. Za to BPEL predviđa upotrebu aktivnosti toka, ili `<flow>`. Ako su neke aktivnosti ovisne o drugima, moraju se definirati posebne veze (element `<link>`) između njih te potom naglasiti koja aktivnost je uvjet tj. izvor veze (element `<source>`) a koja uvjetovana tj. cilj veze (element `<target>`). Može se također definirati i više "izvora" i "ciljeva" ako postoje izvjesni uvjeti koji utječu na tok procesa.

Postoji mogućnost da uvjet spajanja ne bude ispunjen, tj. da rezultat uvjetnog izraza bude neistina (engl. *false*). U tom slučaju se izbacuje greška u spajanju uvjeta koju obrađuje rukovatelj iznimkama, ili se greška propagira sve dok uvjet spajanje ne postane istinit (tzv. "eliminacija mrtvog puta" ili DPE – engl. *Dead Path Elimination*).

Razmjena poruka unutar procesa često uključuje manipuliranje porukama u smislu njihovog rastavljanja, izvlačenja informacija iz određenih dijelova te njihove obrade i slaganja s drugim elementima kako bi se stvorili novi tipovi poruka koje očekuju neki drugi segmenti procesa. Glavni element za ovu potrebu unutar specifikacije BPEL je aktivnost pridjeljivanja, `<assign>`, koja sadržava jednu ili više operacija kopiranja (element `<copy>`). Kopirati se mogu svi podaci iz jedne varijable u drugu, no češće se kopira samo jedan segment poruke korištenjem XPath i XSLT izraza.

Iznimke se javljaju kad u obradi poslovnih procesa dođe do situacija koje se ne smatraju očekivanim događajima u sklopu uspješne provedbe procesa. To su tzv. iznimne situacije koje se također moraju adekvatno obraditi kako bi proces ipak završio na odgovarajući način te vanjski partneri dobili informaciju o tome što se i zašto dogodilo. Takvu obradu izvršavaju tzv. rukovatelji iznimkama (engl. *fault handlers*). Rukovatelji se definiraju globalno ispod elementa `<process>`, lokalno unutar opsega a moguće ih je čak i integrirati unutar aktivnosti pozivanja zbog jednostavnije definicije i preglednijeg izgleda procesa.

Iznimke (ili greške – engl. *faults*) su najčešće definirane unutar samih WSDL sučelja usluga. BPEL proces stoga može predvidjeti do kojih iznimki može doći te shodno tome pripremiti odgovarajuće rukovatelje za svaku očekivanu iznimku, ili jednostavno stvoriti generički rukovatelj ako iznimka nije prepoznata.

2.7 Vrste Usluga weba

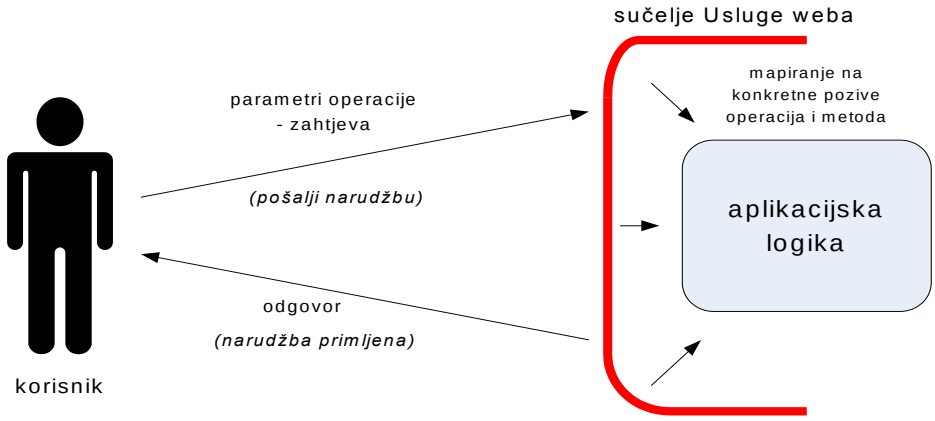
Postoje dvije glavne kategorije Usluga weba: Usluge weba zasnovane na pozivima udaljenih metoda (engl. *RPC-based Web services*) te dokumentno orijentirane Usluge weba (engl. *document-oriented Web services*).

Kao što je rečeno u poglavlju 2.3, protokol SOAP je u početku zamišljen isključivo kao normizirani protokol za pozivanje udaljenih metoda. Slijedeći tu logiku Usluge weba su također isprva bile zamišljene kao programski entiteti koji u distribuiranoj okolini pružaju „usluge“ – metode koje će klijenti pozivati uz pomoć protokola SOAP. Ovakav tip Usluga weba odgovara klasičnom RPC modelu (engl. *RPC – Remote Procedure Call* – udaljeni poziv procedure) koji se i danas često koristi. Ovakve usluge mogu se prepoznati po tzv. „*rpc-literal*“ ili „*rpc-encoding*“ tipu SOAP veze koju potražuju[4] te po metodama čiji su ulazni parametri najčešće skup objekata osnovnog tipa (npr. „*string*“ ili *„integer“)*. **Slika 2-10** prikazuje Uslugu weba zasnovanu na udaljenom pozivu metoda.

Ovakve Usluge weba imaju puno prednosti među kojima je jedna od ključnih jednostavnost izvedbe; WSDL-ovski opis usluge je jednostavan, primatelj poruke ju lako prosljeđuje odabranoj implementacijskoj metodi koja primljeni sadržaj lako interpretira i obrađuje istovjetno kao da se radi o lokalnom pozivu.

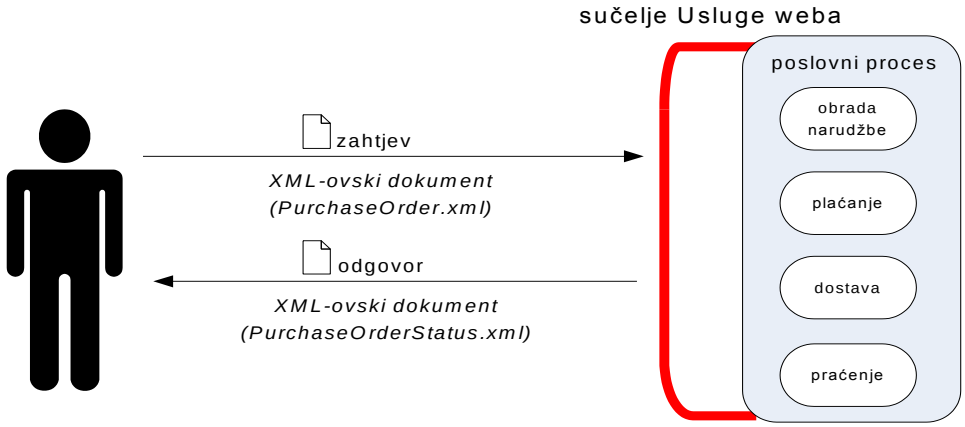
S druge strane, poslovni procesi u sklopu elektroničkog poslovanja temelje se na „porukama“. U ovakvom kontekstu Usluge weba nisu jedinke koje „izvršavaju pozvanu metode“ već izgradbeni blokovi poslovnog procesa koji se provodi razmjenom poslovnih dokumenata – tj. navedenim porukama. Poruka – „poslovni dokument“ – je najčešće XML-ovski dokument definiran pripadnom XML schemom. WSDL-ovski dokument Usluge weba ima točno definirani tip dokumenta koji Usluga očekuje tako da ona ima opciju provjere valjanosti istog. Ovakve Usluge weba prepoznaju se po tzv. *„document-literal“* tipu SOAP veze te najčešće – ako se radi o Usluzi weba koja prati preporuke WS-I Osnovnog Profila – jedinstvenom elementu unutar sadržaja poruke tipa SOAP koji predstavlja spomenuti dokument koji se razmjenjuje.

[4] Razlika između „*encoded*“ i *„literal*“ je u tome što prvi kod slanja ulaznih parametara zahtijeva i definiciju njihovih tipova, dok ih drugi smatra implicitno zadanim pomoću WSDL-ovskog dokumenta. Tip „*encoded*“ nije podržan od strane WS-I Osnovnog Profila.



Slika 2-10 - Usluga weba zasnovana na udaljenom pozivu metoda

Prednosti ovakvog tipa Usluga weba je očigledna – mogućnost provjere valjanosti XML-ovskog dokumenta (poslovne poruke) koju je Usluga primila. Naravno, ovo sa sobom vodi i određene nedostatke. Provjera valjanosti potražuje određene resurse što usporava poslovni proces. WSDL-ovski opis ovakve Usluge je kompleksniji od WSDL-ovskog opisa Usluge zasnovane na udaljenim metodama. Konačno, pošto je sadržaj SOAP poruke zapravo sam dokument, ona ne sadržava naziv metode koju primatelj mora pozvati (a unatoč tome što se ne radi o usluzi zasnovanoj na pozivu udaljenih metoda, implementacijska klasa je i dalje programski objekt s metodama koje uokviruju traženu funkcionalnost). Ovo se rješava implementacijom tzv. „omotača“ poruke (zapravo korištenjem određenog principa oblikovanja poruke koji uključuje i dodavanje imena metode kao atributa poruke iako ono nije dopušteno kroz odgovarajuću XML shemu dokumenta). Ovakav tip Usluge se naziva „*document/literal wrapped*” i podržan je od strane WS-I Osnovnog Profila a prikazuje ga **Slika 2-11**.



Slika 2-11 - Dokumentno-orijentirana Usluga weba

Ovaj posljednji tip Usluge weba smatra se najprikladnijim za uporabu kod dizajna automatiziranih poslovnih procesa, što ne znači da je on uvijek očigledni izbor. Kao prvo, usluge tipa poziva udaljenih metoda su puno duže prisutne na tržištu softvera te kao takve imaju puno jaču implementacijsku podršku gledajući dostupne alate i razvojna sučelja (a i njihova jednostavnija implementacija se često smatra znatnom prednošću). Drugo – tip Usluge weba ne određuje njenu funkcionalnost, tj. svaka funkcionalnost može se implementirati pomoću Usluge weba bilo kojeg od navedenih tipova, tako da ne postoje objektivno dobar ili loš izbor, već samo priklanjanje zahtjevima korisnika i implementatora sustava. Stoga je zastupljenost i popularnost tipova Usluga weba ostavljeno u rukama informatičke zajednice te će budućnost pokazati koji od njih će prevladati i postati svojevrsna norma kad se radi o stvaranju uslužno orijentiranih poslovnih procesa.

2.8 Proširenja specifikacije Usluga weba – WS-Extensions

Usprkos neslaganjima oko službene definicije i opsega, Usluge weba su se dugo vremena definirale kroz trojstvo specifikacija UDDI, SOAP i WSDL uz normu XML kao temeljnu podlogu. Ove specifikacije prilično su dobro pokrivale implementaciju ranih oblika uslužno orijentiranih arhitektura. Moderne uslužno orijentirane arhitekture – obrađene u poglavlju 1.9 – imaju puno širi spektar zahtjeva i principa koje navedene tehnologije ne mogu uspješno implementirati. Zbog toga danas vrijedi općenito mišljenje da Usluge weba treba proširiti dodatnim tehnologijama koje će ih učiniti prikladnim za implementaciju svih zahtjeva koje nalažu moderne uslužno orijentirane arhitekture.

Proširenja Usluga weba najčešće se nazivaju skupnim imenom *WS-Extensions*. Ovo ime označava sve specifikacije koje se naslanjaju na Usluge weba i obogaćuju ih određenom funkcionalnošću. Kako bi se normizirao način nomenklature većina specifikacija nosi prefiks WS- kao npr. *WS-Security*.

Iako se temeljnim normama Usluga weba uglavnom bave neprofitne organizacije kao što su W3C i OASIS, proširenja su u današnje vrijeme većinom pod jakim utjecajem velikih proizvođača softvera kojima je većinom prioritet promocija vlastitih softverskih rješenja dok su normizacija i otvorenost najčešće na drugom mjestu. Ovo je s jedne strane prednost – pošto se korisnicima Usluga weba nudi širok spektar različitih proizvoda koji nude funkcionalnost koje zahtijevaju – ali i veliki nedostatak pošto se gubi glavna prednost Usluga weba kao platformski neovisne i strogo normizirane tehnologije. Dodatni problem je i nesloga velikih proizvođača softvera koji objavljuju "paralelne" specifikacije tj. proširenja koja nude istu funkcionalnost ali s potpuno drukčijom implementacijom. Ovakvi trendovi potencijalno bi mogli Usluge weba dovesti do iste sudbine koja je zadesila CORBU – kvalitetna tehnologija čija je iskoristivost ograničena zbog prevelike količine vlastitih i međusobno nekompatibilnih rješenja na tržištu.

Za početak je potrebno naglasiti da se i temeljne tehnologije Usluga weba – UDDI, WSDL, SOAP – danas u literaturi često nazivaju proširenjima (npr. na portalu koji se bavi proširenjima – [10] – WSDL i SOAP nazivaju se "osnovnim" proširenjima dok se UDDI smatra "proširenjem za otkrivanje" koje je jednakopravno *Microsoft*-ovom proširenju *WS-Inspection*). Isto tako, jezik WS-BPEL se odnedavno

smatra proširenjem što je vjerojatno i glavni razlog preimenovanja specifikacije tako da sadrži prefiks "WS-".

U nastavku će biti dani detalji nekoliko važnijih proširenja Usluga weba. U obzir su uzete sâmo priznate specifikacije koje se već koriste u izvjesnom broju implementacija uslužno orijentiranih arhitektura.

2.8.1 WS-Adressing

Ovo proširenje pokušava normizirati reprezentaciju lokacija krajnjih točaka usluga te navesti korelacijske parametre poruka (tako da se parovi poruka tipa zahtjev-odgovor mogu adekvatno upariti kao dio istog komunikacijskog toka). Specifikacija je rezultat zajedničkog truda tvrtki *IBM, Microsoft, BEA* i *SAP* a predana je na ratifikaciju od strane W3C-a.

Glavna prednost ove specifikacije je imenovanje krajnjih točaka i njihovo prepoznavanje u asinkronoj komunikaciji. Nadalje, autori specifikaciju smatraju "pomoćnom" tehnologijom koja je kompatibilna s temeljnim Uslugama weba i drugim proširenjima te ključnom tehnologijom za stvaranje poruka s visokom razinom samostalnosti i samoopisivosti (engl*. self-sufficient messages*).

2.8.2 WS-ReliableMessaging

Proširenje *WS-ReliableMessaging* također je stvoreno kao zajednički trud tvrtki *IBM, Microsoft, BEA* i *SAP* a priznato je 2007. godine kao norma od strane organizacije OASIS. Ovo proširenje opisuje protokol koji omogućuje *sigurnu dostavu* poruka tipa SOAP između distribuiranih aplikacija u slučaju kada dođe do pogreške bilo softverske komponente, sustava ili same mreže. Ovaj protokol također predstavlja samo jedan izgradbeni blok koji je kompatibilan s temeljnim Uslugama weba te predstavlja pomoćnu funkcionalnost za poslovne scenarije gdje je usluga sigurne dostave potrebna.

Specifikacija *WS-ReliableMessaging* se smatra naprednijom od specifikacije *WS-Reliability* koja je izašla ranije te koja pruža analognu funkcionalnost.

2.8.3 WS-Policy

WS-Policy proširenje predano je na normizaciju organizaciji W3C, a omogućuje definiciju pravila i preferenci vezanih uz sigurnost poruke, način obrade ili sam sadržaj. Specifikacija proširenja sastoji se od tri odvojene specifikacije - *WS-Policy, WS-PolicyAssertions* i *WS-PolicyAttachments.* Ove tri specifikacije definiraju način stvaranja pravila (tj. "politika") koje se onda mogu ugraditi u WSDL-ovski dokument te na adekvatni način interpretirati od strane poslovnih subjekata zainteresiranih za korištenje određene Usluge weba.

2.8.4 WS-MetadataExchange

Česti je prigovor WSDL-ovskim dokumentima da ne posjeduju dodatne informacije koje bi funkcionalnost usluge stavile u određeni semantički kontekst te pružili dodatne informacije o usluzi kao što je kvaliteta usluge i sl. Proširenje *WS-MetadataExchange* omogućuje ugradnju dodatnih metapodataka o usluzi unutar poruke tipa SOAP koje onda poslovni subjekt može analizirati te se dobivenim informacijama poslužiti u daljnjem kontaktu s traženom uslugom.

2.8.5 WS-Security

Sigurnost je jedan od ključnih problema koji se navodi kao nedovoljno zastupljen kod tehnologije Usluga weba. Pošto se često naglašava javna dostupnost Usluga weba kroz sveprisutne internetske protokole opravdano se postavlja pitanje sigurnosnih protokola i mehanizama koji će te usluge zaštititi od zlonamjernih pristupa.

Proširenje *WS-Security* zapravo sadrži veći broj specifikacija za rješavanje raznih problema sigurnosti. Veliki broj tih specifikacija nalazi se u raznim fazama prihvaćenosti te je teško govoriti o proširenju *WS-Security* kao jedinstvenom kompaktnom proširenju koje donosi jasno definirane sigurnosne mehanizme u uslužno orijentirane arhitekture zasnovane na Uslugama weba.

Neke od važnijih specifikacija unutar proširenja *WS-Security* su:

- *WS-Security* - specifikacija koja dijeli ime sa skupnim imenom proširenja a označava glavne jezične elemente koji se koriste unutar sigurnosne infrastrukture
- *XML-Encryption* – nije *de facto* proširenje Usluga weba već je usko povezana s normom XML, no smatra se dijelom *WS-Security* zbog činjenice da se šifriranje poruka unutar uslužno orijentiranih arhitektura najčešće svode na šifriranje XML-ovskog sadržaja
- *XML-Signature* – analogno prethodnoj specifikaciji, *XML-Signature* je povezana s normom XML a pruža mogućnost digitalnog potpisivanja XML-ovskih poruka

Ove tri specifikacije se najčešće spominju kao jezgra proširenja *WS-Security* te se smatra da pružaju dovoljno elemenata za izgradnju adekvatnih sigurnosnih mehanizama unutar informacijskog sustava zasnovanog na uslužno orijentiranoj arhitekturi.

3 Uslužno orijentirane arhitekture i stvarnovremensko skladištenje podataka

Primjena uslužno orijentiranih arhitektura u poslovnom svijetu je u stalnom porastu. Prema [10] dosadašnja iskustva velikih – ali i srednjih i malih poduzetništva – iznimno su optimistična kada se radi o konkretnim primjenama arhitekture SOA. Strateška vrijednost uslužno orijentiranih aplikacija predstavlja idealne temelje za razvoj budućih rješenja takve vrste a zadovoljstvo dosadašnjim rješenjima samo govori tome u prilog.

Zbog svoje fleksibilne i modularne prirode uslužno orijentirani sustavi prodiru u sve aspekte elektroničkog poslovanja, te se konstantno traže nove primjene koje bi dalje iskorištavale prednosti ovakvog tipa dizajna informacijskih sustava. Jedno od područja za koje je korist uslužno orijentiranih principa iznimno očigledna je područje *skladištenja podataka*, većinom zbog svoje heterogene prirode i očitom potrebom za integracijom raznorodnih sustava (što do danas predstavlja najprihvaćeniju primjenu arhitektura SOA [11]).

U ovom poglavlju bit će dan kratki pregled pojma skladištenja podataka, s posebnim naglaskom na problematiku *stvarnovremenskog skladištenja* (engl. *real-time data warehousing*). Stvarnovremensko skladištenje je relativno nova pojava u svijetu skladištenja podataka koja donosi vlastite zahtjeve i probleme i za koje se traže moderna, učinkovita rješenja zasnovana na najnovijim tehnologijama. Jedan tip takvih rješenja zasnovan je na arhitekturi SOA, čime će se baviti i veliki dio ovog rada.

3.1 Skladištenje podataka

Skladište podataka, kako je prvobitno definirao W.H.Inmon u [12], je *"subjektno-orijentiran, integriran, vremenski varijabilan, nepromjenjiv skup podataka koji služi kao podrška za donošenje poslovnih odluka".* R.Kimball u [13] nudi jednostavnu, neformalniju definiciju: "*Skladište podataka je kopija transakcijskih podataka za potrebe upita i analiza.*"

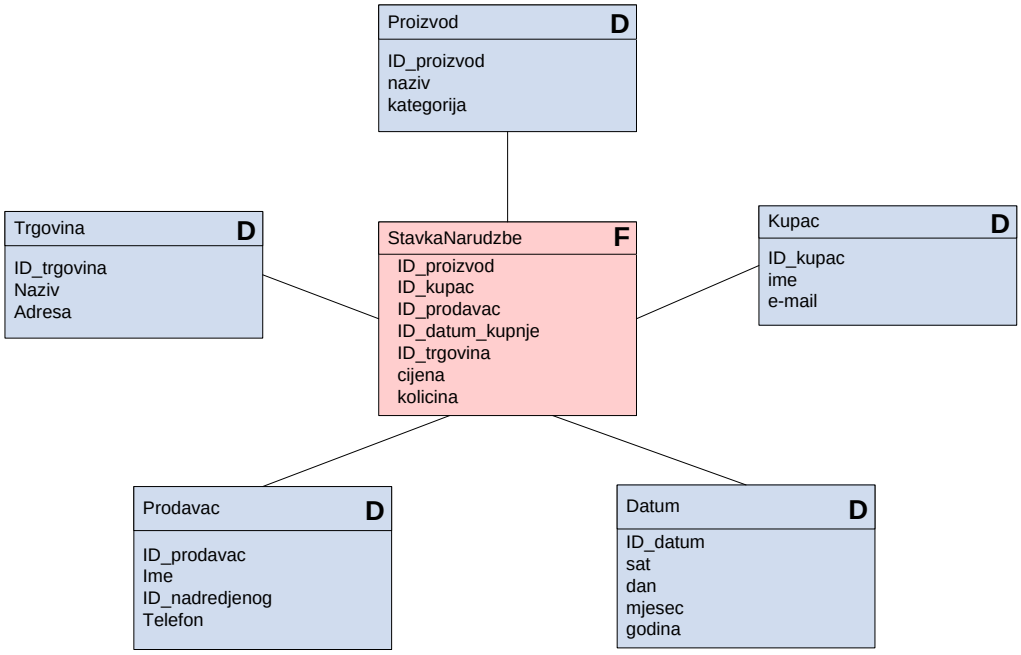
Jedna od osnovnih ideja skladišta podataka jest pružanje temeljne baze podataka za potrebe aplikacija poslovnog odlučivanja. Podaci u skladištu predstavljaju integriranu i prilagođenu verziju podataka iz transakcijskih izvora pohranjenih tako da predstavljaju konzistentnu i točnu sliku povijesnih podataka vezanih uz određeni aspekt poslovanja. Baza podataka koja se koristi kao skladište često je optimizirana za brzo izvođenje velikog broja jednostavnih i složenih upita nad vremenski konstantnim skupom podataka. Skladište podataka često se dodatna razdvaja u skup tzv. *data mart*-ova od kojih je svaki orijentiran određenim poslovnim potrebama. *Data mart* može predstavljati logički podskup skladišta ali također i fizički izdvojenu bazu podataka koja preslikava podatke iz "glavnog" skladišta prije nego ih čini dostupnima za analizu.

Moderna skladišna podataka najčešće koriste tzv. "dimenzijski podatkovni model". Kod ovakvog modela podaci su organizirani u tzv. činjenice (engl. *facts*) i *dimenzije* (engl. *dimensions*).

Činjenice u općenitom slučaju predstavljaju numeričke podatke dobivene kao rezultat nekog mjerenja koje adekvatno reprezentiraju neki poslovni događaj (npr. količina prodanog proizvoda u nekoj trgovini na neki određeni dan). Činjenice predstavljaju ključnu komponentu skladišnog sustava te su najčešće glavni interesni skup podataka nad kojima se rade poslovne analize. Činjenice se pohranjuju u tzv. *činjenične tablice* (engl. *fact tables*). Svaki redak takve jedne tablice predstavlja jedno "mjerenje" vezano uz spomenutu poslovnu činjenicu. Mjerenje je definirano skupom "izmjerenih" vrijednosti te skupom stranih ključeva odgovarajućih dimenzija na koje se činjenica odnosi.

Dimenzije komplementiraju činjenice u smislu da pružaju kontekst izmjerenim vrijednostima. Svaka dimenzija predstavlja opis poslovnog entiteta na kojeg se činjenica odnosi (npr. proizvod, kupac ili datum). Dimenzije se pohranjuju u odgovarajuće *dimenzijske tablice* (engl. *dimension tables*). Svaki redak dimenzijske tablice opisuje određenu instancu poslovnog entiteta pomoću niza atributa. Atributi najčešće sadrže tekstualne opisne vrijednosti koje pobliže određuju dani entitet.

Slika 3-1 prikazuje primjer jednostavnog dimenzionalnog podatkovnog modela koji se sastoji od jedne činjenice i pet dimenzija. Činjenica (označena slovom F kao engl. *fact*) opisuje poslovni događaj prodaje određenog proizvoda. "Izmjerene" vrijednosti su cijena proizvoda i kupljena količina, dok su pripadajuće dimenzije kupac, proizvod, trgovin, zaposlenik koji je izvršio prodaju i vrijeme kupnje.

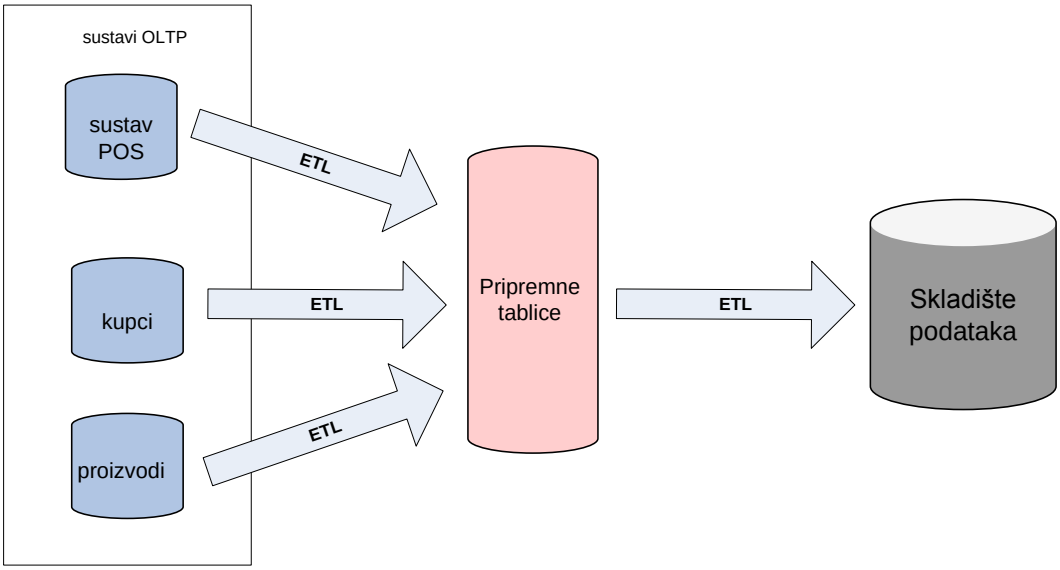


Slika 3-1 - Dimenzionalni model podataka

Sa slike se jasno može uočiti tzv. *zvjezdasta shema* koja je tipična za sustave skladištenja podataka. Za razliku od *normaliziranog modela* klasičnih sustava baza podataka čija je glavna svrha smanjivanje redundancije kod pohrane podataka radi očuvanja konzistentnosti i točnosti podataka, dimenzionalni model se često izvodi u *denormaliziranom obliku* koji sadrži namjerno uključenu redundanciju. Redundancija se uvodi radi redukcije potrebnih spajanja tablica kod vršenja složenih upita što rezultira

optimizacijom performansi skladišnog sustava. Očuvanje konzistentnosti podataka unutar sustava izvršava se kroz strogo kontrolirani proces ažuriranja podataka unutar skladišta.

Spomenuti proces, tj. prikupljanje podataka iz transakcijskih izvora, njihova prilagodba i učitavanje u skladište nazivaju se procesom ETL (engl. *Extraction, Transformation* and *Loading*). Ovaj proces prikazuje **Slika 3-2**.



Slika 3-2 - Proces ETL

Kao što se vidi iz slike, proces ETL komunicira s izvorima podataka koji su najčešće tzv. sustavi OLTP (engl. *Online Transaction Processing*) tj. sustavi za potporu tekućem poslovanju. Podaci se prikupljaju i pohranjuju u tzv. pripremne tablice (engl. *staging tables*). Potom se nad njima izvršavaju transformacije i prilagodbe kako bi se prikupljeni podaci preoblikovali u normu koju nameće skladište podataka. Konačno, podaci se učitavaju u skladište gdje postaju dostupni za poslovne analize.

Kao što je rečeno, skladišta podataka optimizirana su za brzo izvođenje složenih upita nad konstantnim skupom podataka. Skup je konstantan zato što se radi o nepromjenjivim, povijesnim podacima koji se ne mijenjaju niti ažuriraju. Stoga se klasični skladišni sustavi ažuriraju samo u diskretnim, većim vremenskim intervalima za koje vrijeme skladište podataka nije dostupno za izvršavanje upita i analiza. U to vrijeme "mirovanja" skladišta proces ETL prebacuje sve novopristigle podatke iz pripremnih tablica. Zbog činjenice da se često radi o velikim količinama podataka ovaj proces često je resursno i vremenski zahtjevan.

Ovakav model ažuriranja skladišta je u zadnjem desetljeću polako postao nedostatan zbog globalizacije i modernizacije poslovanja te sve većeg vrjednovanja što svježijih poslovnih informacija kod donošenja poslovnih odluka. Sve to dovelo je do pojave koncepta tzv. *stvarnovremenskog skladištenja podataka* koje je tema poglavlja 3.2.

3.2 Stvarnovremensko skladištenje podataka

Kao što je rečeno u poglavlju 3.1, u tradicionalnom smislu skladišta podataka ne sadržavaju najnovije poslovne informacije – tehnološki i poslovni zahtjevi takvih, "klasičnih" skladišnih sustava podrazumijevaju inherentno kašnjenje podataka ovisno o predefiniranom diskretnom vremenskom intervalu ažuriranja novim podacima. Nagli razvoj Interneta i srodnih tehnologija doveo je do znatnih promjena u sustavu vrednovanja ažurnosti poslovnih informacija, što je uvjetovalo i pomakom perspektive s klasičnog tipa skladištenja na noviji, tzv. stvarnovremenski model. Ovakav model prilagođen je modernom konceptu potrebe za dostupnošću pravovremenih poslovnih informacija. Kako su operativne odluke sve značajnije za poslovanje, tako i sustavi poslovne inteligencije sve više poprimaju karakteristike operativnih sustava, a skladišta podataka imaju dodatni imperativ – pružati podršku poslovnih analiza ne samo povijesnih, već i najnovijih poslovnih podataka. Drugim riječima, javlja se potreba za stvarnovremenskim skladištenjem podataka.

3.2.1 Kategorije "stvarnog vremena"

Stvarnovremensko skladištenje podataka nema strogu definiciju, nego se smisao pojma često mijenja ovisno o kontekstu u kojem se koristi. [14] navodi četiri najčešće korištena značenja ovog pojma:

- na vrijeme (engl. *on time*)
- simulirano stvarno vrijeme (engl. *simulated real-time*)
- pravo vrijeme (engl. *right time*)
- stvarno vrijeme (engl. *real time*)

Važno je naglasiti da ova četiri značenja nisu međusobno isključiva niti neovisna – hijerarhijski gledano, gornja značenja su svojevrsna generalizacija nižih.

Koncept "na vrijeme" jednostavno označava dogovor poslovnih korisnika i informatičke potpore skladištenja o dostavi podataka ovisno o trenutnim poslovnim zahtjevima i potrebama. Stvarno vrijeme nije strogo uvjetovano ni poslovno, ni tehnološki, iako nije isključeno – brzina poslovanja je takva da joj tehnološka infrastruktura predstavlja adekvatnu potporu neovisno o konkretnoj brzini premještanja operativnih poslovnih podataka u skladište.

Koncept "simulirano stvarno vrijeme" označava da je nužno osigurati izvršavanje poslovnih analiza u realnom vremenu, dok je određena latencija[5] podataka prihvatljiva. Drugim riječima, poslovni korisnik mora imati interaktivni odnos s aplikacijom za poslovne analize, ali podaci se i dalje mogu učitavati na "klasičan" način masovnog (engl. *bulk*) prijenosa u diskretnim vremenskim intervalima. Ovo se najčešće realizira prediktivnim računanjem tj. računanjem i pohranom rezultata često korištenih poslovnih upita unaprijed unutar skladišta kako bi aplikacijama poslovnog odlučivanja bili dostupni "u stvarnom vremenu"[13].

Koncept "pravog vremena" je najčešće korišteno značenje stvarnovremenskog skladištenja, a označava hitnu dostavu nove poslovne informacije u skladište prema trenutnom dostupnim mogućnostima, npr.

[5] vremenski interval između trenutka pojave poslovnog događaja do trenutka kada podaci koji reprezentiraju opis tog poslovnog događaja postanu dostupni u skladištu.

korištenjem tehnologija integracije poslovnih aplikacija (EAI – *engl. Enterprise Application Integration*). Nulta latencija nije nametnuta kao zahtjev, ali očekuje se da skladište pruža uvid u što svježije poslovne informacije.

Čisto stvarno vrijeme (engl. *real time*) označava najstrožu definiciju stvarnovremenskog skladištenja, gdje je poslovni imperativ informaciju dostaviti u skladište u najkraćem vremenu koje dopuštaju fizička i tehnološka ograničenja. Ovakav način diktira transakcijsku kvalitetu ažuriranja informacija – cijeli informatički sustav je "zaključan" za to vrijeme kako bi se zaista garantirala "stvarnovremenost" poslovne informacije. Poslovni sustavi koji zahtijevaju "čisto" stvarno vrijeme često moraju posezati strogo specijaliziranim hardverskim i softverskim rješenjima koji mogu ovakvo nešto realizirati. Jedno od danas najpopularnijih takvih rješenja nudi tvrtka Netezza.

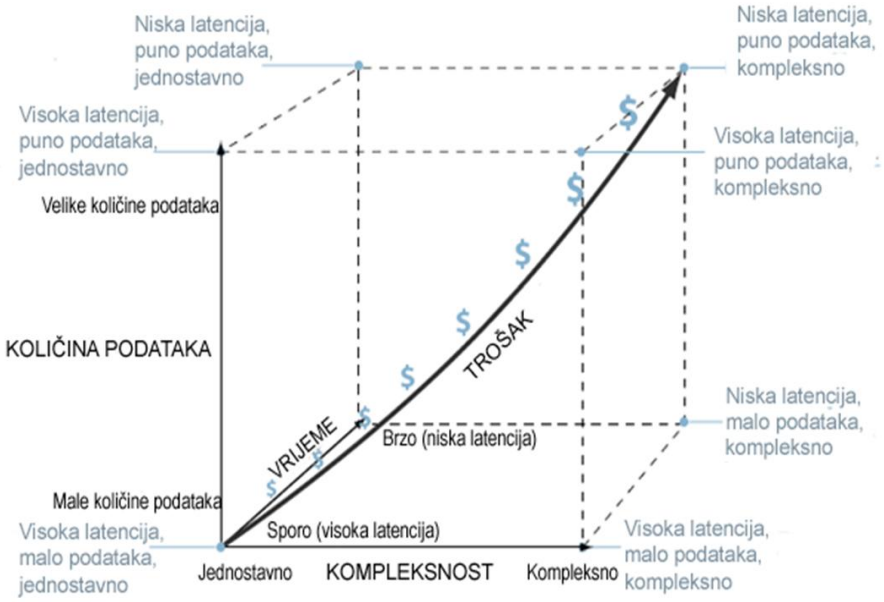
3.2.2 Motivacija, prednosti i problemi stvarnovremenskog skladištenja

Glavni razlog prelaska na stvarnovremensko skladištenje, pored poslovnih imperativa, jest razvoj tehnologije koja takva rješenja čini mogućim. Dok se krajem 20. stoljeća smatralo da su operativne tehnologije i tehnologije skladištenja međusobno polarizirane te se u slučaju poslovnih potreba moralo posezati za kompromisnim rješenjima (kao što je npr. ODS – engl. *Operational Data Store*), danas se općenito smatra da skladište podataka ne treba nužno biti segmentirano niti strogo odvojeno od operativnih sustava da bi vršilo svoju funkciju. Jedinstveni, integrirani pogled na agregirane ali i detaljne, povijesne ali i nove podatke je danas tehnološki izvediv.

Drugi razlog je globalizacija poslovanja koja rezultira sve manjim vremenskim prozorom mirovanja sustava (engl. *downtime*) kada je pogodno izvršavati zadatke koji zahtijevaju izvjesno opterećenje poslovne informatičke infrastrukture kao što je prikupljanje podataka i punjenje skladišta. Količine poslovnih podataka sve više rastu, a dostupni prozor "mirovanja" se smanjuje, tako da koncept učitavanja velike količine podataka (engl. *bulk loading*) povezan s tradicionalnim skladištenjem podataka sve manje odgovara modernom tipu poslovanja.

Stalni tok informacija iz operativnih sustava u skladišne sustave stoga postaje prikladnije rješenje, ali i glavna osnova dodatnog razloga uvođenja stvarnovremenskog skladištenja, a to je tzv. zatvaranje petlje (engl. *closed loop*) između operativnih i skladišnih tj. analitičkih sustava. Sustavi podrške odlukama postaju sve manje strateški a sve više operativni, što je pogotovo uočljivo u sustavima CRM (engl. *Customer Relations Management*), tj. sustavima koji upravljaju odnosom s klijentima. Takvi sustavi često moraju u vrlo kratkom vremenu (npr. dok klijent čeka na telefonskoj liniji) pomoći pri donošenju poslovne odluke zasnovane na složenim poslovnim analizama koje nije dostupne u operativnim sustavima.

S druge strane, stvarnovremensko skladištenje sa sobom nosi i određene probleme. Najveći problem su potrebe za dodatnim ljudskim, financijskim i informatičkim resursima koje eksponencijalno rastu s padom latencije podataka. **Slika 3-3** (preuzeta iz [14]) prikazuje porast troška ovisno o varijabli vremena (tj. smanjenja latencije), kompleksnosti i količine poslovnih podataka.



Slika 3-3 - Ovisnost troška o složenosti, količini i latenciji podataka

Uz to, proces prikupljanja, transformacije i učitavanja podataka u skladište je često iznimno složen process koji zahtijeva znatne resurse ali zato garantira kvalitetu podataka u skladištu. Imperativ izvođenja ETL-a u stvarnom vremenu je jedan od ključnih problema stvarnovremenskog skladištenja (zbog čega se stvarnovremensko skladištenje često smatra ekvivalentnim sa "stvarnovremenskim ETL-om").

Kod stvarnovremenskog skladištenja od najveće je važnosti racionalna evaluacija poslovnih uvjeta i potreba za uvođenjem takvog koncepta. Stvarnovremensko skladištenje **mora** biti opravdano adekvatnim poslovnim potrebama, ali isto tako i podržano tehnološkom i poslovnom infrastrukturom [15]. Na primjer, ako operativni poslovni sustav inherentno prikuplja podatke s određenom latencijom, onda nema smisla ulagati u stvarnovremenski skladišni sustav koji će raditi s latencijom manjom od operativnim poslovnim sustavom određene gornje granice. Također, ako poslovne analize ne ovise bitno o najsvježijim podacima, onda ulaganje u stvarnovremensko skladištenje možda nije potrebno niti opravdano, pogotovo ako uvodi bitno povećanje kompleksnosti sustava – ako je "simulirano" stvarno vrijeme dovoljno, onda možda i tradicionalni sustav skladištenja već odgovara traženim zahtjevima.

3.2.3 Postojeća rješenja stvarnovremenskog skladištenja

Istinsko stvarno vrijeme (kao što je definirano u [14]) karakterizirano je sinkronim ažuriranjem podataka koje zaključava bazu, mreže i resurse dnevnika sve do svog završetka kako bi bilo osigurano da su dostupni podaci uvijek najnoviji. Ideal stvarnovremenskog skladištenja je skladište nulte latencije, tj. ZWDH *(*engl. *Zero-Latency Data Warehouse*) *[16]*. Realizacija ovakvih rješenja je najzahtjevnija i najskuplja te zahtijeva specijalizirane tehnologije (npr. u [17] je predloženo korištenje Grid tehnologija za analizu velikih količina podataka u stvarnom vremenu.)

U realnom slučaju jedino opravdanje za postizanje nulte latencije je nužnost donošenja analitičkih odluka u stvarnom vremenu (kao npr. otkrivanje neovlaštenog korištenja kreditnih kartica), no u većini slučajeva "pravo vrijeme" (*right time*) je adekvatna inačica stvarnovremenskog skladištenja koja odgovara poslovnim potrebama.

[18] navodi neka od rješenja za stvarnovremensko skladištenje: prvo i najjednostavnije je povećanje frekvencije osvježavanja informacije u klasičnom skladištu (npr. umjesto jednom dnevno više puta na dan). Ako ovo odgovara poslovnim potrebama i ako informatički sustav ne dolazi u preopterećenje ovo je adekvatno i preporučeno rješenje.

Ako postojeći sustav ne može podnijeti prečesta osvježavanja onda je potrebno osigurati mehanizam kontinuirane "dostave" informacija u skladište. Ovo se može postići na više načina. Naziv "*microbatch ETL"* [19] označava specijalizirane sustave koji provode "klasične" ETL procedure ali puno većom frekvencijom i na manjoj količini podataka, što u većini slučajeva odgovara poslovnoj slici stvarnovremenskog sustava. Složeniji sustavi mogu koristiti principe integracije poslovnih aplikacija (EAI – *Enterprise Application Integration)* za dostavu i transformaciju podataka, o čemu će više riječi biti u idućem poglavlju.

Jednom kad informacija dođe do sustava skladišta, otvara se problematika politike ažuriranja skladišnih podataka. Jedno od rješenja je tzv. ažuriranje curenjem (engl. *trickle feed*) gdje se informacije odmah pohranjuju u odgovarajuću tablicu unutar skladišta. Ovo rješenje najčešće nije prihvatljivo, jer sustav skladišta nije dizajniran za konstantne promjene, pa može doći do znatnih usporavanja kako kod ažuriranja informacija, tako i kod izvođenja upita nad skladištem od strane aplikacija za poslovno odlučivanje.

Zbog toga se obično preporučuje izgradnja stvarnovremenske particije i korištenje tzv. "curi i okreni" (engl. *trickle & flip*) sistema ažuriranja. Stvarnovremenska particija je spremište za podatke koje ima ekvivalentnu strukturu odgovarajućim tablicama u skladištu i pohranjuje podatke koji kontinuirano dolaze, a u diskretnim intervalima skladište podataka se "ažurira" tim podacima ovisno o mogućnostima postojećeg skladišnog sustava. Kako bi se izbjeglo dodatno opterećenje nad skladištem i negativne posljedice koje dolaze s tim, često se stvarnovremenska particija nalazi u zasebnoj bazi podataka ili u tzv. vanjskoj predmemoriji (RTDC –engl. *external real-time data cache*).

Kod uvođenja stvarnovremenske particije izvjesne se promjene moraju izvesti nad prezentacijskim segmentom skladišta, tj. načinom na koji aplikacije poslovnog odlučivanja pristupaju informacijama u skladištu. Ako poslovna analiza zahtijeva kombinaciju novih i povijesnih podataka, to se može riješiti preko dizajna odgovarajućih pogleda (engl. *view*) nad tim podacima koji će transparentno prikazivati i agregirati oba tipa podataka.

Problemi se mogu pojaviti kod izvođenja složenijih analiza čija se unutrašnja struktura sastoji od izvođenja niza SQL upita i čije izvršavanje nužno zahtijeva određeni vremenski period. Zbog čestog ažuriranja stvarnovremenske particije moglo bi se dogoditi da se taj vremenski period znatno

produži (za razliku od upita nad strogo povijesnim podacima koji su statični za cijelo vrijeme izvođenja upita). Dodatno, za vrijeme izvođenja složenog upita može doći do izmjena nad podacima čime rezultati analize mogu postati nekonzistentni. Zbog svega toga treba posvetiti posebnu pažnju odnosu između aplikacija poslovnog odlučivanja i stvarnovremenskog skladišta, te eventualno razgraničiti koji upiti nužno zahtijevaju nove podatke (te adekvatno razraditi politiku njihovog izvođenja) a kod kojih korištenje stvarnovremenskih podataka nije presudno dok potencijalno može donijeti više problema nego prednosti kad se radi o poslovnom odlučivanju zasnovanom na rezultatima takve analize.

3.3 Integracija poslovnih aplikacija i stvarnovremensko skladištenje podataka

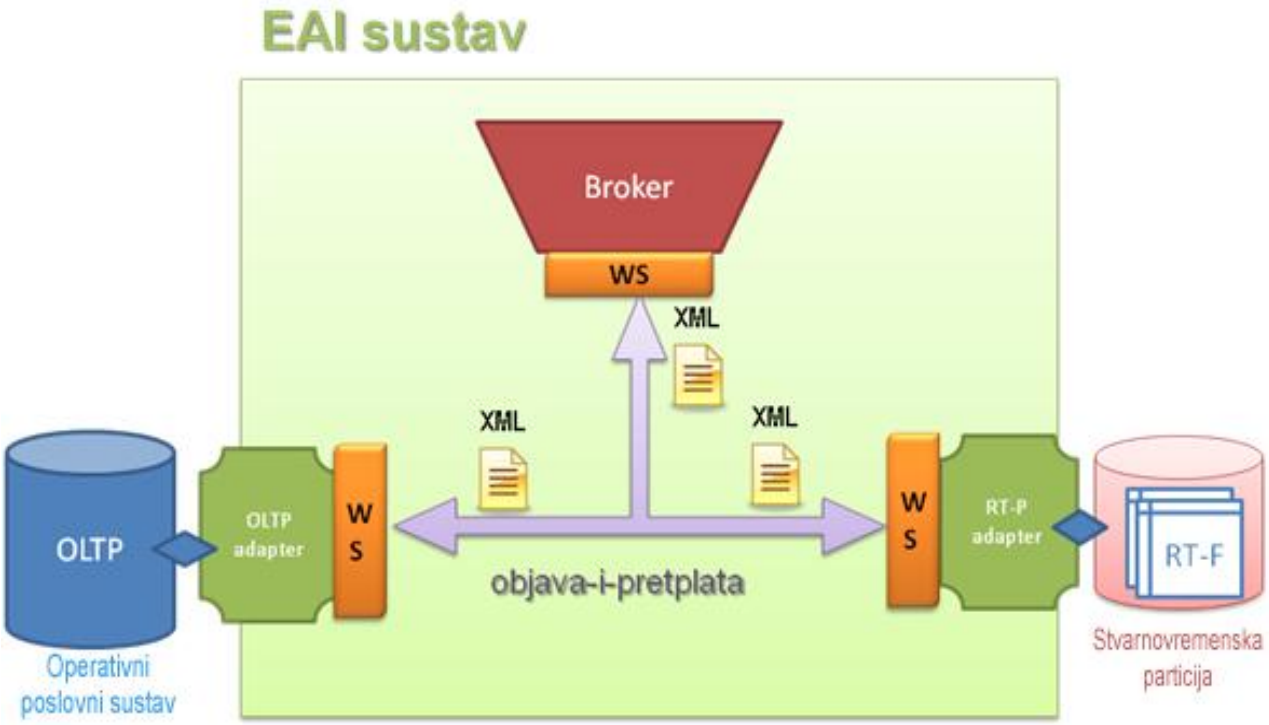
R.Kimball u [13] kao jednu od metoda implementacije stvarnovremenskog skladištenja navodi koncept korištenja stvarnovremenske particije zajedno s principima EAI (engl. *Enterprise Application Integration* – integracija poslovnih aplikacija). Kao što je već rečeno u poglavlju 1, implementacija ovih principa je danas najkorištenija uporaba uslužno orijentirane arhitekture. U ovom poglavlju bit će detaljno razrađena problematika korištenja ovog koncepta za implementaciju sustava stvarnovremenskog skladištenja.

EAI kao princip najčešće podrazumijeva izgradnju sustava sastavljenog od komponenata koje međusobno komuniciraju preko standardiziranih poruka. Određene komponente toga sustava nazivaju se adapterima i one se naslanjaju na "vanjske" komponente koje je potrebno integrirati u sustav. Adapteri omogućuju vanjskim komponentama funkcioniranje bez potrebe za poznavanjem unutarnje logike poslovnih procesa u kojima sudjeluju i integriranog poslovnog sustava općenito, čime se osigurava visoka fleksibilnost integracije. Kao što je rečeno u poglavlju 2, kod korištenja arhitekture SOA ovi adapteri zapravo su omotači Usluga weba.

Poruke se najčešće razmjenjuju po principu objavi-i-pretplati (engl. *publish-and-subscribe*) pri čemu obično postoji središnja upravljačka komponenta (engl. *broker*) čija zadaća je primanje i raspodjela poruka u sustavu. Adapterske i ostale komponente sustava EAI mogu se "pretplatiti" na određeni tip poruka, a brokerska komponenta će nakon primitka svake poruke svim pretplaćenim primateljima tu poruku i proslijediti. Naravno, uvijek je moguća i jednostavnija arhitektura gdje pošiljatelj i primatelj direktno komuniciraju preko poruka, no model „objavi-i-pretplati“ omogućuje dodatnu fleksibilnost i veću otvorenost sustava.

Poruke koje se razmjenjuju moraju biti normizirane unutar sustava kako bi se olakšala integracija. Adapteri, koji predstavljaju glavne točke kontakta s "vanjskim svijetom", posjeduju transformacijsku logiku pretvorbe standardnih poruka u format kojeg očekuju vanjske aplikacije. U najvećem broju slučajeva – pogotovo kada se radi o implementacijama kroz Usluge weba – jezik XML je najpodobniji za implementaciju standardnih poruka, pri čemu se pomoću XML-ovskih shema postiže definicija i restrikcija tipova poruka koje se razmjenjuju u sustavu.

Slika 3-4 prikazuje primjer implementacije stvarnovremenskog skladištenja podataka uz pomoć sustava EAI (pravokutnici sa slovima WS označavaju sučelja Usluga weba).



Slika 3-4 - Uporaba sustava EAI za stvarnovremensko skladištenje

Glavna zadaća ovakvog sustava je prikupljanje podataka iz operativnih poslovnih sustava (u smislu sustava podrške poslovne operative, ne klasičnih "operativnih sustava"), transformacija i učitavanje u skladišni sustav (u ovom primjeru predviđeno je učitavanje u odgovarajuću stvarnovremensku particiju).

S lijeve strane nalaze se adapteri na operativne sustave koji moraju na adekvatan način registrirati pojavu nove poslovne informacije, prikupiti tu informaciju, a sve to uz minimalni utjecaj na performanse operativnog poslovnog sustava. Adapteri potom transformiraju prikupljene podatke u normiziranu XML-ovsku poruku koja se prosljeđuje brokerskoj komponenti, koja nadalje prosljeđuje poruku pretplaćenim primateljima, poglavito adapterskoj komponenti stvarnovremenske particije.

Na primjer, operativni sustav može biti sustav POS (engl. *Point-of-sale*) koji služi kao poslovna podrška prodajnom mjestu tvrtke. Kada adapter sustava POS realiziran kao slušateljska aplikacija (engl. *listener*) nad bazom POS-a registrira pojavu novih činjenica (tj. zapisa o izvršenoj prodaji) on prikuplja te informacije i transformira ih u otprije definirani XML-ovski dokument predviđen za pohranu informacija vezanih uz prodaju. Taj dokument se prosljeđuje sučelju Usluge weba brokerske komponente.

Broker analizira primljenu poruku i prema identifikatoru (i eventualnoj provjeri valjanosti prema XML-ovskoj shemi) utvrđuje da se radi o činjenicama vezanim uz prodaju. Iz popisa adresa Usluga weba primatelja broker pronalazi odgovarajuću stavku (adresu adaptera stvarnovremenske particije) te prosljeđuje poruku na tu adresu, a i ostalim primateljima ako takvi eventualno postoje. Adapter stvarnovremenske particije prima XML-ovsku poruku, prikuplja i transformira informacije iz nje u konačni oblik prilagođen učitavanju u stvarnovremensku particiju (npr. niz SQL naredbi).

Konačno, informacije se pohranjuju u stvarnovremenskoj particiji gdje ih se može koristiti za poslovne analize od strane aplikacija za poslovno odlučivanje.

Izvedba ovakvog sustava postavlja nekoliko problemskih područja. Prvo je vezano uz prikupljanje informacija tj. način na koji adapter operativnih sustava registrira pojavu nove poslovne informacije. Drugo je pitanja prikupljanja i transformacija podataka te odnos između ovih radnji s klasičnim ETL-ovskim sustavom. Konačno, mora se riješiti pitanje životnog toka informacija u stvarnovremenskoj particiji tj. politike njihovog prebacivanja u klasično skladište ili brisanja kada postaju redundantne zbog toga što su se isti podaci preko drugih komunikacijskih kanala dostavili u klasični skladišni sustav.

Registracija pojave novog poslovnog događaja nije trivijalan problem a postoji više rješenja koja su više ili manje prikladna ovisno o poslovnoj situaciji. Prvo se treba definirati izvor "događaja" tj. pokretač procesa prikupljanja – odluka da li će to biti operativni poslovni sustav ili sam adapter.

Prednost odabira operativnog poslovnog sustava kao pokretača – npr. kroz implementaciju okidača (engl. *trigger*) u bazi podataka operativnog sustava - uključuje minimalizaciju latencije (jer se informacija šalje u skladište odmah po pojavi poslovnog događaja koji ju je uzrokovao) i mogućnost implementacije adaptera kao strogo reaktivne komponente koja je bitno jednostavnija za implementaciju. Ovim postupkom izvor iz gledišta skladišnog sustava postaje "inteligentan" [20]. Nedostatak je potreba za direktnim izmjenama operativnih poslovnih sustava što može ne samo negativno utjecati na njihove performanse nego i ugroziti njihovu funkcionalnost.

Ako je adapter inicijator procesa prikupljanja podataka, onda je najčešće jedini način registracije novih podataka provjera stanja operativnog sustava postavljanje upita nad njim u nekim pravilnim vremenskim razmacima. Ovdje se mora pažljivo odrediti frekvencija upita – ako se upiti postavljaju prerijetko, onda opet dolazi do latencije u prikupljanju koja možda nije prihvatlljiva od strane poslovnih zahtjeva. S druge strane, prečesti upiti nad operativnim sustavom mogu ugroziti njegove performanse.

Adapter također mora posjedovati logiku razlučivanja novih poslovnih informacija što je usko vezano za konkretnu implementaciju poslovnog sustava – najčešće se radi o usporedbi vremenske oznake (engl. *timestamp*) posljednje stavke s vremenskom stavkom zadnjih prikupljenih podataka, analizi dnevnika promjena (engl. *change logs*) ako su dostupni, a u nedostatku nikakvih drugih podobnih mehanizama (ili imperativu minimalnog utjecaja na operativne sustave).

Drugi problem nastaje nakon prikupljanja, a to je pitanje transformacije podataka s usporedbom na klasične sustave ETL. Transformacijska logika situirana je na dvije lokacije – prvo unutar ili u blizini adaptera na operativne sustave gdje se informacije pretvara u standardni XML-ovski dokument, a drugo kod adaptera stvarnovremenske particije gdje se ona pretvara u oblik prikladan skladištu podataka. "Klasičan" proces ETL morao bi u velikoj mjeri biti izveden kroz ove dvije transformacije. Iz tog razloga od presudne je važnosti definirati sam izgled – tj. shemu - standardizirane XML-ovske poruke koja će predstavljati svojevrstan spremnik za informacije koje se prikupljaju i kao takav u najvećoj mogućoj mjeri olakšati transformaciju podataka u prikladni oblik za skladištenje. Procedure procesa ETL mogu se koristiti kao glavni predložak za implementaciju kako transformacija tako i shema XML-ovskih dokumenta. Ako su neki segmenti tih procedura neizvedivi glede uvjeta latencije podataka, onda se

moraju uvesti određeni kompromisi, tj. mora se prihvatiti izvjesna nepotpunost informacija u stvarnovremenskoj particiji. Konačno, uvijek postoji i opcija ulaganja u usko specijalizirana rješenja koja će odgovarati strogim poslovnim zahtjevima.

Treći problem je pitanje premještanja u glavno, statično skladište podataka. Ovo izlazi iz okvira sustava EAI (čija je glavna zadaća dostava podataka u stvarnovremenku particiji) i zahtjeva primjenu odabrane politike ažuriranja koja je u funkciji nad statičnim skladištem podataka. Stvarnovremenska particija može služiti striktno kao podrška za omogućavanje dostupnosti podataka niske latencije koji se nakon izvjesnog vremena brišu, a "klasični" sustav ETL tradicionalnom i otprije uspostavljenom metodom iz operativnih izvora prikuplja te iste podatke odjednom u velikoj količini te ih provodi kroz standardnu proceduru transformacije i učitavanja. Ovaj pristup povoljan je zbog toga što ne zahtijeva nikakvu izmjenu postojećeg, statičnog skladišnog sustava i što je zajamčena uobičajena kvaliteta povijesnih podataka u skladištu.

S druge strane, skladište može stvarnovremensku particiju koristiti kao novi "izvor" podataka za skladištenje tj. kao alternativni izvor umjesto operativnih poslovnih sustava. Glavna prednost ovoga rješenja je veće razgraničenje skladišnog i operativnog sustava i neovisnost o stanju operativnih sustava u vrijeme prikupljanja podataka. Nedostatak je nužnost izmjena postojećeg skladišnog sustava, ali i problem u situaciji kad se stvarnovremenska particija puni samo izvjesnim dijelom svih dostupnih podataka potrebnih za skladištenje.

3.4 Prednosti i nedostaci korištenja uslužno orijentiranog sustava za implementaciju stvarnovremenskog skladištenja

Stvarnovremensko skladištenje podataka predstavlja viziju skladištenja budućnosti, no u današnjim uvjetima svaka implementacija zahtijeva racionalnu analizu poslovnih zahtjeva i svojevrsno odolijevanje trendu krajnjeg smanjivanja latencije ako za to nema realnih poslovnih opravdanja. Iako tehnologija čini ovakav vid skladištenja sve prihvatljivijom i dostupnijom opcijom, povećanje troškova implementacije zajedno sa zakonom opadajućih prinosa nakon izvjesne minimalne granice latencije još uvijek diktiraju potrebu za pažljivim odabirom adekvatnog rješenja koje često može u sebi sadržavati izvjesne kompromise i/ili odstupanja od inicijalne ideje implementacije skladištenja u stvarnom vremenu.

Uslužno orijentirani sustav izveden pomoću otvorenih i široko dostupnih tehnologija (programski jezik Java, internetski protokoli i tehnologije) predstavlja kvalitetnu podlogu za izradu vlastitih prilagođenih rješenja implementacija stvarnovremenskog skladištenja, pogotovo kad se radi o problemu dostavljanja podataka iz izvora u skladište (ili njegovu stvarnovremensku particiju). Normizirane usluge uvelike olakšavaju integraciju heterogenih informacijskih sustava a kroz cijeli proces skladištenja mogu se koristiti uobičajene prednosti uslužno orijentiranih principa – modularnost, fleksibilnost, lakše održavanje, nadgledanje i proširivanje sustava. Ovakvim pristupom proces prikupljanja, transformacije i pohrane podataka za potrebe skladištenja postaje poslovni proces ekvivalentan ostalim procesima tvrtke te može transparentno koristiti usluge izvedene za potrebe ostalih poslovnih procesa (i obrnuto – poslovni procesi tvrtke koji nemaju direktne veze sa skladištenjem mogu koristiti usluge integrirane u proces skladištenja ako su izvedene tako da njihova funkcionalnost nije ograničena na ulogu koju preuzimaju u samom skladišnom procesu). Dodatno, proces skladištenja se može proširiti i vanjskim

uslugama koje pružaju dodatne informacije interesantne za poslovne analize, kao što su vremenske prilike na određenoj lokaciji u doba odvijanja poslovnih događaja, stanje dionica na burzi, tečaj stranih valuta i sl.

Naravno, treba imati u vidu da ovakva rješenja u nekim slučajevima ne mogu biti adekvatna u poslovnim situacijama koje zahtijevaju minimalnu (tj. nultu) latenciju. Implementacija uslužno orijentiranih sustava pomoću Usluga weba najčešće neće biti optimizirana za minimalno vrijeme procesiranja informacija – pogotovo kada se radi o složenoj manipulaciji XML-ovskim dokumentima što je integralni dio funkcionalnosti Usluga weba. Ako je minimalizacija latencije kritična za donošenje poslovnih odluka u određenom poslovnom okruženju tada se ipak mora posegnuti za specijaliziranim rješenjima koja odgovaraju danim zahtjevima (iako najčešće po cijenu duge, zahtjevne i skupe implementacije). No za znatni broj poslovnih okolina "skoro stvarno vrijeme" ili "pravo vrijeme" predstavlja pristupačan i adekvatan kompromis, a uslužno orijentirani sustavi ove uvjete mogu lako ispuniti te prema svojim karakteristikama znatno ubrzati i olakšati implementaciju stvarnovremenskog skladištenja. Konačno, kroz izvedbu studijskog primjera implementacije uslužno orijentiranog sustava za stvarnovremensko skladištenje (nazvanog SOA RT-ETL sustav) u poglavlju 6.7 prikazat će se da moderne tehnologije i informacijski sustavi mogu provoditi stvarnovremenski ETL s latencijom od svega nekoliko sekundi, što je najčešće sasvim dovoljno niska latencija za provođenje stvarnovremenskih poslovnih analiza.

4 Metapodaci i metamodeliranje

Dosadašnja poglavlja bavila su se uglavnom uslužno orijentiranim principima te njihovom implementacijom u svrhu integracije raznorodnih sustava. Poglavlje 3 prikazalo je kako se principi SOA mogu potencijalno učinkovito iskoristiti u svrhu stvarnovremenskog skladištenja podataka.

Kod integracije poslovnih sustava postoji još jedan element, nužan da bi se integracija mogla uspješno provesti. To su informacije koje opisuju karakteristike sustava koji sudjeluju u integraciji te problem razmjene informacija između poslovnih sustava. Usluge weba tj. principi SOA uvelike pomažu kod izgradnje "mostova" između sustava tj. olakšavaju implementaciju sučelja između sustava pružajući normizirane protokole komunikacije. No da bi jedan sustav uspješno komunicirao s drugim sustavom čestu su mu potrebne detaljne informacije o tom sustavu, informacije koje pružaju više informacija od jednostavnog opisa sučelja s kojim sustav komunicira. Te informacije pružaju metapodaci.

Ovo poglavlje bavit će se problematikom metapodataka – između ostalog njihovom normizacijom, metamodelima koji se danas nalaze na tržištu softvera te razmjenom metapodataka između sustava. Poglavlja 5 i 6 će potom prikazati scenarij sinergije uslužno orijentiranih principa i metapodataka za izvedbu jedne od mogućih implementacija za rješavanje problema stvarnovremenskog skladištenja.

4.1 Metapodaci i registri metapodataka

Pojam *metapodaci* doslovno znači "podaci o podacima" (počevši od činjenice da grčki prefiks *meta-* znači "poslije", "pored", "o" ili "uz"). David Marco u [21] daje sljedeću definiciju metapodataka gledano iz perspektive tvrtke-korisnika informacijskog sustava:

> *"Metapodaci su svi fizički podaci i informacije koje sadrže znanje o poslovnim i tehničkim karakteristikama procesa, podataka i struktura koje koristi tvrtka."*

Postoji više čimbenika zbog kojih se u poslovnim tvrtkama današnjice javlja sve veća potreba za kvalitetnim sustavom prikupljanja, pohrane i upravljanja metapodacima :

- integracija poslovnih sustava
- rast skladišta podataka i područnih skladišta (*engl. "data mart"*)
- potrebe poslovnih korisnika
- dokumentacija u slučaju odlaska zaposlenika iz tvrtke
- veće povjerenje korisnika informacijskih sustava u dostupne podatke

Veliki broj aplikacija unutar poslovnih informacijskih sustava izgrađenih tokom 80-tih i 90-tih godina prošlog stoljeća rađene su kao zatvoreni sustavi orijentirani vlastitoj funkcionalnosti bez potrebe za znanjem o poslovnoj okolini u kojoj se koriste i ostalim aplikacijama koje postoje u istom poslovnom okruženju. Takve aplikacije su kao svojevrsni "otoci" koji imaju vlastitu hardversku platformu, baze podataka te načine oblikovanja i pohrane podataka. Moderni poslovni sustavi zahtijevaju integraciju informacijskih sustava radi lakšeg upravljanja sustavom, održavanja te centraliziranog pogleda na podatke u sustav u cjelini, no zbog prirode poslovnih aplikacija to često može biti veliki implementacijski

problem. Velike tvrtke koje imaju dosta akvizicija pogotovo nailaze na velike poteškoće zbog raznolikosti informacijskih sustava koje tim načinom nasljeđuju a koji svi trebaju funkcionirati u istoj poslovnoj okolini. Dostupnost metapodataka koji na pregledan način dokumentiraju svojstva i ponašanje informacijskih sustava umnogome pomaže kod integracije takvih raznolikih sustava.

Što se skladišta podataka i poslovne inteligencije tiče, trendovi porasta kako samih skladišta tako i zahtjeva za količinom, raznolikošću i kvalitetom podataka su evidentni. Sustavi skladištenja i analitičke aplikacije potrebuju kvalitetne metapodatke kako bi se osiguralo adekvatno rješenje ovih zahtjeva.

Potrebe poslovnih korisnika također se mijenjaju – tehnologije analize podataka, metode dubinskog pretraživanja i sl. mijenjaju način na koji poslovni korisnici gledaju informacijski sustav i podatke općenito. Više nije dovoljno da znanje o sustavu posjeduju samo informatički stručnjaci, poslovni korisnici danas često imaju potrebu znati više detalja o samom sustavu kako bi mogli provoditi kvalitetnije analize a time i donositi bolje poslovne odluke. Kvalitetni metapodaci u ovom slučaju su nužni kako bi se poslovnim korisnicima omogućila dostupnost takvih informacija.

Metapodaci također predstavljaju dokumentaciju o informacijskom sustavu. Ovo je bitno ne samo kao referenca na postojeći sustav, nego i kao osiguranje tvrtke da je znanje o sustavu trajno pohranjeno i dostupno u slučaju odlaska zaposlenika odgovornih za upravljanje i održavanje sustava.

Konačno, metapodaci osiguravaju semantičku vezu između podataka i njihovih korisnika. Postavljanje podataka u odgovarajući kontekst te saznavanje informacija o njihovom porijeklu osigurava veće povjerenje u korisnika u podatke i veću sigurnost u njih kad se radi o donošenju poslovnih odluka čija je učinkovitost osjetljiva na kvalitetu i točnost ulaznih podataka.

4.2 Potreba za strategijom upravljanja metapodacima

"Strategija upravljanja metapodacima" je definicija i provedba konkretnih poteza koji se tiču zahtjeva i politika upravljanja s ciljem uspješne integracije poslovne okoline na razini metapodataka. Sam model metapodataka nije dovoljan, kao ni postojanje podrške za metapodatke od strane korištenih softvera ili tehnoloških normi koje oni koriste. Tehnologija sama po sebi ne definira niti omogućuje strategiju upravljanja metapodacima – upravo obrnuto; strategija upravljanja metapodacima diktira model i tehnologiju metapodataka koji će se koristiti.

Strategija upravljanja metapodacima može sadržavati sljedeće:

- sigurnosnu politiku upravljanja metapodacima
- način identifikacije izvora metapodataka te njihovih korisnika
- način identifikacije elemenata koji čine same metapodatke
- dogovor o semantici korištenih metapodataka
- riješen problem vlasništva nad pojedinim metapodacima
- pravila razmjene i unošenja promjena nad metapodacima
- pravila objavljivanja metapodataka
- način automatizacije procesa koji se tiču metapodataka
- eliminacija nepotrebne redundancije među metapodacima

Sigurnosna politika ključna je kod svake strategije upravljanja metapodacima. Po svojoj prirodi, metapodaci su osjetljivi podaci jer pružaju uvid u strukturu i strateški vrijedne informacije o tvrtki. Zbog toga model i/ili tehnološka podrška metapodacima mora neizostavno sadržavati sigurnosne mehanizme ako se oni koriste u poslovnoj okolini.

Prepoznavanje izvora metapodataka te njihovih korisnika umnogome utječe na kvalitetu samog sustava metapodataka. Njihova kvaliteta ovisi o kvaliteti procesa njihovog nastajanja, a znanje o korisnicima omogućuje bolje modeliranje sustava i njegovo prilagođavanje prema korisničkim zahtjevima.

U sustavu upravljanja metapodacima često se korisnicima sustava ili softverskim komponentama koje mu pristupaju zadaju određene uloge. Postoji pet osnovnih uloga:

- *autor (*engl*. author)* – izvor metapodataka – korisnik koji prikuplja metapodatke i oblikuje ih unutar okvira odabrane norme ili softverska komponenta koja stvara metapodatke
- *objavnik (eng. publisher)* – korisnik ili softverska komponenta koji na određeni način omogućuje dostupnost metapodataka
- *vlasnik (*engl*. owner)* – korisnik ili softverska komponenta koja ima vlasništvo tj. odgovornost nad određenim metapodatkovnim elementom
- *potrošač (*engl*. consumer)* – korisnik ili softverska komponenta koji metapodatke dohvaća i koristi za neku svoju svrhu
- *upravljač (*engl*. manager)* – korisnik ili softverska komponenta koja je zadužena za životni ciklus i upravljanje nad određenim metapodatkovnim elementom

Ove uloge i njihova implementacija najčešće nisu dio modela metapodataka ali su ključan dio same strategije upravljanja metapodacima.

Sustav upravljanja metapodacima mora omogućiti identifikaciju njegovih elemenata. Zbog toga takvi sustavi najčešće podržavaju korištenje tzv. UUID-ova (engl*. Universally Unique ID* – univerzalno jedinstveni identifikator) kako bi svaki element imao svoju jedinstvenu oznaku koja bi služila kao njegov "primarni ključ". Upravljanje identifikatorima također je jedan od bitnih problema strategije upravljanja zbog potrebe definiranja politika eventualne izmjene identifikatora kod objave elementa, pojave nove verzije i sl.

Semantika je ključna u strategiji upravljanja jer svaki metapodatak mora sustavu koji ga koristi označavati istu stvar (tj. metapodatkovni element mora biti semantički ekvivalentan u svakoj okolini koja ga koristi). Ovo se najčešće postiže pažljivim odabirom norme koja ima dovoljnu ekspresivnost kako bi omogućila semantičku anotaciju ali i dodatnim "semantičkim dogovorom" koji nastaje neovisno od modela metapodataka a kojeg donose sami sustavi – korisnici metapodataka.

Vlasništvo, objava, razmjena te izmjena metapodatkovnih elemenata su također bitni problemi sustava za upravljanje metapodacima. Kao što svaka baza podataka mora imati učinkovitu politiku upravljanja podacima u njoj, tako i registar metapodataka mora ove probleme riješiti na odgovarajući način.

Automatizacija poslovnih procesa ključna je za učinkovitost informatičkih tehnologija koje služe kao podrška poslovanju. Stoga je i strategija upravljanja metapodacima to učinkovitija što je razina automatizacije sustava veća.

Konačno, treba izbjegavati redundanciju u sustavu metapodataka zbog lakšeg održavanja konzistentnosti sustava i uklanjanja potencijalnih grešaka. Zbog toga se treba potruditi održavati jedinstvenu primarnu kopiju svakog elementa, strogo odrediti vlasništvo te definirati upravljačku politiku nad tim elementom kako bi se redundancija smanjila na najmanju moguću mjeru.

4.3 Jezici, metamodeliranje i četverorazinska meta-arhitektura

Pojam jezika općeprisutan je u ljudskoj svakodnevici, pa tako i u informatičkom svijetu. Jezici u informatici imaju razne svrhe, ali glavna ideja jezika je mogućnost izražavanja određenog skupa informacija koje imaju određenu svrhu na konzistentan način razumljiv entitetima sposobnim interpretirati odabrani jezik.

Jezik se može gledati i kao univerzalna apstrakcija koja omogućuje destiliranje ključnih karakteristika interesnog objekta u oblik koji odgovara perspektivi korisnika objekta. Ključna ideja je filtriranje nebitnih detalja uz zadržavanje odgovarajućeg konteksta kako bi apstrakcija imala smisla. Jezik opisuje trajnu pohranu tj. zapis apstrakcije te njezinu daljnju obradu. Ono što je bitno je zadržavanje *konzistentnosti* onoga što se opisuje kroz jezik – apstrakcija mora *sakrivati detalje* ali postojanje detalja je uvijek implicirano i slika visoke razine se ne smije ni na koji način razlikovati od slike nižih razina gledano iz perspektive koncepata koje opisuje i elemenata na koje se odnosi.

Jezik može pružati *konkretnu sintaksu* – tj. način zapisivanja informacija (npr. programskih naredbi) na određeni način – npr. tekstualni ili vizualni. Tekstualna sintaksa pogodna je zbog lakoće zapisa kroz normizirane simbole (slova) iako razumljivost i upravljivost takvih zapisa opada s količinom informacija koje se zapisuju; vizualna omogućuje intuitivni i razumljivi prikaz kompleksnih sustava, ali zbog svoje prirode ne može sadržavati previše detalja ako se određena razina razumljivosti želi održati. U praksi jezici najčešće koriste kombinaciju tekstualne i vizualne sintakse kako bi iskoristili prednosti oba načina prikaza.

Apstraktna sintaksa jezika opisuje vokabular koncepata koje jezik pruža te način na koji se oni koriste kako bi se formirali modeli. Vokabular se sastoji od koncepata, odnosa i pravila koja opisuju kako se koncepti i odnosi mogu pravilno kombinirati čime bi se dobile dobro oblikovane "rečenice". Apstraktna sintaksa je neovisna od konkretne sintakse i semantike – ključni su oblik i struktura dok su prezentacija i značenje nebitni.

U informatičkom svijetu često se stvara razlika između "jezika za modeliranje" i "programskih jezika", iako oni po svojim svojstvima nemaju velike različitosti. Oba tipa jezika najčešće imaju konkretnu i apstraktnu sintaksu te svoju semantiku, iako se tradicionalno smatra da jezici za modeliranje zbog svoje prirode imaju neformalnu semantiku dok programski jezici imaju strogo definirana semantička svojstva (pošto stvaraju izvršive aplikacije). Isto tako, jezici za modeliranje često se povezuju s vizualnom

sintaksom dok se programski jezici povezuju s tekstualnom; ovo je također arbitrarna usporedba jer jezici za modeliranje najčešće pružaju način pohrane modela u tekstualnom zapisu (npr. kroz XML-ovske dokumente) dok programski jezici mogu koristiti vizualne alate koji omogućuju stvaranje programa kroz vizualne simbole umjesto unošenja tekstualnih programskih naredbi. Konačno, uzrok stvaranja razlike između njih je ta što nema očiglednog izvornog jezika, tj. meta-jezika koji bi ukazao na srodnost njihovih svojstava ili njihovog porijekla.

Pojam **metamodel** zapravo označava "model jezika za modeliranje". Prefiks "meta" ponovno označava da se radi o pojmu na višoj razini apstrakcije od samog jezka za modeliranje. Metamodel može pružati načine za opisivanje apstraktne sintakse, konkretne sintakse ili semantike nekog jezika. Koncepte koji se koriste za opisivanje navedenog je opet potrebno tražiti na višoj razini aptrakcije, tj. za njih je potrebno odabrati meta-metamodel.

OMG grupa (engl. *Object Management Group*) je nezavisna organizacija koja se bavi izdavanjem normi poglavito vezanih uz modeliranje programa, poslovnih procesa, informacijskih sustava i sl. Njihova najpoznatija i najprihvaćenija specifikacija jest specifikacija jezika UML (engl. *Unified Modelling Language*). U svom konceptu "arhitekture zasnovane na modelu" (engl. *MDA – Model-driven Architecture*) – čiji je jedan od temelja i sam jezik UML – OMG grupa predlaže četverorazinsku metaarhitekturu u koju se mogu smjestiti jezici prema svojim svojstvima tj. odnosu prema modelima koje pružaju te drugim jezicima s kojima su eventualno na određeni način povezani. Ovu metaarhitekturu prikazuje **Tablica 4-1** [22]**.**

Tablica 4-1 - Četverorazinska metaarhitektura

Razina metamodela	Jezik metamodela	Primjer objekta
M3	MOF, XMI	meta-metamodel
M2	UML, XMI, CWM	metamodel
M1	XMI	relacijska shema, E-R dijagram, klasa
M0	Uređaj za pohranu	objekt, podaci u relacijskoj tablici

Razine su nazvane Mn (n=0,1,2,3) gdje se broj "n" može interpretirati kao "broj prefiksa 'meta'" ispred opisa razine. Isto tako, objekti na nižim razinama mogu se gledati kao "instance" objekata na višim, dok su objekti na višim razinama "apstraktni opisi" objekata na nižim.

Na razini M0 nalaze se sami podaci. To mogu biti objekti u programskom jeziku, retci u tablicama unutar baze podataka i sl.

Razina M1 sadrži "model" podataka na razini M0. U slučaju objektnih programskih jezika na razini M1 su predlošci objekata, tj. klase. Ako se radi o sustavu za upravljanje bazama podataka na razini M1 su

definicije tablica u koju se smještaju podaci (npr. SQL DDL naredbe). Na ovoj razini također se može naći model entitet-veza nekog sustava i sl.

Na razini M2 nalaze se metamodeli, tj. jezici koji pružaju sintaksu za stvaranje modela. Konačno, razina M3 je razina meta-metamodela – jezika za definiciju metamodela.

Ono što je bitno uočiti jest da elementi viših metarazina pružaju izgradbene blokove za definiciju elemenata nižih razina. Kada se bira način na koji će se modelirati neki sustav, najčešće se izbor svodi na odabir jezika na razini M2; taj odabir diktira kako će modeli sustava izgledati tj. koje će elemente sadržavati, koja ograničenja će biti na raspolaganju te koju će svrhu taj model naposljetku imati. Odabirom jezika na razini M2 može se početi s modeliranjem na razini M1 te konačnom implementacijom na razini M0.

Opravdano je postaviti pitanje zašto se razina M3 smatra "vršnom razinom" pogotovo jer se i elementi te razine – tzv. meta-metamodeli - moraju opisati pomoću određenih izgradbenih konstrukata koji se po definiciji složaja metarazina moraju nalaziti na razini *iznad*, tj. na razini koja bi se trebala nazvati M4. U principu nema razloga da se ne naznače i daljnje razine apstrakcije tj. broj metarazina teoretski može biti neograničen. Odabir četvrte kao konačne razine uzet je većinom iz praktičnih razloga – ograničenje vršne razine omogućuje definiciju meta-jezika kao polazišne točke dok neograničene metarazine imaju teoretski značaj, ali ne omogućuju praktičnu implementaciju.

OMG je definirao jezik razine M3 i nazvao ga MOF (engl. *Meta-Object Facility*). Jezik je rekurzivno opisiv tj. može se sam opisati pomoću elemenata koje definira, čime je riješen je problem egzistencije viših metarazina. Jezik MOF predstavlja generičku polazišnu točku za izgradbene blokove pomoću kojih se mogu definirati jezici na razini M2.

U sljedećem poglavlju bit će dan detaljni pregled meta-jezika MOF. Potom slijede osvrti na dva najpopularnija jezika koji su definirani kroz meta-jezik MOF – UML (engl. *Unified Modelling Language*) i CWM *(*engl. *Common Warehouse Metamodel*). Potonji predstavlja temeljni polazišni model metapodataka za stvaranje metamodela za potrebe uslužno orijentiranog sustava stvarnovremenskog skladištenja.

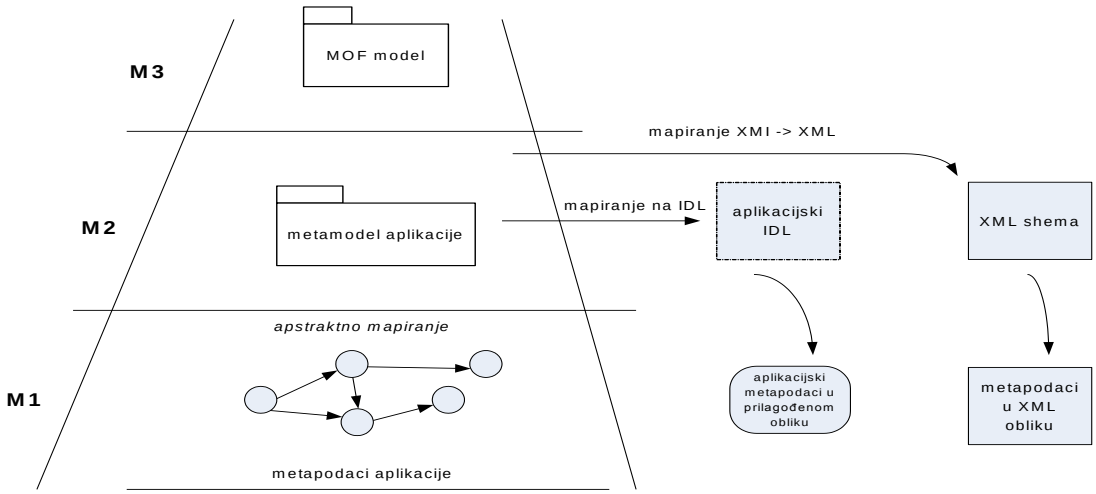
4.4 MOF – Meta Object Facility

MOF je prihvaćena norma grupe OMG koja pruža okvir za upravljanje metapodacima te skup usluga za razvoj i interoperabilnost modela i sustava zasnovanih na metamodelima. Puno tehnologija (UML, CWM, XMI, razni UML profili) koriste MOF i tehnologije zasnovane na MOF-u (kao npr. XMI i JMI) za razmjenu i manipulaciju metapodataka [22].

MOF je najpoznatiji po tome što predstavlja meta-jezik pomoću kojeg je definiran UML, danas najpopularniji jezik za stvaranje modela informacijskih sustava. Kronološki gledano, UML je nastao prije MOF-a; MOF je zapravo definiran kao rezultat potrebe za formalizacijom UML-a. Naime, smatralo se da usprkos sadržajnosti UML-a njegovi konstrukti nisu dostatni za samodefiniciju tj. za definiciju jezika UML *kroz* jezik UML. Zbog toga je i definirana navedena četverorazinska meta-arhitektura (**Tablica 4-1**). UML kao jezik za modeliranje zauzeo je mjesto na razini M2 dok je MOF smješten na vršnu razinu M3. Pošto

se MOF nalazi na višoj razini od jezika za definiciju modela, uobičajeno je MOF referencirati kao "*meta-jezik*", a ako se smatra da se na razini M2 nalaze "metamodeli" (pomoću kojih se definiraju modeli na razini M1), članovi razine M3 se smatraju "meta-metamodelima".

Specifikacija MOF 1.1 prihvaćena je zajedno s revizijom UML-a 1997. godine kada je i specifikacija UML preuzela isti broj verzije – UML 1.1. Paralelni razvoj specifikacija se nastavio i dalje, tako da su specifikacije UML 2.0 i MOF 2.0 razvijane sinkrono s posebnim naglaskom na zajedničkim temeljnim konceptima i što većom razinom ponovne iskoristivosti. MOF i UML danas predstavljaju osnovu principa MDA (engl. *Model Driven Arhitecture*) – tj. arhitekture zasnovane na modelu [23]. Glavna ideja ovog principa je implementacija sustava kroz modele; umjesto stvaranja sustava na određenoj platformi kroz određeni programski jezik, ideja je stvoriti generički model sustava koji bi se onda kroz adekvatne alate i definirana mapiranja transformirao u konkretnu implementaciju, kao što prikazuje **Slika 4-1.**



Slika 4-1 – Koncept mapiranja po principima MDA

MDA principi danas su djelomično podržani u razvojnim alatima ali univerzalnog modela i implementacije još uvijek nema.

4.4.1 Arhitektura jezika MOF

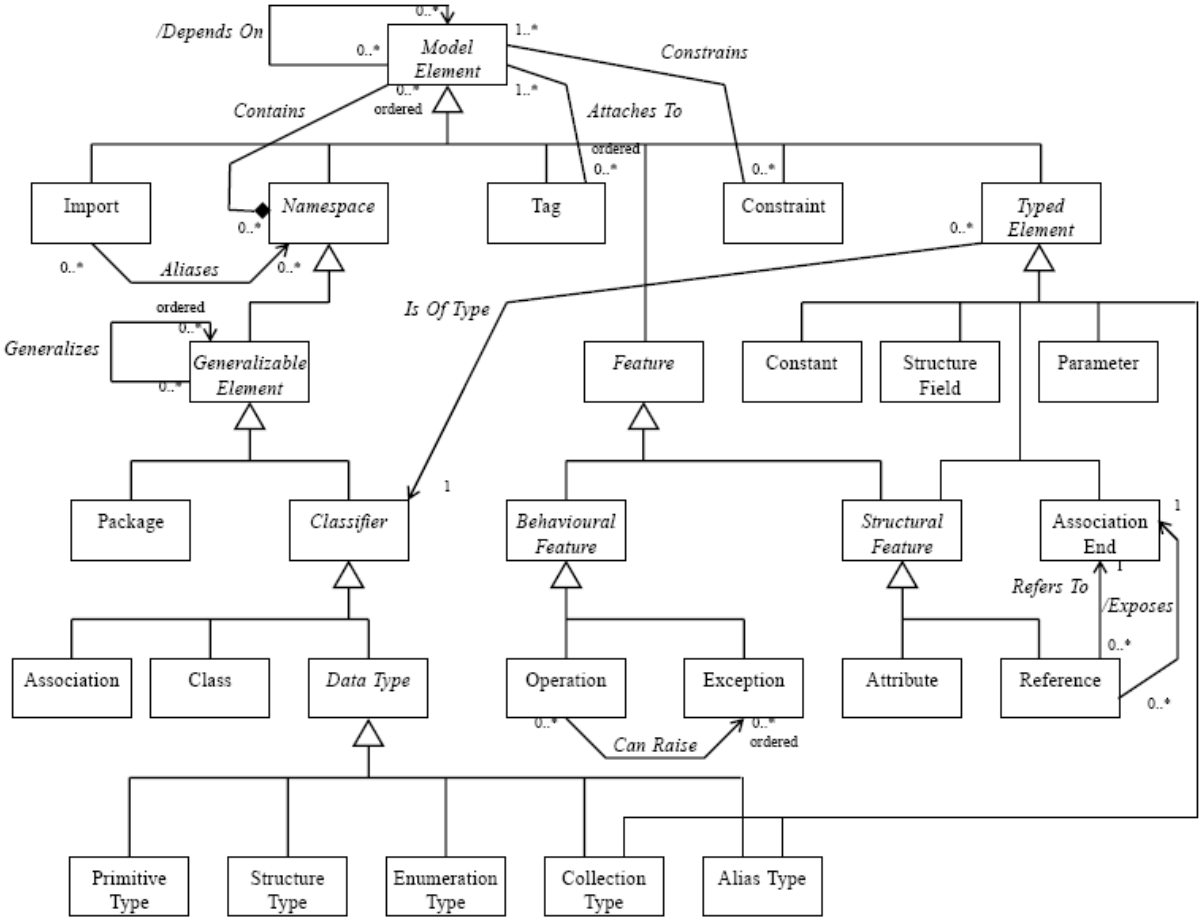
MOF arhitektura ima nekoliko ključnih svojstava po kojima se razlikuje od ostalih arhitektura meta-metamodeliranja:

- MOF model je objektno orijentiran s konstruktima koji su zajednički konstruktima UML modela
- metarazine MOF modela nisu fiksne – broj metarazina je praktično neograničen i ovisi o potrebama korisnika
- MOF model ima svojstvo refleksivnosti, tj. može se opisati pomoću samog sebe

Sposobnost samoopisivanja prikazuje činjenicu da MOF ima dovoljnu snagu ekspresivnosti za praktično metamodeliranje. Isto tako, ono omogućuje definiciju MOF sučelja i ponašanja pomoću mapiranja

sučelja nad MOF modelom. Ovo omogućuje unifikaciju semantike između objekata koji prikazuju modele i metamodele. Naravno, čim se definira mapiranje na novu tehnologiju, sučelja za upravljanje metamodelima unutar tog konteksta su također implicitno definirana.

Osnovni elementi jezika MOF mogu se vidjeti na slici (**Slika 4-2**).



Slika 4-2 - Osnovni elementi jezika MOF

Kao što je već rečeno, MOF 2.0 i UML 2.0 dizajnirani su tako da dijele veliki dio osnovnih koncepata. Zbog toga osnovni elementi jezika MOF predstavljaju podskup jezgrenih UML koncepata koji se nalaze u infrastrukturnoj knjižnici jezika UML. MOF koristi četiri glavna koncepta:

- **Klase** –modeliranje samih metaobjekata
- **Asocijacije** – modeliranje veza između metaobjekata
- **Tipovi podataka** – modeliranje tipa podataka npr. primitivni tipovi, vanjski tipovi i sl.
- **Paketi** – konstrukt za lakšu modularizaciju modela

Klase su MOF metaobjekti koji opisuju tipove "prve instance klasa". Klase definirane na razini M2 imaju instance na razini M1. Instance imaju svoj identitet, stanje i ponašanje. Stanje i ponašanje instanci razine M1 definirano je od strane klase razine M2 unutar odgovarajućeg konteksta. Klase imaju tri tipa svojstava: *atribute*, o*peracije* i *reference*. One također mogu imati *iznimke*, *konstante, tipove podataka, ograničenja* te druge elemente.

Atribut (engl. *Attribute)* definira imenovano mjesto za pohranu određene vrijednosti. Svojstva atributa daje **Tablica 4-2** .

Tablica 4-2 - Svojstva atributa

Svojstvo	**Opis**
name	Jedinstveno ime unutar opsega klase.
type	Klasa ili neki drugi tip podatka.
isChangeable (flag)	Zastavica koja određuje da li je klijent omogućio postavljanje vrijednosti atributa pomoću eksplicitne operacije.
isDerived (flag)	Određuje da li je atribut dio "eksplicitnog stanja" instance ili je deriviran iz nekog drugog stanja.
multiplicity	Specifikacija multipliciranosti atributa.

Operacije (engl. *Operation*) su pristupne točke ponašanja *klasa*. *Operacije* ne specificiraju samo ponašanje niti metode koje implementiraju to ponašanje, već samo imena i signature tipova pomoću kojih se to ponašanje poziva. Svojstva operacije prikazuje **Tablica 4-3**.

Tablica 4-3 - Svojstva operacije

Svojstvo	**Opis**
name	Jedinstveno ime unutar opsega Klase.
tip parametra	*Klasa* ili *Tip Podatka*.
tok parametra (*in, out, in out)*	Određuje tok prosljeđivanja argumenata.
multiplicity	Specifikacija multipliciranosti atributa.
return parameter	Povratna vrijednost operacije (opcionalno)
exception	Pojava iznimke

Atributi i operacije mogu se definirati na "razini instance" ili "razini klasifikatora". Atribut na razini instance ima različito vrijednosno mjesto za svaku instancu klase, dok atribut na razini klasifikatora ima jedinstveno vrijednosno mjesto koje dijele sve klase. Isto tako, operacija na razini instance se može pozvati samo nad instancom i tipično će utjecati samo na stanje instance, dok se operacija na razini klasifikatora može pozvati neovisno o bilo kojoj instanci te se može primijeniti nad bilo kojom ili nad svim instancama unutar ekstenta klase.

MOF dozvoljava **generalizaciju,** tj. mogućnost da klasa naslijedi jednu ili više klasa. Pod-klasa nasljeđuje sve atribute, operacije i reference te ugniježđene tipove podataka, iznimke i konstante). Instanca klase na M1 razini ima tip instance klase na M2 razini. Postoje određena ograničenja i termini kod generalizacije. Klase definirane kao "apstraktne" postoje samo kao generalizacija drugih klasa. Korijenske klase se ne mogu dalje generalizirati. Klase-listovi se ne mogu nasljeđivati.

Asocijacije su primarni konstrukt MOF modela za izražavanje odnosa u metamodelu. MOF asocijacija razine M2 definira odnos (vezu) između parova instanci na razini M1. Konceptualno, ove veze nemaju identitet objekta tako da ne mogu imati atribute i operacije.

Svaka asocijacija ima točno dva *završetka* čija svojstva prikazuje **Tablica 4-4**.

Tablica 4-4 - Svojstva završetaka veze

Svojstvo	**Opis**
name	Jedinstveno ime unutar Asocijacije.
type	Mora biti Klasa.
multiplicity	Specifikacija multipliciranosti završetka.
aggregation	Specifikacija agregacije.
postavka "navigabilnosti"	Kontrolira da li se mogu definirati Reference.
postavka "izmjenjivosti"	Određuje da li se postavljeni Završetak može ažurirati.

U MOF metamodelu klase i tipovi podataka mogu biti u međusobnim odnosima pomoću asocijacija i atributa. U oba slučaja aspekti ponašanja odnosa mogu se opisati pomoću semantike agregacije.

MOF podržava dvije vrste agregacije za odnose između instanci ("*composite"* i *"non-aggregate*"). Neagregatni odnos je konceptualno slaba povezanost između instanci bez posebnih ograničenja na multipliciranost, porijeklo instanci te semantiku životnog ciklusa instanci (konkretno: brisanje instance ne uzrokuje brisanje instanci koje su u odnosu s tom instancom). Kompozitni odnos je konceptualno snažnija veza između instanci sa složenijim svojstvima:

- veza je asimetrična; jedan završetak definira cjelinu (kompozit) dok drugi definira dijelove (komponente)
- instanca ne može biti komponenta više od jedne cjeline, niti može biti komponenta same sebe ili neke od svojih komponenti
- brisanje cjeline uzrokuje brisanje komponenti
- pravilo krajnosti kompozicije: instanca ne može biti komponenta instance iz ekstenta nekog drugog paketa

Semantika agregacije asocijacije se eksplicitno definira pomoću atributa "*aggregation*". Ona ovisi o tipu atributa; atribut čiji je tip primitivni tip podatka ima neagregatnu semantiku, dok atribut čiji je tip klasa ima kompozitnu semantiku.

Tipovi podataka postoje kako bi definicije unutar metamodela mogle koristiti vrijednosti parametara operacija i atributa koje imaju tipove čije vrijednosti nemaju identitet objekta. Postoje dvije vrste tipova podataka:

- primitivni tipovi (unutar MOF modela ih ima 6)
- kompleksni tipovi (enumeracije, strukture, kolekcije, aliasi)

Paketi služe za grupiranje elemenata metamodela. Paketi imaju dvije svrhe: na razini M2 služe za particioniranje i modulariziranje prostora metamodela, dok na razini M1 služe kao vanjski kontejneri za metapodatke.

MOF metamodel pruža četiri glavna mehanizma kompozicije i ponovnog korištenja paketa: generalizacija, gniježđenje, uvođenje i klasteriranje.

Generalizacija je identična ekvivalentnom konceptu kod klasa. Naslijeđeni paket sadržava sve elemente modela super-paketa. Vrijede ista pravila glede imenovanja elemenata. Pod-paket ima isti tip kao i nad-paket, a instanca pod-paketa ne ovisi o instanci nad-paketa.

Gniježđenje znači da paket može sadržavati druge pakete. Postoje neka ograničenja – ugnježđeni paket ne može biti na nijednoj od strana generalizacije, niti može biti uveden ili klasteriran.

Uvođenje omogućuje ponovno iskorištavanje podskupa elemenata nekog paketa.

Klasteriranje je snažnija varijanta uvođenja; uvedeni paketi postaju dio "klastera" paketa, koji se ponašaju kao da su ugnježđeni unutar jednog velikog paketa.

Ograničenja su potrebna kako bi MOF model imao sposobnost očuvanja konzistentnosti. MOF specifikacija ne postavlja ograničenja nad jezikom; no ograničenja unutar MOF Modela se opisuju u jeziku OCL (engl. *Object Constraint Language*) koji je dio UML specifikacije. Politika evaluacije određuje kad se pravilo treba provoditi (odmah ili "kasnije", npr. ako je vrijednost koja se treba provjeravati izraz, onda bi se pravilo trebalo provesti nakon što je dovršeno ažuriranje svih komponenti izraza, ne prije).

Poslužitelj metapodataka nema definirana stroga ograničenja provođenja provjere konzistentnosti; jedino strogo pravilo je provođenje svih strukturalnih provjera koje su označene kao provjere koje se moraju izvesti odmah.

Pored nabrojanih, postoje još i drugi elementi, kao npr. **konstante, iznimke, oznake** (koje služe kao glavni mehanizam proširenja i/ili modifikacije MOF metamodela) itd.

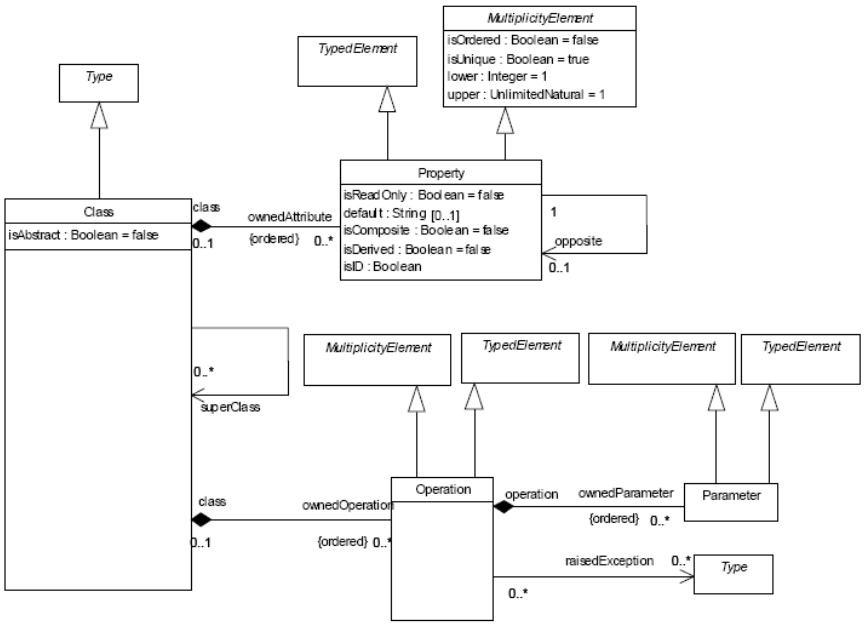
4.4.2 EMOF i CMOF

Postoji nekoliko verzija jezika MOF. OMG je definirao dvije varijante:

- EMOF (engl*. Essential MOF)*
- CMOF (engl*. Complete MOF*)

OMG je razmatrao i treću inačicu zvanu SMOF (*engl. Semantic MOF*) no u vrijeme pisanja ovog rada ona još nije objavljena.

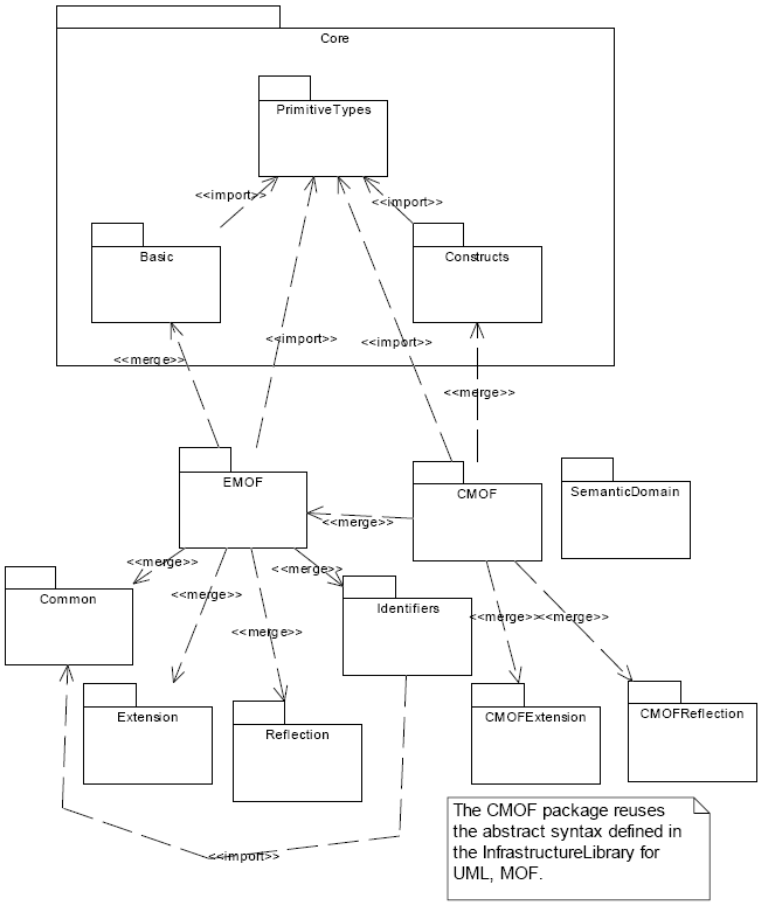
Slika 4-3 prikazuje klasni dijagram varijante EMOF.



Slika 4-3 - Klase varijante EMOF

EMOF (ili esencijalni MOF) je podskup elemenata jezika MOF odabran tako da pokriva sve ključne elemente vezane uz objektno-orijentirane programske jezike te jezik XML. EMOF uglavnom preuzima koncepte osnovnog paketa specifikacije UML, zajedno s paketima koji omogućuju refleksivnost, identifikaciju te proširivanje. Primarni cilj EMOF-a je pružiti jednostavni metamodel za modeliranje osnovnih koncepata uz zadržavanje svih prednosti koje pruža sam MOF, poglavito proširivost i ponovnu iskoristivost.

CMOF (ili kompletni MOF) predstavlja cjelokupni jezik MOF, tj. EMOF zajedno sa svim ostalim modelima koji pružaju mogućnost metamodeliranja i stvaranja jezika na razini M2. U praksi, kada se govori o jeziku MOF misli se na varijantu CMOF. Važno je napomenuti da su modeli izrađeni kroz EMOF potpuno kompatibilni s varijantom CMOF. **Slika 4-4** prikazuje pakete varijante EMOF te proširenje koje donosi CMOF, zajedno s elementima specifikacije UML koje koriste obje varijante.



Slika 4-4 - Odnos između EMOF i CMOF varijanti

4.4.3 Korištenje jezika MOF

Postoje dva različita gledišta na MOF:

- gledište modeliranja – ovo je gledište dizajnera, koji gleda "niz" metarazine. MOF se koristi za definiciju informacijskog modela neke interesne domene. Definicija potom služi za daljnji dizajn softvera i konkretne implementacijske korake razvoja softverskog rješenja
- gledište podataka – gledište programera, koji kreće od trenutne metarazine i gleda prema višim razinama.

Specifikacija MOF pruža mogućnost otvorenog informacijskog modeliranja. Set konstrukata koji definira je ograničen i relativno mal (iako ne i minimalan). Nasljeđivanjem i kompozicijom on se može proširivati do zahtijevane razine kompleksnosti.

Jedna od najvažnijih predviđenih funkcija MOF-a je podrška razvoju distribuiranog objektno-orijentiranog softvera iz visokorazinskih modela. Sustav za takvu podršku bi se sastojao od repozitorija

modela i alata za manipulaciju tim modelima; takvi alati služili bi s jedne strane za stvaranje modela, a s druge za prevođenje tih modela u konkretne softverske implementacije.

MOF nije zamišljen kao ultimativni jezik za modeliranje informacijskih sustava, već prije kao podesan alat za razvoj sofisticiranijih sustava za modeliranje. Scenarij definicije novog jezika – npr. jezika UDL (engl. *Universal Definition Language*) može izgledati ovako: softverski inženjer koristi konstrukte koje pruža model MOF-a za definiciju metamodela nekog jezika UDL. Potom bi konstruktor UDL jezika mogao koristiti jednostavni alat zasnovan na MOF-u za prevođenje UDL metamodela u IDL za UDL repozitorij, te stvoriti softver koji implementira UDL repozitorij i alate za stvaranje UDL modela. Nadalje, alati koji će funkcionirati na nižim metarazinama (tj. koji će prevoditi UDL modele u konkretni programski kod) moći će imati koristi od repozitorija koji će im moći pružiti dodatne informacije vezane uz dizajn, verzioniranje, rokove i sl.

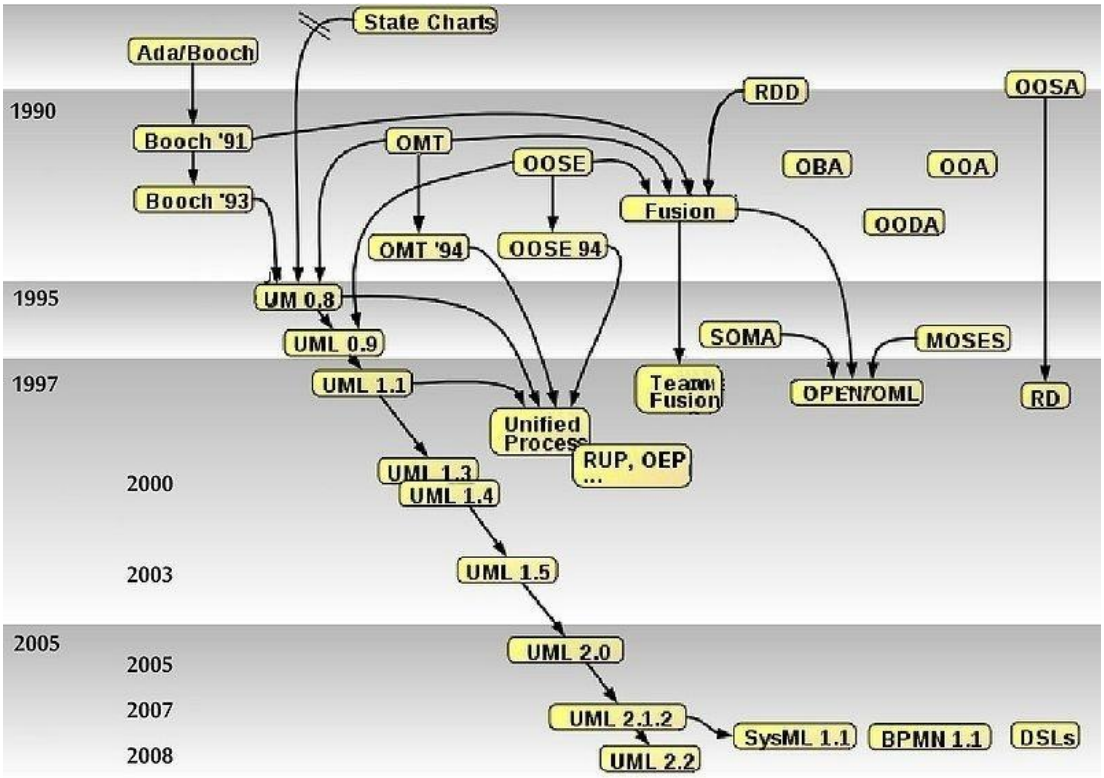
MOF može imati ulogu u sustavima za upravljanje informacijama, npr. kao alternativa semantičkom webu u svrhu anotacije web stranica. MOF također može biti vrlo koristan u skladištenju podataka, no konstrukti MOF-a su previše generički da bi se lako stvorio normizirani, lako razmjenjivi model metapodataka zasnovan direktno na MOF-u. Realniji scenarij je definicija jezika kroz MOF koji bi pružao normizirane izgradbene blokove za opis sustava skladištenja – što je OMG grupa i učinila kroz specifikaciju CWM, detaljnije objašnjenu u poglavlju **4.6**.

Kao što je rečeno, MOF je najpoznatiji kao "formalni jezik kroz kojeg je definiran jezik UML". Upravo jezik UML tema je idućeg poglavlja.

4.5 UML – Unified Modeling Language

UML je jezik za specifikaciju, vizualizaciju, konstrukciju i dokumentiranje elemenata različitih sustava, poglavito informacijskih [24].

Inicijalna verzija UML-a nastala je iz tri vodeće metode za dizajn objektno orijentiranih sustava: Booch, OMT (engl. *Object Modelling Technique)* i OOSE (engl. *Object-Oriented Software Engineering).* Ideja UML-a bila je ujediniti najbolje koncepte i prakse dizajna jezika za modeliranje, objektno-orijentiranih sustava te jezika za opis arhitekture sustava. **Slika 4-5** (autor slike: Axel Scheithauer, Dienstleistungen für Innovative Informatik GmbH, Hamburg) prikazuje evoluciju UML norme kao unifikacije postojećih metoda od 1990.-tih pa sve do danas.



Slika 4-5 - Evolucija UML norme

Trenutna verzija (UML 2.2) pruža sljedeće:

- formalnu definiciju zajedničkog metamodela zasnovanom na jeziku MOF koja definira apstraktnu sintaksu jezika UML. Apstraktna sintaksa definira skup modelirajućih koncepata, njihove atribute, međusobne veze te skup pravila koja omogućuju slaganje ovih koncepata u djelomične ili potpune UML modele.
- detaljno razjašnjenje semantike svakog od UML-ovih koncepata. Semantika definira način realizacije koncepata na tehnološki neovisan način.
- specifikaciju reprezentacije UML-ovih koncepata na način čitljiv ljudskim bićima te pravila slaganja takvih notacija u širok spektar dijagrama različitih tipova od kojih svaki odgovara određenom aspektu modeliranja informacijskih sustava
- detaljne definicije načina na koji se UML alati mogu smatrati kompatibilnim sa specifikacijom UML uključujući i specifikaciju XML-ovskih dokumenata za razmjenu UML-ovskih modela

Kao što mu i samo ime govori, jedan od glavnih ciljeva UML-a je *unifikacija*, pod čim se ne misli samo na unifikaciju najboljih postojećih praksi i metoda nego i unifikaciju načina modeliranja kroz različite tipove sustava (softverske i hardverske), domena (poslovne i informacijske), razvojnih faza (od zahtjeva pa do postavljanja i testiranja) i sl. [25].

UML je definiran kroz niz dokumenata za koje su zadužene organizacije *Rational Software Corporation* i već spomenuti *Object Management Group.* Neki od osnovnih dokumenata su

UML Semantics – definira semantiku UML metamodela.

- **UML Notation Guide** – definira grafičku sintaksu kao reprezentaciju spomenute semantike.
- **Object Constraint Language Specification** – definira sintaksu OCL jezika (jezik za definiciju i izražavanje ograničenja).
- **UML XMI DTD/Schema Specification** – mehanizam razmjene UML-ovskih modela uz pomoć XML tehnologije.
- **UML Standard Profiles** – definira UML profile za procese razvoja softvera i profile za poslovno modeliranje.

UML se najčešće gleda kao specifikacija koja nudi širok spektar izgradbenih blokova i dobro formiranih pravila pomoću kojih se mogu stvarati precizno specificirani modeli. UML se može koristiti za modeliranje različitih aspekata sustava, kao npr.:

- **Strukturalno modeliranje** – naglašava strukturu objekata unutar sustava kao što su klase, odnosi, atributi i operacije. Strukturalno modeliranje uključuje modeliranje statične strukture kroz klasne dijagrame i objektne dijagrame te modeliranje implementacije kroz komponentne dijagrame i dijagrame postavljanja sustava.
- **Modeliranje slučajeva korištenja** – naglašava funkcionalnost sustava prema njegovom međudjelovanju s korisnicima. Particionira funkcionalnost sustava na transakcije (slučajeve korištenja) koje su smislene iz perspektive korisnika (sudionika).
- **Modeliranje ponašanja** – naglašava ponašanje objekata unutar sustava, uključujući njihove interakcije, događaje i kontrolu toka podataka. Koriste se dijagrami sekvenci, dijagrami suradnje, modeliranje događaja korištenjem dijagrama stanja te modeliranje toka podataka korištenjem dijagrama aktivnosti.

Za modeliranje metapodataka koristi se uglavnom modeliranje statičke strukture. Jezgreni elementi statičke strukture vrlo su sličnim onima u MOF-u: to su *klase, objekti, atributi* i *operacije. Klasa* je opis skupa objekata koji dijele iste atribute, operacije i semantiku. Svi *objekti* su instance klasa. Klasa ima *atribute* koji opisuju karakteristike klase. Atributi imaju ime i tip (koji može biti primitivan ili druga klasa). Također mogu imati opseg klase ili opseg instance, te različitu vidljivost (javnu, zaštićenu ili privatnu). *Operacija* ima ime, tip te nula ili više parametara. Također ima svoj opseg i vidljivost.

Poput MOF-a, glavni odnosi unutar statičkih struktura su *asocijacija, generalizacija, ovisnost* i *poboljšanje. Asocijacija* je odnos između dvije ili više klasa koja označava nekakvu povezanost. Može biti jednosmjerna ili dvosmjerna te može sadržavati različite kardinalnosti. *Agregacija* je poseban tip asocijacije koja označava povezanost između cjeline i njezinih komponenata. Agregacija se može dijeliti, što značio da njezina komponenta može biti dio bilo koje cjeline, ili može biti *Kompozicija*, što znači da posjeduje svoje komponente. *Generalizacija* je taksonomijski odnos između općenitijeg i preciznije definiranog elementa. *Ovisnost* je takav odnos među elementima gdje promjena jednog elementa utječe na drugi, njemu ovisan element. *Realizacija* je odnos između specifikacije i implementacije.

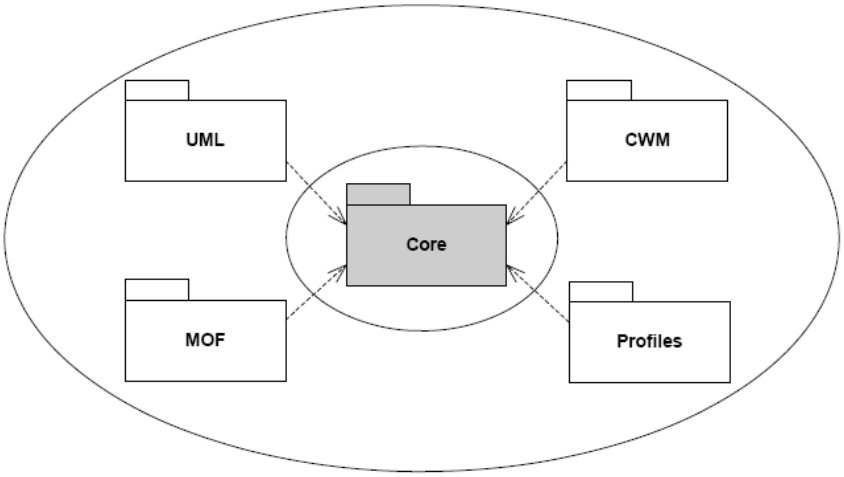
4.5.1 Arhitektura jezika UML

Arhitektura jezika UML slijedi nekoliko važnih principa:

- *modularnost* – specifikacija slijedi princip jake kohezije i slabe povezanosti; konstrukti se grupiraju u precizno određene pakete, tj. module koji su međusobno slabo povezani ali unutar kojih postoji snažna veza između konstrukata.
- *razine* – razine se u arhitekturi UML-a primjenjuju na dva načina. Prvo – jezgreni koncepti su izdvojeni na posebnu razinu od koncepata viših razina koji ih koriste. Drugo – UML koristi već spomenutu četverorazinsku metaarhitekturu kako bi razdvojio razine apstrakcije te jasno razdijelio objekte od njihovih instanci.
- *particioniranje* – unutar iste razine stvorene su particije kako bi se razgraničila područja koja međusobno konceptualno odgovaraju.
- *proširivost* – UML se može proširiti na dva načina: definicijom novog "dijalekta" pomoću normiziranih profila čime se jezik prilagođava pojedinoj platformi te specifikacijom potpuno novog jezika korištenjem dostupnih klasa u glavnom paketu koji se zove *InfrastructureLibrary.*
- *ponovna iskoristivost* – jezgreni koncepti UML-a mogu se koristiti za definiciju drugih metamodela, kao što je i izvedeno s jezicima MOF i CWM.

Kao što je spomenuto, paket pod nazivom *InfrastructureLibrary* je osnovni paket jezika UML. On se sastoji od dva paketa – *Core* (jezgreni paket) i *Profiles* (paket profila).

Paket *Core* je jezgreni paket koji definira potpuni metamodel čija je primarna svrha iznimno visoka razina ponovne iskoristivosti. Kao što prikazuje **Slika 4-6**, paket *Core* predstavlja zajedničku polazišnu točku samog UML-a ali i drugih jezika kao što su MOF i CWM. Glavna ideja jest da drugi metamodeli djelomično ili potpuno preuzmu elemente paketa *Core* što uvelike ubrzava proces razvoja i normizaciju zbog već postojeće apstraktne sintakse i semantike.

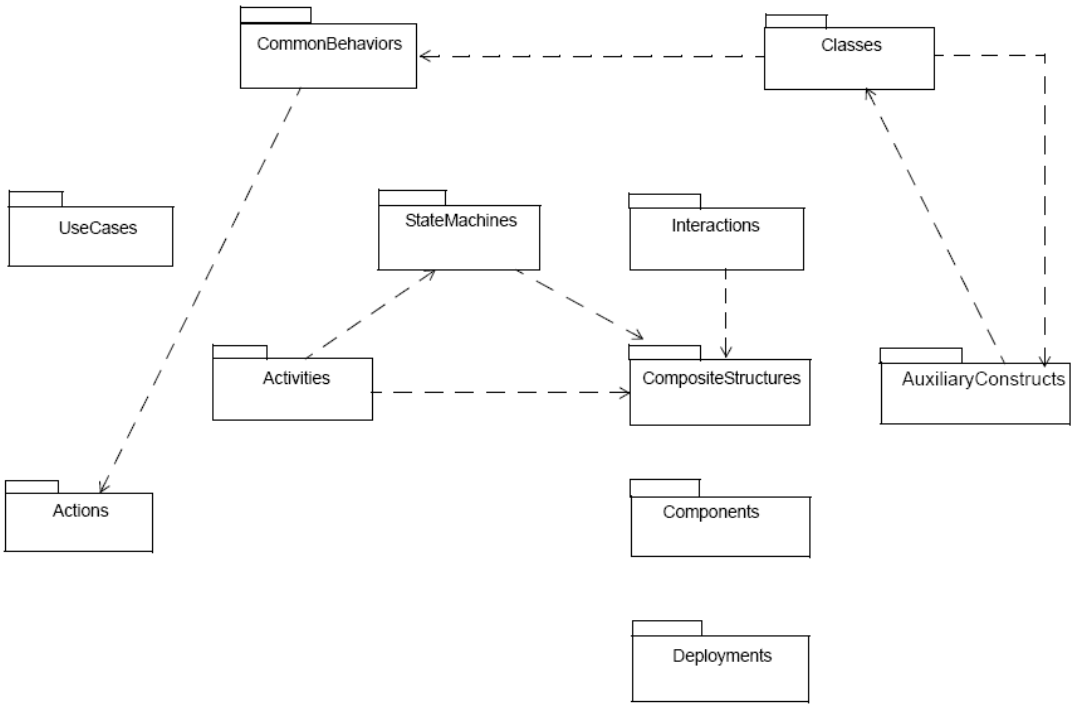


Slika 4-6 - Paket *Core* jezika UML

Paket *Core* se također sastoji od manjih paketa: *PrimitiveTypes, Abstractions, Basic* i *Constructs. PrimitiveTypes* sadrži kolekciju primitivnih tipova koji se najčešće koriste pri metamodeliranju i dizajniran je poglavito za potrebe UML-a i MOF-a. *Abstractions* sadrži niz asptraktnih metaklasa koje su predviđene za daljnju specijalizaciju, a *Constructs* sadrži konkretne metaklase predviđene primarno za objektno-orijentirano modeliranje. *Basic* paket sadrži konstrukte za potrebe XMI-ja (specifikacija za potrebe razmjene UML-ovskih modela kroz XML-ovske dokumente).

Kada se govori o jeziku UML, osim njegove infrastrukture (definirane kroz paket *InfrastructureLibrary*) govori se i o tzv. *superstrukturi*. Superstruktura jezika UML (**Slika 4-7**) ima vlastitu zasebnu specifikaciju a njezina svrha je proširenje UML-a s većim brojem paketa koji omogućuju već spomenuta strukturalna modeliranja, modeliranja ponašanja te modeliranja slučajeva korištenja.

Ono što je bitno napomenuti jest da se razvojni alati za UML modele zasnivaju na grafičkim reprezentacijama podskupova elemenata superstrukture jezika UML. Takvi alati najčešće ne podržavaju cijelu UML specifikaciju, što se najčešće smatra i svojevrsnim nedostatkom – parcijalna implementacija ne garantira normiziranost i prijenos potpune informacije pri razmjeni modela. Usprkos tome, UML-ovski alati danas predstavljaju glavnu podršku pri razvoju raznih modela širokog spektra primjene.



Slika 4-7 - Superstruktura jezika UML

4.5.2 Prednosti i nedostaci UML-a

Prednosti jezika UML već su navedene tako da će ovdje biti samo ukratko ponovljene: UML predstavlja krajnji rezultat unifikacije najboljih metoda, praksi i koncepata koji se tiču objektno-orijentiranih principa te dizajna modela informacijskih sustava. UML je prihvaćen kao standardni jezik za modeliranje opće primjene te što je najbitnije ima snažnu podršku kako velikih proizvođača softvera tako i softverskih inženjera i arhitekata. Širok spektar primjene, veliki broj različitih tipova dijagrama orijentiranih na različite aspekte dizajna sustava te primjenjivost UML-a u svim razvojnim fazama čine ovaj jezik jednim od najpopularnijih jezika za modeliranje današnjice.

Usprkos svojoj popularnosti, UML ima i svojih nedostataka. Jedna od najčešćih jest tvrdnja da je UML – pogotovo nakon verzije 2.0 – postao prevelik, preopširan i prekompleksan [26]. Smatra se da UML sadrži veliki broj elemenata koji su nepotrebni ili redundantni te da je zbog toga implementacija i usklađenost razvojnih alata sa specifikacijom – kao i učenje i usvajanje UML-a od strane informacijskih inženjera - nepotrebno otežana.

Još jedna od čestih primjedbi jest da UML s jedne strane pokušava biti previše "univerzalan" dok se s druge trudi biti kompatibilan s velikim brojem implementacijskih jezika, što su semantički suprotstavljeni ciljevi [27]. Smatra se da je opravdano postaviti pitanje isplativosti vertikalne prilagodbe univerzalnih koncepata ako postoji specijalizirana alternativa prilagođena implementacijskom jeziku.

Na kraju, postoje kritike oko načina razmjene modela definiranih kroz UML. Iako postoji norma za razmjenu pomoću XML-ovskih dokumenata – tzv. XMI (engl. *XML Metadata interchange*) on se pokazao neučinkovitim u praktičnoj primjeni. XMI nije dizajniran primarno za UML, već kao mehanizam šire primjene poglavito zasnovan na MOF-u. Dodatno, XMI ne nudi dovoljno elemenata za sigurnu razmjenu svih UML-ovskih notacija unutar modela, pogotovo ako se uzme u obzir njihova vizualna reprezentacija. Konačno, već spomenut problem parcijalne implementacije UML-a od strane alata (što je i rezultat glomazne specifikacije) inherentno rezultira gubitkom informacije kod razmjene modela. Učinkovita razmjena modela koji odgovaraju specifikaciji UML 2.x je dakle jedan od problema čije rješenje se očekuje u bližoj budućnosti.

4.6 CWM – Common Warehouse Metamodel

Kao što je rečeno u prethodnom poglavlju, jezik UML danas je jedan od najraširenijih i najprihvaćenijih jezika za modeliranje informacijskih sustava. Brojni profili i dijagrami omogućuju veliku fleksibilnost i širok spektar mogućnosti u svim fazama modeliranja sustava.

No usprkos svojoj popularnosti UML još nije univerzalno prepoznat kao norma za modeliranje metapodataka u svrhu razmjene i integracije iz već spomenutih razloga – veličina specifikacije otežava njenu potpunu implementaciju a ne postoji učinkovit mehanizam razmjene UML-ovskih modela koji bi garantirao čuvanje svih nužnih informacija te potpuno razumijevanje podataka s obje komunikacijske strane. Dodatno, UML zbog svoje unifikacijske prirode promovira *generički* način modeliranja koji ne mora nužno odgovarati vertikalno orijentiranim implementacijama. Jedna od češćih vertikalnih implementacija koja je danas imperativ modernog poslovanja jest *skladištenje podataka*.

Zbog ove činjenice grupa OMG je 2000.-te godine izdala specifikaciju nazvanu CWM (engl. *Common Warehouse Metamodel*) [28]. Prema [29]: *CWM je OMG-ova norma za razmjenu metapodataka u okolinama skladištenja podataka i poslovne inteligencije. Radi se o jeziku za opis metapodataka i mehanizmima razmjene metapodataka zasnovanim na normi XML. Gledano s tehničke strane, CWM pruža pristup razmjeni metapodataka zasnovan na modelima gdje se formalni modeli koji reprezentiraju dijeljene metapodatke izgrađuju preko specifikacija CWM metamodela. Ti modeli se pohranjuju i razmjenjuju u obliku XML-ovskih dokumenata.*

Drugim riječima, CWM je jezik precizno definiran za opisivanje informacija vezanih uz skladištenje podataka kako bi omogućio lakšu razmjenu metapodataka te time i integraciju sustava uključenih u skladišni sustav. CWM je zasnovan na MOF-u te usko povezan s UML-om čije osnovne koncepte intenzivno koristi. CWM se može gledati i kao svojevrsna vertikalno orijentirana specijalizacija (tj. proširenje ili skup proširenja) jezika UML te općenito principa metamodeliranja koje pruža grupa OMG. Srodnost CWM-a i UML-a je tolika da je često moguće transparentno koristiti UML-ovske razvojne alate za modeliranje CWM-ovskih elemenata.

U nastavku će biti pobliže opisana arhitektura norme CWM.

4.6.1 Arhitektura

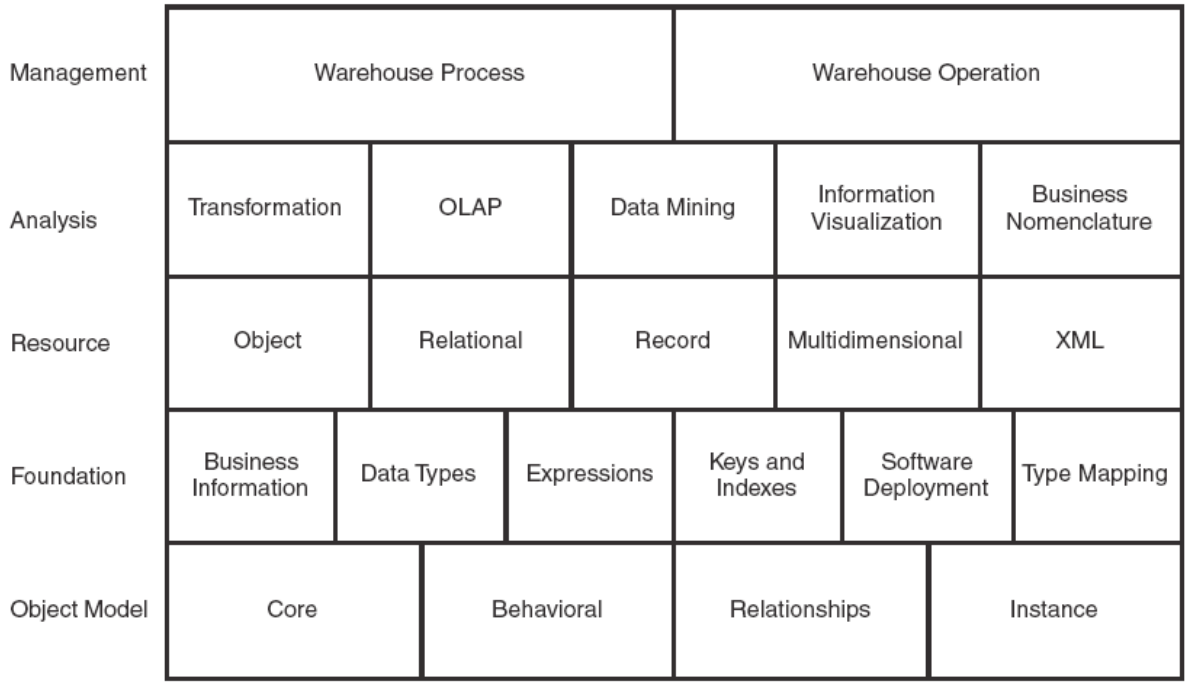
Kao što je rečeno, CWM je skup proširenja nad OMG arhitekturom za metamodeliranje koja su posebno prilagođena potrebama skladištenja podataka i domenama poslovne inteligencije. CWM je modularna arhitektura zasnovana na objektno-orijentiranim temeljima. U četverorazinskoj metaarhitekturi CWM se nalazi na razini M2, zajedno s jezikom UML.

Skladišni sustavi su prema svojim inherentnim karakteristikama iznimno heterogeni. CWM stoga mora biti u mogućnosti opisati metapodatke koji podržavaju široki spektar implementacijskih tehnologija. Nije nužno da CWM opiše sve detalje i kompleksnost heterogenih sustava, no određena razina kompleksnosti mora biti podržana kako bi se omogućila svrhovita razmjena podataka. Zbog svega toga je CWM i sam po sebi iznimno kompleksni sustav.

Skupno gledano, specifikacija CWM definira više od 200 klasa od kojih je svaka orijentirana na određeni aspekt unutar skladišnog sustava. Zbog lakše razumljivosti i brže implementacije klase su raspodijeljene u 21 zasebni paket ili modul. Poput UML-ovskih paketa, svaki paket sadrži klase, asocijacije i ograničenja koja su bitna za određeno interesno područje u sklopu skladištenja podataka i poslovne inteligencije. Paketi nisu samostojeći; čak 20 paketa zahtijeva prisustvo drugih paketa u konkretnoj implementaciji.

Zbog povećanja razumljivosti CWM paketi su organizirani u 5 razina. Istorazinski paketi igraju slične uloge u cjelokupnoj CWM arhitekturi. **Slika 4-8** prikazuje svih 21 paketa i njihove pozicije u 5-razinskom složaju (na slici se zapravo nalazi 22 paketa; to je zato što iako paket *Object* nije zastupljen kao zasebni paket on je eksplicitno naveden iz razloga kompletnosti).

Pošto se aplikacija izrađena tokom ovog rada uvelike temelji na CWM-ovskoj specifikaciji u nastavku će biti dani kratki detalji o svakom od CWM-ovskih paketa, gledano po razinama.



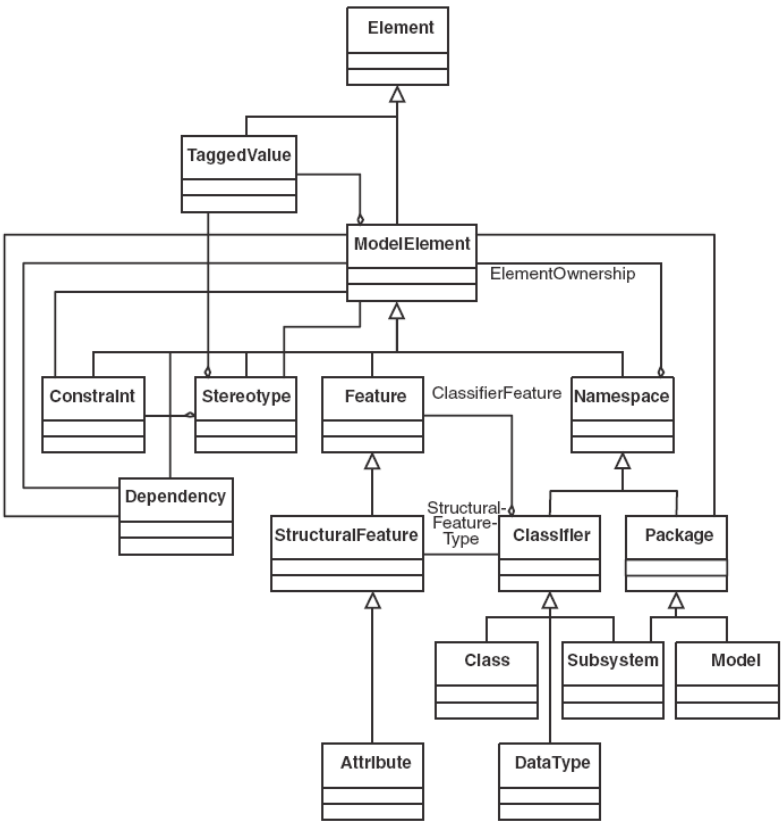
Slika 4-8 - CWM-ovska 5-razinska arhitektura

4.6.2 Razina objektnog modela

Prije nego se krene u detalje o razini objektnog modela potrebno je objasniti način na koji će se grafički prikazivati elementi pojedine razine. CWM koristi UML-ovske koncepte i konvencije za grafičku notaciju modela, primjer kojih se može vidjeti na klasnom dijagramu jezgrenog paketa kojeg prikazuje **Slika 4-9**. CWM-ovske klase se prikazuju korištenjem ikone analogno UML-ovskim klasama (pravokutnik podijeljen na tri dijela, ime u gornjem dijelu, atributi u srednjem i operacije u donjem dijelu[6]). Prazna strelica označava nasljeđivanje. Asocijacije se označavaju linijama. Klasične linije prikazuju klase koje su u jednostavnom odnosu dok romboid označava kompozitnu asocijaciju. Crni romboid označava "snažno" vlasništvo, tj. međuovisnost, dok prazan označava slabo vlasništvo (iako su u praksi gotovo sva CWM vlasništva snažna pa nerijetko prazni romboidi označavaju i takav tip a slabo, ako se posebno navodi kao izuzetak). Asocijacije imaju kardinalnost koja se – ako je navedena - označava klasičnom notacijom.

Razina objektnog modela sadrži pakete koji definiraju temeljne koncepte metamodeliranja, odnose i ograničenja koje koriste ostali CWM paketi. Ovi paketi predstavljaju kompletan skup temeljnih usluga i omogućuju paketima na višim razinama lakše usredotočavanje na konkretno interesno područje. S obzirom na činjenicu da se SOA RT ETL sustav (čiju specifikaciju i implementaciju opisuje poglavlje 6) intenzivno služi elementima razine objektnog modela specifikacije CWM kao temeljnim izgradbenim blokovima ova razina će u nastavku biti detaljno obrađena, dok će o ostalim razinama i njihovim pripadnim paketima biti dan samo kratki pregled.

[6] Na složenijim dijagramima atributi i operacije često se ne prikazuju zbog jasnoće prikaza.



Slika 4-9 - Osnovne klase jezgrenog modula specifikacije CWM

Razina objektnog modela je snažno povezana (i u principu je podskup) UML-ovskog paketa *Core* unutar glavnog UML-ovskog paketa *InfrastructureLibrary*. UML pruža zahvalan temeljni model za izgradnju ovakvih sustava, no cijela UML specifikacija je preglomazna za njezino kompletno ubacivanje u CWM složaj; stoga se koristi samo jedan njezin segment.

Glavni paket razine objektnog modela je jezgreni paket – paket ***Core***. On sadrži klase i asocijacije koje koriste svi ostali CWM-ovski paketi i nije ovisan ni o kojem drugom paketu. Također sadrži i osnovnu UML-ovsku infrastrukturu koja je potrebna za definiranje neobjektno-orijentiranih spremišta podataka (relacijske baze podataka, slijedne datoteke) ali ne uvodi ekskluzivne objektno-orijentirane koncepte. Paket *Core* predstavlja glavnu infrastrukturalnu podršku ostalim paketima.

U CWM-u, svaka klasa u svakom paketu je po definiciji pod-klasa klase *Element*. Ta klasa nema posebne atribute već služi samo kao podesni korijen stablastoj strukturi klasa. Isto tako, gotovo sve klase su pod-klase od klase *ModelElement* koja sadrži neke osnovne atribute (kao npr. atribut *name*) i služi kao uobičajena točka spajanja za sve klase (npr. klasa ograničenja *Constraint* definirana je nad klasom *ModelElement* kako bi bila primjenjiva za širok spektar različitih klasa na nižim razinama specijalizacije). U praksi svaki imenovani element je element modela, tj. instanca klase *ModelElement.*

Još neke osnovne klase su *Dependency* (definicija međuovisnosti dva elementa; jedan element je *dobavljač,* drugi je *klijent)*, *TaggedValue* (pruža jednostavan mehanizam proširenja koji omogućuje

dodatak atributa bilo kojem elementu modela) te *Stereotype* (mehanizam proširenja koji omogućuje dodatno imenovanje klase kako bi se točnije označila njena uloga).

U svojoj srži ovaj paket pruža način opisivanja stvari koje imaju nekakvu prepoznatljivu strukturu. Prema UML-ovskim terminima, stavka u nekoj strukturi se naziva *svojstvom* (*feature*) a reprezentira ju klasa *StructuralFeature* (npr. strukturalna stavka tablice je njezin redak, a zapisa uređena lista polja u zapisu). Klasa *Attribute* prikazuje strukturalna svojstva koja mogu imati inicijalnu vrijednost.

Svojstva (uključujući i strukturalna) su u vlasništvu klasifikatora (*Classifier*) preko asocijacije *ClassifierFeature*. Klasifikator je "nešto što ima strukturu" (vrlo slično ideji "tipa", npr. *integer, character* kao jednostavni klasifikatori ili *adresa* kao složeni). Strukturalno svojstvo ima dva tipa asocijacije s klasifikatorom – jedna označava "vlasnika", a druga "tip" samog svojstva. Primjer: tablica je klasifikator, njena svojstva su stupci; adresa je klasifikator a njegova svojstva su adresa, poštanski broj, grad i država i sl.

Klasa pod nazivom Klasa (*Class*)[7] je zapravo klasifikator koji može imati više instanci (analogno polju u programskom jeziku). Tako je tablica zapravo Klasa jer može imati više redova s podacima. Klasa *DataType* predstavlja klasifikatore koji mogu imati samo jednu instancu, npr. *integer* ili *character*, koji su instance te Klase.

Imenski prostori (*Namespace)* su ključni elementi paketa i specifikacije CWM uopće jer reprezentiraju jedinstveni način identifikacije objekata. Posljedica toga je da je gotovo svaki element CWM modela u vlasništvu imenskog područja. Kompozitna asocijacija između elementa *ModelElement* i elementa *Namespace* pod nazivom *ElementOwnership* omogućuje imenskom području da bude u vlasništvu drugog imenskog područja, a direktna i ključna posljedica toga jest mogućnost organiziranja elemenata CWM-a u stablaste hijerarhijske imenske strukture. Važno je primijetiti da se ova asocijacija ***nasljeđuje*** tako da se može koristiti između pod-klasa na nižim razinama, npr. između klasifikatora i sučelja.

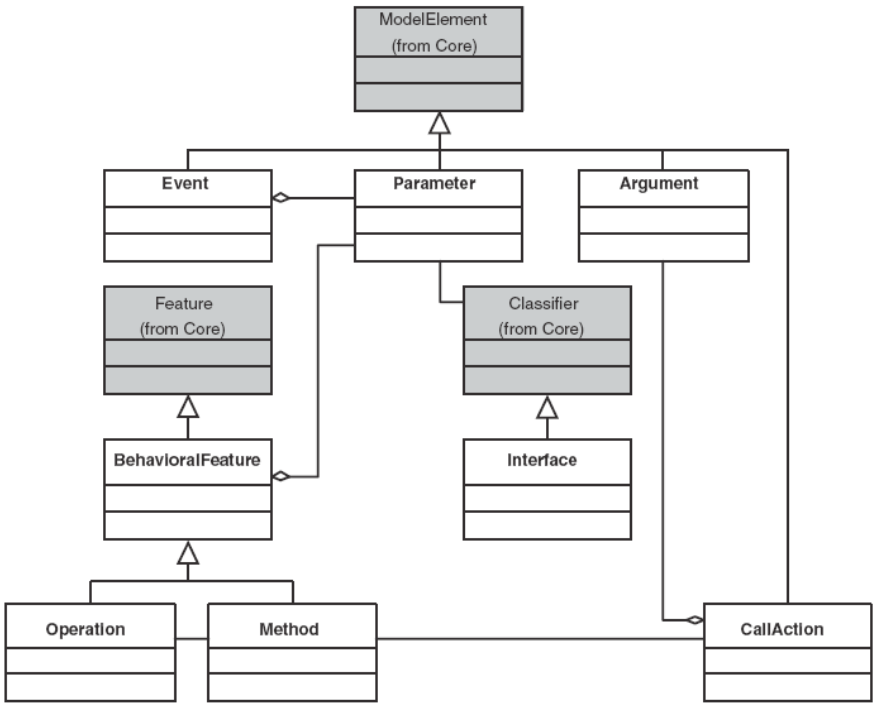
Klasa *Package* nasljeđuje klasu *Namespace* i omogućuje referenciranje objekata u drugim paketima. Pošto element modela može pripadati samo jednom imenskom području, pomoću paketa mogu se *uvesti* elementi iz drugih paketa.

Klase *ModelElement, Class, Attribute, DataType* i *Package* su glavna pogonska snaga specifikacije CWM i temelj na kojem su izgrađene glavne strukturalne sposobnosti CWM-a.

Paket ***Behavioral*** (**Slika 4-10**) omogućuje CWM-ovskim klasama karakteristike "ponašanja" koje se često povezuju s objektno-orijentiranim jezicima; objekti u takvim jezicima najčešće imaju *operacije* (ili *metode*) kojima se pristupa preko *sučelja* i koje mogu biti pokrenute ili mogu uzrokovati određene *događaje*.

[7] Ovdje se može uočiti problem manjkavosti jezične ekspresivnosti jer se pojam "klasa" pojavljuje u dva svojstva, ovisno o metarazini na koju se pojam odnosi. Kako bi se izbjegla konfuzija, Klasa koja je element paketa *Core* će se označavati velikim slovom.

Objektno-orijentirana svojstva su organizirana oko klase *BehavioralFeature* koja se može intuitivno predstaviti izvedivom jedinicom programske logike koju neki objekt može izvoditi. Ona ima ime, skup definicija parametara (sa ulaznim i/ili izlaznim vrijednostima) te opcionalnu povratnu vrijednost. Ova klasa može biti "operacija" ili "metoda". Razlika između "operacije" i "metode" je sljedeća: "operacija" je specifikacija izvedive jedinice programske logike koju dijele sve "metode" koje implementiraju tu operaciju u određenom programskom jeziku. *CallAction* klasa zapisuje konkretne pozive metoda, dok klasa *Argument* zapisuje konkretne vrijednosti parametara. Sučelje (*Interface*) je skup operacija koja definira uslugu koju klasifikator pruža svojim klijentima. Klasa "*Event*" predstavlja događaj kojeg je moguće na neki način primijetiti i koji može imati parametre.



Slika 4-10 - Struktura paketa *Behavioral*

Klase u paketu ***Relationships*** opisuju odnose između CWM-ovskih objekata. Postoje dva tipa odnosa u CWM-u: **generalizacija** i **asocijacija.** Pošto su CWM klase definirane na metarazini M2, ove klase omogućuju instancama drugih CMW klasa odnose i hijerarhijski složaj na razini M1 (analogno tome, MOF klase na razini M3 pružaju istu uslugu CWM klasama na razini M2).

Generalizacija omogućuje organizaciju objekata u *hijerarhiju*, s tim da se treba uzeti u obzir da takva hijerarhijska struktura semantički postavlja objekte u nasljedne odnose (gdje je objekt na višim slojevima "tip" objekta na nižim) za razliku od klasične organizacijske strukture. U CWM-u djeca mogu imati više od jednog roditelja-klasifikatora, iako to nije česti slučaj.

Asocijacije definiraju odnos između dva ili više klasifikatora (binarne asocijacije su daleko najčešće). Svaka asocijacija ima pripadni završetak, kojeg asocijacije posjeduju u svom vlasništvu preko asocijacije *ClassifierFeature*. Završeci identificiraju klasifikatore na koje su "naslonjeni" preko asocijacije tipa između svojstva *StructuralFeature* i klase *Classifier.*

Ponekad je uz metapodatke podesno slati i konkretne podatke, kao npr. skup dozvoljenih vrijednosti za neki stupac, ili razmjenu instanci klase tipa podatka. Podršku za to pruža paket ***Instance*.**

4.6.3 Razina *Foundation*

Najdonja razina CWM-vskog složaja je zapravo podskup specifikacije UML, koja pruža općenite, generičke usluge gornjim razinama. Temeljna razina (*foundation*) se više orijentira konkretnim uslugama vezanih uz arhitekturu CWM-a.

Paket ***Business Information*** pruža generičke usluge vezane uz vođenje evidencije o najnužnijim detaljima poslovne okoline unutar koje se provodi skladištenje podataka. Tri najvažnije klase ovog paketa su opis (*Description*), dokument (*Document*) i odgovorna stranka (*ResponsibleParty).* Opis sadrži kratki opisni tekst nekog elementa modela. Dokument označava dokumentaciju koja se ne nalazi unutar sustava CWM, te se najčešće opisuje njezina lokacija u virtualnom ili stvarnom svijetu. Odgovorna stranka označava organizacijsku jedinicu koja je odgovorna (ili se interesira o stanju) za neki element modela; stranka se može identificirati preko URI-ja, ali i lokacije, telefona ili adrese elektroničke pošte. Pomoću asocijacije *ElementOwnership* moguće je uspostaviti adekvatnu hijerarhijsku strukturu organizacijskih jedinica koje su uključene u CWM sustav.

Tip objekta je koncept koji je u samoj srži principa programiranja; taj koncept u biti opisuje što neka druga stvar jest. Koncept klase nije ništa drugo nego korisnički definirani tip podatka. *Sustavom tipova* zovemo princip pridjeljivanja primitivnih ili strukturiranih tipova elementima nekog sustava. Nesklad sustava tipova je glavni uzrok problema kod uspostavljanja komunikacije između dva heterogena informacijska sustava. Paket ***DataTypes*** zato pruža mehanizme za opis primitivnih i strukturiranih tipova podataka unutar informacijskih sustava CWM arhitekture. Definicijom sustava tipova i mehanizmima razmjene bavi se paket *TypeMapping.*

Izrazi su određeni skupovi vrijednosti i operacija koji su izračunljivi i kao rezultat daju određenu vrijednost. Izrazi su najčešće izrazito semantički ovisni o implementacijskoj tehnologiji i kao takvi predstavljaju crnu kutiju, tj. mehanizam koji funkcionira unutar odabrane tehnologije ali čije se unutarnje karakteristike izvana ne mogu razumjeti niti se izraz kao takav može razmjenjivati. Kroz paket ***Expressions*** CWM dozvoljava opisivanje takvih izraza (tzv. *black box* izrazi) tako što se zapisuje identifikator jezika te izraz izveden u semantici tog odabranog jezika. Drugi tip izraza, tzv. *white box* izrazi su univerzalni i opisuju se u takvom obliku koji se može razumjeti i razmjenjivati. To se radi jednostavnim prevođenjem izraza u niz funkcijskih poziva. Prvo se definiraju funkcije (kao operacije nad nekim klasifikatorima), a potom instance atributa koje opisuju elemente izraza.

Indeks je najčešće popis elemenata izveden na logički izabran način koji se najčešće razlikuje od fizičke poredanosti elemenata koje opisuje. *Ključ* je skup jedne ili više vrijednosti koji jedinstveno identificira neki entitet (npr. zapis u bazi podataka). Ove koncepte definira paket ***Keys and Indexes***. Koncept

ključeva i indekasa je često korišten od strane paketa unutar CWM specifikacije tako da je paket koji ih podržava definiran na temeljnoj razini.

Konačno, paket ***Software Deployment*** pruža usluge za vođenje dokumentacije o softverskoj i hardverskoj okolini sustava za skladištenje podataka. Opseg usluga ograničen je na samo one koje su nužne drugim paketima CWM specifikacije za identifikaciju i lokaciju potrebnih softverskih i hardverskih resursa vezanih uz elemente modela.

4.6.4 Razina *Resursa*

Paketi na ovoj razini opisuju strukture podatkovnih resursa koji djeluju kao izvori ili kao ciljevi razmjene podataka unutar procesa skladištenja.

Ova razina ne sadrži paket za modeliranje objektno-orijentiranih resursa. Pošto paketi na temeljnoj razini CWM specifikacije pružaju više nego dovoljnu podršku za objektno-orijentirano modeliranje, definiranje dodatnog paketa za istu svrhu je redundantno. Ako sustav skladištenja sadrži objektno-orijentirani izvor podataka za čiji opis spomenuti temeljni paketi nisu dovoljni, moguće je definirati proširenje koje će pružiti dodatna svojstva.

Sheme relacijskih baza podataka mogu se vrlo detaljno opisati pomoću komponenata paketa ***Relational***. Metamodel paketa izrađen je prema okvirima modela SQL99 relacijskih baza podataka, uključujući i objektno-orijentirana proširenja. Ovaj model podesan je za podršku razmjene podataka između većine prihvaćenijih sustava za upravljanje relacijskim bazama podataka. Važno je napomenuti da je ovaj paket predviđen za opisivanje logičke strukture relacijske baze podataka; fizička struktura je vrlo usko povezana s konkretnom implementacijskom tehnologijom. Ako je nužno pomoću specifikacije CWM opisati i fizičku strukturu potrebno je definirati adekvatni paket proširenja.

Ovaj paket je najveći unutar CWM specifikacije te ima najveći broj paketa o kojima ovisi. Konkretno, paket sadrži ukupno 24 vlastite i 44 međuovisne klase, što čini trećinu cjelokupne specifikacije CWM.

Prije relacijskih baza podataka podaci su se najčešće pohranjivali u obliku zapisa – uređenih skupova podataka predefinirane strukture i tipa. Pohrana podataka unutar zapisa se i danas široko koristi, posebno zbog toga što je vrlo bliska principima fizičke pohrane podataka na medije. Zbog toga se takvim načinom zapisa bavi paket ***Record.***

Multidimenzionalne baze podataka su fizičke strukture izvedene kao podrška OLAP sustavima. Ovakve baze najčešće imaju vrlo specifične strukture ovisno o implementacijskoj tehnologiji; zbog toga je definiran paket ***Multidimensional*** koji pruža podršku za strogo generičku evidenciju opisa multidimenzionalne baze podataka. Ovo je tzv. *stub*-paket jer je predviđen upravo za implementaciju uz pomoć adekvatnog modula proširenja.

Konačno, paket ***XML*** pruža podršku za opis resursa pohranjenih u obliku XML-ovskih dokumenata.

4.6.5 Razina Analize (*Analysis)*

Paketi na ovoj razini nisu direktno povezani uz izvore podataka, koliko uslugama *nad* izvorima i mjestima pohrane podataka koji su definirani na resursnoj razini.

Česta realnost skladišnih sustava je disjunktnost operativnih poslovnih sustava i informacijskih sustava predviđenih za pohranu podataka i pripremu za analitičke procese. Posljedica toga je potreba za premještanjem i prilagodbom podataka prije nego se oni mogu pohraniti na adekvatan način unutar tablica skladišta podataka. Paket ***Transformation*** unutar analitičke razine upravo pruža usluge vezane uz dokumentiranje ovog transformacijskog procesa.

OLAP (*Online Analytical Processing*) je analitička tehnika konsolidacije poslovnih podataka iz različitih izvora i njihovog izlaganja u multidimenzionalnom obliku koji odgovara potrebama poslovnih analitičarima za provođenje raznih analiza čija učinkovitost izvedbe ovisi o dostupnosti potrebnih podataka. Paket ***OLAP*** služi kao podrška vođenju evidencije o OLAP sustavima.

Paket ***Data Mining*** opisuje rezultate aktivnosti dubinskih pretraga podataka. Metamodel ovog paketa podijeljen je u tri glavne skupine: jezgreni model, postavke i atributi. Jezgreni model predstavlja rezultate. Model postavki opisuje ulazne parametre i vrijednosti atributa koje će se koristiti (ili su korištene) za konstruiranje modela dubinske pretrage. Model atributa produbljuje klasu *MiningAttribute* i dozvoljava podrobniji opis i kategorizaciju atributa.

Prezentacija informacija na razumljiv način ključan je segment procesa skladištenja podataka. Podaci se mogu prezentirati na različite načine koristeći različite tehnologije. Stoga paket ***Information Visualization*** pruža okvir za definiciju i razmjenu složenih modela M1 razine za okoline prikazivanja informacija i skupove alata.

Konačno, paket ***Business Nomenclature*** pruža način za stvaranje strukturiranog poslovnog vokabulara – generičke organizirane strukture neovisne o konkretnoj tehnološkoj implementaciji. Termini se slažu u rječnike ili taksonomije, te mogu biti međusobno povezani (npr. sinonimskom vezom).

4.6.6 Razina Upravljanja (*Management)*

Paketi na ovoj razini daju podršku dnevnoj operaciji skladišta podataka. Isto tako, zamišljeni su kao temeljne polazišne točke za izgradnju proširenja za složenije aktivnosti vezanih uz skladištenje.

Paket ***Warehouse Process*** opisuje tok informacija unutar skladišta podataka. Tokovi informacija prikazani su preko transformacija (iz paketa *Transformation*). Transformacije se dokumentiraju na razini aktivnosti ili koraka. Događaji unutar skladišta su okidači koji započinju tok informacija i mogu biti ili raspoređeni ili rezultat nekog internog ili vanjskog događaja.

Skladišni proces je ili aktivnost (*WarehouseActivity*) ili korak (*WarehouseStep*) ovisno o tome koji segment iz transformacijskog paketa reprezentira. Procesi se mogu grupirati u pakete.

Konačno, paket ***Warehouse Operation*** zapisuje događaje unutar skladišta koji su na neki način interesantni iz poslovne ili druge perspektive. Zapisuju se tri glavna tipa događaja – izvedbe transformacija, mjere i zahtjevi za izmjenama.

4.6.7 Uzorci razmjene i semantički kontekst CWM-ovskih metapodataka

Sintaksa jezika CWM sama po sebi nije dovoljna za uspostavu učinkovite razmjene metapodataka između različitih sudionika informacijskog sustava. Uz sintaksu, veliku ulogu igra semantički kontekst interpretacije modela a također i veličina same CWM instance. Specifikacija CWM ne pruža mehanizme definicije semantičkog konteksta, a jedna od posljedica njezine otvorenosti i fleksibilnosti je izgradnja neograničeno velikih modela koji, iako sintaktički ispravni, ne mogu biti prihvatljivi kao realno implementabilni modeli unutar poslovnog sustava.

CWM metamodel pruža izgradbene blokove za izgradnju modela podataka koji su zasnovani na nasljeđivanju i ponovnoj iskoristivosti pojedinih zajedničkih koncepata. Tako je većina elemenata modela zapravo podklasa elemenata *Classifier, Namespace* i *Feature* a povezani su preko veza na najvišoj razini (veza koja opisuje svojstvo ili veza za definiciju vlasničkog odnosa). Posljedica toga je postojanje impliciranih svojstvenih i vlasničkih veza između različitih elemenata modela, koje mogu ali i ne moraju biti implementirane. Drugim riječima, ako neki element ima definiranog vlasnika, to ne znači nužno da dotični vlasnik mora postojati u konkretnom modelu niti da uopće mora biti definiran.

Npr. sustav koji koristi relacijski model može baratati pojmovima "tablice" i "stupca". Drugi sustav može zahtjevati da "tablice" budu organizirane u "sheme" koje se dalje pohranjuju u "kataloge"; tj. on koristi drugačiji **kontekst** metapodataka. Iako metapodatci i jednog i drugog sustava sintaktički odgovaraju specifikaciji CWM pojavljuju se problemi u razmjeni podataka - podaci drugog sustava očekivat će definiciju kataloga i sheme koje prvi sustav neće pružati i obrnuto; podaci koje drugi sustav šalje prvom sadržavat će elemente koje prvi sustav neće znati interpretirati.

Jedno od rješenja ovoga problema je definicija *uzoraka razmjene metapodataka* u obliku strukturalnih uzoraka koji se postavljaju nad sadržaj metapodataka oko kojega se moraju dogovoriti sudionici poslovnog sustava, bilo da se radi o stvarateljima ili potrošačima metapodataka koji se koriste za razmjenu. Konkretno, sudionici sustava moraju definirati zajednički kontekst unutar kojega će se ti metapodaci razmjenjivati. Uzorak također može uvesti ograničenja na kardinalnost pojedinih elemenata modela kako bi se riješio problem sintaktički valjanih modela koje je nemoguće procesirati.

U [30] definirano je devet ključnih stavki koje se tiču uzoraka razmjene metapodataka.

1. Instanca metamodela naziva se **rješenjem** problema modeliranja metapodataka unutar domene metamodela. Rješenje se **generira** iz metamodela.
2. **Prostor rješenja metamodela** je skup svih mogućih rješenja generiranih iz tog metamodela.
3. **Podskup metamodela** je dio metamodela koje particioniran ili izdvojen na način koji je dozvoljen od strane samog metamodela (particioniranje je izvedivo gdje god završeci asocijacija definiraju donju granicu kao nulu).
4. **Prostor podskupova metamodela** je kolekcija svih podskupova metamodela.
5. **Projekcija metamodela** je specificirana kolekcija poskupova metamodela. Podskupovi u projekciji ne moraju biti povezani. Projekcija je zapravo specifikacija navedenog semantičkog konteksta za razmjenu. Particioniranje se izvodi na razini M2.

6. **Restrikcija projekcie** je specificirani podskup rješenja koja se mogu generirati iz projekcije. Restrikcija je ograničenje na razini M2 koje djeluje na instance razine M1.
7. **Parametar** je specifikacija neke vrijednosti koja se mora koristiti za formiranje restrikcije nad skupom rješenja projekcije.
8. **Uzorak razmjene metapodataka** je specificirana projekcija metamodela s opcionalnim restrikcijama i skupom parametara.
9. **Realizacija** uzorka je instanca metapodataka koji odgovaraju, ili su generirani iz uzorka. Uzorak je *povezan* s realizacijom, a realizacija *realizira* uzorak.

Dakle – usprkos činjenici da CWM pruža normizirani model za stvaranje metapodataka, analogno specifikaciji UML često se specifikacija ne koristi u cijelosti već se definira samo njen podskup uz dodatna ograničenja. Alati koji koriste ove metapodatke ne moraju samo biti kompatibilni sa specifikacijom CWM već i s konkretnim kontekstom unutar kojeg se metapodaci koriste te ograničenjima koja su nametnuta nad istima.

4.6.8 Nedostaci CWM-a

Najveći nedostatak CWM-a kao specifikacije jest činjenica da se ona prestala razvijati nakon verzije 1.0 tako da je veliki broj koncepata koja ona opisuje u izvjesnom smislu zastario. Npr. modul XML naglasak stavlja na opis DTD-a kao "metapodataka" XML-ovskog dokumenta iako se od izlaska specifikacije „XML Schema“ DTD smatra zastarjelom tehnologijom koju više ne bi trebalo koristiti. Isto tako, CWM specifikacija ne prepoznaje arhitekturu SOA niti Usluge weba. Iako se mogu naći dokumenti o različitim inicijativama za proširenjem i modernizacijom specifikacije CWM, do sada to nije učinjeno niti postoje javne obavijesti da će se to ostvariti u skoroj budućnosti.

Usprkos tome, veliki proizvođači softvera kao što su *Oracle* i *Microsoft* te zajednice vezane uz otvorene tehnologije (kao npr. *Java* zajednica) i dalje u svoje proizvode integriraju podršku za CWM tj. omogućuju stvaranje metapodataka koji prate normu CWM.

Uzevši sve to u obzir opravdano je postaviti pitanje iskoristivosti CWM-a u modernom informatičkom svijetu, npr. pogodnosti koje ta specifikacija može donijeti i doprinijeti kod novih softverskih rješenja zasnovanih na otvorenim tehnologijama i principima SOA.

Jedna od zadaća sljedećih poglavlja je upravo demonstrirati iskoristivost CWM specifikacije ali i naglasiti njezinu ograničenost, jer će primjena CWM-a biti podobna samo uz znatna proširenja i preinake; CWM pruža dobro definirane i kvalitetne *temelje* za stvaranje metapodataka, ali zbog svoje zastarjelosti u originalnom obliku je u izvjesnoj mjeri neprimjenjiv za opisivanje modernijih koncepata i korištenje tih opisa za izvedbu funkcionalnih rješenja.

4.7 Norma XMI

XMI (*XML Metadata Interchange*) je norma grupe OMG koji omogućuje razmjenu metapodataka definiranim kroz MOF modele pomoću XML-ovskih dokumenata.

XMI definira pravila za stvaranje DTD-ovskih dokumenata (verzije XMI 1.x) ili XML-ovska shema (verzije XMI 2.x) koje opisuju kako XML-ovski dokumenti koji sadrže metapodatke za razmjenu moraju izgledati. Također, XMI specifikacija definira pravila stvaranja samih dokumenata. Ova pravila su reverzibilna tj. mogu se koristiti za dekodiranje XMI dokumenata i rekonstrukciju metapodataka [31].

Osnovni principi specifikacije XMI su sljedeći:

1) Sve deklaracije XML-ovskih elemenata, atributa, tipova te grupa koji se definiraju u XML-ovskim shemama za razmjenu metapodataka moraju odgovarati specifikaciji XMI. Imensko područje XML-ovskih elemenata specifikacije XMI je uvijek "http://schema.omg.org/spec/XMI/*verzija*" .

2) Svaka klasa unutar modela reprezentirana je XML-ovskim elementom čije ime odgovara imenu klase te kompleksnim tipom (*eng. complexType*) čije ime također odgovara imenu klase. Deklaracija tipa navodi svojstva klase koja ne moraju biti navedena određenim redoslijedom. Svojstva klase mogu biti elementi ili atributi, osim kompozitnih svojstava ili svojstava s više mogućih vrijednosti koje se moraju definirati kao elementi,

3) Svaki element može biti proširen. Sloboda proširenja je neograničena što se omogućuje dodavanjem elementa ANY u deklaraciji tipa koji odgovara modelu klase.

Ono što je bitno napomenuti jest činjenica da XMI ne nudi konkretne XSD-ovske sheme niti bilo koji drugi način provjere valjanosti XML-ovskih reprezentacija modela. XMI se više svodi na naputke i principe koje XML-ovski dokumenti moraju slijediti kako bi se omogućila što točnija reprezentacija metapodataka koji se opisuju.

Slika 4-11 (preuzeta iz [32]) prikazuje primjer korištenja specifikacije XMI. U gornjem lijevom kutu vidi se UML-ovski klasni dijagram koji opisuje jednostavni automat s dva stanja (čija se slobodna vizualna reprezentacija vidi u gornjem desnom kutu). S donje strane nalazi se isti model ali u XML-ovskoj reprezentaciji koja koristi XMI principe zapisa modela.

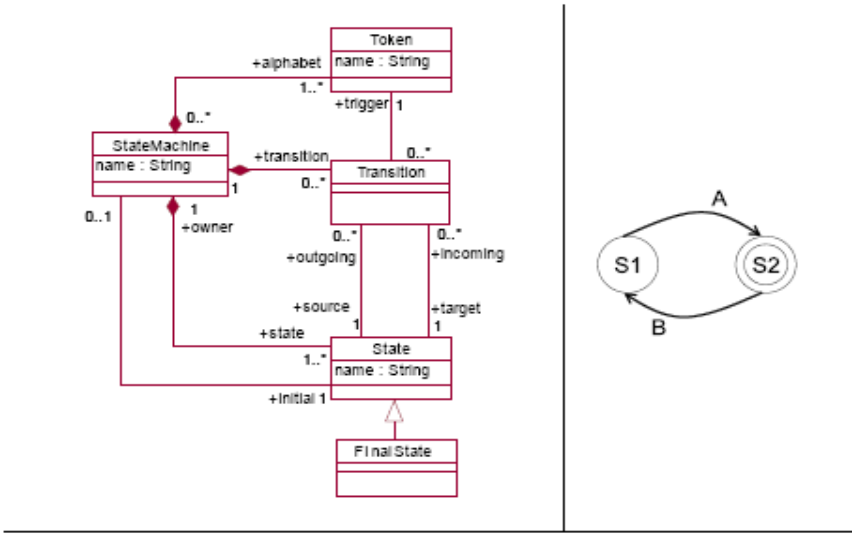
Na slici se također jasno vidi jedan od ključnih koncepata norme XMI - i*dentifikacija objekata.* Identifikacija je glavni mehanizam pronalaska određenog elementa te određivanja ekvivalencije između pojedinih objekata. XMI definira tri načina identifikacije:

1. Globalna identifikacija
2. Identifikacija unutar dokumenta
3. Deskriptivna identifikacija

Globalna identifikacija se radi pomoću tzv. "univerzalnog identifikatora" – UUID (engl. *Universal Unique Identifier*). UUID je vrijednost za koju postoji gotovo apsolutna garancija da je jedinstvena u globalnom smislu. U praksi UUID se stvara pomoću posebno definiranog algoritma implementiranog pomoću

odabranog alata. Svakom elementu može se pridružiti njegov UUID a oznaka unutar dokumenta je "**XMI:uuid**".

Identifikacija unutar dokumenta koristi oznaku "**XMI:id**" i služi za jednoznačnu identifikaciju elementa unutar opsega jednog XML-ovskog dokumenta unutar kojeg se taj element nalazi. Nedostatak ove identifikacije je nemogućnost garancije jedinstvenosti ako dođe do spajanja više dokumenata gdje može doći do neželjenog ponavljanja identifikatora.




```
<?xml version='1.0' encoding='UTF-8'?>
<xmi:XMI xmlns:xmi="http://schema.omg.org/spec/XMI/2.0"
         xmlns:xlink="http://www.w3.org/1999/Xlink"
         version='2.0' timestamp='Sun, 06 Mar 2005 18:21:02 +0200'>
    <documentation>. . .</documentation>
    <FSM:StateMachine xmlns:FSM="http://www.abo.fi/FSM/1.0"
                      xmi:id="e1" name="Example">
        <alphabet xmi:id="e2" name="A">
            <transition xmi:idref="e3" />
        </alphabet>
        <alphabet xmi:id="e4" name="B">
            <transition xmi:idref="e5" />
        </alphabet>
        <state xmi:id="e6" name="S1">
            <incoming xmi:idref="e5" />
            <outgoing xmi:idref="e3" />
        </state>
        <state xmi:type="AcceptingState" xmi:id="e7" name="S2">
            <incoming xmi:idref="e3" />
            <outgoing xmi:idref="e5" />
        </state>
        <transition xmi:id="e3" source="e6" target="e7" trigger="e2" />
        <transition xmi:id="e5" source="e7" target="e6" trigger="e4" />
    </FSM:StateMachine>
</xmi:XMI>
```

Slika 4-11 - Primjer korištenja norme XMI

Konačno, *deskriptivna identifikacija* ne garantira jedinstvenost već samo pruža dodatnu informaciju o elementu. Oznaka koja se koristi je "**XMI:label**".

XMI se danas najviše koristi za razmjenu XML-ovskih dokumenata koji opisuju UML-ovske modele. Iako razvojni alati često koriste ovu normu, postoje i određeni prigovori. Najveći problem je veliki broj službenih verzija specifikacije XMI koje nisu međusobno kompatibilne. Trenutno postoje četiri verzije te specifikacije (XMI 1.0, 1.1, 1.2 i 2.0) što je pogotovo problematično kada se uzme i veći broj izdanih verzija specifikacije UML što dovodi do velikog broja mogućih kombinacija te time i otežanim uspostavljanjem kompatibilnosti. Isto tako, brza evolucija ovih normi znači da se vrlo malo proizvođača softvera odlučuje na praktičnu implementaciju.

Usprkos tome, nedostatak drugih normi za sličnu primjenu čine XMI glavnom polazišnom točkom za stvaranje normizirane XML-ovske reprezentacije metapodataka zasnovanim na normi MOF i srodnim jezicima.

5 Metamodel za sustav stvarnovremenskog skladištenja podataka zasnovan na Uslugama weba

5.1 Mehanizmi proširenja specifikacije CWM

Jedan od ključnih preduvjeta prihvaćanja norme jest njezina fleksibilnost. Ovaj pojam najčešće označava sposobnost norme da se mijenja kako ovisno o potrebama industrije, ali i sposobnost proširenja kako bi se prilagodila uvjetima iz stvarnog svijeta u okviru kojih se ta norma koristi. Specifikacija CWM koristi normizirane OMG-ove mehanizme i specifikacije koji podržavaju (a i pretpostavljaju) znatne mogućnosti proširivanja danih normi.

Postoje dva glavna mehanizma proširena CWM specifikacije: **proširenje kroz pod-klase te proširenje pomoću klasa *TaggedValue* i *StereoType*.**

Proširenje kroz pod-klase je standardni način proširivanja preuzet iz objektno-orijentiranih paradigmi te je kao takav intenzivno korišten i u dizajnu same specifikacije CWM. Ponekad se ovaj mehanizam proširenja jednostavno svodi na preimenovanje CWM-ovskih klasa kako bi odgovarale terminologiji koja će to proširenje koristiti.

Proširenje pomoću klasa *TaggedValue* i *Stereotype* je tehnika uvedena kako bi se omogućio "jednostavniji" način proširivanja modela, npr. u situacijama kad je potrebno dodati pokoji atribut i sl. Stereotipi su jednostavno nazivi koji dodaju semantičku informaciju nekoj klasi kako bi pobliže opisali za što će se klasa koristiti. Označne vrijednosti (*TaggedValues*) su parovi ime-vrijednost koji se mogu dodati bilo kojem elementu modela. Nedostatak ovakvog mehanizma proširenja je nemogućnost izvedbe asocijacija s postojećim klasama te neka dodatna ograničenja nad vrijednostima koje označne vrijednosti mogu poprimiti.

Od ovih elemenata proširenja u nastavku je gotovo isključivo korištena metoda proširenja kroz pod-klase. Ova metoda je prirodan odabir prvenstveno zbog toga što je cijela specifikacija CWM izgrađena pomoću nasljeđivanja postojećih elemenata tako da proširenje dodatnim elementima zapravo nastavlja izgradnju modela na utvrđeni i dokazano učinkovit način, ali i iz razloga što je nasljeđivanje vrlo fleksibilan mehanizam koji predstavlja kompromis između korištenja postojećih elemenata i uvođenja novih konstrukata za potrebe opisivanja metapodataka elemenata sustava kojeg se opisuje.

U nastavku su dani prijedlozi i razlozi proširenja pojedinih paketa koji su odabrani kao korisni za integraciju s uslužno orijentiranim sustavom za stvarnovremensko prikupljanje podataka. Proširenja su dana opisno te UML-ovskim klasnim dijagramom dok je formalizacija provedena kroz XML-ovske sheme. Sheme jezgrenih elemenata dane su u poglavlju **5.2,** dok se sheme svih novoizvedenih elemenata mogu naći u Prilogu 1.

Jedna od osnovnih teza ove disertacije jest korištenje metapodataka za dinamičku prilagodbu uslužno orijentiranog sustava za stvarnovremensko skladištenje (nazvanog u poglavlju 6 SOA RT-ETL sustav) svojoj okolini. Prema spomenutom principu integracije kroz model sustav prikuplja metapodatke te saznaje koji je idući korak procesa, koje operacije treba provesti te s kojim parametrima. Pošto se radi o konkretnoj primjeni metapodataka, potrebno je stvoriti kvalitetan model koji će služiti kao kvalitetna podrška SOA RT-ETL sustavu. Model metapodataka stvara se kroz:

1) Analizu postojećeg modela metapodataka koje pruža specifikacija CWM
2) Stvaranje proširenja specifikacije CWM koja će odgovarati potrebama SOA RT-ETL aplikacije
3) Formalizaciju stvorenih proširenja kroz odgovarajuću normu obradivu od strane sustava SOA RT-ETL

Kao što se vidjelo u poglavlju 4.6, specifikacija CWM svoje elemente definira kroz formalni model zasnovan na MOF-u i UML-u. Svaki paket CWM-a opisan je preko UML-ovske notacije tj. reprezentacijom kroz klasne dijagrame. U skladu s tim opis proširenja će također koristiti takvu notaciju tj. proširenja će se inicijalno definirati pomoću UML-ovskih klasnih dijagrama dok će mehanizmi proširenja odgovarati onima navedenim u poglavlju 5.1 s tim da će mehanizam nasljeđivanja biti preuzet kao glavni način stvaranja novih elemenata.

Specifikacija CWM koristi specifikaciju XMI za serijalizaciju metapodataka tj. njihovo pretvaranje u oblik pogodan za razmjenu i obradu podataka. XMI omogućuje serijalizaciju modela u XML-ovski oblik pomoću pravila stvaranja DTD-ova ili XML-ovskih shema te samih dokumenata. Nažalost, u vrijeme stvaranja specifikacije CWM korištena je tadašnja verzija specifikacije XMI – verzija 1.0 – koja koristi DTD-ove i ne poznaje pojam XML-ovske sheme tako da ne postoji reprezentacija specifikacije CWM u obliku XSD-ovskog dokumenta. Zbog svega toga proširenje i formalizacija slijedit će ove korake:

1) Prepoznavanje elemenata specifikacije CWM koji su ključni za funkcioniranje SOA RT-ETL aplikacije
2) Opisivanje prepoznatih elemenata u obliku XML-ovske sheme, tj. definicija kompleksnih tipova koji će opisivati elemente modela pri čemu će se pratiti principi koje predlaže specifikacija XMI
3) Definicija novih elemenata kroz UML-ovske klasne dijagrame
4) Opisivanje novih elemenata također u obliku XML-ovske sheme proširenjem definiranih tipova (što prati predviđeni mehanizam nasljeđivanja)

Sljedeće poglavlje bavi se točkama 1) i 2), tj. formalizacijom podskupa CWM-ovskih jezgrenih elemenata kroz XML-ovske sheme.

5.2 Formalizacija opisa elemenata jezgrenog paketa specifikacije CWM kroz XML-ovske sheme

Kao što je rečeno, jedan od problema specifikacije CWM leži u činjenici da je za serijalizaciju metapodataka zasnovanih na normi CWM predviđena specifikacija XMI verzije 1.0 koja je nastala prije objave specifikacije XML Schema. Direktan rezultat ovoga jest taj da ne postoje formalni opisi elemenata specifikacije CWM u obliku XML-ovskih shema. Jedini objavljeni oblik službene formalizacije elemanata CWM-a jest DTD-ovski dokument koji se ne može direktno prevesti u oblik XML-ovske sheme zbog neodgovarajuće norme imenovanja elemenata te korištenja dvotočja što je u sukobu sa sintaksom XML-ovskih imenskih područja.

Zbog toga će unutar ovog poglavlja biti izvedena formalizacija podskupa jezgrenog paketa CWM specifikacije kroz XML-ovske sheme tj. definirat će se kompleksni tipovi koji će odgovarati elementima specifikacije CWM predviđenim za proširenje novim elementima koje će koristiti SOA RT-ETL aplikacija. Ovo je nužan preduvjet kako bi se uopće mogla provoditi kasnija formalizacija novih elemenata kroz XML-ovske sheme ako se želi koristiti princip nasljeđivanja.

Za formalizaciju opisa koriste se sljedeći principi:

1) Pratit će se principi specifikacije XMI verzije 2.1.1.

2) Svojstva elemenata (tj. klasa unutar modela) bit će opisana XML-ovskim elementima neovisno o tipu svojstva (specifikacija XMI dozvoljava i atribute i elemente ako se radi o svojstvu s jedinstvenom vrijednosti); ovo se provodi zbog očuvanja konzistentnosti serijalizacije

3) Veze između elemenata također se izvode pomoću XML-ovskih elemenata. Ovisno o tipu veze element koji je povezan s drugim elementom može se navesti *unutar* elementa (npr. u slučaju vlasničke veze) ili preko reference, u kojem slučaju se koristi atribut "*href*" i lokalni identifikator

4) Svaki element implicitno definira elemente koje nasljeđuje, tj. ako postoji hijerarhijski složaj od nekoliko elemenata oni se neće definirati svaki zasebno te povezati kroz reference već će se samo definirati element koji je najniži u roditeljskom slijedu a roditeljski elemente se iz njega mogu izgraditi jednostavnom projekcijom podskupa roditeljskih svojstava.

5) Glavni cilj formalizacije opisa jest korištenje opisa za potrebe SOA RT-ETL sustava. XML-ovske sheme ne pružaju apsolutni i konačni oblik normiziranog opisa elemenata specifikacije CWM, već samo lokalnu normizaciju podskupa potrebnih elemenata i njihovih svojstava kako bi se omogućila njihova praktična implementacija.

Formalizacija strukture svakog elementa jezgrenog paketa CWM-a izvest će se pomoću definicije elemenata specifikacije XML Scheme uz pomoć imenovanog konstrukta *ComplexType*. Ovaj konstrukt između ostalog omogućuje generičku definiciju "složenog tipa" elementa koji se može pojaviti u XML-ovskom dokumentu. Imenovani složeni tipovi mogu se proširivati novim tipovima što se naziva

derivacijom složenog tipa. Ovaj mehanizam umnogome je sličan principu nasljeđivanja u objektno-orijentiranim paradigmama te će se u tu svrhu i koristiti – tj. element će se definirati pomoću složenih tipova a nasljeđivanje će se preslikavati u derivaciju složenih tipova. Dokumenti koji će koristiti metapodatkovne elemente moći će stoga unutar svojih shema koristiti navedene složene tipove kako bi ograničili sintaksnu strukturu elemenata koje sadržavaju.

Formalizacija počinje vršnim elementom specifikacije CWM – klasom *CWM_Element*[8].

Element je vršna klasa bez vlastitih svojstava. XML-ovska shema stoga postoji samo zbog potpunosti nasljednog lanca. No uzevši u obzir da specifikacija XMI diktira imperativ mogućnosti identifikacije svakog elementa modela u XML-ovsku shemu se stavljaju atributi *id* (lokalni identifikator unutar dokumenta), *uuid* (univerzalni identifikator ako je nužna precizna definicija identiteta elementa) te *href* kao univerzalni mehanizam povezivanja elemenata unutar dokumenta kako bi se spriječilo redundantno ponavljanje istih informacija. Zbog jednostavnosti prikaz shema neće uključivati imensko područje osim područja "*xs:*" koje predstavlja uobičajeni prefiks imenskog područja XML Scheme te se neće posebno naznačavati unutar ispisa.

Slika 5-1 prikazuje formalizaciju elementa CWM_Element pomoću XSD Scheme.

```
<xs:element name="CWM_Element" type="CWM_Element_type"/>

<xs:complexType name="CWM_Element_type">
      <xs:attribute name="id" type="xs:string" use="optional"/>
      <xs:attribute name="uuid" type="xs:string" use="optional"/>
      <xs:attribute name="label" type="xs:string" use="optional"/>
      <xs:attribute name="href" type="xs:string" use="optional"/>
</xs:complexType>
```

Slika 5-1 - Shema elementa CWM_Element

Element *CWM_TaggedValue* predstavlja mehanizam proširivanja elemenata modela parovima oznaka/vrijednost. Ovo je najjednostavniji način proširenja elementa koji se najčešće koristi kada alati trebaju metapodatke za neke specifične namjene koje zahtijevaju dodatne informacije poput grafičkog prikazivanja na ekranu, dodatnog opisa i sl. Za očekivati je da će SOA ETL poslovni proces imati koristi od ovakvog mehanizma te se stoga definira XML-ovska shema ovog elementa.

Slika 5-2 prikazuje shemu elementa *CWM_TaggedValue*.

[8] DTD-ovski dokument koji daje formalni opis elemenata specifikacije CWM koristi princip imenovanja CWM:*ImeElementa* koji nije podesan za definiciju kroz XML-ovsku shemu zbog dvotočja koje označava imensko područje i ne može se koristiti kao jedinstveno ime elementa. Zbog toga je u formalizaciji kroz XML-ovske sheme korišten princip *CWM_ImeElementa* za elemente jezgrenog paketa te *CWM_ImePaketa_ImeElementa* za elemente ostalih paketa.

```
<xs:element name="CWM_TaggedValue" type="CWM_TaggedValue_type"/>

<xs:complexType name="CWM_TaggedValue_type">
      <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="value" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string" use="optional"/>
</xs:complexType>
```

Slika 5-2 - Shema elementa CWM_TaggedValue

Dodavanje atributa "*type*" predstavlja odstupanje od normizirane specifikacije koja ne omogućuje definiranje tipa označne vrijednosti već se strogo ograničava na navođenje imena i vrijednosti reprezentirane pomoću niza znakova. "Pravilan" postupak koji ne bi uvodio preinake norme uključivao bi nasljeđivanje elementa CWM_*TaggedElement* novim elementom koji bi sadržavao spomenuti atribut. Odabir "namjernog" odstupanja izveden je strogo zbog očuvanja jednostavnosti shema novih elemenata kojima će se u narednim poglavljima proširivati specifikacija; element *CWM_ModelElement* će u svojoj definiciji podrazumijevati proširenje pomoću označne vrijednosti te će se ovaj mehanizam transparentno proširivati na elemente koji se nalaze niže u hijerarhijskom stablu nasljednih veza. Definicija zasebnog "proširenog" elementa uvela bi znatnu redundanciju kod proširenja specifikacije novim elementima koja se ovakvim postupkom uspješno izbjegla. Naposljetku, implementacija registra metapodataka ne mora odstupati od CWM specifikacije – dovoljno je definirati vezu između elementa *CWM_TaggedValue* i elementa CWM_*DataType* koji će se preslikavati u atribut tipa od strane aplikacije koja će koristiti "prošireni" *CWM_TaggedValue* element.

Klasa *ModelElement* predstavlja imenovani entitet unutar modela. XML-ovska shema sadržavat će elemente *name* (ime) kako bi se očuvala semantička svrha klase te CWM_*TaggedValue* kao već spomenuti osnovni mehanizam jednostavnog proširenja. Tip elementa *CWM_ModelElement* derivira se iz tipa *CWM_Element_type.*

Slika 5-3 prikazuje shemu elementa *CWM_ModelElement.*

```
<xs:element name="CWM_ModelElement" type="CWM_ModelElement_type"/>

<xs:complexType name="CWM_ModelElement_type">

      <xs:complexContent>
      <xs:extension base="CWM_Element_type">
            <xs:sequence>
            <xs:element name="name" type="xs:string" minOccurs="0"/>
            <xs:element name="taggedValue" type="CWM_TaggedValue_type"
                  minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:extension>
      </xs:complexContent>

</xs:complexType>
```

Slika 5-3 - Shema elementa CWM_ModelElement

Element *NameSpace* zapravo je element modela čije je glavno svojstvo mogućnost posjedovanja drugih elemenata modela. Njegova XML-ovska shema izvodi se iz tipa *CWM_ModelElement_type* a dodaje mu se jedan element čija je semantička uloga zapravo definicija elementa-vlasnika. Shemu prikazuje **Slika 5-4**.

```
<xs:element name="CWM_Namespace" type="CWM_Namespace_type"/>

<xs:complexType name="CWM_Namespace_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
            <xs:sequence>
                  <xs:element name="ownedElement" minOccurs="1"
                  maxOccurs="unbounded">
                        <xs:complexType>
                        <xs:sequence>
                        <xs:element name="CWM_ModelElement"
                        type="CWM_ModelElement_type"
                        minOccurs="1" maxOccurs="1"/>
                        </xs:sequence>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika 5-4 - Shema elementa CWM_Namespace

Element *Expression* definira izraz tj. izjavu izraženu u određenom jeziku koja se može interpretirati i obraditi od jedinke koja poznaje jezik tog izraza. Ovaj element nije nasljednik elementa *ModelElement* pošto ne mora imati definirano ime i vrijednost tako da se derivira direktno iz vršnog elementa. Shemu prikazuje **Slika 5-5** .

```
<xs:element name="CWM_Expression" type="CWM_Expression_type"/>

<xs:complexType name="CWM_Expression_type">
      <xs:complexContent>
      <xs:extension base="CWM_Element_type">
      <xs:sequence>
      <xs:element name="body" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="language" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika 5-5 - Shema elementa CWM_Expression

Element *CWM_Dependency* opisuje međuovisnost dvaju elemenata. Ovaj element koristit će se kao glavni mehanizam uspostavljanja veze između dva elemenata modela koja nije "nasljedna" ili "vlasnička" veza tj. sve veze koje ne ulaze u ova dva tipa rješavat će se na razini M1 u toku stvaranja modela. Shema ovog elementa dana je na slici (**Slika 5-6**).

```
<xs:element name="CWM_Dependency" type="CWM_Dependency_type"/>

<xs:complexType name="CWM_Dependency_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement">
      <xs:sequence>
      <xs:element name="kind" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="client" type="CWM_ModelElement" minOccurs="0"
      maxOccurs="unbounded"/>
      <xs:element name="supplier" type="CWM_ModelElement" minOccurs="0"
      maxOccurs="unbounded"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika 5-6 - Shema elementa CWM_ Dependency

Ovi temeljni elementi (tj. njihovi tipovi) koristit će se za definicije proširenja koja se opisuju u poglavljima koja slijede.

5.3 Specifikacija elemenata novog metamodela

Podskup jezgrenog paketa specifikacije CWM formaliziran pomoću XML-ovskih shema (opisanih u prethodnom poglavlju) korišten je kao temelj definicije metapodataka za potrebe SOA RT-ETL aplikacije. Sva proširenja pohranit će se u paket nazvan CWMX_SOA.

Svi elementi paketa CWMX_SOA izvedeni su nasljeđivanjem elemenata definiranih u poglavlju 5.2. Kako bi se olakšala formalizacija pored nasljedne veze postoji samo još jedan predefinirani tip veze između elemenata – vlasnička veza koja se opisuje direktnim navođenjem elementa u vlasništvu unutar tijela elementa-vlasnika ili pomoću reference. Nadalje, veze su ograničene na tip 1:1, također zbog jednostavnije definicije formalnog opisa a s time i lakše obrade i interpretacije.

Sve ostale veze opisuju se pomoću elementa *CWM_Dependency* na razini M1 tj. pri modeliranju samog sustava.

Opis proširenja počet će definicijom XML-ovskih elemenata paketa CWMX_SOA. U ovom poglavlju bit će dani UML-ovski prikazi proširenih elemenata. Formalizacija izvedena kroz XML-ovske sheme može se pronaći u Prilogu 1**.**

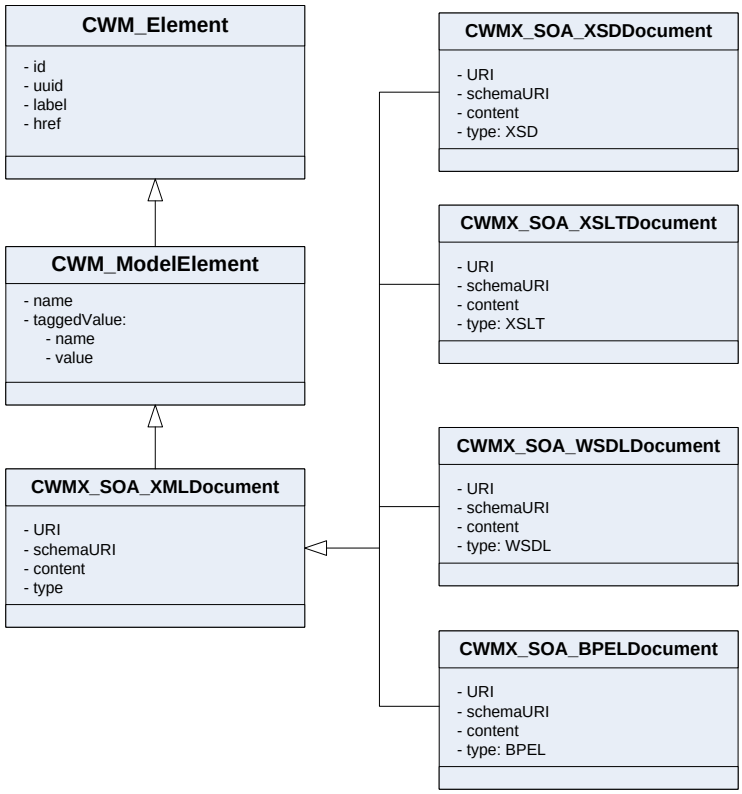
5.3.1 XML-ovski elementi paketa CWMX_SOA

XML-ovski dokumenti ključni su elementi uslužno orijentiranog procesa za stvarnovremenski ETL. Oni između ostalog opisuju:

- XML-ovske sheme dokumenata koji će se razmjenjivati
- poslovne poruke koje proces koristi i koje odgovaraju navedenim shemama
- same metapodatke koji će se dohvaćati iz registra
- WSDL-ovske dokumente koji opisuju Usluge weba (uključujući i WSDL-ovske dokumente poslovnog procesa)
- BPEL-ovske dokumente koji opisuju sam poslovni proces prikupljanja, transformacije i pohrane podataka u stvarnom vremenu

Paket CWM:XML koji već postoji unutar specifikacije CWM je dosta problematičan zbog svoje uske povezanosti s DTD-om (već spomenutim oblikom opisivanja ograničenja XML-ovskih dokumenata koji se danas smatra zastarjelim i čiju je ulogu gotovo potpuno preuzela *XML Schema*). Dodatno, paket sadrži niz elemenata koji opisuju strukturu pojedinog XML-ovskog dokumenta što je u praktičnoj primjeni redundantno, pogotovo ako se uzme u obzir činjenica da je XML glavni mehanizam razmjene metapodataka u sustavi koji se opisuje unutar ovog rada te da XML Schema sama za sebe predstavlja "metapodatke" o strukturi dokumenta. Konačno, ovaj paket sadrži element *CWM_XML_Schema* koji unatoč istom nazivu nema nikakve veze sa specifikacijom XML Schema što uvodi dodatnu konfuziju. Zbog toga je ovaj paket zamijenjen alternativnim rješenjem koje će omogućiti što učinkovitiju pohranu potrebnih metapodataka bez potrebe za prilagodbom zastarjelim konceptima.

Slika 5-7 prikazuje UML-ovski dijagram XML-ovskih elemenata paketa CWMX_SOA.



Slika 5-7 – Dijagram XML-ovskih elemenata paketa CWMX_SOA

Ovdje je važno uočiti visoku razinu fleksibilnosti kod unosa stavke metapodatka koja opisuje XML-ovski dokument. On se može zadati samo URI-jem tj. identifikatorom lokacije na kojoj se taj dokument može pronaći. Isto tako, URI se ne mora navesti već se cjelokupni sadržaj dokumenta može pohraniti u samom registru metapodataka (što je npr. poželjno kod XML-ovskih shema koje i same predstavljaju svojevrsne metapodatke). Konačno, može se navesti i vanjski URI ali i sam sadržaj, s tim što onda sustav upravljanja registrom metapodataka mora voditi računa o konzistentnosti podataka vezanih uz XML-ovski dokument.

Elementi koji predstavljaju XML-ovsku shemu, XSLT-ovski, WSDL-ovski i BPEL-ovski dokument su sve zapravo XML-ovski dokumenti te se kao takvi trebaju i reprezentirati odgovarajućom metapodatkovnom shemom. Važno je napomenuti da nije nužno ove elemente definirati na razini M2 već je moguće to ostaviti za razinu M1 tj. uvesti stavku generičkog XML-ovskog dokumenta te postaviti odgovarajući tip pri modeliranju sustava. No pošto se radi o tipovima XML-ovskih dokumenata koji su usko povezani s elementima sustava SOA RT-ETL očekuje se da će definiranje zasebnih stavki na razini M2 olakšati modeliranje na razini M1 – posebno kada se radi o definiranju veza kroz element *CWM_Dependency.*

Iz elementa *CWMX_SOA_XMLDocument* derivirat će se sljedeći elementi:

- *CWMX_SOA_XSDDocument* (reprezentacija XSD-ovskog dokumenta tj. XML-ovske sheme)

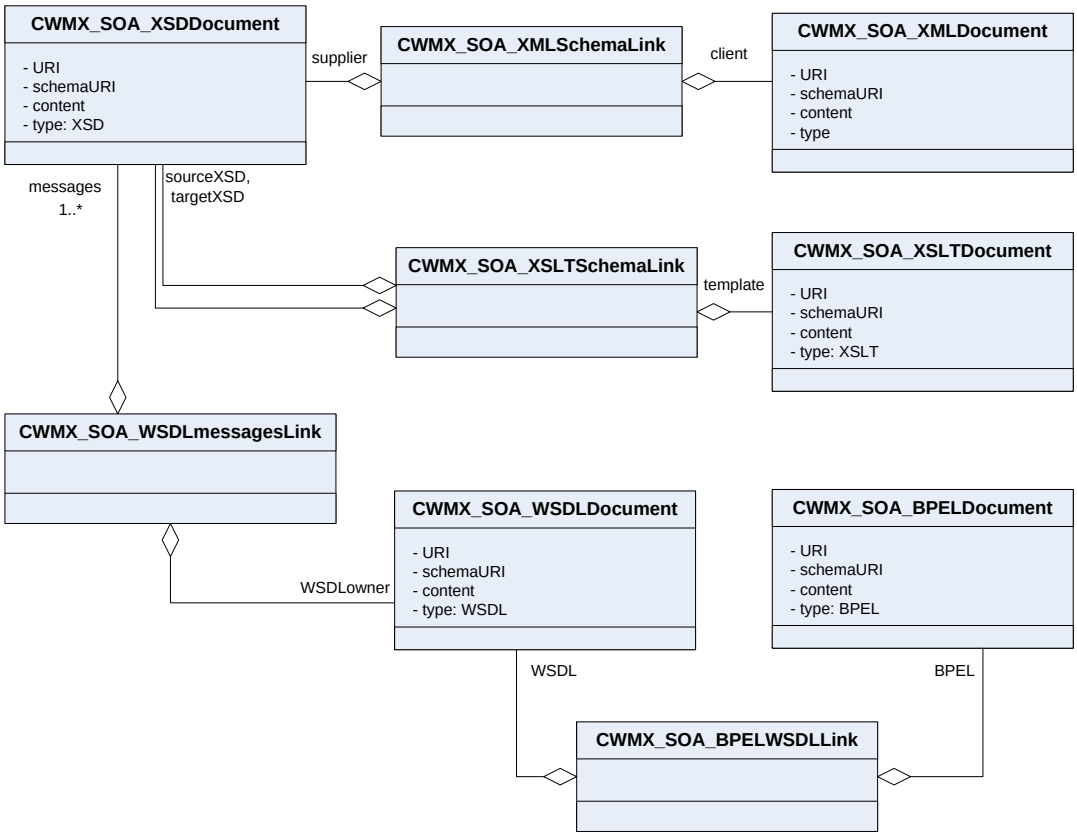
- *CWMX_SOA_WSDLDocument* (reprezentacija WSDL-ovskog dokumenta koji opisuje Uslugu weba)
- *CWMX_SOA_BPELDocument* (reprezentacija BPEL-ovskog dokumenta koji opisuje izvođenje poslovnog procesa)
- *CWMX_SOA_XSLTDocument* (reprezentacija XSLT-ovskog dokumenta koji opisuje postupak transformacije određenog XML-ovskog dokumenta u drugi XML-ovski dokument ili neki drugi oblik)

Derivacija ovih elemenata izvest će se postavljanjem atributa tipa XML-ovskog dokumenta kroz odgovarajuću predefiniranu konstantu. Definicijom zasebnog kompleksnog tipa za svaki od ovih elemenata omogućuje laku nadogradnju u slučaju ukazane potrebe tokom korištenja sustava.

Metapodatkovni element koji predstavlja XML-ovsku shemu je sam po sebi vrlo interesantan jer sadržaj XML-ovske sheme nekog dokumenta sam po sebi predstavlja metapodatke o tom elementu. Specifikacija CWM inicijalno nudi način "opisa" strukture XML-ovskog dokumenta kroz CWM-ovske elemente što je samo po sebi redundantno iz više razloga – prvo jer se te informacije mogu jednostavno dobiti iz samog XML-ovskog dokumenta i drugo što su te informacije nepotpune jer one samo predstavljaju strukturu dokumenta (koja je inherentna karakteristika samog dokumenta ako je on dobro oblikovan; inače se strogo govoreći niti ne radi o XML-ovskom dokumentu). To znači da ti metapodaci ne govore ništa o svrsi samog dokumenta ni o ograničenjima ili pravilima prema kojima je on stvoren. S druge strane XML-ovska shema nudi konkretne metapodatke o XML-ovskom dokumentu ali kroz svoj sadržaj tj. kroz elemente koji sami po sebi nisu u sprezi s CWM-ovskom normom. Zbog toga je preporučeni način unosa zapisa o XML-ovskoj shemi nekog dokumenta stvaranje stavke *CWMX_SOA_XSDDocument* u čijem će sadržaju biti navedena sama shema.

Poveznica XML-a i njegove sheme može se napraviti preko URI-ja koji će pokazivati na vanjsku lokaciju te sheme ili na lokaciju preko koje se može doći do samog metapodatkovnog elementa unutar registra metapodataka. No pošto se na ovaj način izlazi iz okvira norme poželjno je stvoriti i zasebni element koji će povezivati XML-ovski dokument s njegovom shemom.

Slika 5-8 prikazuje UML-ovski dijagram elemenata koji omogućuju modeliranje semantičkih poveznica XML-ovskih metapodatkovnih elemenata. Detaljne formalizacije ovih elemenata dane su dalje u tekstu. Svi elementi poveznica su naslijeđeni iz elementa *CWM_ModelElement* što nije posebno naznačeno zbog čistoće prikaza.



Slika 5-8 - Dijagram metapodatkovnih elemenata-poveznica XML-ovskih elemenata

Element koji povezuje XML-ovski dokument i njegovu shemu izvodi se prema predlošku elementa *CWM_Dependency* i imenuje se *CWMX_SOA_XMLSchemaLink.* Ovo znači da se veza između XML-ovske sheme i XML-ovskog dokumenta opisanog tom shemom zapravo može izraziti na dvije razine – na razini M0 kroz obradu samog XML-ovskog dokumenta te na razini M1 preko metapodataka. Ovime je omogućena dodatna fleksibilnost no opet se mora naglasiti važnost održavanja konzistentnih podataka u metapodatkovnom registru.

Element *CWMX_SOA_XSLTDocument* opisuje predložak za preoblikovanje XML-ovskog dokumenta u drugi XML-ovski dokument ili neki drugi oblik.

Za opis preslikavanja također će se stvoriti zasebni metapodatkovni element pod nazivom *CWMX_SOA_XSLTLink* koji povezuje XSLT-ovski dokument, izvorni XML-ovski te ciljni XML-ovski dokument (pošto se XSLT naslanja na strukturu dokumenta izvor i cilj se zapravo definiraju shemom dokumenta). Rezultat XSLT-ovske transformacije ne mora nužno biti XML-ovski dokument tako da element *CWMX_SOA_XSLTLink* omogućuje definiciju alternativnog "cilja" transformacije čiji je tip postavljen na *ModelElement* tj. točna priroda cilja transformacije ostavlja se otvorenom i definira se na razini M1.

Element *CWMX_SOA_WSDLDocument* između ostalog sadrži i definicije poruka koje Usluga weba razmjenjuje (ako se radi o dokument-centričnoj Usluzi weba). Metapodaci bi mogli na razini M1 opisati koje XSD-ovske dokumente Usluga weba koristi tako da se za tu potrebu razvija element *CWMX_SOA_WSDLmessageLink.*

Konačno, BPEL-ovski dokument povezan je sa svojim pripadnim WSDL-ovskim dokumentom.[9] Za to se koristi element *CWMX_SOA_BPELWSDLLink.*

Ovim elementom završena je formalizacija elemenata metamodela za potrebe opisa XML-ovski orijentiranih elemenata SOA ETL sustava. U nastavku će se modelirati element koji opisuje izraze (engl. *Expressions*) koje će sustav koristiti tokom svog rada.

5.3.2 Element opisa izraza

Specifikacija CWM nudi nekoliko elemenata koji opisuju "izraze", tj. izjave izrečene u određenom jeziku koje se na neki način mogu obraditi što može rezultirate određenom vrijednošću.

Element *CWM_Expression* već je ukratko objašnjen u poglavlju **4.6.3** te formaliziran u obliku XML-ovske sheme u poglavlju **5.2**. Unutar specifikacije CWM ovaj element se u nekoliko dodatnih paketa specijalizira kroz element *CWM_ProcedureExpression* (element istog oblika ali preimenovan zbog dodatne semantičke karakteristike da će obrada izraza kojeg opisuje rezultirati izvjesnom izmjenom okoline unutar koje se obrađuje) te *CWM_QueryExpression* (također analogan elementu *CWM_Expression* ali preimenovan kako bi dodatno naznačio da se radi o *upitu* a ne generičkom izrazu).

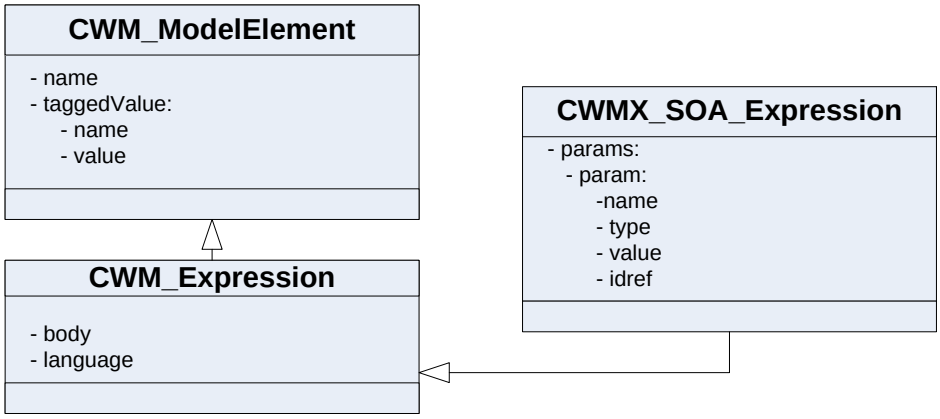
Dodatno, specifikacija CWM nudi kompletni metamodel za opisivanje izraza koji pretpostavlja raščlambu bilo kojeg izraza na čvorove (čvor svojstva, čvor konstante ili čvor elementa) kako bi se kroz metapodatke opisao te svaki pojedini segment izraza. Takav izraz nazvan je "izraz bijele kutije" (engl. *white box expression*).

SOA RT-ETL sustav treba izraze kako bi obavljao svoje uobičajene zadaće: izraz može opisivati način kako prikupiti podatke, kako ih obraditi te kako ih pohraniti u određeni spremnik. U tradicionalnom scenariju takvi izrazi bili bi zapravo SQL-ovske izjave koje bi elementi SOA RT-ETL sustava izvršavali nad određenom relacijskom bazom podataka.

Modeliranje SQL-ovskih izjava kao "izraza bijele kutije" rezultiralo bi prevelikom razinom redundancije pošto bi se svaki takav izraz morao posebno rekonstruirati iz pripadnih metapodataka. Zbog toga se za potrebe SOA RT-ETL sustava koriste klasični izrazi (tj. izrazi crne kutije) koji su opisani samo kroz svoj sadržaj te oznaku jezika u kojem su opisani.

[9] Točnije rečeno, BPEL-ovski dokument je povezan sa više WSDL-ovskih dokumenata: dokumentom koji opisuje Uslugu weba koja implementira sam proces te dokumentima koji opisuju Usluge weba s kojima on surađuje. Za ove potonje neće se izrađivati metapodatkovni element veze zbog redundancije – proces će dobiti svoj vlastiti metapodatkovni opis koji će se na "višim" razinama povezati sa Uslugama weba s kojima komunicira te će se spuštanjem na niže razine moći doprijeti do njihovih WSDL-ovskih dokumenata. (Ovdje je također potrebno napomenuti da se radi o višim razinama apstrakcije koje još uvijek ostaju na razini M2 jer se radi o apstrahiranju koncepata a ne njihovih opisnih elemenata)

Element *CWM_Expression* će se zato naslijediti elementom *CWMX_SOA_Expression.* **Slika 5-9** prikazuje UML-ovski klasni dijagram ovog elementa.



Slika 5-9 – Dijagram elementa CWMX_SOA_Expression

Struktura elementa *CWMX_SOA_Expression* identična je elementu kojeg nasljeđuje s razlikom da će se izraz pohranjivati u *generičkom obliku* tj. izvjesni segmenti izraza bit će zamijenjeni posebnom oznakom umjesto koje će se ubaciti stvarni parametri u trenutku izvršavanja upita. Ovakav način modeliranja izabran je ponajviše zbog lakše implementacije Usluga weba koje za izvršavanje SQL-ovskih upita mogu klasu *PreparedStatement* koja se nalazi u paketu JDBC jezika *Java.* Ova klasa može direktno koristiti ovakve generičke upite koje tokom izvođenja dinamički proširuje danim parametrima što se htjelo i postići modeliranjem ovog elementa.

5.3.3 Element opisa tablice – međuspremišta podataka

Jedan od glavnih elemenata ETL-ovskog procesa prikupljanja i pripreme podataka za skladište su međuspremišta podataka koja služe kao privremena mjesta za pohranu podataka "u obradi". U klasičnom ETL-ovskom procesu ovakva međuspremišta najčešće se izrađuju u obliku jednostavnih tekstualnih datoteka (koja omogućuju brzu obradu velike količine podataka) ili relacijskih tablica (čija strukturirana priroda omogućuje lakšu kontrolu i upravljanje nad međupodacima ali ponekad uz cijenu veće potrebe za resursima i dužeg vremena obrade) [19].

SOA RT-ETL aplikacija koristi XML-ovske poruke te će se shodno tome međuspremišta će također biti XML-ovski dokumenti. Naravno, potrebno je definirati strukturu XML-ovskog međuspremišta za što će se zbog očuvanja konzistentnosti i lakše obrade koristiti relacijski model tj. podatke će se smještati u XML-ovsku reprezentaciju relacijske tablice podataka. Ova tablica može ali ne mora biti povezana s tablicama koje koristi klasični ETL-ovski proces; u svakom slučaju otvorena je mogućnost bržeg modeliranja sustava ako se jednostavno preuzimaju već gotovi modeli međuspremnika iz klasičnog ETL-ovskog sustava.

Zbog svega toga potrebno je definirati metapodatkovni element koji će omogućiti modeliranje ovakvog tipa međuspremnika. Međuspremnik mora imati sljedeće karakteristike;

- puni se pomoću procesa prikupljanja podataka ili transformacije podataka tokom procesa
- strukturalno odgovara tablici relacijske baze podataka
- čuva podatke o izrazu izvršavanjem kojeg su podaci prikupljeni, strukturi samog međuspremnika te ima mogućnost čuvanja prikupljenih podataka ako je to potrebno

Element *CWMX_SOA_StagingTable* neće biti direktno izveden nasljeđivanjem elemenata paketa *Relational* zbog velike redundancije i činjenice da će se ovaj element intenzivno koristiti unutar stvarnovremenskog SOA R-ETL sustava te shodno tome mora biti dizajniran tako da omogućuje brzu obradu informacija. Zbog svega toga izvjesni koncepti paketa *Relational* specifikacije CWM će se uokviriti u ograničeni skup atributa elementa *CWMX_SOA_StagingTable* sa sljedećim preduvjetima:

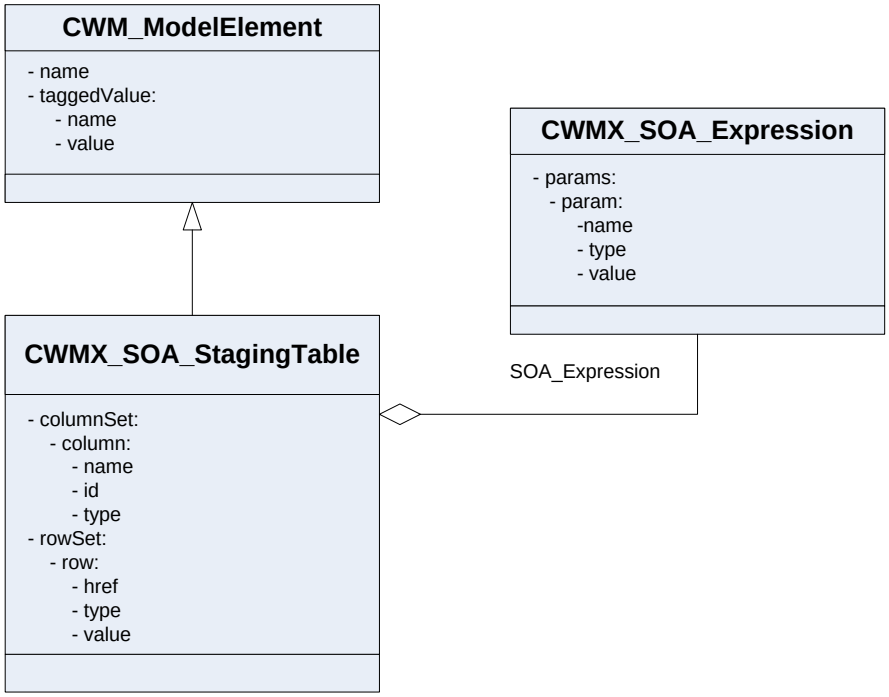
- element *CWMX_SOA_StagingTable* mora se moći ekstrapolirati iz postojećih metapodatkovnih elemenata koji u potpunosti odgovaraju specifikaciji CWM (konkretno paketu *Relational*)
- element mora uključiti mogućnost prenošenja relacijske informacije tj. definicija elementa mora osigurati "mjesto" za pohranu podataka koji će se prenositi unutar sustava

Razlog za odmak od specifikacije CWM jest sličan dosadašnjim razmatranjima - XML-ovski oblik reprezentacije složenih metapodatkovnih elemenata posjeduje veliku količinu redundancije, što nije dobra karakteristika za implementaciju unutar stvarnovremenskih sustava (pogotovo ako se uzme u obzir činjenica da XML kao norma sam po sebi inherentno posjeduje znatnu razinu redundantnih elemenata). Iako je u [30] naglašeno da se CWM *može* koristiti za prijenos manjih količina relacijskih podataka, element *CWMX_SOA_StagingTable* ipak predstavlja "kondenziranu" verziju originalnih CWM-ovskih elemenata kako bi se omogućila jednostavnija i brža računalna obrada.

Element *CWM_Relational_QueryColumnSet* predstavlja osnovni predložak formalizacije elementa *CWMX_SOA_StagingTable.* **Slika 5-10** prikazuje UML-ovski klasni dijagram novog elementa.

Ono što je važno napomenuti jest da iako atribut *id* elementa *<Column>* ima vrijednost *"optional*" tj. prema XML-ovskoj shemi nije nužno da svaki stupac ima svoj identifikator, on je obavezan ako se instanca ove sheme koristi kao međuspremnik podataka unutar procesa. Ovu "uvjetnu" restrikciju nije moguće opisati preko XML-ovske sheme te se mora voditi računa da se ista uspostavi na aplikacijskoj razini.[10]

[10] Ova napomena vrijedi za identifikator *id,* a ne za identifikator *uuid*, kojeg element *<Column>* također može posjedovati ako je prikupljen iz registra metapodataka modeliranog po specifikaciji CWM. Identifikator *uuid* će jedinstveno definirati stupac unutar navedenog registra ali to nije nužno za prijenos i pohranu podataka unutar elementa *CWMX_SOA_StagingTable;* bitno je samo uspostaviti vezu između elemenata *<columnValue>* i *<column>* tako da atribut *id* jednostavno predstavlja poveznicu te nije nužno da bude univerzalno jedinstven, već samo jedinstven unutar dokumenta.



Slika 5-10 – Dijagram elementa CWMX_SOA_StagingTable

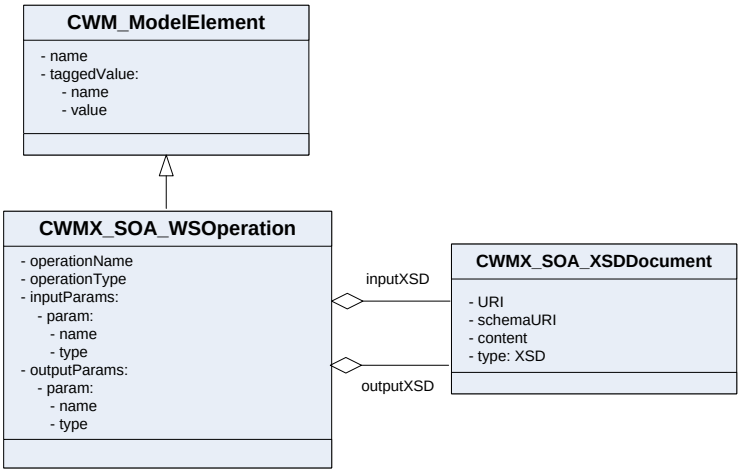
Ovdje se isto tako može uočiti da element *<rowSet>* preuzima ključnu ulogu unutar stvarnovremenskog SOA RT-ETL sustava pošto on direktno pohranjuje podatke koji se prikupljaju, transformiraju i konačno pohranjuju u skladište, tj. njegovu stvarnovremensku particiju. Većina transformacija koja će se raditi nad podacima će se upravo raditi nad elementom *<rowSet>* tj. nad nizom "redaka" koji predstavljaju međuspremište prikupljenih podataka.

5.3.4 Elementi opisa Usluga weba

Usluge weba su ključni element stvarnovremenskog SOA RT-ETL sustava. Cijeli proces pripreme podataka za stvarnovremensku particiju počiva nad sustavom međusobno povezanih Usluga weba. Zbog toga jedan od najvažnijih metapodatkovnih elemenata predstavlja upravo element koji će čuvati metapodatke o korištenim Uslugama weba.

Kao što je spomenuto u poglavlju 2.1, Usluge weba se ugrubo dijele na dvije kategorije: Usluge weba orijentirane na udaljene pozive metoda (engl. *RPC-oriented Web Services*) i dokumentno-orijentirane Usluge weba (engl. *document-oriented Web Services*). Glavna razlika je u tipu operacija koje dotične usluge nude – prve nude klasične operacije zasnovane na ulaznim i izlaznim parametrima određenog tipa, dok se potonje zasnivaju na razmjeni XML-ovskih dokumenata.

Shodno tome definirat će se element *CWMX_SOA_WSOperation* koja će predstavljati oba tipa operacija. **Slika 5-11** prikazuje UML-ovski klasni dijagram ovog elemenata.

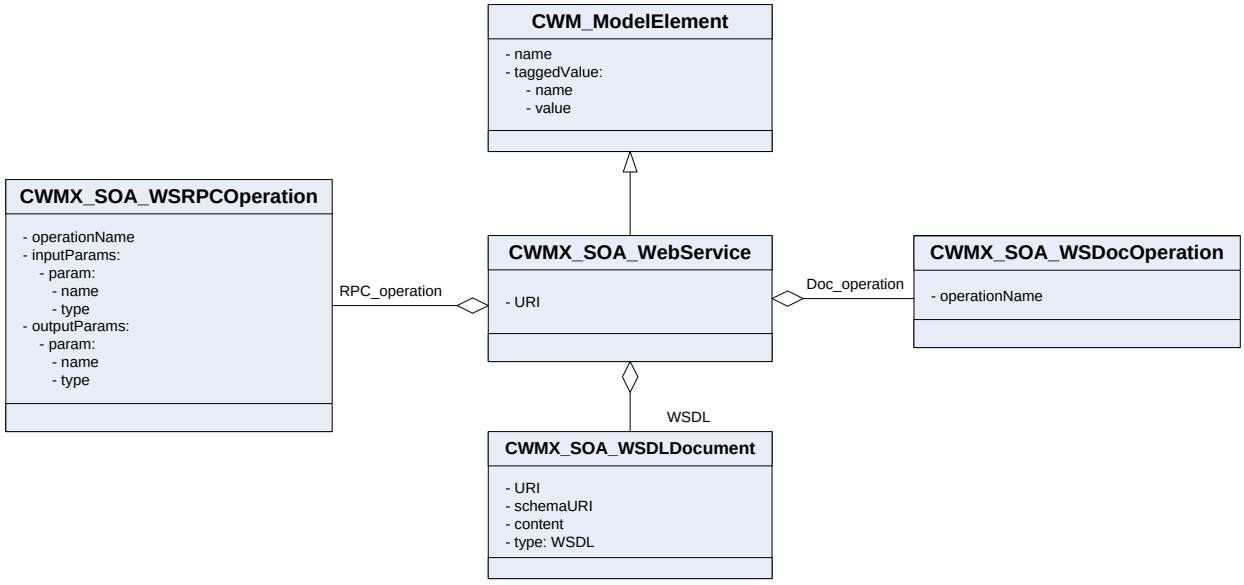


Slika 5-11 – Dijagram elementa CWMX_SOA_WSOperation

Atribut *„operationType“* definira tip operacije. Ako je postavljen na „RPC“ onda ovaj element opisuje „klasičnu“ operaciju čiji su metapodaci njezino ime te ulazni i izlazni parametri određenog tipa. *„Document“* predstavlja operaciju čiji su metapodaci njezino ime a ulazni i izlazni parametar su određeni XML-ovski dokument definiran svojom XML-ovskom shemom.

Osim operacija element opisa Usluge weba mora biti povezan sa svojim WSDL-ovskim dokumentom, koji *de facto* daje sve nužne metapodatke o Usluzi weba. Dodatni metapodaci o samoj Usluzi weba nisu potrebni pošto će se njezina uloga unutar SOA ETL sustava očitovati pomoću veza s drugim elementima koji će biti opisani u narednim poglavljima.

Slika 5-12 prikazuje UML-ovski klasni dijagram elementa *CWMX_SOA_WebService.*



Slika 5-12 – Dijagram elementa CWMX_SOA_WebService

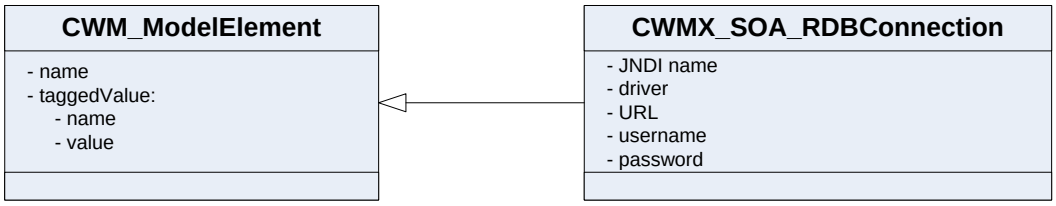
5.3.5 Element opisa veze sa spremištima podataka

Specifikacija CWM nudi pakete pod nazivom *SoftwareDeployment* i *Business Information* čiji elemente omogućuju detaljni zapis softvera i hardvera koji se koristi unutar sustava skladištenja podataka.

Iako elementi ovih paketa imaju veliku izražajnu moć, njihova uloga je prvenstveno namijenjena dokumentaciji hardverskih i softverskih resursa te ih zbog njihove složene strukture nije jednostavno računalno obraditi u stvarnovremenskom sustavu kakav je po svojoj funkcionalnosti SOA RT-ETL sustav.

Zbog toga se za potrebe SOA RT-ETL sustava razvija novi element pod nazivom *CWMX_SOA_RDBConnection* koji na jednostavan i kompaktan način opisuje lokaciju i način spajanja na udaljeni sustav relacijske baze podataka. Ovaj element može se izvesti i postupkom transformacije i sinteze niza elemenata koji već postoje u specifikaciji CWM a njegova ključna zadaća je sustavu objasniti gdje se nalazi relacijska baza kojoj mora pristupiti, protokol spajanja na nju te autorizacijske elemente koji će omogućiti pristup.

Slika 5-13 prikazuje UML-ovski klasni dijagram elementa *CMWX_SOA_RDCConnection.*



Slika 5-13 – Dijagram elementa CWMX_SOA_RDBConnection

5.3.6 Elementi za opis prikupljanja, transformacije i pohrane podataka

Svi do sada izvedeni metapodatkovni elementi odnosili su se - na neki način - na konkretne resurse koji se nalaze unutar sustava skladištenja podataka. XML-ovski dokumenti, veze na izvore podataka, izrazi u određenom jeziku, relacijske tablice kao međuspremnici podataka – sve su to entiteti koji u semantičkom smislu imaju nekakvu fizičku reprezentaciju te ih se kao takve može identificirati, dohvatiti i njima se u određenom smislu poslužiti.

ETL-ovski proces uključuje logička grupiranja ovih elemenata. Npr. proces prikupljanja podataka uključuje povezivanje na izvor podataka, izvođenje izraza pomoću kojeg se podaci dohvaćaju te spremanje tih podataka u međuspremnik. Implicitno se podrazumijeva i programska logika tj. jedinica softvera koja će taj postupak moći i izvršiti – u slučaju SOA RT-ETL sustava to je određena Usluga weba.

Opis procesa dakle uključuje i grupiranje elemenata u logički kategorizirane skupine. ETL-ovski proces se intuitivno može podijeliti u tri procesa (vidljiva i iz samog naziva):

- proces prikupljanja
- proces transformacije
- proces pohrane

Svaki od ovih potprocesa modelirat će se pomoću zasebnog metapodatkovnog elementa, a skupina potprocesa činit će jedan ETL-ovski proces, tj. skup metapodataka koji opisuje proceduru premještanja određenih podataka iz transakcijskih izvora u stvarnovremensku particiju, naravno u obliku prilagođenom definiranom skladištu.

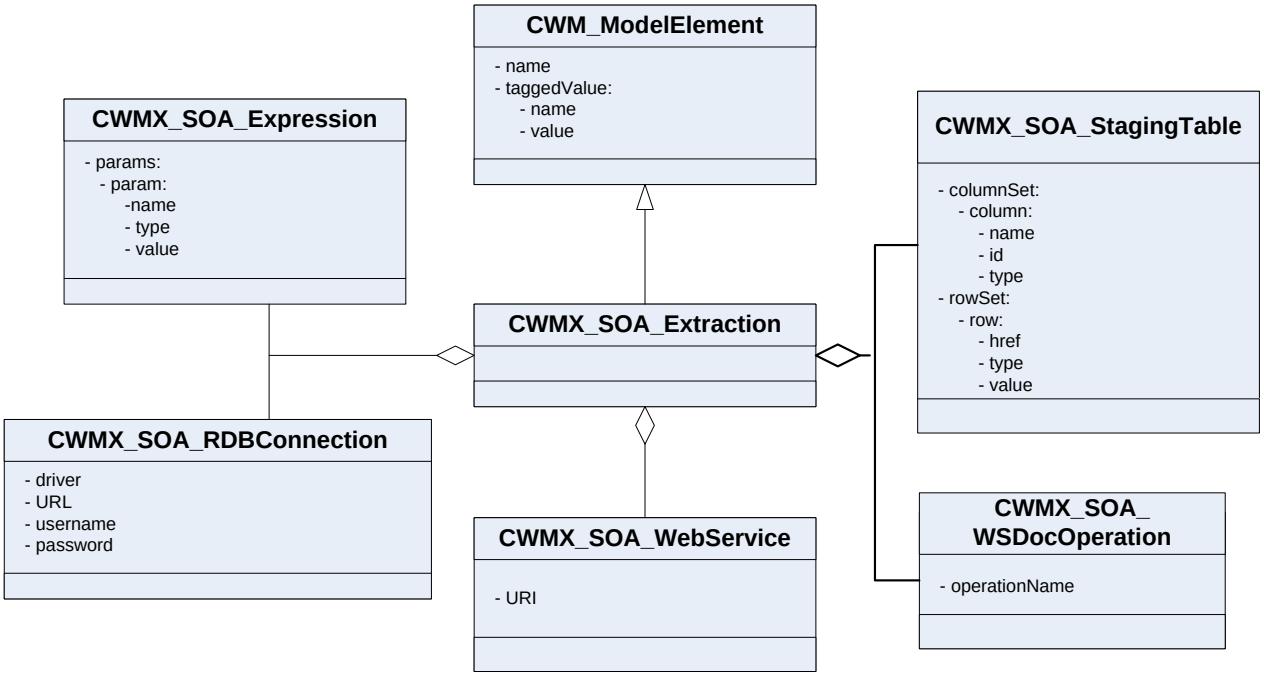
Osim logičkog grupiranja ovi elementi imaju još jednu ulogu. Pošto je jedna od osnovnih ideja vodilja razvoja metapodatkovnih opisa stvarnovremenskog skladištenja bila njihova sprega sa SOA RT-ETL sustavom, potrebno je već u ovoj fazi razlučiti najprihvatljiviji način korištenja danih metapodataka od strane Usluga weba. Elementi opisani u ovom poglavlju mogu predstavljati spremnike za enkapsulaciju podataka koje će primati određene Usluge weba. Drugim riječima, XML-ovske sheme formalizacije elemenata koji predstavljaju prikupljanje, transformaciju i pohranu podataka mogu se koristiti i kao predlošci XML-ovskih poruka koje će se razmjenjivati u SOA RT-ETL sustavu. Primjer ovoga može se vidjeti u poglavlju 6, a ovo poglavlje nadalje će se baviti samo navođenjem pojedinih shema bez daljnjeg osvrta na njihovu konkretnu uporabu unutar sustava.

Prvi element jest *prikupljanje podataka*. Programska jedinka koja izvodi prikupljanje jest – kao što je već rečeno – Usluga weba. U prethodnom poglavlju je spomenuto da SOA ETL sustav prepoznaje dva tipa Usluga weba – usluge temeljene na meta-podacima te samostalne Usluge weba. Kao što će biti prikazano u narednim poglavljima koja će se baviti konkretnom implementacijom sustava, Usluga weba za prikupljanje podataka temeljena na metapodacima bit će izvedena u generičkom obliku tj. njezina funkcionalnost ovisit će o dobivenim parametrima iz registra metapodataka – konkretno: s kojim repozitorijem se mora povezati, koji izraz mora izvršiti, kako mora izgledati međuspremnik u koji će pohraniti podatke. Samostalna Usluga weba ima specijaliziranu ulogu – ona zahtjeva minimalni skup ulaznih parametara (npr. vremensku oznaku zadnjeg prikupljanja) ali je zato strogo prilagođena prikupljanju određenih podataka iz određenog izvora[11].

UML-ovski klasni dijagram elemenata koji opisuju prikupljanje, transformaciju i pohranu podataka prikazuje **Slika 5-14**.

Element *CWMX_SOA_Extraction* reprezentira prikupljanje podataka pomoću određene Usluge wbea (tip Usluge weba je implicitno zadan definicijom elementa; npr. usluga temeljena na metapodacima zahtjevat će elemente *CWMX_SOA_Connection* i *CWMX_SOA_Expression* dok oni nisu nužni za samostalnu Uslugu weba).

[11] Ono što je važno uočiti jest činjenica da su i jedna i druga usluga modelirane tako da kao ulazni parametar primaju XML-ovsku reprezentaciju elementa *CWMX_SOA_Expression* a kao izlazni vraćaju podatke u obliku *CWMX_SOA_StagingTable* – ova restrikcija je nužna kako bi se standardizirao tip poruka unutar SOA ETL sustava (što će biti prikazano u poglavlju 6 koje se bavi implementacijom sustava). Ovo ne znači da se podaci ne mogu prikupljati preko Usluga weba koja ne koristi zadane formate – za takav tip usluga predviđena je izvedba zasebne "prevoditeljske" Usluge weba koja će predstavljati komunikacijski kanal između SOA RT-ETL sustava i usluge koja ne poznaje njegove standarde.

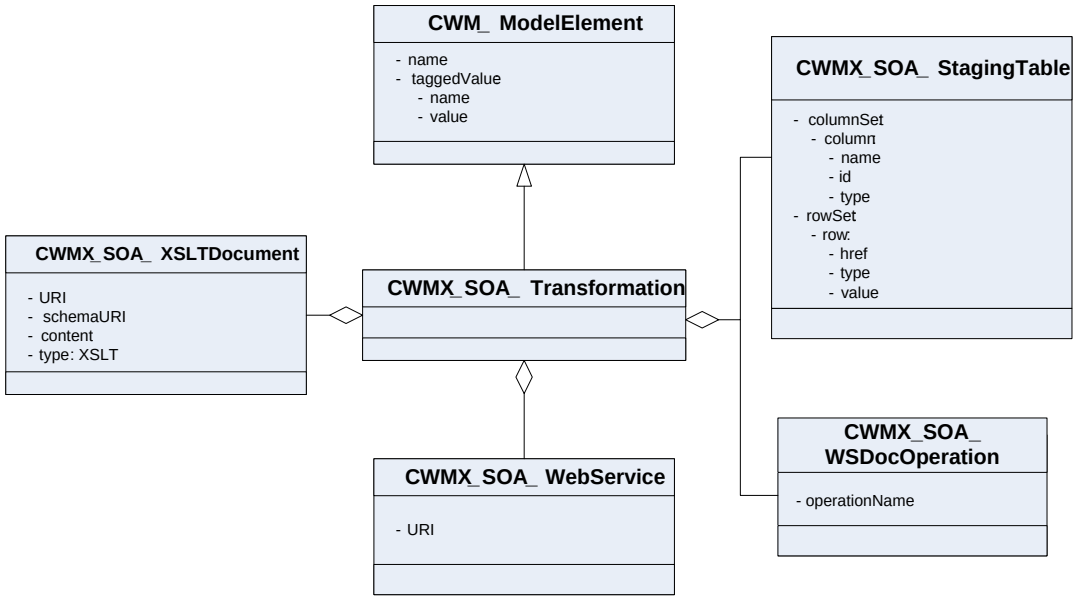


Slika 5-14 - Dijagram elementa CWMX_SOA_Extraction

Ovdje se važno prisjetiti činjenice da element *CWM_ModelElement* omogućuje proširenja pomoću elementa *CWM_TaggedValue*. Usluzi weba koja djeluje samostalno moguće je dodatne parametre poslati preko toga elementa kako bi se izbjeglo korištenje *CWMX_SOA_Expression* u svrhe za koje on nije namijenjen.

Sljedeći element je element transformacije *CWMX_SOA_Transformation*. On također pretpostavlja korištenje zasebne Usluge weba koja može biti temeljena na metapodacima ili funkcionirati samostalno. Element transformacije pored Usluge weba koja se mora pozvati sadrži ulazni međuspremnik podataka, izlazni međuspremnik te izraz koji opisuje način transformacije u zadanom jeziku. Pošto se transformacija često svodi na manipulaciju XML-ovskih podataka, element sadrži i opcionalni XSLT-ovski predložak koji se primjenjuje nad podacima. Ako transformacija potrebuje dodatne podatke, preporučeno ih je postaviti u element *CWM_TaggedValue.*

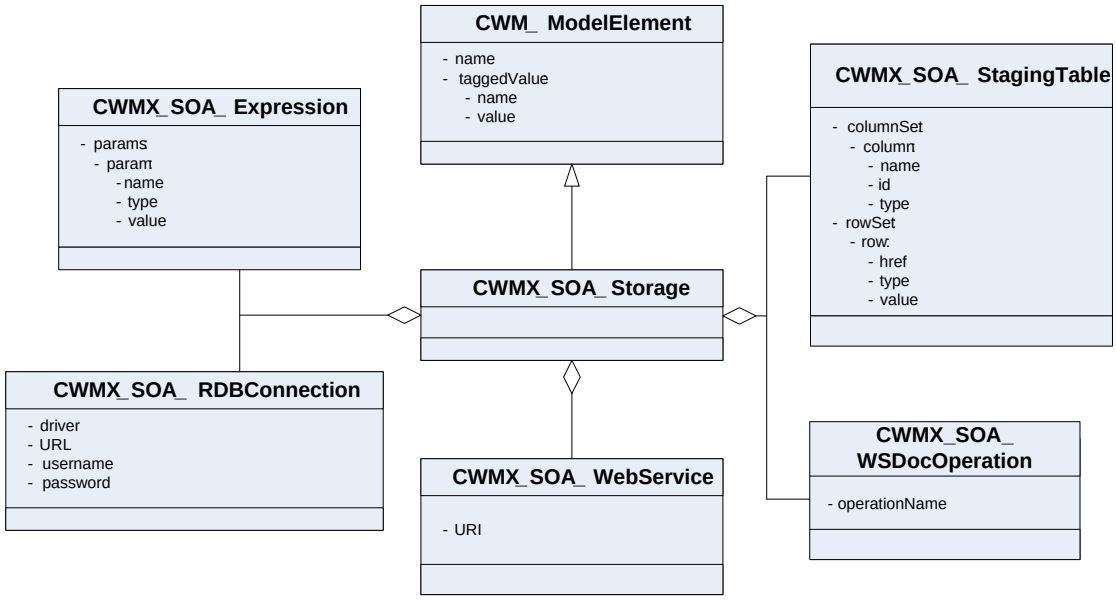
Slika 5-15 prikazuje UML-ovski klasni dijagram elementa *CWMX_SOA_Transformation.*



Slika 5-15 - Dijagram elementa CWMX_SOA_Transformation

Konačno, element pohrane – *CWMX_SOA_Storage* služi za pohranjivanje podataka u stvarnovremensku particiju. On je po svojem opisu vrlo sličan elementu prikupljanja – zahtjeva izraz, opis Usluge weba te međuspremnik podataka. Razlika je samo u tome što međuspremnik kojeg prima nije prazan već napunjen podacima transformiranim u određeni oblik a Usluga weba ne vraća međuspremnik već samo informaciju o uspješnom upisu.

Slika 5-16 prikazuje UML-ovski dijagram elementa *CWMX_SOA_Storage*.



Slika 5-16 - Dijagram elementa CWMX_SOA_Storage

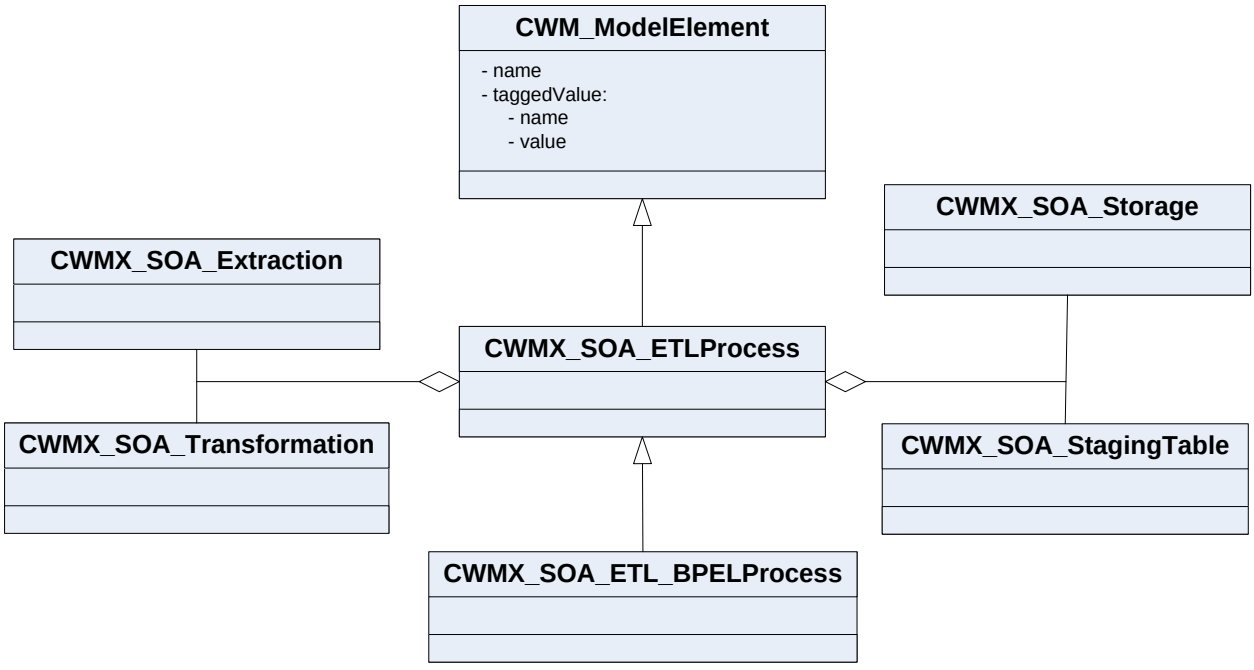
Ova tri elementa čine glavne izgradbene blokove ETL-ovskog procesa. Programski entiteti koji pristupaju repozitoriju metapodataka mogu iz ovih elemenata saznati puno informacija o detaljima samog procesa tj. toku podataka od izvora prema stvarnovremenskoj particiji. Transparentnost procesa je to veća što je veća zastupljenost Usluga weba temeljenih na metapodacima.

Nedostaje samo element koji će logički povezati elemente u konkretni ETL-ovski proces, tj. koji će pružiti opis procesa kao niza poziva Usluga weba čiji je konačni rezultat premještanje podataka u stvarnovremensku particiju.

5.3.7 Element opisa ETL-ovskog procesa

Kao što je već navedeno, elementi opisani u prošlom poglavlju služe kao metapodaci o fazama ETL-ovskog procesa (prikupljanje, transformacija, pohrana) ali i kao predlošci za XML-ovske poruke koje će se razmjenjivati u SOA ETL sustavu. Analogno tome, element opisa cjelokupnog ETL-ovskog procesa osim kao logičko grupiranje elemenata iz prošlog poglavlja služit će i kao glavna referenca na te elemente koju će SOA ETL dohvaćati. Drugim riječima, element metapodataka procesa bit će glavni izvor metapodataka koje će koristiti uslužno orijentirani ETL-ovski proces, tj. one Usluge weba koje svoju funkcionalnost zasnivaju na metapodacima.

Element je nazvan *CWMX_SOA_ETLProcess* a njegov UML-ovski klasni dijagram (zajedno s elementom *CWMX_SOA_ETL_BPELProcess* koji je objašnjen dalje u tekstu) može se vidjeti na slici (**Slika 5-17**).



Slika 5-17 - Dijagram elemenata CWMX_SOA_ETLProcess i CWMX_SOA_ETL_BPELProcess

Ako se radi o poslovnom procesu zasnovanom na Uslugama weba, pretpostavlja se da će element *CWMX_SOA_ETLProcess* biti zasnovan na jeziku WS-BPEL (*Web Services – Business Process Execution Language*). Pošto je za sada WS-BPEL glavna implementacijska tehnologija unutar sustava SOA ETL izvedeni element *CWMX_SOA_ETLProcess* nasljeđuje se elementom *CWMX_SOA_ETL_BPELProcess* u koji će sadržavati i reference na njegov BPEL-ovski opis i implementacijsku Uslugu weba (koji se također nalaze u registru metapodataka).

Derivacija ovih elemenata izvest će se postavljanjem atributa tipa XML-ovskog dokumenta kroz odgovarajaću predefiniranu konstantu. Definicijom zasebnog kompleksnog tipa za svaki od ovih elemenata omogućuje laku nadogradnju u slučaju ukazane potrebe tokom korištenja sustava.

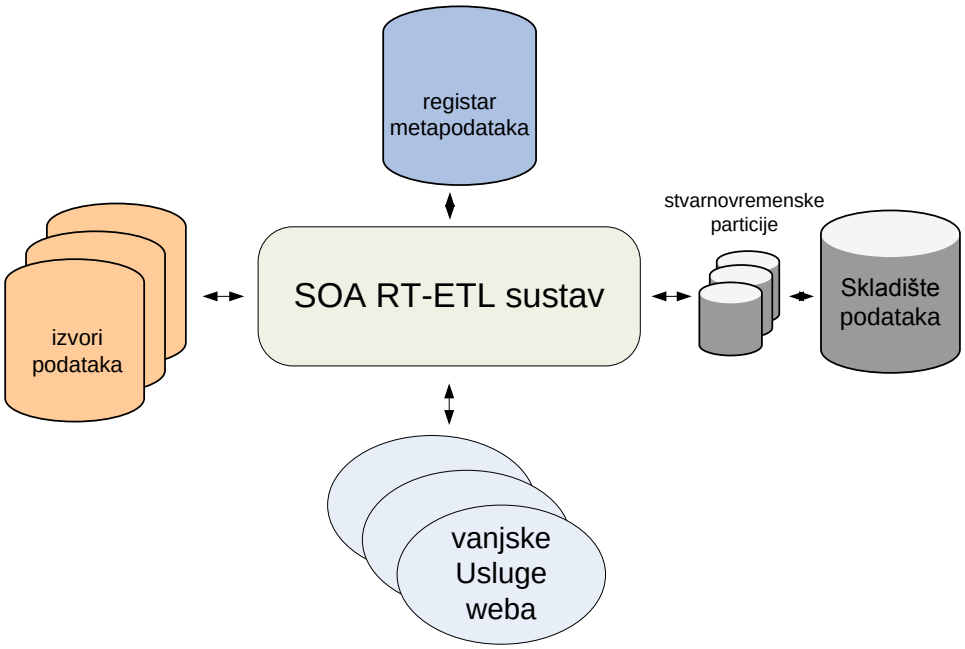
Ono što treba napomenuti jest da se ETL-ovski proces ne mora sastojati od jednog poziva prema izvoru te jedne pohrane podataka. Realni poslovni scenarij uključuje prikupljanje podataka iz više izvora te njihovo slaganje i kombiniranje kako bi se došlo do pripremljenih podataka za stvarnovremensko skladište. Ovakva manipulacija najčešće se izvodi unutar samog BPEL-ovskog procesa te se neće posebno naznačiti pomoću metapodataka. Glavni razlog tome je taj što je i sam BPEL-ovski proces Usluga weba, te bi se on istovremeno morao modelirati kao "transformacija" podataka i u metapodatkovnom opisu zapravo pozivati samog sebe. Kako bi se izbjegli problemi ovakvih rekurzivnih poziva u trenutnoj inačici ovog elementa navodi se samo komunikacija s vanjskim Uslugama weba, a sve interne manipulacije mogu se saznati iz samog BPEL-ovskog procesa tj. na metarazini M1.

U idućem poglavlju bit će riječi o implementaciji SOA RT-ETL sustava te njegovom međudjelovanju s metapodatkovnim elementima tj. njihovim XML-ovskim reprezentacijama zasnovanim na formalizaciji kroz XML-ovske sheme izvedenih u ovom poglavlju čiji se ispisi mogu naći u Prilogu 1**.**

6 Specifikacija i implementacija uslužno orijentiranog sustava za stvarnovremensko skladištenje pogonjenog metapodacima

Poglavlje 5 pružilo je temeljni metamodel za metapodatke čija je glavna primjena prikupljanje i interpretacija od strane sustava SOA za stvarnovremenski ETL (skraćeno – SOA RT-ETL sustav). U ovom poglavlju izvršit će se specifikacija glavnih komponenata toga sustava te će se na studijskom primjeru prikazati primjer konkretne implementacije istog.

Slika 6-1 prikazuje SOA RT-ETL sustav u odnosu na vanjske komponente s kojima komunicira.



Slika 6-1- Vanjski partneri SOA RT-ETL sustava

Na slici se vidi kako je glavna uloga SOA RT-ETL sustava integracija s vanjskim partnerima. Postoji četiri glavne grupe vanjskih komponenata koje SOA RT-ETL sustav povezuje:

- vanjski izvori podataka
- registar metapodataka
- „vanjske“ Usluge weba (usluge koje nisu dio SOA RT-ETL sustava)
- skladište podataka (tj. stvarnovremenske particije)

Ono što je važno uočiti jest činjenica da registar metapodataka sadržava metapodatke o *svim* ovim vanjskim sudionicima, te da SOA RT-ETL sustav potencijalno može u komunikaciji s njim saznati sve o svojoj okolini te joj se dinamički prilagoditi. U idealnom slučaju, sustav bi bio implementiran potpuno generički tj. sve usluge unutar njega bile bi potpuno pogonjene metapodacima čime bi se postigla maksimalna razina potpune iskoristivosti sustava. No – kako će biti pokazano u narednim poglavljima -

tehnološke restrikcije, ograničenost metamodela i nepostojanje jezika za općenito opisivanje transformacijskih izraza još uvijek predstavljaju ozbiljne prepreke konačnom cilju. SOA RT-ETL sustav prikazan tokom ovog rada će u najvećoj mogućoj mjeri iskoristiti potencijal definiranog metamodela te shodno tome izvesti generičku specifikaciju onih usluga koje su obzirom na korišteni metamodel podobne za takvu izvedbu.

Imajući to u vidu, SOA RT-ETL sustav će grupirati Usluge od kojih se sastoji u dvije glavne kategorije:

- usluge pogonjene metapodacima
- samostalne usluge

Usluge pogonjene metapodacima bit će izvedene po principu *pomoćnih usluga* (engl. *utility services*) kao što je spomenuto u poglavlju 1.7. Te usluge imat će generičku izvedbu te će analizom primljenih metapodataka izvršavati svoju funkciju.

6.1 Prikupljanje podataka

Uloga usluga za prikupljanje podataka je uspostavljanje komunikacije s izvorima podataka, prikupljanje skupa podataka pomoću izvršavanja određenog upita/izraza nad izvorima te elementarna transformacija prikupljenih podataka u XML-ovski oblik kojeg će koristiti preostali dio SOA RT-ETL procesa.

Usluge se specificiraju pomoću apstraktnog WSDL-ovskog dokumenta, tj. WSDL-ovskog dokumenta koji nije povezan s nekom konkretnom pristupnom točkom već samo definira sučelje usluge, ulazne i izlazne parametre. U nastavku će se definirati nekoliko vrsta usluga za prikupljanje podataka no svima njima je zajednički isti WSDL-ovski opis kojeg prikazuje **Slika 6-2**.

Prikazani WSDL-ovski dokument je *apstraktan*, u smislu da ne povezuje opis usluge s konkretnom komunikacijskom točkom niti određenim protokolom, ali točno definira tip usluge, operacije i ulazno-izlazne parametre.

Iz dokumenta se mogu pročitati zajednička svojstva svih usluga za prikupljanje podataka koje su dio SOA RT-ETL sustava:

- radi se o dokumentno-orijentiranim Uslugama weba
- prepoznaju elemente XML-ovske sheme *CWMX_SOA.xsd,* što je dokument koji uključuje sve elemente metamodela razvijene u poglavlju 5.3 a čije sheme se mogu vidjeti u Prilogu 1**.**
- sadržavaju operaciju imenovanu *ExtractData*
- ulazni parametar operacije je XML-ovski dokument koji odgovara elementu *Extraction* metamodela CWMX_SOA
- izlazni parametar operacije je XML-ovski dokument koji odgovara elementu *StagingTable* metamodela CWMX_SOA

Drugim riječima, da bi se neka Usluga weba mogla koristiti kao usluga za prikupljanje podataka unutar sustava SOA RT-ETL, onda mora implementirati operaciju *ExtractData* koja će primiti metapodatke koji opisuju proces prikupljanja a vratiti prikupljene podatke u XML-ovskom obliku kojeg također diktira dani metamodel.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="ExtractionServiceWSDL"
      targetNamespace=http://j2ee.netbeans.org/wsdl/ExtractionServiceWSDL
      xmlns=http://schemas.xmlsoap.org/wsdl/
      xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:tns="http://j2ee.netbeans.org/wsdl/ExtractionServiceWSDL"
      xmlns:ns="http://xml.netbeans.org/schema/CWMX_SOA"

    <types>
      <xsd:schema
      targetNamespace="http://SOA_RT_ETL/wsdl/ExtractionServiceWSDL">
         <xsd:import namespace="http://xml.netbeans.org/schema/CWMX_SOA"
                        schemaLocation="CWMX_SOA.xsd"/>
      </xsd:schema>
    </types>

    <message name="ExtractDataRequest">
      <part name="part1" element="ns:CWMX_SOA_Extraction"/>
    </message>

    <message name="ExtractDataResponse">
      <part name="part1" element="ns:CWMX_SOA_StagingTable"/>
    </message>

    <portType name="ExtractionServiceWSDLPortType">
      <operation name="ExtractData">
            <input name="input1" message="tns:ExtractDataRequest"/>
            <output name="output1" message="tns:ExtractDataResponse"/>
      </operation>
    </portType>

</definitions>
```

Slika 6-2 - WSDL-ovski dokument generičke usluge za prikupljanje podataka

Apstraktni WSDL-ovski dokument ne diktira implementaciju same usluge. U općenitom smislu svaka usluga koja sadrži gore navedena svojstva može se koristiti kao dio SOA RT-ETL sustava. No kako bi se konkretizirao sustav te olakšala implementacija predviđena su tri tipa usluga za prikupljanje:

- generička usluga pogonjena metapodacima za prikupljanje podataka iz relacijskih baza podataka
- specijalizirana usluga pogonjena metapodacima za prikupljanje podataka iz izvora za kojeg je usluga prilagođena
- posrednička usluga za prevođenje i prosljeđivanje metapodataka usluzi koja će izvršiti prikupljanje podataka

U nastavku će se detaljnije objasniti svaki od ovih tipova.

6.1.1 Generička usluga za prikupljanje podataka pogonjena metapodacima

Generička usluga za prikupljanje pogonjena metapodacima mora biti izgrađena po principu pomoćne usluge (engl. *utility service*).

Slika 6-3 prikazuje primjer XML-ovskog dokumenta kojeg ovakva usluga prima kao ulazni parametar.

```
<?xml version="1.0" encoding="UTF-8"?>

<CWMX_SOA_Extraction xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      id="1423">

      <CWMX_SOA_Expression>
            <body>SELECT stavka, proizvod, cijena, kupac
                  FROM prodaja
                  WHERE datum_prodaje=[31]
                  AND vrijeme_prodaje=[32]>
            </body>
            <language>SQL99</language>/>
            <params>
             <param type="date" id="31" value="30-may-2009"
                  name="datum_prodaje"/>
             <param type="date" id="32" value="10:01:12am"
                  name="vrijeme_prodaje"/>
            </params>
      </CWMX_SOA_Expression>

      <CWMX_SOA_RDBConnection>
            <driverName>oracle.jdbc.driverOracleDriver</driverName>
            <url>jdbc:oracle:thin:hr/hr@salesserver.sales.hr:1521/ORCL</url>
            <username>realtimeETL</username>/>
            <password>fhagyrazlk</password>/>
      </CWMX_SOA_RDBConnection>

      <output id="76">
            <CWMX_SOA_StagingTable id="76">
            <ColumnSet id="153">
                  <Column type="xs:string" id="175" name="stavka"/>
                  <Column type="xs:string" id="176" name="proizvod"/>
                  <Column type="xs:string" id="177" name="cijena"/>
                  <Column type="xs:string" id="178" name="kupac"/>
            </ColumnSet>
            <CWMX_SOA_StagingTable id="76">
      </output>
</CWMX_SOA_Extraction>
```

Slika 6-3 - Primjer ulaznog XML-ovskog dokumenta za prikupljanje podataka

XML-ovski dokument kojeg usluga prima kao ulazni parametar a koji sadrži metapodatke o prikupljanju podataka pruža sljedeće informacije:

- lokaciju spremišta podataka kojem usluga mora pristupiti
- autorizacijsku informaciju za pristup spremištu
- izraz za prikupljanje podataka zajedno s oznakom jezika u kojem je izraz napisan
- spremnik za prikupljene podatke u obliku XML-ovske reprezentacije relacijskih tablica

Iako specifikacija XMI nalaže mogućnost jednoznačne identifikacije svakog metapodatkovnog elementa, na slici su dani samo identifikatori onih elemenata koje je nužno identificirati dalje tokom procesa. Najvažnija je mogućnost identifikacije izvora podataka, kako zbog daljnje koordinacije procesa i manipulacije nad primljenim podacima, tako i zbog povezivanja stupaca i samih podataka unutar XML-ovskog dokumenta pošto su oni ipak pohranjeni u serijaliziranom obliku.

Slika 6-4 prikazuje primjer izlaznog dokumenta.

```
<?xml version="1.0" encoding="UTF-8"?>

<CWMX_SOA_ExtractionResult xmlns:xsi=http://www.w3.org/2001/XMLSchema-
instance id="1423">

      <CWMX_SOA_StagingTable id="76">
            <ColumnSet id="153">
                  <Column type="xs:string" id="175" name="stavka"/>
                  <Column type="xs:string" id="176" name="proizvod"/>
                  <Column type="xs:string" id="177" name="cijena"/>
                  <Column type="xs:string" id="178" name="kupac"/>
            </ColumnSet>

            <RowSet href="153">
                  <Row href="175" value="11322751"/>
                  <Row href="176" value="sapun"/>
                  <Row href="177" value="8,69"/>
                  <Row href="178" value="134177621"/>
            </RowSet>

            <RowSet href="153">
                  <Row href="175" value="11376512"/>
                  <Row href="176" value="gel za kosu"/>
                  <Row href="177" value="15,78"/>
                  <Row href="178" value="134177621"/>
            </RowSet>
      </CWMX_SOA_StagingTable>

</CWMX_SOA_ExtractionResult>
```

Slika 6-4 - Primjer izlaznog XML-ovskog dokumenta za prikupljanje podataka

U primjeru je prikazan ulazni dokument koji pretpostavlja spajanje na relacijsku bazu *Oracle*. Na koje će se sve izvore podataka usluga moći spajati te u kojoj će mjeri biti generička ovisi o samoj implementaciji usluge. Izvori ne moraju biti ograničeni samo na relacijske baze podataka – usluga može pristupati i podatkovnim sistemima ili sustavima koji podatke organiziraju u zapise. Naravno, softverski inženjer koji se bavi konkretnom implementacijom mora uzeti u obzir sve moguće slučajeve te za svaki od njih unijeti potrebne pomoćne klase te prevoditeljsku logiku koja će se pomoću primljenih metapodataka prilagoditi određenom izvoru, izvršiti izraz kako bi prikupio podatke te ih transformirao u određeni oblik.

Najveća prednost ovakve izvedbe usluge jest njena ponovna iskoristivost. Usluga nije ograničena na jedan konkretan izvor pa čak niti na tip izvora. Isto tako, ako dođe do izmjene u ETL-ovskom procesu usluga se dinamički prilagođava novim uvjetima bez potrebe za rekonfiguracijom izvornog koda i ponovnim prevođenjem. Generička priroda usluge omogućuje njezino korištenje i od strane drugih poslovnih procesa kojima je potreban pristup određenim izvorima podataka – adekvatno oblikovana ulazna poruka će apstrahirati protokole pristupa te tako potencijalno olakšati integraciju i drugih poslovnih aplikacija koje nisu uključene u sustav skladištenja.

Nedostatak ovakve izvedbe jest prvenstveno kompleksnost implementacije same usluge koja mora predvidjeti i biti prilagođena velikom broju scenarija prikupljanja podataka koji se mogu pojaviti u nekom informacijskom sustavu. Ako sustav ne uključuje veliki broj različitih tipova izvora onda to i ne mora predstavljati veliki problem no porastom tog broja raste i složenost implementacije. To se može riješiti i izvedbom usluge tako da ona igra posredničku ulogu, tj. ona interpretira metapodatke te ih prosljeđuje uslugama koje su posebno prilagođene određenom izvoru podataka. Ovakav način zapravo predstavlja generičku aproksimaciju pristupa drugom tipu usluge za prikupljanje koja će biti objašnjena u poglavlju 6.1.2.

6.1.2 Specijalizirana usluga za prikupljanje pogonjena metapodacima

U slučaju da se generička usluga pokaže presloženom za implementaciju, ili ako je pristup podacima izveden tako da se ne može jednostavno opisati kroz prikazani metamodel, onda se usluge mogu izvesti tako da budu na određeni način specijalizirane za prikupljanje iz određenog izvora. Stupanj specijalizacije usluge ovisi o implementaciji – usluga može biti prilagođena određenom tipu izvora podataka ali još uvijek dozvoliti fleksibilnost kod autorizacije pristupa i izraza koji će izvršavati. Isto tako, usluga može biti izvedena tako da provodi strogo specijalizirano prikupljanje podataka uz čvrsto definirani izvor i izraz za prikupljanje te kao ulazni parametar eventualno potrebuje inicijalizacijske parametre koji će pomoći pri identifikaciji informacija koje se moraju prikupiti.

Ono što je bitno napomenuti jest da se ulazni dokument ove usluge ne razlikuje bitno od dokumenta kojeg prikazuje **Slika 6-3**. On prati istu shemu koju definira izvedeni metamodel te se jednostavno izabire onaj podskup opcionalnih elemenata koji će omogućiti prikupljanje podataka. Izlazni dokument je identičan onome kojeg prikazuje **Slika 6-4** – serijalizirani prikaz tablice prikupljenih podataka. Ovakav izlazni dokument davat će gotovo sve usluge koje su dio SOA RT-ETL sustava.

6.1.3 Posrednička usluga za prikupljanje podataka

Principi uslužno orijentiranih arhitektura uključuju i komunikaciju s Uslugama weba koje se nalaze izvan granica određenog informacijskog sustava. SOA RT-ETL sustav tako uključuje mogućnost prikupljanja podataka uz pomoć Usluga weba koje same za sebe nisu dio SOA RT-ETL sustava. Takve usluge naravno ne prate definirani metamodel te koriste vlastite ulazne i izlazne parametre.

Posrednička usluga za prikupljanje podataka ima zadaću prevesti ulazne parametre nastale prema metamodelu u parametre koje očekuje „vanjska“ usluga, proslijediti zahtjev, primiti odgovor te ga konačno preoblikovati u oblik koji se koristi kroz ostatak procesa (tj. u dokument kakvog prikazuje **Slika 6-4**).

6.2 Transformacija podataka

Usluge za prikupljanje podataka osim komunikacije s izvorima provode i elementarnu transformaciju podataka u XML-ovski spremnik definiran elementom *CWMX_SOA_StagingTable*. No prije spremanja u stvarnovremensku particiju podaci najčešće moraju proći kroz dodatne transformacije kako bi na kraju poprimili oblik očekivan od strane poslovnih aplikacije koje za svoje analize koriste skladište podataka. Usluge za transformaciju podataka preuzimaju prikupljene podatke i provode ove transformacije.

Sve transformacijske usluge SOA RT-ETL sustava imaju sljedeće karakteristike:

- radi se o dokumentno-orijentiranim Uslugama weba
- prepoznaju elemente XML-ovske sheme *CWMX_SOA.xsd,* što je dokument koji uključuje sve elemente metamodela razvijene u poglavlju 5.3 a čije sheme se mogu vidjeti u Prilogu 1**.**
- sadržavaju operaciju imenovanu *TransformData*
- ulazni parametar operacije je XML-ovski dokument koji odgovara elementu *Transformation* metamodela CWMX_SOA
- izlazni parametar operacije je XML-ovski dokument koji odgovara elementu *StagingTable* metamodela CWMX_SOA

Za razliku od klasičnih ETL-ovskih sustava koji podatke obrađuju uz pomoć relacijskih tablica ili posebnih tekstualnih datoteka, transformacijske usluge SOA RT-ETL sustava manipuliraju isključivo nad XML-ovskim podacima. Pošto je XML-ovska reprezentacija u pravilu opširnija s velikom količinom redundantne informacije opravdano je postaviti pitanje da li će ovakav način usporiti sam proces transformacije podataka, što je pogotovo bitno za stvarnovremenski sustav u kojem je često ključno minimalizirati latenciju. No mora se uzeti u obzir dvije činjenice:

- stvarnovremenski ETL sustav ne transformira velike količine podataka već male količine koje stižu u kratkim vremenskim razmacima
- sustav SOA implementiran uz pomoć Usluga weba nužno se svodi na upravljanje XML-ovskim podacima tako da se redundancija koju XML kao norma uvodi u sustav ne može izbjeći

Sve transformacijske usluge unutar sustava SOA RT-ETL slijede apstraktni WSDL-ovski dokument kojeg prikazuje **Slika 6-5**.

Kao što je rečeno, sve transformacije se rade nad elementom pripremne tablice - *CWMX_SOA_StagingTable*, tj. nad elementom koji uz pomoć elemenata *„ColumnSet“* i *„RowSet“* definira stupce te podatke u njima. Shodno tome predviđeno je tri tipa transformacija:

- transformacija strukture
- transformacija sadržaja
- složena transformacija

Transformacija strukture će presložiti informaciju iz jedne pripremne tablice u drugu bez izmjene sadržaja tj. podataka koji se nalaze u pojedinim stupcima. *Transformacija sadržaja* ostavit će strukturu „tablice“ istom ali će izmijeniti tj. prilagoditi sadržaj određenih stupaca. *Složena transformacija* provest će obje transformacije tj. rezultat transformacije bit će nova tablica s novim sadržajem unutar nje.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TransformationServiceWSDL"
      targetNamespace=http://j2ee.netbeans.org/wsdl/ExtractionServiceWSDL
      xmlns=http://schemas.xmlsoap.org/wsdl/
      xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:tns="http://j2ee.netbeans.org/wsdl/TransformationServiceWSDL"
      xmlns:ns="http://xml.netbeans.org/schema/CWMX_SOA"

    <types>
      <xsd:schema
      targetNamespace="http://SOA_RT_ETL/wsdl/TransformationServiceWSDL">
         <xsd:import namespace="http://xml.netbeans.org/schema/CWMX_SOA"
                        schemaLocation="CWMX_SOA.xsd"/></xsd:schema>
    </types>

    <message name="TransformDataRequest">
      <part name="part1" element="ns:CWMX_SOA_Transformation"/>
    </message>

    <message name="TransformDataResponse">
      <part name="part1" element="ns:CWMX_SOA_StagingTable"/>
    </message>

    <portType name="TransformationServiceWSDLPortType">
      <operation name="TransformData">
            <input name="input1" message="tns:TransformDataRequest"/>
            <output name="output1" message="tns:TransformDataResponse"/>
      </operation>
    </portType>
</definitions>
```

Slika 6-5 - WSDL-ovski dokument generičke usluge za transformaciju podataka

Proširena norma XML uključuje i specifikaciju XSLT (engl. *eXtensible StyleSheet Transformation)* koja je kratko spomenuta u poglavlju 2.2.3. Svrha ove specifikacije jest definiranje predložaka pomoću kojih se XML-ovski dokumenti mogu transformirati u druge XML-ovske dokumenata. XSLT-ovski predlošci predviđeni su za glavnu temeljnu tehnologiju za definiciju transformacija pogonjenih metapodacima koje će koristiti SOA RT-ETL sustav.

Analogno uslugama za prikupljanje podataka predviđena su tri tipa usluga za transformaciju:

- generička usluga pogonjena metapodacima za transformaciju podataka pomoću XSLT-ovskih predložaka
- specijalizirana transformacijska usluga pogonjena metapodacima
- posrednička usluga za prevođenje i prosljeđivanje metapodataka usluzi koja će izvršiti transformaciju podataka

U nastavku su dana kratka objašnjenja ovih tipova transformacijskih usluga.

6.2.1 Generička transformacijska usluga pogonjena metapodacima

Poput generičke usluge za prikupljanje podataka, ova usluga također je izvedena prema principima „pomoćne“ usluge, tj. usluge čija je glavna karakteristika ponovna iskoristivost i univerzalnost. Razlika između usluge za prikupljanje i transformacijske jest ta što je puno teže eksplicitno pomoću metapodataka iskazati transformaciju nego opisati izvor podataka. Još uvijek ne postoje univerzalni opisi transformacija niti modeli koji bi na osnovu danih metapodataka mogli samostalno preinačiti dokument kako bi odgovarao predviđenom rezultantnom obliku. Neka teoretska istraživanja i strogo specijalizirane implementacije postoje – kao što je dano u [31] – ali za sada univerzalnih implementacija nema.

Stoga je u inicijalnoj inačici SOA RT-ETL sustava usluga generičke transformacije pogonjene metapodacima svedena na uslugu koja kao metapodatke prima u obliku XSLT-ovskih predložaka. XSLT je danas glavni način za enkapsulaciju opisa transformacije XML-ovskih dokumenata pomoću predložaka koji su također XML-ovskom obliku te kao takav je vrlo podoban za ovu svrhu.

Slika 6-6 prikazuje skraćeni prikaz ulaznog XML-ovskog dokumenta kojeg prima generička transformacijska usluga pogonjena metapodacima.

Izlazni dokument identičan je dokumentu kojeg prikazuje **Slika 6-4**, tj. radi se o elementu *CWMX_SOA_StagingTable* koji pohranjuje transformirane informacije.

Usluga izvedena na ovaj način primjenjiva je i u svim drugim poslovnim procesima koji trebaju izvršiti XSLT-ovsku transformaciju tako da je njezina razina ponovne iskoristivosti vrlo visoka. S druge strane, ekspresivnost XSLT-a nije velika tako da se očekuje da će u budućim inačicama SOA RT-ETL sustava generička usluga transformacije uključivati i dodatne jezike za opis transformacije sa širim spektrom mogućnosti od onih koje nudi specifikacija XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>

<CWMX_SOA_Transformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" id="1423">


            <XSLTTemplate>
                  <xsl:stylesheet version="1.0"
                      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                  <xsl:template match="/">
                  ...
            </XSLTTemplate>
            <input id="76">
                  <CWMX_SOA_StagingTable id="76">
                  <ColumnSet id="153">
                        <Column type="xs:string" id="175" name="stavka"/>
                        ...
                  </ColumnSet>
                  <RowSet>...</RowSet>
                  </CWMX_SOA_StagingTable id="76">
            </input>
            <output id="77">
                  <CWMX_SOA_StagingTable id="76">
                  <ColumnSet id="154">
                        <Column type="xs:string" id="192" name="id_st"/>
                        ...
                  </ColumnSet>
                  </CWMX_SOA_StagingTable>
            </output>
</CWMX_SOA_Transformation>
```

Slika 6-6 – Primjer ulaznog dokumenta generičke usluge za transformaciju podataka

6.2.2 Specijalizirana transformacijska usluga

Kao što je rečeno, mogućnosti XSLT-a su ograničene. Kroz XSLT se relativno jednostavno mogu opisati transformacije strukture ali transformacije sadržaja i složene transformacije mogu predstavljati veći problem ako ih se treba izvesti kroz jedinstveni XSLT-ovski predložak. Dodatno, usluga koja izvršava XSLT-ovsku transformaciju može oduzimati previše procesorskog vremena što se u stvarnovremenskom ETL sustavu često ne može tolerirati.

Specijalizirana transformacijska usluga izvedena je za točno određeni tip transformacije. WSDL-ovski opis usluge je identičan onom za generičku transformaciju a ulazni XML-ovski parametar je isti osim što ne sadrži XSLT-ovski predložak. Rezultat transformacije je također identičan. Konkretan tip transformacije je ovisan o implementaciji; u pravilu nema ograničenja nad transformacijskim koracima koji se mogu izvoditi pošto implementacija usluge može koristiti sve programske konstrukte koji su softverskom inženjeru na raspolaganju.

Nedostatak ovakve usluge je naravno njezina ograničena primjena te nemogućnost prilagodbe kod eventualnih izmjena ETL procesa. Specijalizirana usluga koristi sve prednosti arhitekture SOA ali iskoristivost povezanosti s metapodacima strogo ovisi o konkretnoj implementaciji.

6.2.3 Posrednička usluga za transformaciju podataka

Analogno posredničkim uslugama za prikupljanje podataka, takve usluge za transformaciju zapravo služe kao prevoditelji između SOA RT-ETL sustava i „vanjskih" usluge koje izvode samu transformaciju ali ne prepoznaju XML-ovske elemente koje odgovaraju CWMX_SOA metamodelu. Korištenje vanjskih usluga može biti nužno kada se radi o transformacijama sadržaja kao npr. kad transformacija zahtijeva prevođenje valute prema trenutnom dnevnom tečaju ili kompleksnu kalkulaciju koju izvodi specijalizirana partnerska Usluga weba.

6.3 Spremanje podataka

Usluge za spremanje podataka po svojoj prirodi su vrlo slične uslugama za prikupljanje podataka u smislu da komuniciraju sa spremištima podataka te pomoću danih izraza provode svoju funkcionalnost. No dok usluge za prikupljanje imaju dodatnu zadaću elementarne transformacije prikupljenih podataka u oblik kojeg očekuje i koji se koristi kroz SOA RT-ETL sustav, usluge za spremanje podataka nemaju taj zadatak; spremanje podataka u odgovarajuće spremište – stvarnovremensku particiju skladišta podataka - predstavlja završni korak ETL procesa. Isto tako, dodatni parametri koji se šalju usluzi za prikupljanje pomažu pri identifikaciji podataka unutar izvora koji se prikupljaju; parametri koje prima usluga za spremanje odnose se na podatke sadržane u samoj poruci tj. podatke koji se moraju na adekvatan način pohraniti.

Usluge za pohranu podataka unutar sustava SOA RT-ETL imaju sljedeće karakteristike:

- radi se o dokumentno-orijentiranim Uslugama weba
- prepoznaju elemente XML-ovske sheme *CWMX_SOA.xsd*
- sadržavaju operaciju imenovanu *StoreData*
- ulazni parametar operacije je XML-ovski dokument koji odgovara elementu *Storage* metamodela CWMX_SOA

Sve usluge spremanja podataka slijede WSDL-ovski dokument kojeg prikazuje **Slika 6-7**.

Analogno uslugama za prikupljanje i transformaciju podataka, predviđene su tri okvirne kategorije usluga za spremanje podataka:

- generička usluga pogonjena metapodacima za spremanje podataka
- specijalizirana usluga za spremanje pogonjena metapodacima
- posrednička usluga za prevođenje i prosljeđivanje metapodataka usluzi koja će izvršiti spremanje podataka

U nastavku su dani detalji o ove tri kategorije.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="StorageServiceWSDL"
      targetNamespace=http://j2ee.netbeans.org/wsdl/ExtractionServiceWSDL
      xmlns=http://schemas.xmlsoap.org/wsdl/
      xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:tns="http://j2ee.netbeans.org/wsdl/StorageServiceWSDL"
      xmlns:ns="http://xml.netbeans.org/schema/CWMX_SOA"

    <types>
      <xsd:schema
      targetNamespace="http://SOA_RT_ETL/wsdl/StorageServiceWSDL">
         <xsd:import namespace="http://xml.netbeans.org/schema/CWMX_SOA"
                     schemaLocation="CWMX_SOA.xsd"/>
      </xsd:schema>
    </types>

    <message name="StoreDataRequest">
      <part name="part1" element="ns:CWMX_SOA_Storage"/>
    </message>

    <message name="StoreDataResponse">
      <part name="part1" element="ns:CWMX_SOA_StagingTable"/>
    </message>

    <portType name="StorageServiceWSDLPortType">
      <operation name="StoreData">
            <input name="input1" message="tns:StoreDataRequest"/>
            <output name="output1" message="tns:StoreDataResponse"/>
      </operation>
    </portType>

</definitions>
```

Slika 6-7 - WSDL-ovski dokument generičke usluge za spremanje podataka

6.3.1 Generička usluga za spremanje podataka pogonjena metapodacima

Generička usluga za spremanje podataka očekuje sljedeće metapodatke za izvršavanje svoje funkcije:

- lokaciju spremišta podataka kojem usluga mora pristupiti
- autorizacijsku informaciju za pristup spremištu
- SQL-ovski izraz pomoću kojeg će se izvršiti pohrana podataka

- podatke za spremanje u XML-ovskom obliku

Slika 6-8 prikazuje primjer XML-ovskog dokumenta kojeg ovakva usluga može primiti kao ulazni parametar.

```
<?xml version="1.0" encoding="UTF-8"?>

<CWMX_SOA_Storage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      id="1423">
      <CWMX_SOA_Expression>
            <body>INSERT INTO prodaja_fact_table
                  VALUES [211],[212],[213],[214];
            </body>

            <language>SQL99</language>/>

            <params>
              <param type="CWMX_SOA_StagingTable" href="87"/>
              ...
            </params>
      </CWMX_SOA_Expression>

      <CWMX_SOA_RDBConnection>
            <driverName>oracle.jdbc.driverOracleDriver</driverName>
            ...
      </CWMX_SOA_RDBConnection>
      <input>
      <CWMX_SOA_StagingTable id="87">
            <ColumnSet id="153">
                  <Column type="xs:string" id="211" name="proizvod_id"/>
                  <Column type="xs:string" id="212" name="trgovina_id"/>
                  ...
            </ColumnSet>
            <RowSet id="273">
                  <Row type="xs:string" href="211" value="132"/>
                  <Row type="xs:string" href="212" value="7"/>
                  ...
            </RowSet>
      </CWMX_SOA_StagingTable>

</CWMX_SOA_Storage>
```

Slika 6-8 - Primjer ulaznog XML-ovskog dokumenta za spremanje podataka

Činjenice koje vrijede za generičke usluge prikupljanja podataka uglavnom vrijede i za usluge za spremanje podataka – razina ponovne iskoristivosti je velika i usluga se može koristiti za općenite potrebe pristupa i pohrane podataka u skladište. Ovisno o implementaciji može se izraditi i jedinstvena

usluga koja izvršava i prikupljanje i spremanje podataka pošto dijele veliki dio programske logike. Najveći nedostatak je opet kompleksnost implementacije koja mora predvidjeti širok spektar mogućih tipova spremišta.

6.3.2 Specijalizirana usluga za spremanje podataka pogonjena metapodacima

Za specijaliziranu uslugu spremanja podataka također vrijedi većina činjenica vezanih uz specijaliziranu uslugu prikupljanja podataka koja je kratko obrađena u poglavlju 6.1.2. Razina specijalizacije te stupanj iskoristivosti metapodataka ovise strogo o implementaciji; za SOA RT-ETL sustav bitno je samo da se priklanja danom metamodelu te da koristi WSDL-ovski dokument kojeg prikazuje **Slika 6-7**.

6.3.3 Posrednička usluga za spremanje podataka

Ovdje se opet mogu ponoviti sve činjenice koje vrijede za posredničku uslugu prikupljanja i transformacije podataka koje su dane u poglavljima 6.1.3 i 6.2.3. Posrednička usluga ima prevoditeljsku funkciju tj. priprema podatke koji odgovaraju metamodelu u oblik kojeg očekuje usluga koja će provesti samo spremanje. Razlika od ostalih posredničkih usluga jest što u pravilu usluga za spremanje podataka ne mora vraćati nikakvu vrijednost (osim naravno u slučaju pogreške kada dolazi do pojave iznimke koja će prikazati o kakvoj se pogrešci radi).

6.4 Pristup registru metapodataka

Prethodna poglavlja bavila su se glavnim „poslovnim“ uslugama SOA RT-ETL sustava, tj. uslugama koje obavljaju posao vezan uz sam ETL-ovski proces. Kao što je rečeno, očekuje se da će te usluge u izvjesnoj mjeri koristiti metapodatke sustava kako bi se dinamički prilagođavale svojoj okolini te tako povećale ukupnu ponovnu iskoristivost i fleksibilnost sustava.

Metapodaci koje te usluge primaju kao ulazni parametri nalaze se u registru metapodataka. Izvedba registra metapodataka te upravljanje njim je složen problem o kojem je već bilo riječi u poglavlju 4.1 te o čemu se više informacija može pronaći u [21]. Registri metapodaka mogu imati različita sučelja i protokol pristupa te mogu posluživati podatke u različitim oblicima, ovisno o aplikaciji koja im pristupa, namjeni metapodataka i sl.

Za potrebe SOA RT-ETL sustava pretpostavljen je registar koji je kompatibilan s CWM specifikacijom te koji podržava proširenje CWMX_SOA. Sama implementacija registra te pristupno sučelje simulirano je kroz datotečni sustav te pretragu i dohvat uz pomoć imena datoteke izvedenog po principu *„tip_id.xml“*. Također, pretpostavljena je uporaba tehnologija JMI (engl. *Java Metadata Interchange*) [33] i već spomenute XMI. Pošto je u poglavlju 5 opširno opisan metamodel CWMX_SOA te u Prilogu 1 dana njegova formalizacija kroz XML-ovske sheme, očekivana je mogućnost prikupljanja metapodataka u obliku koji odgovara danim shemama. Kako bi se registru moglo pristupati kroz uslužno-orijentirani sustav, za dohvat metapodatkovnih elemenata koristi se pripadna Uslugu weba.

Specifikacija CWMX_SOA između ostalog sadrži i element pod nazivom *CWMX_SOA_ETLProcess*, čija UML-ovska shema se može vidjeti u poglavlju 5.3.7. Ovaj element je pogodan pošto on zapravo predstavlja spremnik svih metapodataka vezanih uz proces. On konkretno sadrži skup elemenata tipa *CWMX_SOA_Expression, CWMX_SOA_Transformation* i *CWMX_SOA_Storage* čija koordinirana interpretacija omogućuje izvršavanje cijelog procesa.

Usluga za pristup registru metapodacima očekivano omogućuje pristup svim elementima u registru, prvenstveno preko poznatog identifikatora. **Slika 6-9** prikazuje WSDL-ovski prikaz usluge s minimalnom potrebnom funkcionalnosti koju SOA RT-ETL sustav očekuje – usluga koja na zahtjev vraća skup metapodataka vezanih uz ETL-ovski proces koji se mora izvršiti.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="MetadataServiceWSDL"
      targetNamespace=http://j2ee.netbeans.org/wsdl/ExtractionServiceWSDL
      xmlns=http://schemas.xmlsoap.org/wsdl/
      xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
      xmlns:xsd=http://www.w3.org/2001/XMLSchema
      xmlns:tns="http://j2ee.netbeans.org/wsdl/MetadataServiceWSDL"
      xmlns:ns="http://xml.netbeans.org/schema/CWMX_SOA"

    <types>
      <xsd:schema
      targetNamespace="http://SOA_RT_ETL/wsdl/MetadataServiceWSDL">
         <xsd:import namespace="http://xml.netbeans.org/schema/CWMX_SOA"
                     schemaLocation="CWMX_SOA.xsd"/>
      </xsd:schema>
    </types>

    <message name="ETLProcessRequest">
      <part name="part1" element="ns:CWMX_SOA_ModelElement"/>
    </message>

    <message name="ETLProcessResponse">
      <part name="part1" element="ns:CWMX_SOA_ETLProcess"/>
    </message>

    <portType name="MetadataServiceWSDLPortType">
      <operation name="GetETLProcessMetadata">
            <input name="input1" message="tns:ETLProcessRequest"/>
            <output name="output1" message="tns:ETLProcessResponse"/>
      </operation>
    </portType>

</definitions>
```

Slika 6-9 - WSDL-ovski dokument usluge za pristup registru metapodataka

Za ulazni parametar usluge izabran je element *CWMX_SOA_ModelElement*. Ovaj element je podesan iz dva razloga:

- svaki element unutar metapodatkovnog registra izveden je iz te klase tako da se ovakav zahtjev može koristiti kao univerzalni ulazni parametar;
- svi elementi/atributi unutar elementa *CWMX_SOA_ModelElement* su opcionalni tako da je moguće poslati samo podskup elemenata prema kojima usluga može uspješno pronaći metapodatke koje će poslati kao povratnu informaciju; pretpostavlja se da će se u većini slučajeva slati primarni identifikator elementa (atribut „id“ korijenskog elementa).

Usluga pristupa registru metapodataka s potpunom funkcionalnosti jednostavno bi implementirala po jednu operaciju za svaki metapodatkovni element kojem je potrebno pristupiti. Odabir ovakvog tipa usluge koji ima samo jednu operaciju odabran je zbog jednostavnosti – kroz samo jedan poziv vraćaju se svi nužni metapodaci nužni za provedbu procesa. Naravno, ako se u implementaciji sustava ovakav način pokaže neučinkovit tj. ako se prikupljanje metapodataka u manjim koracima smatra pogodnijim, usluga se može implementirati i na taj način, samo je potrebno prema danom predlošku definirati dodatne operacije.

6.5 Komunikacija s vanjskim Uslugama weba

Jedno od važnih svojstava uslužno orijentiranih arhitektura je mogućnost povezivanja sa svim globalno dostupnim uslugama koje slijede dane norme vezane uz tip poruka i komunikacijske protokole. Iako se sustav SOA RT-ETL može izvesti kao samostalan sustav koji sadrži i koristi samo usluge unutar jedinstvenog informacijskog sustava, nije uputno zanemariti potencijal kojeg pruža vanjska funkcionalnost dostupnih Usluga weba koje mogu doprinijeti pripremi podataka u sustav skladištenja. ETL proces bi mogao tako pomoću vanjskih usluga saznati dnevni tečaj stranih valuta, vrijednosti vezane uz vremenske prilike na nekoj lokaciji ili trenutne vrijednosti dionica na nekoj od burzi. Pošto poslovne analize vezane uz podatke spremljene u skladište često trebaju što opširnije i sadržajnije podatke, ovakve informacije dostupne preko vanjskih usluga se mogu pokazati vrlo korisnima (pa čak i biti glavni inicijatori za izvedbu uslužno orijentiranog sustava za dopremanje podataka u skladište).

Komunikacija s vanjskim Uslugama weba nosi sa sobom i određene rizike koji mogu biti vrlo važni, pogotovo za stvarnovremenske sustave skladištenja gdje je minimalizacija vremena latencije često ključna. Najveći problem jest u činjenici da ne postoji mehanizam kontrole kvalitete vanjskih Usluga weba i to ne samo glede točnosti informacija, već i u smislu dostupnosti usluge te vremena procesiranja zahtjeva. Vanjska Usluga weba tako može postati usko grlo i glavni uzrok kašnjenja ili nedolaska podataka u stvarnovremensku particiju, stoga treba biti pažljiv kod modeliranja procesa koji koriste takve usluge te razmotriti opciju naknadnog ažuriranja skladišnog sustava „vanjskim“ informacijama kako bi se minimalizirali utjecali vanjskih čimbenika na odvijanje ETL-ovskog procesa.

Metamodel CWMX_SOA omogućuje zapisivanje metapodataka o svim Uslugama weba, tako da se u registru metapodataka mogu čuvati i informacije vezane uz vanjske Usluge weba koje sustav koristi. Problem samo nastaje kada se kod odabira načina pristupa takvim uslugama. Postoji dva tipa takvog pristupa:

- proces koji direktno pristupa vanjskoj usluzi i prikuplja podatke

- proces koristi posredničku uslugu koja umjesto njega prikuplja potrebne podatke te ih prosljeđuje procesu

Prvi pristup je jednostavniji za implementaciju pošto se komunikacija s vanjskom Uslugom weba može ugraditi u sam proces. Isto tako, jednostavnije je modelirati ulazne i izlazne poruke te upravljati prikupljenom informacijom. S druge strane, ovakav pristup ne omogućuje dinamičku prilagodbu kroz metapodatke, tj. vanjska Usluga weba se modelira kao „partnerska veza“ i ona je fiksirana u toku procesa. Metapodaci ovdje imaju samo klasičnu ulogu opisa podataka o procesu, oni nemaju nikakvu integracijsku funkciju.

Posrednička usluga se može implementirati tako da prosljeđuje pozive vanjskim uslugama te procesu vraća primljene podatke. Prednost ovakvog pristupa je ostanak u okvirima danog metamodela i veća fleksibilnost pošto bi se takva usluga mogla izvesti generički, tj. da tek nakon primitka ulaznih parametara saznaje adresu usluge koju mora pozivati. Ovakav način omogućuje visoku razinu fleksibilnosti i ponovne iskoristivosti ali uz cijene veće kompleksnosti implementacije te teže konfiguracije pošto se ulazni parametri moraju izgrađivati uz pomoć postojećih metapodataka.

Ključni problem postoji u činjenici da WS-BPEL – glavna implementacijska tehnologija za izvedbu poslovnih procesa kroz Usluge weba obrađena u poglavlju 2.6 – ne podržava dinamički izbor Usluga weba već se sve partnerske veze moraju unaprijed odrediti u modelu procesa a njegova ekspresivnost ograničava mogućnosti dinamičkog slaganja poruka. Zbog toga je izvedba „direktnog“ pristupa za sada puno jednostavnija dok se posrednički pristup može prikazati prezahtjevnim a potencijalno i redundantnim pošto nove verzije specifikacije mogu ponuditi i nove mogućnosti koje će ukinuti potrebu za takvim posredničkim pristupom.

6.6 Poslovni proces

Dok se Usluge weba koje će sustav SOA RT-ETL koristiti za provedbu stvarnovremenskog ETL-ovskog procesa mogu relativno egzaktno specificirati, poslovni proces se ne može usko definirati točno određenom specifikacijom. Razlog tome jest prvo u prirodi samog procesa (koji se mora modelirati prema konkretnim uvjetima poslovne okoline) ali i u tehnološkim ograničenjima (tehnologija WS-BPEL ne dozvoljava dinamičko slaganje procesa već zahtijeva točno definirani tok procesa uz fiksirane „partnerske veze“ – usluge s kojima komunicira tj. koje povezuje u orkestrirani poslovni proces).

Informacije koje se mogu prikupiti o karakteristikama poslovnog procesa imaju za to definirani metapodatkovni element *CWMX_SOA_ETLProcess*. Ovaj element može sadržavati sve detalji vezane uz prikupljanje podataka, njihovu transformaciju te konačno pohranu u stvarnovremensku particiju. No stvaranje generičkog procesa koji bi prihvaćao taj element te se dinamički prilagodio njegovim karakteristikama može se izvršiti samo djelomično, korištenjem partnerskih usluga generičke prirode. No svaki poslovni proces imat će svoje vlastite karakteristike koje se teško može obuhvatiti i opisati generičkim podacima – pogotovo takvim koji bi omogućili automatsku prilagodbu. Proces može koristiti različite mehanizme inicijalizacije (npr. kontinuirano postavljanje upita nad transakcijskim sustavom u nekom vremenskom intervalu, vanjski CDC sustav (engl. *Change Data Capture*) koji će pokrenuti poslovni proces i koji može ali ne mora uključivati povratnu vezu i sl.). Proces može zahtijevati složene

manipulacije primljenim podacima koje se ne mogu jednostavnu opisati generičkim izrazima te naknadno interpretirati. Konačno, određivanje stroge strukture procesa umnogome bi otežalo njegovu konkretnu implementaciju pošto bi se softverski inženjeri morali prilagođavati ne samo poslovnoj okolini, već i ograničenjima tako definiranog procesa.

Zbog toga se za poslovne procese koji čini središnji dio sustava SOA RT-ETL mogu dati neke općenite karakteristike, smjernice te norme koje će koristiti, no njihovo modeliranje će se ipak morati prilagoditi svakom posebnom poslovnom slučaju. Neke od karakteristika svakog procesa koji će biti dio SOA RT-ETL dane su u nastavku.

Poslovni proces koji pruža funkcionalnost stvarnovremenskog ETL-a u sklopu SOA RT-ETL sustava mora:

1) Poznavati shemu koja definira metapodatkovne elemente koji su u sklopu ovoga rada grupirani u paketu *CWMX_SOA*
2) Koristiti poruke oblikovane prema navedenoj shemi, poglavito poruke koje se tiču prikupljanja, transformacije i pohrane podataka a koje opisuju elementi *CWMX_SOA_Extraction*, *CWMX_SOA_Transformation* i *CWMX_SOA_Storage*
3) Moći interpretirati metapodatke pohranjene unutar elementa *CWMX_SOA_ETLProcess* tj. moći razložiti ovaj proces na sastavne elemente te ih proslijediti partnerskim uslugama koje će izvršavati poslovne akcije unutar ETL-ovskog procesa
4) U potpunosti provesti sve zadatke zadane definirane unutar navedenog metapodatkovnog elementa

Kao što će se vidjeti iz studijskog primjera SOA RT-ETL sustava, ovi zahtjevi nisu previše restriktivni a dizajn procesa je prilično jednostavan te većinom koristi samo najosnovnije konstrukte koje nudi specifikacija WS-BPEL – poglavito se radi o segmentaciji primljenih poruka i prosljeđivanju na pravu adresu. Uz kvalitetne temelje koje nudi formalizacija kroz XML-ovske sheme te adekvatnu funkcionalnost vanjskih partnera modeliranje poslovnog procesa najčešće predstavlja najjednostavniji element modeliranja implementacije sustava SOA RT-ETL.

6.7 Studijski primjer implementacije SOA RT-ETL sustava

6.7.1 Poslovni scenarij i korištene tehnologije

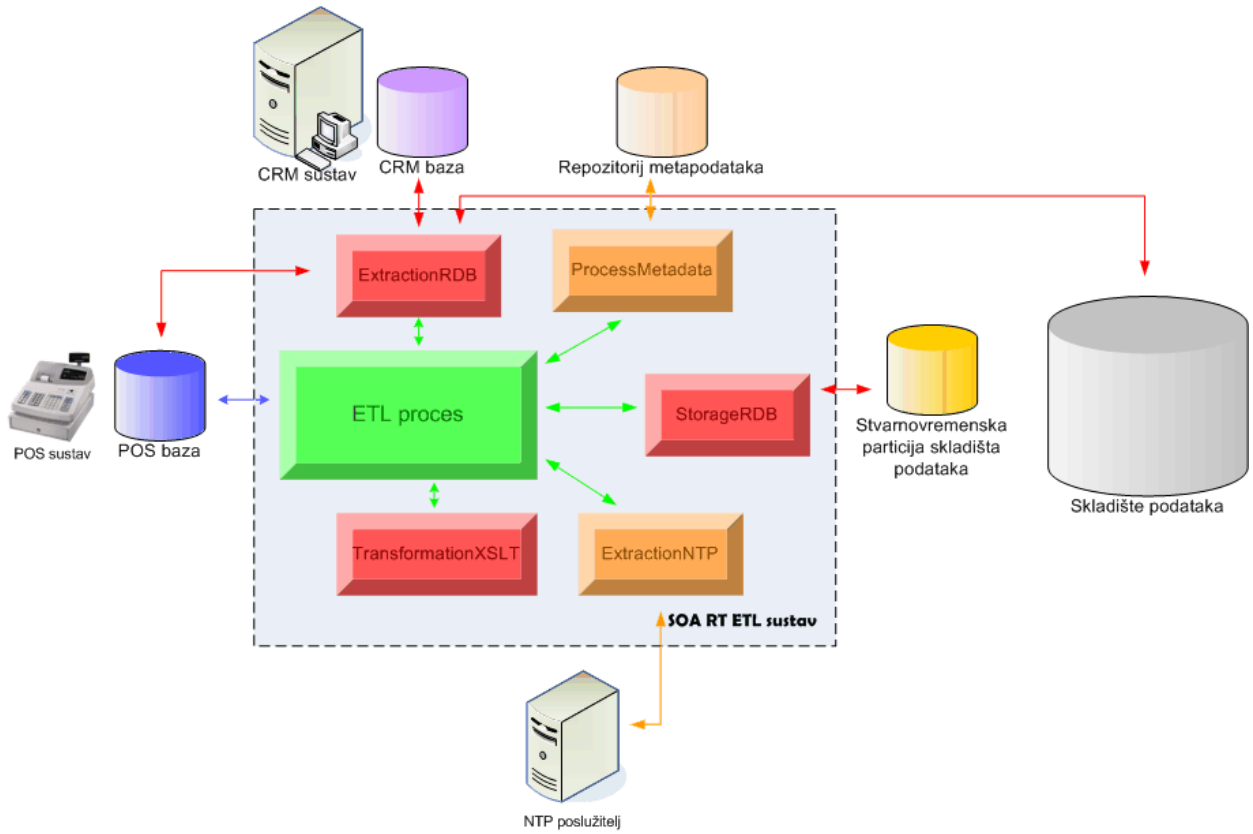
Ideja studijskog primjera jest demonstracija uporabljivosti formalizacija i koncepata razvijenih u prethodnim poglavljima, provjera iskoristivosti implementacijskih tehnologija te izvođenje elementarnih mjerenja učinkovitosti ovakvog sustava za prikupljanje i pripremu podataka u stvarnom vremenu. Studijski primjer pretpostavljat će scenarij proširenja klasičnog skladišta stvarnovremenskom particijom u koju je potrebno u stvarnom vremenu smještati nove činjenice vezane uz definirani poslovni scenarij.

Za izvedbu studijskog primjera koristit će se sljedeće tehnologije:

- *baza i sustav za upravljanje bazama podataka* ***Oracle 10g*** – nad ovom bazom izvest će se dvije zasebne sheme koje će predstavljati transakcijski sustavi i stvarnovremensku particiju

- *aplikacijski poslužitelj* ***Apache Tomcat 6.0*** – Tomcat će se koristiti kao poslužitelj Usluga weba koje koristi ETL-ovski proces u sklopu SOA RT-ETL sustava
- *aplikacijski poslužitelj* ***Glassfish 2.0*** – ovaj poslužitelj bit će zadužen za provođenje samog ETL-ovskog procesa
- *razvojna okolina* ***NetBeans 6-5*** - glavna razvojna okolina za implementaciju SOA RT-ETL sustava
- ***JAX-WS*** *(engl. Java Api for XML Web Services)* – proširenje jezika *Java* za razvoj Usluga weba zasnovan na anotacijama

Poslovni scenarij studijskog primjera uključivat će prikupljanje podataka o izdanom računu iz POS[12] sustava, proširivanje podataka informacijama iz sustava za podršku CRM-u (*engl. Customer Relationship Management* – upravljanje odnosa s klijentima), parcijalnu integraciju s podacima iz skladišta pomoću prikupljanja surogata ključeva, dodavanje vremenske oznake trenutka prikupljanja podataka, prilagodbu podataka činjeničnoj tablici u stvarnovremenskoj particiji te pohranu podataka u istu nakon čega su oni dostupni za analizu. **Slika 6-10** prikazuje elemente takvog poslovnog sustava koji sudjeluju u spomenutom procesu.



Slika 6-10 - Arhitektura poslovnog sustava studijskog primjera

[12] POS (*engl. Point of Sale*) – sustav za obradu transakciju vezanih uz prodaju, najčešće smješte na fizičkoj lokaciji gdje se proces prodaje izvršava.

Komponente unutar pravokutnika omeđenog isprekidanom linijom (tj. Usluge weba koje sačinjavaju SOA RT-ETL sustav) implementirane su u potpunosti. Elementi *izvan* sustava SOA RT-ETL simulirani su u onolikoj mjeri koliko zahtjeva demonstracija funkcionalnosti spomenutog sustava. U nastavku je objašnjena uloga pojedinih segmenata studijskog primjera te su definirane karakteristike implementiranih elemenata.

6.7.2 Implementacija izvora, međuspremišta i odredišta podataka

Studijski primjer sastoji se od nekoliko izvora podataka te jednog odredišta – stvarnovremenske particije. Izvori podataka su redom:

1) POS baza podataka – izvor podataka vezanih uz izdani račun nastao kao rezultat izvršene prodaje
2) CRM baza podataka – izvor podataka o kategorizaciji korisnika
3) NTP poslužitelj – izvor podataka o konkretnom trenutku prikupljanja podataka koje je sinkronizirano na razini cijelog poslovnog sustava tvrtke
4) Skladište podataka – izvor podataka o surogat ključevima dimenzija koji se koriste u samom skladištu
5) Registar metapodataka

Svaki od ovih izvora pruža određene podatke koji se tokom ETL-ovskog procesa integriraju te u obliku činjenica upisuju u stvarnovremensku particiju. Razlog korištenja većeg broja izvora je demonstracija prikupljanja podataka iz heterogenih izvora sa sljedećim konkretnim obrazloženjima:

1) Podaci iz POS baze podataka čine osnovne podatke koje činjenica opisuje
2) Podaci iz CRM baze podataka omogućuju proširenje činjenica informacijama o kategoriji korisnika koji je izvršio kupnju; podaci o kategoriji prikupljaju se direktno iz sustava CRM) zbog pretpostavke da dotični sustav sadržava najnovije podatke za razliku od skladišta koje se puni sa određenom latencijom
3) Vremenska oznaka prikupljena od strane NTP poslužitelja omogućuje označavanje podataka točnom oznakom vremena kada je prikupljanje podataka izvršeno što je često koristan podatak u stvarnovremenskim sustavima
4) Skladište podataka predstavlja izvor podataka koji omogućuju parcijalnu integraciju činjenice sa postojećim podacima u skladištu; ovaj postupak olakšava posao analitičkim aplikacijama koje kombiniraju podatke iz stvarnovremenske particije s podacima u skladištu podataka

U nastavku su dani primjeri konkretnih relacijskih tablica (ili tipa podataka) koje se nalaze u određenom izvoru. Studijski primjer ograničit će se samo na podatke koji se direktno koriste od strane SOA RT-ETL sustava.

POS baza podataka sadrži tablicu RACUN kojeg prikazuje **Tablica 6-1.**

Tablica 6-1 - primjer tablice RACUN unutar POS sustava

RACUN	PROIZVOD	KUPAC	CIJENA	KOLICINA
112657	221	154312	123,00	2
112657	243	154312	15,00	1
112657	125	154312	37,00	1
112658	256	127313	254,00	3
…	…	…	…	…

Ova tablica čuva podatke o izdanom računu tj. podatke koji opisuju izvršenu prodaju u obliku niza stavki za svaki tip prodanog proizvoda. Stavka računa definira broj računa, identifikator kupca ako je kupac poznat (tj. ako je koristio tzv. karticu lojalnosti), ključ kupljenog proizvoda, količinu i jediničnu cijenu. Pretpostavlja se da se za jedan račun podaci upisuju transakcijski tj. sve stavke vezane uz jedan račun se upisuju (i eventualno prikupljaju) zajedno.

POS baza podataka također sadržava i pokretač poslovnog procesa u obliku PL/SQL-ovskog okidača. Pokretanje procesa prikupljanja započinje na sljedeći način: okidač je povezan uz tablicu RACUN na takav način da se nakon upisivanja novih redaka u istu poziva Usluga weba tj. BPEL-ovski proces prikupljanja podataka (čija je implementacija dana u poglavlju 6.7.7). Inicijalizacijski mehanizam detaljnije je objašnjen u poglavlju 6.7.6.

CRM baza podataka sadrži podatke o korisnicima i predstavlja glavni izvor podataka za dimenziju KUPAC koja se nalazi u skladištu podataka. U scenariju studijskog primjera ova baza se koristi kako bi se činjenica koja opisuje prodaju proširila informacijama o kategoriji kupca prema nekoj otprije utvrđenoj kategorizaciji. Pretpostavlja se da CRM baza sadržava najsvježije podatke o kategoriji kupaca te se zato i koristi umjesto samog skladišta podataka (koje bi u slučaju preopterećenosti sustava CRM bilo i podesnija alternativa za prikupljanje ovog podataka).

Ova baza sadrži tablicu KUPAC koja može sadržavati veliku količinu informacija vezanih uz korisnike sustava. U sklopu studijskoj primjera bitan je samo podatak o kategorizaciji. Korištenu tablicu prikazuje **Tablica 6-2**.

Stupac KATEGORIJA_FK sadržava strani ključ povezan s otprije definiranom kategorijom. Sama kategorizacija nije bitna za studijski primjer, važno je samo naglasiti pretpostavku da su podaci o kategorizaciji preslikani i unutar skladišta podataka unutar odgovarajuće dimenzijske tablice (inače prikupljanje podataka u kategorizaciji ne bi imalo smisla s gledišta analitičkih aplikacija ako se ne prikupe i dodatne informacije o kojoj je kategoriji riječ).

Tablica 6-2 - primjer tablice KUPAC unutar CRM sustava

KUPAC_PK	KATEGORIJA_FK	…
154312	2	…
154313	2	…
154314	4	…
…	…	…

NTP poslužitelj simuliran je prikupljanjem lokalnog vremena aplikacijskog poslužitelja na kojem je postavljena usluga za prikupljanje podataka od strane NTP poslužitelja. Više informacija o implementaciji ove usluge nalazi se u poglavlju 6.7.3.

Konačno, skladište podataka se također koristi kao izvor kako bi se izvršila parcijalna integracija s postojećim podacima u skladištu pomoću dodavanja surogata ključa korisnika u samu činjenicu što omogućuje lako kombiniranje podataka iz stvarnovremenske particije s podacima unutar stvarnovremenskog skladišta. Racionalizacija korištenja surogata prirodnih ključeva umjesto samih prirodnih ključeva dana je u [13] a predstavlja standardnu komponentu ETL-ovskog procesa koji pohranjuje nove činjenice u skladište podataka. U općenitom slučaju, ovaj korak nije nužan već se integracija podataka iz stvarnovremenske particije s podacima iz skladišta može ostaviti na odgovornost nekom drugom sustavu ili samoj analitičkoj aplikaciji. No ovakva prilagodba podataka od strane ETL-ovskog procesa osigurava veću kvalitetu podataka u stvarnovremenskoj particiji, uz određene kompromise zbog dodavanja izvjesne latencije pošto podaci prolaze dodatnu obradu prije nego su dostupni na analizu.

Analogno CRM bazi podataka, implicirano je da dimenzija KUPAC koju koristi SOA RT-ETL sustav za prikupljanje surogata ključeva sadrži veći broj podataka o korisnicima u obliku niza stupaca s različitim atributima vezanim uz pojedinog korisnika, no za studijski primjer interesantni su samo stupci koji povezuju prirodni ključ korisnika (tj. broj njegove kartice lojalnosti) sa surogatom ključa kojeg izdaje samo skladište. Shodno tome, **Tablica 6-3** prikazuje tablicu KUPAC_DIM s naznačenim stupcima koji su interesantni s gledišta studijskog primjera.

Tablica 6-3 - primjer tablice KUPAC_DIM unutar skladišta podataka

KUPAC_NK	KUPAC_DW_PK	…
154312	1007	…
154313	1008	…
154314	1009	…
…	…	…

Prikupljeni podaci spremaju se u međuspremnike. **Međuspremnici podataka** su konstrukti unutar ETL-ovskog sustava koji pohranjuju podatke za vrijeme trajanja ETL-ovskog procesa. SOA RT-ETL sustav za tu potrebu koristi element metamodela *CWMX_SOA_StagingTable* opisan u poglavlju 5.3.3 čija XSD-ovska shema se može vidjeti u Prilogu 1, **Slika P1-8**. Ovaj element zapravo predstavlja XML-ovski oblik relacijske tablice podataka zajedno s podacima o broju i tipovima podataka unutar stupaca te eventualno i samim podacima.

Studijski primjer definira tri međuspremišta podataka: tablicu koja čuva podatke o činjenicama koje se opisuju u stvarnovremensku particiju, tablicu koja privremeno pohranjuje podatke o korisniku koji je kupovao proizvode te tablicu koja čuva podatke o vremenu prikupljanja podataka. XML-ovske dokumente koji čine ove tablice prikazuje **Slika 6-11**.

```
<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<CWMX_SOA_StagingTable id="76">
       <ColumnSet id="153">
              <Column type="xs:string" id="175" name="racun"/>
              <Column type="xs:string" id="176" name="proizvod"/>
              <Column type="xs:string" id="177" name="kupac"/>
              <Column type="xs:float" id="178" name="cijena"/>
              <Column type="xs:int" id="179" name="kolicina"/>
              <Column type="xs:string" id="180" name="kupac_kat_FK"/>
              <Column type="xs:string" id="181" name="kupac_PK"/>
       </ColumnSet>
</CWMX_SOA_StagingTable>

<CWMX_SOA_StagingTable id="77">
       <ColumnSet id="154">
              <Column type="xs:string" id="177" name="kupac_PK"/>
              <Column type="xs:string" id="180" name="kategorija_FK"/>
              <Column type="xs:string" id="181" name="kupac_FK"/>
       </ColumnSet>
</CWMX_SOA_StagingTable>

<CWMX_SOA_StagingTable id="78">
       <ColumnSet id="155">
              <Column type="xs:date" id="182" name="timestamp"/>
       </ColumnSet>
</CWMX_SOA_StagingTable>
```

Slika 6-11 – Međuspremnici podataka sustava SOA RT-ETL

Registar metapodataka predstavlja specijalizirani izvor podataka kojeg koristi proces kako bi omogućio prikupljanje potrebnih informacija za segmente procesa pogonjenih metapodacima. U sklopu studijskog primjera registar metapodataka izveden je pomoću niza XML-ovskih dokumenata koji opisuju elemente poslovnog sustava prateći metamodel razvijen u poglavlju 5.3. Ključni metapodaci tiču se ETL-ovskog procesa koji se mora provesti. Potpuni ispis XML-ovskog dokumenta sa svim elementima procesa nalazi se u Prilogu 2.

Konačno, spremište podataka je stvarnovremenska particija u koju se upisuju nove činjenice zajedno sa svim ostalim prikupljenim podacima. Međuspremnik s identifikatorom "76" kojeg prikazuje **Slika 6-11** dizajniran je upravo prema strukturi činjenične tablice u stvarnovremenskoj particiji (koja opet ima odgovarajuću komplementarnu tablicu u skladištu podataka. **Tablica 6-4** prikazuje izgled tablice unutar stvarnovremenske particije

Tablica 6-4 – konačni izgled činjenične tablice RACUN_FACT unutar stvarnovremenske particije

RACUN	PROIZVOD	KUPAC	KUPAC_KAT_FK	KUPAC_PK	CIJENA	KOLICINA	TIMESTAMP
112657	221	154312	2	1007	123,00	1	2009-05-24-06:23
112657	243	154312	2	1007	15,00	4	2009-05-24-06:23
112657	125	154312	2	1007	37,00	1	2009-05-24-06:23
112658	256	127313	3	945	54,00	3	2009-05-24-06:25
…	…	…			…	…	

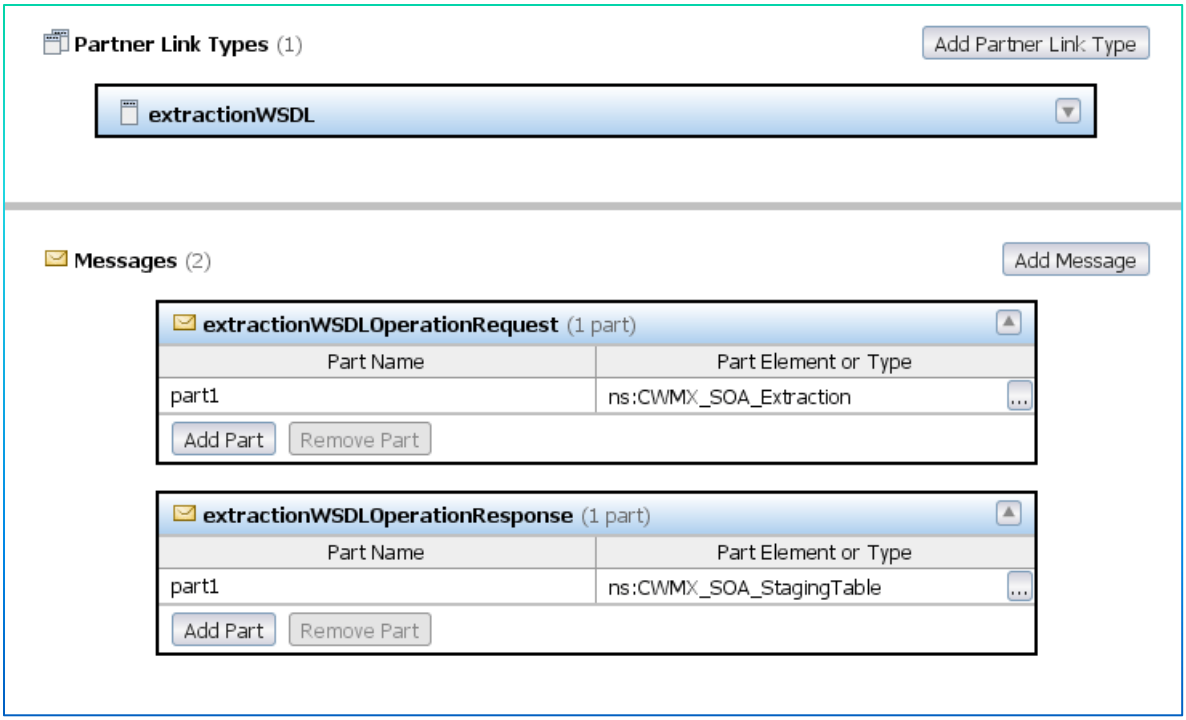
Dakle – izvori i odredište podataka implementirani su pomoću tablica unutar relacijskih baza podataka, od kojih svaka ima vlastite tehničke karakteristike glede uspostavljanja veze i autorizacije. Međuspremišta podataka su XML-ovski dokumenti koji se pune prikupljenim podacima. S gledišta sustava SOA RT-ETL, cijeli proces svodi se na obradu poruka u obliku XML-ovskih dokumenata, stoga usluge za prikupljanje podataka osim povezivanja s bazom i provedbe izraza koji će podatke prikupiti imaju i zadaću transformacije podataka u oblik u kojem će se koristiti u daljnjim fazama procesa (drugim riječima, prikupljene podatke na adekvatan način spremiti u pružene XML-ovske međuspremnike). Implementacija usluga za prikupljanje podataka tema je idućeg poglavlja.

6.7.3 Implementacija Usluga weba za prikupljanje podataka

Studijski primjer uključio je implementaciju tri usluge za prikupljanje podataka. One su redom:

1. ***ExtractionRDBWebService* –** usluga koja prima metapodatke o načinu prikupljanja podataka iz relacijskih izvora, interpretira ih, izvršava proces prikupljanja i prilagođava rezultate u oblik koji je također definiran metamodelom – konkretno elementom *CWMX_SOA_StagingTable*.
2. ***ExtractionNTPWebService*** – specijalizirana usluga čija je uloga komunikacija s NTP poslužiteljem i prikupljanjem vremenske oznake trenutka koja je sinkronizirana na nivou cijelog poslovnog sustava.

Usluga ***ExtractionRDBWebService*** jedna je od tri usluge pogonjene metapodacima implementirane u studijskom primjeru (pored nje izvedena je usluga za transformaciju i usluga za spremanje). Ova usluga izvedena je prema predlošku generičke usluge za prikupljanjem podataka obrađenom u poglavlju 6.1.1. Usluga kao ulazni parametar prima element *CWMX_SOA_Extraction* koji sadrži detalje o tome gdje i kako prikupiti podatke koji će se dalje obrađivati tokom ETL-ovskog procesa. Povratna poruka usluge je element *CWMX_SOA_StagingTable* koji sadržava prikupljene podatke. Vizualni prikaz WSDL-ovskog opisa usluge kakvog koristi razvojno sučelje ***Netbeans6.5*** *prikazuje* **Slika 6-12**.



Slika 6-12 - Vizualni prikaz WSDL-ovskog opisa usluge *ExtractionRDBWebService*

Ova usluga izvedena je tako da se pomoću primljenih metapodataka automatski prilagođava relacijskim izvorima podataka. Implementacija usluge uključuje veći broj tzv. JDBC (*engl. Java Database Connectivity*) pogonitelja koji odgovaraju trenutno najčešće korištenim sustavima za upravljanje relacijskim bazama podataka. Ovisno o uvjetima poslovne okoline, usluga se lako proširuje dodatnim pogoniteljima ako se za to ukaže potreba.

Element *CWMX_SOA_Extraction* između ostalog sadrži elemente *CWMX_SOA_RDBConnection* (element koji opisuje način pristupa relacijskom izvoru podataka), *CWMX_SOA_Expression* (element koji opisuje izraz čije izvršavanje u sprezi s danim parametrima rezultira prikupljanjem potrebnih podataka) te jedan ili više elemenata *CWMX_SOA_StagingTable* (koji predstavljaju ulazne i izlazne spremnike podataka a dizajnirani su da prate relacijski model podataka). Primjer XML-ovskog dokumenta koji odgovara ovom elementu već je dan u poglavlju 6.1.1, **Slika 6-3**. Usluga ***ExtractionRDBWebService*** interpretaciji ovih metapodaka prilazi na sljedeći način:

1) Analizira se element *CWMX_SOA_RDBConnection*. Ukoliko element definira tip relacijskog izvora za kojeg usluga nema odgovarajući pogonitelj, proces prikupljanja se prekida i vraća se odgovarajuća poruka o greški.
2) Ako se radi o poznatom tipu izvora, usluga koristi primljene autorizacijske podatke kako bi pokušala uspostaviti vezu s izvorom. Alternativno, ako tip izvora nije naveden ali je navedeno JNDI-ovsko (engl. *Java Naming and Directory Interface*) ime izvora usluga će pokušati pristupiti

kontekstu poslužitelja na kojeg je postavljena te tako uspostaviti vezu s izvorom[13]. U svakom slučaju, ako uspostavljanje veze nije uspješno također dolazi do prekida prikupljanja te se vraća poruka o greški.

3) Nakon uspješnog uspostavljanja veze analizira se element *CWMX_SOA_Expression*. On mora opisivati izvedivi SQL-ovski izraz s ispravno zadanim parametrima. Parametar može biti i tablica (opisana elementom *CWMX_SOA_StagingTable*) u kojem slučaju se SQL-ovski izraz izvršava iterativno za svaki redak ulazne tablice čiji stupci definiraju parametre izraza.
4) Neuspješno izvršavanje izraza također rezultira greškom. Rezultat uspješnog izvođenja izraza bit će relacijska tablica podataka.
5) Ukoliko nije navedena izlazna tablica, usluga sama generira rezultantnu tablicu koja prati model elementa *CWMX_SOA_StagingTable*. Imena i tipovi stupaca zaključuju se iz primljenih podataka.
6) Ukoliko postoji otprije definirana izlazna tablica, usluga provjerava da li struktura rezultantne tablice odgovara strukturu navedene izlazne tablice. Ako se uoči neslaganje proces se također prekida[14]. U slučaju da izlazna tablica odgovara rezultantnoj, ona se puni primljenim podacima i vraća pozivatelju usluge.

Ponovna iskoristivost ove usluge očituje se u činjenici da se njezina funkcionalnost može koristiti u bilo kojem procesu koji zahtijeva pristup i prikupljanje podataka iz relacijske baze podataka koja je dostupna kroz mrežu na kojoj je usluga postavljena. Preduvjet je jedino prilagodba zadanom metamodelu, tj. dizajn ulaznih i izlaznih poruka prema predlošcima danim od strane metamodela, poglavito predložak za stvaranje ulazne poruke tipa *CWMX_SOA_Extraction* te izlazne tipa *CWMX_SOA_StagingTable.* Dodatno, usluga se može lako proširiti dodatnom funkcionalnošću kao što je npr. pristupanje podacima koji se nalaze u tekstualnim datotekama unutar određenog datotečnog sustava i sl. Potrebno je samo slijediti način na koji metamodel opisuje takve izvore podataka te implementirati pripadnu logiku koja će dotične zadatke izvršavati.

Gledajući međuspremnike koje prikazuje **Slika 6-11** usluga ***ExtractionRDBWebService*** zadužena je za punjenje međuspremnika "76" (prikupljanje podataka iz sustava POS) i punjenje međuspremnika "77" (koji se dijelom puni iz sustava CRM a dijelom iz skladišta podataka).

Usluga ***ExtractionNTPWebService*** također prati isti WSDL-ovski predložak, no od usluge ***ExtractionRDBWebService*** razlikuje se u svojoj implementaciji. Ona je primjer usluga koja se *ne* naslanja na metapodatke tj. implementirana je za točno određenu funkciju unutar poslovnog procesa (prikupljanje podataka o vremenskoj oznaci trenutka nastanka činjenice). Korištenje protokola NTP, tj. egzistencija zasebnog NTP poslužitelja unutar poslovne okoline preporučljivo je zbog očuvanja jedinstvenog i konzistentnog sustava računanja vremena na razini cijelog informatičkog sustava. S

[13] Ovakva metoda spajanja je u određenim poslovnim okolinama preferirana pošto se na taj način odgovornost upravljanja vezom s bazom podataka može prebaciti na aplikacijski poslužitelj koji često nudi dodatne pogodnosti kao što su optimizacija veze, upravljanje uspostavom i raskidanjem i sl.

[14] Izlazna tablica može imati više stupaca nego rezultantna, pri čemu se stupci sa nepoznatim podacima ispunjavaju NULL vrijednostima. Ovo je implementacijska karakteristika sustava izvedena zbog lakše uporabe međuspremišnih tablica koje se moraju inkrementalno popunjavati u toku ETL-ovskog procesa (alternativa je korištenje većeg broja međuspremišta i potreba za izvedbom kompleksnijih transformacija).

gledišta poslovnog primjera NTP poslužitelj simulirat će se prikupljanjem vremenske oznake od strane aplikacijskog poslužitelja na kojem je usluga postavljena.

Iako usluga ***ExtractionNTPWebService*** nije pogonjena metapodacima, to ne znači da se ona ne naslanja na izvedeni metamodel. Kao dio SOA RT-ETL sustava ona i dalje mora komunicirati s ostalim segmentima sustava pomoću normiziranih poruka unutar procesa – konkretno njezina povratna informacija mora također biti element *CWMX_SOA_StagingTable*.

Iako usluga ***ExtractionNTPWebService*** vraća samo jedan podatak, struktura procesa nalaže da se i on mora postaviti u odgovarajući međuspremnik. Ponovo pogledavši međuspremnike koje prikazuje **Slika 6-11** može se vidjeti da postoji međuspremnik "78" posebno predviđen za korištenje od strane ove usluge.

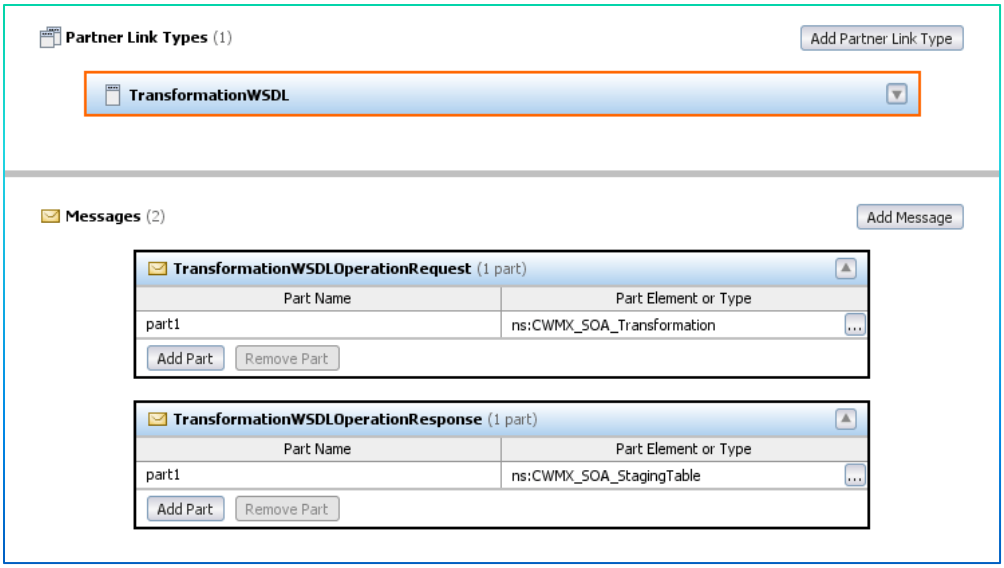
U sljedećem poglavlju prikazat će se izvedba transformacijske usluge koja omogućuje "preslagivanje" podataka iz navedenih međuspremnika te konačno integriranje podataka u činjenice koje će se upisivati u stvarnovremensku particiju.

6.7.4 Implementacija transformacijske Usluge weba

U poglavlju 6.2.1 opisan je predložak generičke usluge za transformaciju podataka te je naveden primjer takve usluge koja kao glavni pogonitelj transformacije koristi tehnologiju XSLT. Pošto se sva informacija unutar SOA RT-ETL sustava prenosi u obliku XML-ovskih dokumenata, ova tehnologija prirodan je izbor za izvedbu univerzalnog načina transformacije podataka pogonjenog metapodacima.

Usluga je implementirana pomoću WSDL-ovskog predloška kojeg prikazuje **Slika 6-5** u poglavlju 6.2.1. **Slika 6-13** prikazuje vizualni prikaz spomenutog WSDL-ovskog dokumenta kako ga prikazuje razvojno sučelje *Netbeans 6.5.*

Funkcionalnost ove usluge izvedena je na sljedeći način: očekivani ulazni parametar tipa *CWMX_SOA_Transform* sadržava ulazni element tipa *CWMX_SOA_StagingTable* koji sadržava podatke, element tipa *CWMX_SOA_XSLTDocument* koji predstavlja predložak na osnovu kojeg se izvršava transformacija te izlazni element tipa *CWMX_SOA_StagingTable* koji ne sadržava podatke već opisuje očekivanu strukturu tablice podataka nakon transformacije. Nakon izvedbe transformacije pomoću klase *java.xml.Transformer* rezultat transformacije prolazi provjeru gdje se pregledava da li odgovara tipu elementa *CWMX_SOA_StagingTable* te da li njegova struktura odgovara očekivanoj izlaznoj strukturi. Ako je rezultat provjere pozitivan proces se nastavlja, dok se u suprotnom slučaju proces prekida i javlja se poruka o greški.



Slika 6-13 - Vizualni prikaz WSDL-ovskog opisa usluge *TransformationXSLTWebService*

Potpuni izgled transformacijskog elementa može se vidjeti na XML-ovskom ispisu cjelokupnog procesa u Prilogu 2. **Slika 6-14 i Slika 6-15** prikazuju XSLT-ovske predloške korištene unutar studijskog primjera. Ovi predlošci naslanjaju se na međuspremnike koje prikazuje **Slika 6-11.** Prva transformacija iz primljenih podataka o računu (međuspremnik "76") preuzima prirodni ključ kupca (tj. broj kartice lojalnosti) te ga preslikava u drugi međuspremnik (sa identifikatorom "77"). Druga transformacija kombinira podatke iz sva tri prikupljena međuspremnika u jedan (inicijalni međuspremnik s identifikatorom "76") iz kojeg se podaci naknadno prenose u činjeničnu tablicu u stvarnovremenskoj particiji.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">

<CWMX_SOA_StagingTable id="77">
            <ColumnSet id="154">
                  <Column type="xs:string" id="177" name="kupac_PK"/>
                  <Column type="xs:string" id="180" name="kategorija_FK"/>
                  <Column type="xs:string" id="181" name="kupac_FK"/>
            </ColumnSet>

            <RowSet>
                  <Row href="177" value="{//RowSet[1]/Row[@href='177']/@value}"/>
                  <Row href="180" value="null"/>
                  <Row href="181" value="null"/>
            </RowSet>
</CWMX_SOA_StagingTable>
</xsl:template>
</xsl:stylesheet>
```

Slika 6-14 – XSLT-ovski predložak prve transformacije

```
<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<CWMX_SOA_StagingTable id="76">
       <ColumnSet id="153">
              <Column type="xs:string" id="175" name="racun"/>
              <Column type="xs:string" id="176" name="proizvod"/>
              <Column type="xs:string" id="177" name="kupac"/>
              <Column type="xs:float" id="178" name="cijena"/>
              <Column type="xs:int" id="179" name="kolicina"/>
              <Column type="xs:string" id="180" name="kategorija_FK"/>
              <Column type="xs:string" id="181" name="kupac_PK"/>
              <Column type="xs:date" id="182" name="timestamp"/>
       </ColumnSet>

       <xsl:for-each select="//RowSet[@href=153]">
       <RowSet>
              <Row href="175" value="{./Row[@href=175]/@value}"/>
              <Row href="176" value="{./Row[@href=176]/@value}"/>
              <Row href="177" value="{./Row[@href=177]/@value}"/>
              <Row href="178" value="{./Row[@href=178]/@value}"/>
              <Row href="179" value="{./Row[@href=179]/@value}"/>
              <Row href="180"
                     value="{/input/CWMX_SOA_StagingTable[@id=77]
                     /RowSet/Row[@href=180]/@value}"/>
              <Row href="181"
                     value="{/input/CWMX_SOA_StagingTable[@id=77]
                     /RowSet/Row[@href=181]/@value}"/>
              <Row href="182"
                     value="{/input/CWMX_SOA_StagingTable[@id=78]
                     /RowSet/Row[@href=182]/@value}"/>
              </RowSet>
       </xsl:for-each>

</CWMX_SOA_StagingTable>
</xsl:template>
</xsl:stylesheet>
```

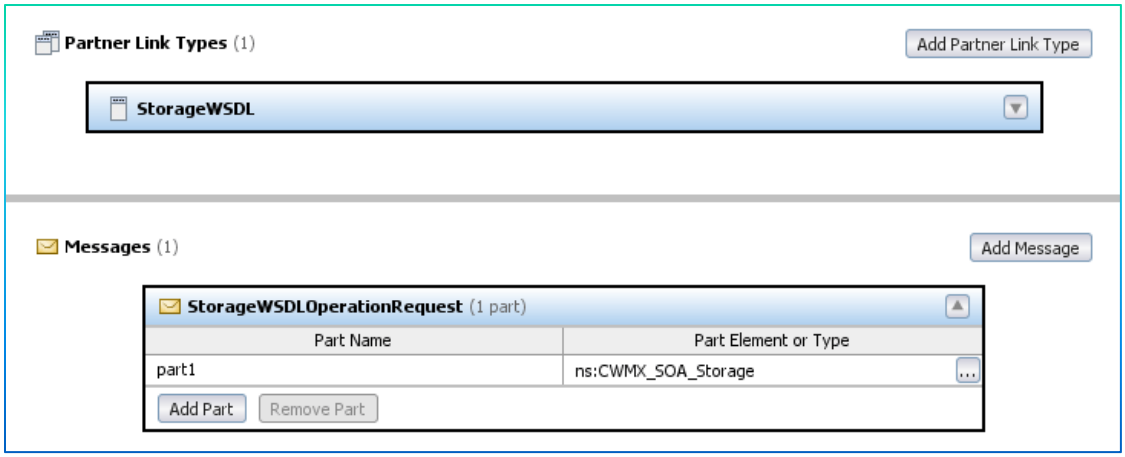
Slika 6-15 – XSLT-ovski predložak druge transformacije

Sličnom izvedbom vrlo je lako izvesti uslugu koja nije ograničena samo na transformacije elemenata tipa *CWMX_SOA_StagingTable* već vrši transformaciju bilo kojih XML-ovskih dokumenata uz pripadni XSTL-ovski predložak. Ograničavanje funkcionalnosti na elemente tipa *CWMX_SOA_StagingTable* izvedeno je strogo zbog povećanja robustnosti sustava SOA RT-ETL, no proširena funkcionalnost nudi privlačnu alternativu zbog puno većeg stupnja ponovne iskoristivosti takve usluge unutar poslovnog sustava.

6.7.5 Implementacija Usluge weba za spremanje podataka

Usluga *StorageRDBWebService* izvedena je prema predlošku razvijenom u poglavlju 6.3.1. Funkcionalnost usluge vrlo je slična usluzi *ExtractionRDBWebService* pošto je element metamodela *CWMX_SOA_Extraction* strukturalno vrlo sličan elementu *CWMX_SOA _Storage* a procedura prikupljanja podataka i procedura spremanja se obje zasnivaju na uspostavljanju veze sa spremištem podataka te izvršavanjem određenih SQL-ovskih izraza uz pomoć odgovarajućih ulaznih parametara.

Kao što se vidi na vizualnom prikazu WSDL-ovskog opisa ove usluge kojeg prikazuje **Slika 6-16** (a koji prati WSDL-ovski dokument razvijen u poglavlju 6.3.1, **Slika 6-7**) ulazni parametar je element *CWMX_SOA_Storage* koji sadrži sve nužne podatke za pohranu prikupljenih podataka u stvarnovremensku particiju. Konkretno, on sadrži element *CWMX_SOA_RDBConnection* koji opisuje način povezivanja s particijom, element *CWMX_SOA_Expression* koji opisuje SQL-ovski izraz čije izvođenje će izvršiti spremanje podataka te konačno prikupljene podatke unutar elementa *CWMX_SOA_StagingTable.*



Slika 6-16 - Vizualni prikaz WSDL-ovskog opisa usluge *StorageRDBWebService*

Algoritam obrade elementa *CWMX_SOA_Storage* od strane usluge *StorageRDBWebservice* analogan je algoritmu kojeg izvodi usluga *CWMX_SOA_Extraction* prikazanom u poglavlju 6.7.3. Razlika je samo u tome što izvođenje izraza ne rezultira povratnom tablicom, tj. ova usluga nema izlaznih parametara. Ako je upisivanje uspješno, nije potrebno vraćati nikakvu vrijednost nego je uspješna pohrana implicirana nedostatkom poruka o grešci. S druge strane, pojava bilo kakvog problema rezultirat će stvaranjem poruke o grešci koja će konkretno precizirati o kakvoj se grešci radi, u kojem dijelu metode je nastala i što je uzrok.

U općenitom slučaju uspješnom pohranom podataka ETL-ovski proces završava. U sljedećem poglavlju pružit će se više detalja o samom početku procesa, tj. inicijalizacijskom mehanizmu koji rezultira pokretanjem prikupljanja podataka te Usluzi weba koja omogućuje pristup metapodacima korištenim unutar samog procesa od strane usluga pogonjenim metapodacima.

6.7.6 Inicijalizacijski mehanizam i usluga prikupljanja metapodataka

Jedno od bitnih pitanja kod stvarnovremenskog skladištenja podataka jest način uočavanja nove informacije tj. mehanizam koji će registrirati pojavu podataka koji se trebaju prikupiti te inicirati sam stvarnovremenski ETL-ovski proces. Pitanje registracije novih podataka nije trivijalno pošto direktno utječe i na latenciju podataka tj. njihovo kašnjenje s obzirom na trenutak kada se događaj dogodi i trenutak kada se informacije o njemu nađu u skladištu tj. stvarnovremenskoj particiji.

[15] ugrubo definira dvije kategorije mehanizma uočavanja novih podataka, tzv. mehanizam guranja (engl. *push*) i mehanizam povlačenja (engl. *pull*). Glavna razlika je u mjestu gdje se uočavanje novih podataka odvija. Ako se odvija na strani samog izvora podataka radi se o mehanizmu guranja – tj. izvor je aktivni sudionik procesa i on objavljuje činjenicu da se podaci mogu prikupljati. Mehanizam povlačenja znači da neki drugi entitet (ili sam proces) preuzima odgovornost za uočavanje novih podataka.

Studijski primjer koristit će prvi mehanizam pomoću implementacije okidača koji će pri pojavi novih podataka inicijalizirati novi poslovni proces za prikupljanje podataka. Okidač će pratiti sekvencu dodjele novih identifikatora redaka tablice RACUN unutar POS baze podataka.

Za potrebe inicijalizacije procesa stvorena je procedura koja koristi *Oracle*-ov paket za poziv Usluga weba – *Callout Utility for Oracle 10g and 11g.* Procedura će za svaki novi redak konstruirati poruku tipa SOAP i proslijediti ju sustavu SOA RT-ETL pomoću poznate adrese procesa (koji će biti izvedene u sljedećem poglavlju). **Slika 6-17** prikazuje primjer ulazne poruke za poslovni proces.

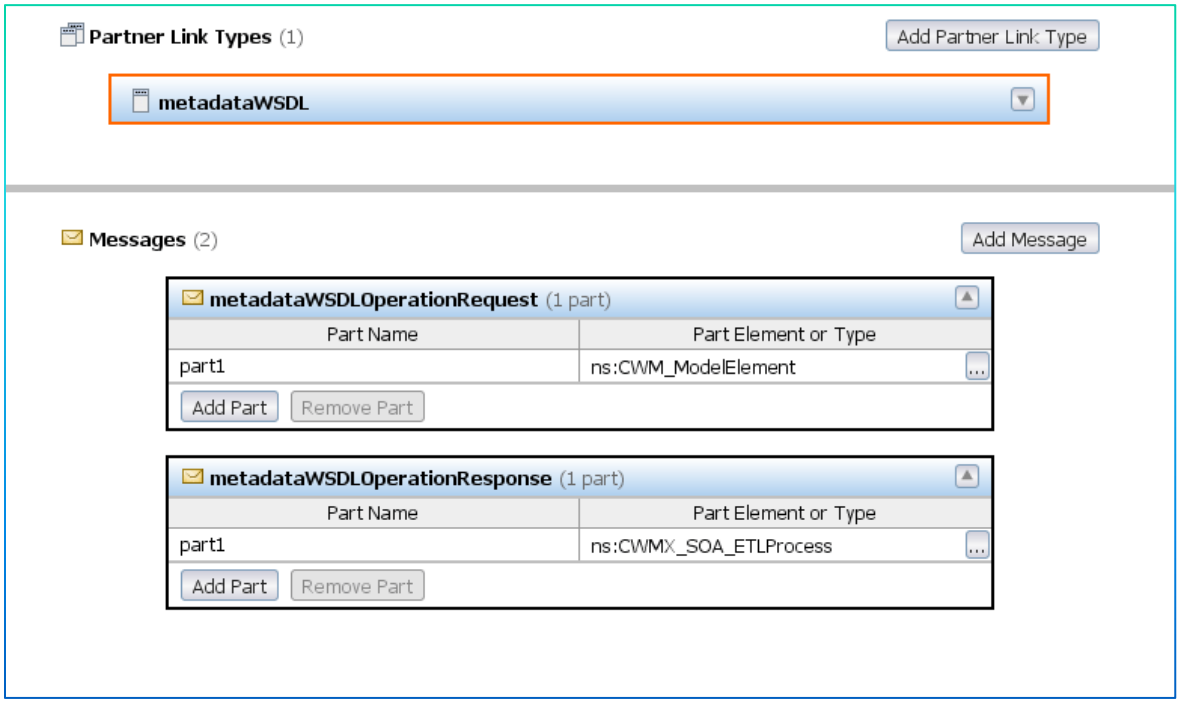
```
<soapenv:Envelope xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cwmx="http://xml.netbeans.org/schema/CWMX_SOA">

  <soapenv:Body>

    <cwmx:CWM_ModelElement id="2810">
       <taggedValue>
              <name>racun_id</name>
              <value>14</value>
       </taggedValue>
    </cwmx:CWM_ModelElement>

  </soapenv:Body>

</soapenv:Envelope>
```

Slika 6-17 - Inicijalizacijska poruka poslovnog procesa

Procedura zapravo stvara dokument tipa *CWMX_SOA_ModelElement*. Bitni dijelovi su identifikator (atribut „id“) koji će procesu omogućiti pronalaženje odgovarajućih metapodataka iz registra, te pridružena vrijednost koja definira parametre koji će poslužiti kod prikupljanja podataka iz izvora. U ovom slučaju definira se parametar „racun_id“ koji preuzima vrijednost „novog“ identifikatora retka koji identificira podatke predviđene za prikupljanje, preoblikovanje i spremanje u stvarnovremensku particiju.

Prihvaćanje ove poruke predstavlja početak provođenja poslovnog procesa. Nakon što je provoditelj procesa (o kojem će više riječi biti u idućem poglavlju) primio ulazni parametar za inicijalizaciju, njegova prva zadaća je prikupljanje metapodataka koji će opisati korake ETL-ovskog procesa. Za to poziva uslugu *ProcessMetadataWebService.*

Usluga za prikupljanje metapodataka prati WSDL-ovski predložak razvijen u poglavlju 6.4, **Slika 6-9**. Vizualni prikaz WSDL-ovskog predloška u razvojnom sučelju ***NetBeans 6.5*** prikazuje **Slika 6-18**.



Slika 6-18 - Vizualni prikaz WSDL-ovskog opisa usluge *ProcessMetadataWebService*

Pošto sve usluge unutar sustava SOA RT-ETL svoje ulazne i izlazne parametre zasnivaju na metamodelu razvijenom u poglavlju 5, tako i ova usluga kao ulazni parametar ima element *CWM_ModelElement* a kao izlazni *CWMX_SOA_ETLProcess*. *CWM_ModelElement* izabran je zato što je najjednostavniji element metamodela a može sadržavati vlastite parametre u obliku označnih vrijednosti (*Tagged Value*). S implementacijske razine ova bi usluga mogla koristiti i ulazne parametre nevezane za metamodel, no ova izvedba izabrana je zbog očuvanja konzistentnosti sustava. Izlazni parametar *CWMX_SOA_ETLProcess* zapravo je spremnik svih elemenata procesa – poglavito elemenata *CWMX_SOA_Extraction, CWMX_SOA_Transformation, CWMX_SOA_Storage* te

CWMX_SOA_StagingTable koji redom predstavljaju prikupljanje, transformaciju, spremanje i međuspremnike podataka.

Usluga *ProcessMetadataWebService* ima zadaću pristupa registru metapodataka i prikupljanja navedenih elemenata vezanih uz ETL-ovski proces. Izvedba samog registra i sučelja prema njemu izlazi iz okvira ove disertacije, važna je samo činjenica da registar mora biti kompatibilan s normom CWM te proširenim metamodelom razvijenim u poglavlju 5. S gledišta studijskog primjera registar metapodataka simuliran je nizom XML-ovskih dokumenata izvedenih pomoću XSD-ovskih shema koje se mogu naći u Prilogu 1. Usluga *ProcessMetadataWebService* pristupa XML-ovskim dokumentima i preko pruženog identifikatora prikuplja potrebi dokument koji opisuje ETL-ovski proces, tj. XML-ovski oblik elementa *CWMX_SOA_ETLProcess.*

Potpuni ispis XML-ovskog dokumenta koji predstavlja element *CWMX_SOA_ETLProcess* može se naći u Prilogu 2. U nastavku će se pobliže objasniti provedba samog procesa, tj. izvedbe Usluge weba koja djeluje kao upravljač procesom i čija je uloga upravljanje razmjenom poruka koje će rezultirati uspješnom izvedbom ETL-ovskog procesa tj. prikupljanjem, obradom i konačnom pohranom novih činjenica u stvarnovremensku particiju.

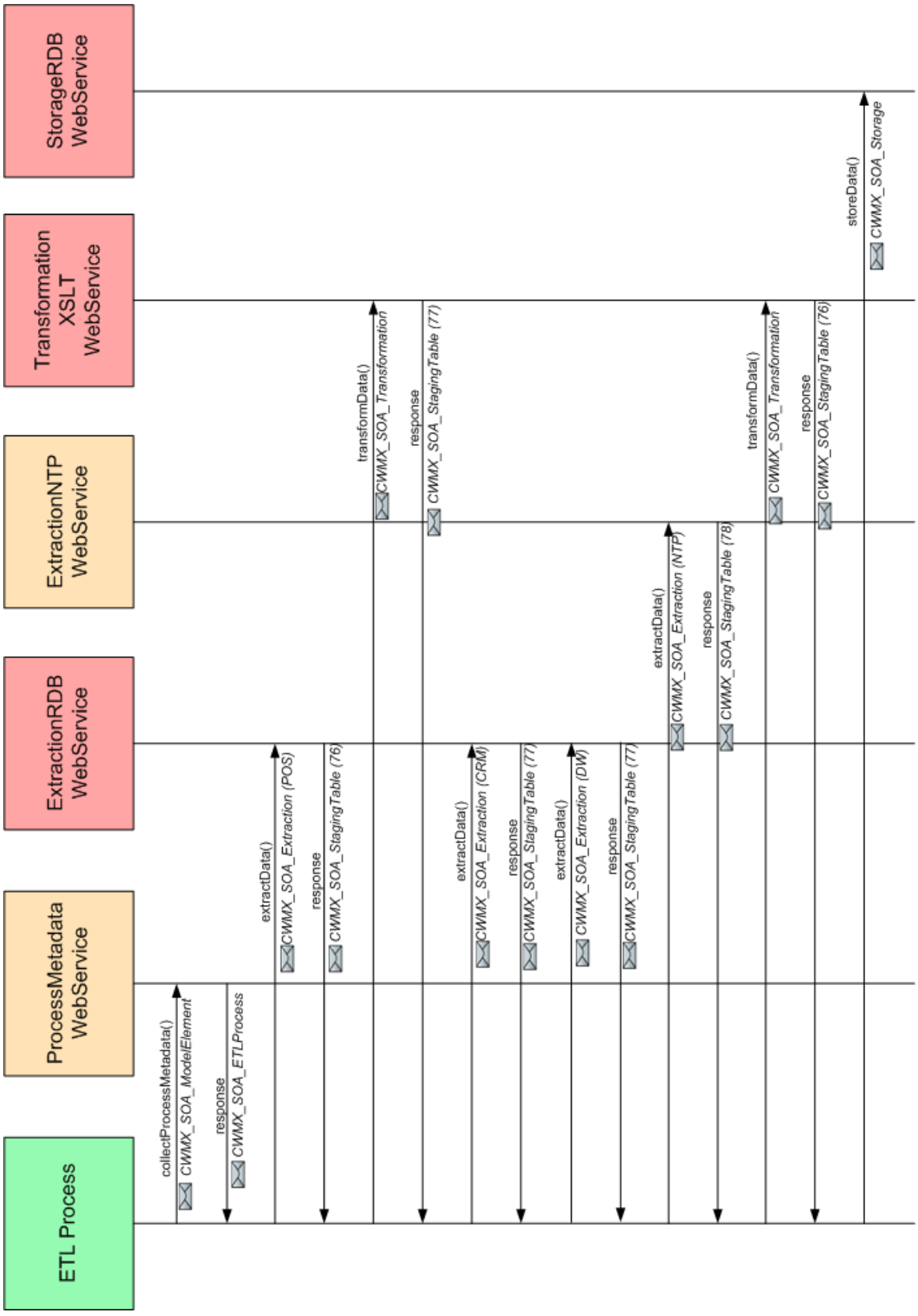
6.7.7 Implementacija ETL-ovskog procesa

Vanjske komponente sustava su implementirane te je preostalo samo izvesti poslovni proces koji će omogućiti organizaciju dosad izvedenih izgradbenih blokova u jedinstvenu cjelinu – ETL-ovski proces.

Poslovni proces izvest će se pomoću WS-BPEL tehnologije. Ova tehnologija omogućuje izvedbu procesa u obliku upravljačke Usluge weba koja prateći izvedeni predložak provodi proces kroz upravljanje razmjenom poruka između sudionika procesa. BPEL-ovski predložak omogućuje definiranje varijabli, pridjeljivanje i upravljanje elementima poruka, uvjetovano grananje te još dodatne složenije funkcionalnosti.

Slika 6-19 prikazuje UML-ovski sekvencijalni dijagram izvedbe poslovnog procesa. Slanjem inicijalizacijske poruke (koju prikazuje **Slika 6-17** u poglavlju 6.7.6) aplikacijski poslužitelj koji implementira okolinu za izvršavanje BPEL-ovskih procesa aktivira Uslugu weba koja djeluje kao upravljač ETL-ovskim procesom. Ova usluga (nazvana *ETLProcess)* zadužena je za koordiniranu razmjenu poruka između ostalih sudionika sustava i kao takva predstavlja jezgru sustava SOA RT-ETL.

Prvi korake je pozivanje usluge *ProcessMetadataWebService* i prikupljanje elementa tipa *CWMX_SOA_ETLProcess* koji sadrži sve nužne metapodatke za provođenje ostatka ETL-ovskog procesa. Potpuni sadržaj ovog elementa u XML-ovskom obliku može se naći u Prilogu 2. Primivši metapodatke procesa, upravljač procesom preslikava metapodatke vezane uz prikupljanje, transformaciju i pohranu podataka u za to predviđene varijable.



Slika 6-19 - UML-ovski sekvencijalni dijagram provedbe procesa

Prvo se povlače podaci iz sustava POS slanjem elementa *CWMX_SOA_Extraction* Usluzi weba *ExtractionRDBWebService* koja je obrađena u poglavlju 6.7.3. Rezultat uspješnog poziva je povezivanje s bazom sustava POS te prikupljanje podataka nakon čega su podaci adekvatno spremljeni u međuspremnik tipa *CWMX_SOA_StagingTable s* identifikatorom "76" (kojeg prikazuje **Slika 6-11** u poglavlju 6.7.2). Nakon toga međuspremnik se šalje usluzi *TransformationXSLTWebService* (obrađene u poglavlju 6.7.4) koja pomoću XSTL-ovskog predloška preslikava potrebne informacije u međuspremnik s identifikacijskom oznakom "77". Ovaj međuspremnik se prosljeđuje usluzi *ExtractionRDBWebService* koja prvo pristupa bazi sustava CRM a potom skladištu podataka. Svi prikupljeni podaci također se pohranjuju u međuspremnik "77". Proces također poziva specijaliziranu uslugu *ExtractionNTPService* koja puni međuspremnik "78" vremenskom oznakom trenutka prikupljanja podataka.

Nakon toga svi međuspremnici prosljeđuju se ponovo transformacijskoj usluzi *TransformationXSTLWebService* koja pomoću proslijeđenog XSLT-ovskog predloška preslikava sve prikupljene podatke u međuspremnik "76". Konačni sadržaj ovog međuspremnika odgovara činjeničnoj tablici u stvarnovremenskoj particiji te se on zajedno s elementom *CWMX_SOA_Storage* (koji sadrži detalje o načinu spremanja) prosljeđuje usluzi *StorageRDBWeBService* koja konačno pohranjuje podatke u stvarnovremensku particiju. Uspješno izvršavanje ovog posljednjeg zadataka ujedno predstavlja i završetak procesa.

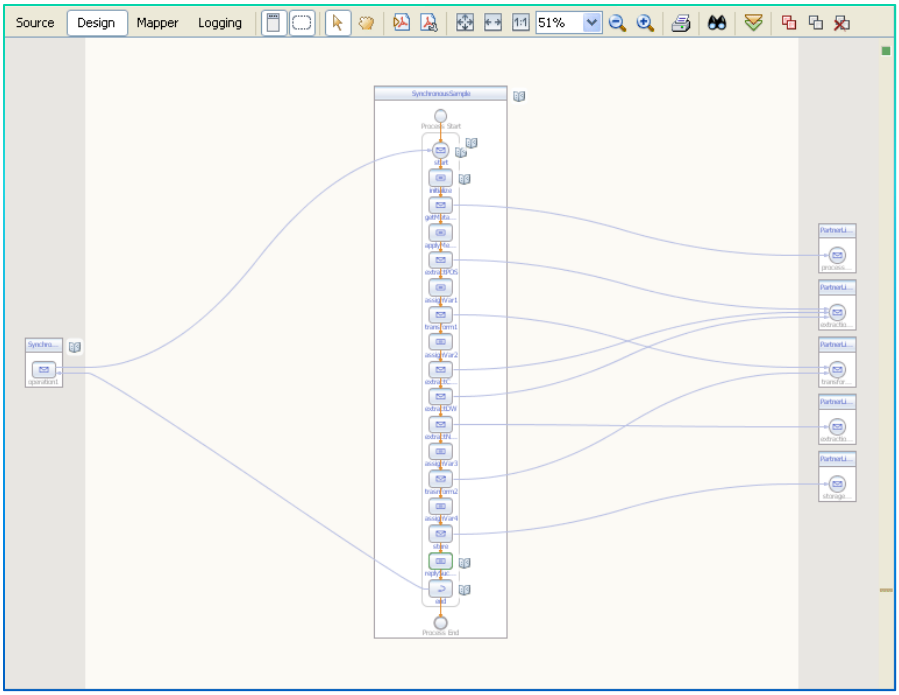
Slika 6-20 prikazuje model procesa kako izgleda u razvojnom sučelju *NetBeans 6.5.* S lijeve strane nalaze se „klijenti“ procesa – partnerske veze s informatičkim entitetima koje traže uslugu od procesa. Kao što je već rečeno, sam proces predstavlja Uslugu weba zajedno sa svojim WSDL-ovskim opisom i ponuđenim metodama. Pozivanje metode stvara instancu procesa i započinje njegovo provođenje.

Struktura procesa je vrlo jednostavna. Zadaća procesa je jednostavno pridjeljivanje vrijednosti pripadnim varijablama te prosljeđivanje odgovarajućih poruka partnerskim Uslugama weba. Drugim riječima, proces koristi gotovo isključivo elemente `<invoke>` i `<assign>`, što njegovo modeliranje čini iznimno jednostavnim.

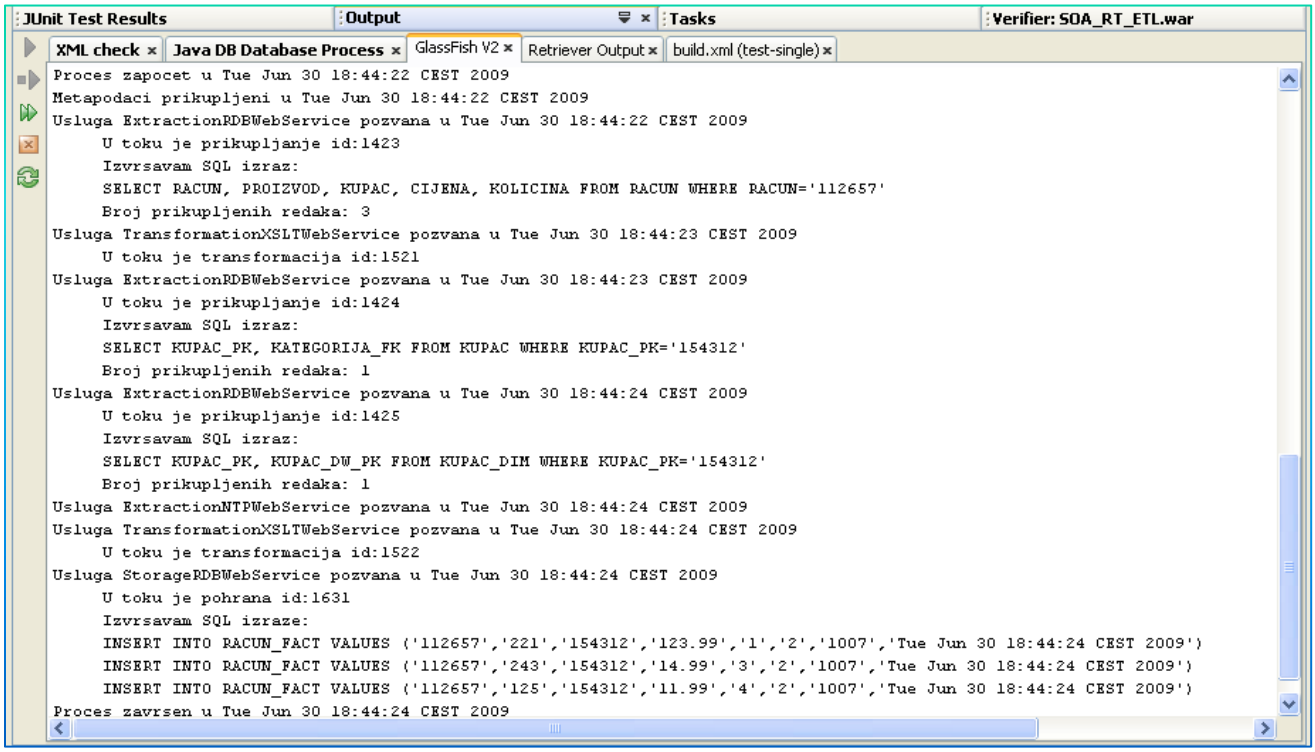
Implementirani proces postavljen je na aplikacijski poslužitelj *Glassfish v2.0* koji predstavlja jednu od preporučenih platformi za izvođenje BPEL-ovskih procesa. Nakon toga provedeno je testiranje kroz dodavanje novih redaka u simulirani sustav POS (uz već postojeće podatke u simuliranim sustavima CRM i dimenzijskoj tablici skladišta podataka) te provjeru njihove uspješne dostave u stvarnovremensku particiju uz dijagnostičke poruke koje omogućuju mjerenje latencije. Analiza rezultata provedena je u sljedećem poglavlju.

6.7.8 Testiranje i analiza rezultata

U programski kôd Usluga weba unesena je logika koja ispisuje dijagnostičke poruke nakon izvođenja određenih segmenata procesa. **Slika 6-21** prikazuje dijagnostički ispis nakon što su u POS bazu podataka unesene nove činjenice u obliku prva tri retka koje prikazuje **Tablica 6-1** u poglavlju 6.7.2.



Slika 6-20 - Model procesa u razvojnom sučelju *NetBeans 6.5*



Slika 6-21- Dijagnostički ispis sustava SOA RT-ETL

Iz dijagnostičkog ispisa može se vidjeti da u ovom slučaju izvođenje cjelokupnog procesa prikupljanja nekoliko novih činjenica traje tek nekoliko sekundi. To znači da su novi podaci u stvarnovremenskoj particiji pohranjeni sa vrlo malom latencijom, vjerojatno prihvatljivom za sve poslovne sustave osim onih koji zahtijevaju krajnju minimalizaciju do ideala nulte latencije. Iako se rezultat mjerenja latencije unutar studijskog primjera može uzeti tek kao referentna vrijednost ona ipak predstavlja dobar primjer ogledne vrijednost koja se može očekivati kod implementacije sustava SOA RT-ETL u realnim poslovnim uvjetima.

Neke od karakteristika sustava koje mogu negativno utjecati na latenciju su:

- relacijska baza podataka (koja igra ulogu transakcijskih izvore i stvarnovremenske particije) je udaljena baza podataka opće namjene kojoj se pristupalo preko mrežnih protokola
- aplikacijski poslužitelji dio je razvojnog sučelja umjesto da je postavljan na namjenski poslužitelj za tu ulogu
- ETL-ovski proces svodi se na niz manipulacija nad XML-ovskim dokumentima, koji sami po sebi ne predstavljaju optimalni način reprezentacije informacije kada je ključna karakteristika informacijskog sustava brzina obrade informacija
- temeljni jezik za izvedbu elemenata sustava je programski jezik *Java* koji je često predmet kritika zbog sporijeg izvođenja s obzirom na druge programske jezike koji ne koriste princip virtualnog stroja već su prilagođeni konkretnoj implementacijskoj platformi

S druge strane u realnim poslovnim uvjetima može se očekivati i povećanje latencije, pogotovo ako se uzme u obzir:

- preopterećenost transakcijskih baza – izvora podataka
- veća složenost ETL-ovskog procesa s većim brojem prikupljanja i složenijim transformacijama
- potencijalno korištenje vanjskih Usluga weba kao izvora podataka ili pružatelja transformacijskih usluga nad kojima matični poslovni sustav nema administrativnu kontrolu i koje mogu izazvati usko grlo i znatno povećanje latencije
- općenite karakteristike i učinkovitost aplikacijskih poslužitelja – platforme na koju se postavlja sustav SOA RT-ETL.

Realna učinkovitost sustava može se zaključiti samo iz daljnjeg iscrpnog testiranja, tj. modeliranja većeg broja ETL-ovskih procesa raznih karakteristika te njihovog provođenja u uvjetima realnih poslovnih okolina.

Kroz izvedbu studijskog primjera između ostalog došlo se do sljedećih zaključaka koji mogu utjecati na daljnji rad na ovom području:

1) Integracija pogonjena modelom izvediva je pomoću modernih tehnologija; odabir kvalitetnog metamodela ključan je za izvedbu integracije te daljnje održavanje i evoluciju sustava
2) Skladište podataka može se bez većih zahvata i izmjena postojećeg sustava proširiti stvarnovremenskom podrškom zasnovanom na uslužno orijentiranim principima te navedenim principom integracije

3) Komunikacija sa vanjskim izvorima podataka te, prikupljanje i spremanje lako se može izvesti pomoću generičkih usluga i pripadajućih metapodataka - glavnina problema odnosi se na kvalitetan opis načina uspostavljanja veze te izraza kojim se podacima upravlja
4) Izvedba transformacije i integracije predstavlja nešto veći problem za generičku izvedbu većinom zbog složenih procesa čišćenja i prilagodbe tj. nepostojanja jezika koji bi mogao takve procese opisati na generički način te adekvatnih mapiranja koja bi takve opise mogli automatski transformirati u izvedive programske rutine; metapodaci su i dalje u ovim slučajevima pogodni za upotrebu u obliku dokumentacije, dok se samo provođenje lakše izvodi kroz specijalizirane usluge
5) tvrtke koje svoje elektroničko poslovanje zasnivaju na uslužno orijentiranim poslovnim procesima vrlo lako mogu postojeću hardversku i softversku podršku te prateće tehnologije iskoristiti i kod implementacije procesa SOA RT-ETL, što uvelike smanjuje početne troškove i potrebne resurse

Studijski primjer prikazao je učinkovitost i podesnost sustava SOA RT-ETL te na realnom poslovnom primjeru pokazao njegovu konkretnu funkcionalnost tj. način na koji se navedeni stvarnovremenski ETL-ovski proces provodi i kako može komplementirati postojeći sustav skladištenja podataka. Očekivani razvoj korištenih normi i tehnologija utjecat će na njegovu daljnju evoluciju i realizaciju novih rješenja za zadane probleme koji se javljaju kod izvedbe stvarnovremenskih sustava. Ono što je potencijalno najveća prednost sustava SOA RT-ETL jest činjenica da se on – zbog svoje uslužno-orijentirane prirode - sastoji od slabo povezanih elemenata od kojih je izvjesni dio generičkog tipa što rezultira visokom razinom ponovne iskoristivosti. Drugim riječima, izvedeni sustav ima znatni potencijal korištenja postojećih principa, normi i usluga za lakšu implementaciju drugih tipova poslovnih procesa koji ne moraju nužno biti usko povezani uz proces stvarnovremenskog skladištenja podataka.

Zaključak

U posljednjih nekoliko desetljeća poslovanje je doživjelo izvjesne promjene. Uspon Interneta doveo je do sve većeg pomaka prema elektroničkom poslovanju tj. prema korištenju novih informacijskih tehnologija kao glavne potpore provođenju modernog poslovanja. Internet je doveo do otvaranja i globalizacije tržišta dok su nova hardverska i softverska rješenja učinila mnoge zahtjeve – prethodno neizvedive zbog tehnoloških ograničenja – mogućim, poželjnim pa čak i očekivanim. Integracija poslovnih sustava danas je imperativ, pošto izolirani informacijski sustav predstavlja prepreku kako u organizaciji internog poslovanja tvrtke, tako i u uspostavljanju poslovanja s partnerima i klijentima tvrtke. Nadalje, poslovna inteligencija tj. analitičke aplikacije koje pomažu pri donošenju poslovnih odluka počinju igrati ključnu ulogu u očuvanju konkurentnosti tvrtki na tržištu a vrijednost pravovremene informacije znatno je porasla.

Sukladno tome, sustavi skladištenja podataka također su doživjeli velike promjene. Broj potencijalnih izvora podataka se znatno povećao a kašnjenje podataka s obzirom na trenutak pojave poslovnih događaja koje opisuju se sve manje tolerira. Poslovni korisnici žele donositi odluke zasnovane na najnovijim informacijama a analitičke aplikacije često potražuju što veće količine najrazličitijih podataka koji bi se mogli uključiti u složene analize. Novije grane poslovne inteligencije – kao što je npr. dubinsko pretraživanje podataka koje traži skrivene pravilnosti među velikim količinama podataka – najčešće potražuju i podatke koji nisu usko vezani uz aspekt poslovanja kojeg analiziraju. Sve to dovodi do dva imperativa novijih skladišnih sustava – *lakša integracija s izvorima podataka* te *brže dohvaćanje relevantne informacije.*

Kada se govori o integraciji heterogenih sustava, odmah se nameće pojam *uslužno orijentirane arhitekture* kao glavnog integracijskog alata današnjice. Od svoje inicijalne pojave kroz tehnologije CORBA i DCOM, preko općeg i entuzijastičnog prihvaćanja nakon pojave Usluga weba kao glavne implementacijske tehnologije uslužno orijentirana arhitektura postaje jedan od glavnih principa integracije modernih poslovnih informacijskih sustava. Sustavi skladištenja podataka mogu imati velike koristi od prihvaćanja uslužno-orijentiranih mehanizama koji predstavljaju učinkovitu i poželjnu alternativu prilagođenim metodama koje koriste monolitni ETL-ovski sustavi. Ovo se pogotovo odnosi na problematiku prikupljanja, transformacije i pohrane podataka u stvarnom vremenu, što predstavlja vrlo pogodan scenarij uporabe kroz arhitekturu SOA koja najčešće i podrazumijeva uvođenje poslovnih procesa koji funkcioniraju pomoću protoka poslovnih poruka.

Integracija pogonjena modelom je drugi pristup integraciji heterogenih sustava koji kao premisu uzima činjenicu da je znanje o drugom sustavu ključno za uspješno uspostavljanje komunikacije a time i integracije u jedinstveni informacijski sustav. Drugim riječima, interpretacija metapodataka o drugom sustavu može poslužiti kao učinkovit alat za dinamičko provođenje integracije dvaju različitih sustava. Kako bi se ovo omogućilo potrebno je odabrati normizirani način predstavljanja metapodataka, omogućiti spremanje metapodataka u za to predviđeno spremište (registar metapodataka) definirati protokole kroz kojih bi se metapodaci prikupljali te konačno implementirati programsku logiku

interpretacije tih metapodataka. Ovo predstavlja složen i zahtjevan proces koji dugoročno može polučiti vrlo učinkovite rezultate.

Kroz istraživanje provedeno u okviru ovog rada provela se svojevrsna sinergija oba pristupa integraciji heterogenih sustava kroz realizaciju sustava za podršku stvarnovremenskom skladištenju pomoću uslužno orijentiranih principa te normiziranih metapodataka. Na temelju proširenja specifikacije CWM izveden je normizirani i formalizirani model metapodataka koji mogu poslužiti za opisivanje konkretnih aspekata sustava za podršku stvarnovremenskom skladištenju. Potom je definiran uslužno orijentirani sustav koji je koristio dane metapodatke za dinamičku prilagodbu poslovnoj okolini i provođenje stvarnovremenskog ETL-ovskog procesa. Konačno, izvedena je implementacija na studijskom primjeru koji je demonstrirao učinkovitost navedenog pristupa integraciji te kroz testiranje prikazao kako tako oblikovan sustav može zadovoljiti stroge poslovne zahtjeve za minimalizacijom latencije podataka dostupnih za analizu.

Daljnji rad na ovom području orijentirat će se na istraživanje novih scenarija uporabe razvijenog modela metapodataka te njegovu prilagodbu na potencijalni izlazak budućih verzija specifikacija CWM. Nadalje, sustav SOA RT-ETL će se proširiti novim mogućnostima a jedno od bitnih područja nastavka istraživanja bit će problematika generičkog načina definiranja i implementacije transformacija te optimizacija provođenja stvarnovremenskog ETL-a pomoću sustava SOA RT-ETL. Konačno, istražit će se mogućnost izvedbe sličnih poslovnih procesa koji će pokrivati druge aspekte pored problematike stvarnovremenskog skladištenja [34] te integracije takvih poslovnih procesa u jedan organizirani informacijski sustav široke primjene unutar modernog elektroničkog poslovanja.

Literatura

[1] Pintar, Damir; Vranić, Mihaela; Skočir, Zoran, "Metadata-Driven SOA-Based Application for Facilitation of Real-Time Data Warehousing", *Lecture notes in computer science*. 5692 (2009) ; pp. 108-119

[2] Raghu R. Kodali, "What is service-oriented architecture? An introduction to SOA", www.*JavaWorld.com*, 13/06/2005

[3] Joe McKendrick, "A definition of SOA we can live with", www.*zdnet.com*, 01/12/2005

[4] Thomas Erl, "Service-Oriented Architecture: Concepts, Technology and Design", Prentice Hall, 2005., ISBN:0-13-185858-0

[5] Jason Bloomberg, "Divorcing SOA and Web Services", www.*zapthink.com*, document_id:ZAPFLASH-2007618, 20/06/2007

[6] Thomas Erl, "Service-Oriented Architecture: A field guide to Integrating XML and Web Services", Prentice Hall, 2005., ISBN:0-13-142898-5

[7] David Chapell, "Who cares about UDDI?", www.*informIT.com*, 12/04/2002

[8] Pintar, Damir; Vranić, Mihaela; Skočir, Zoran. Business Service Interface Structure // *Proceedings of the 12th Mediterranean Electrotechnical Conference, MELECON 2004 /* Matijašević, Maja; Pejčinović, Branimir; Tomšić, Željko; Burković, Željko (ur.). Zagreb: The Institute of Electrical and Electronics Engineers, INC., 2004. 619-622

[9] Matasić, Ivan; Pintar, Damir; Skočir, Zoran. A Tool for e-Business Process Definition // *11. International Conference on Software, Telecommunications & Computer Networks – SoftCOM 2003 /* Begušić, Dinko; Rožić, Nikola (ur.). Split: Slobodna Dalmacija, 2003. 440-444

[10] Dotnet collective, "Summary of Web Service Extensions Specifications", *www.dotnet-collective.com*, 16/08/2005

[11] Randy Heffner, Jonathan Browne, Tim Sheedy, Gene Leganza, Jacqueline Stone, "Planned SOA Usage Grows Faster Than Actual SOA Usage", Forrester Research Inc., 28/02/2007

[12] W.H.Inmon, "Building the Data Warehouse 3rd Edition", Kindle Book, 2002., ISBN:978-0471081302

[13] R.Kimball, Margy Ross, "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd Edition", Wiley Books, 2002, ISBN:978-0471200246

[14] L. Agosta and K. Gile, “Real-Time Data Warehousing: The Hype And The Reality”, Forrester Research Inc., December 2004.

[15] Langseth, J., "Real-Time Data Warehousing: Challenges and Solutions", DSSResources.COM, 02/08/2004

[16] N.M. Tho, A.M Tjoa, “Zero-Latency Data Warehousing: Continuous Data Integration and Assembling Active Rules”, In *Proceedings of 5th International. Conference on Information Integration and Web-based Applications and Services (IIWAS2003)*, Jakarta, Feb. 2003.

[17] T.M.Nguyen, A Min Tjoa, "Zero-Latency Data Warehouse (ZLDWH): The State-of-the-art and experimental implementation approaches", *International Conference on Research, Innovation and Vision for the Future, 2006*, Volume I, Feb. 12-16, 2006 Page(s): 167 – 176 Digital Object Identifier 10.1109/RIVF.2006.1696434

[18] IdeaByte, "Defining Realistic Requirements for Real-time Data Integration", December 29 2003

[19] R. Kimball, J.Caserta, "The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning Conforming and Delivering Data", Wiley Books, 2004., ISBN 13:978-0764567575

[20] D.Pintar, Z.Skočir, M.Vranić. "Intelligent Data Sources and Integrated Data Repository as a Foundation for Business Intelligence Analysis", *Proceedings of 15. International Conference on Software, Telecommunications & Computer Networks – Softcom 2007.,* Split:FESB, 2007

[21] David Marco, Building and Managing the Meta Data Repository: A Full Lifecycle Guide, Wiley, 2000, ISBN 0-471-35523-2

[22] OMG Group, "Meta Object Facility 2.0 Specification", OMG Modelling and Metadata Specifications, 01/06/2001

[23] Model Driven Architecture Richard Soley and the OMG Staff Strategy Group, Object Management Group, White Paper, Draft 3.2 – November 27, 2000

[24] OMG Group, "Unified Modelling Language 2.2: Superstructure and Infrastructure Specification", OMG Modelling and Metadata Specifications, 04/02/2009

[25] Dan Pilone, Neil Pitman: "UML 2.0 in a Nutshell", O'Reilly Publishing, 2005, ISBN 13: 978-0596007959

[26] Jason Stamper, "UML 2.0 'bloated' but effective", Computergram International, 24/11/2003

[27] UML Forum, "UML: Frequently Asked Questions", www.uml-forum.com

[28] OMG Group, "Common Warehouse Metamodel 1.1 Specification", OMG Modelling and Metadata Specifications, 02/03/2003

[29] John Poole, Dan Chang, Douglas Tolbert, David Mellor, "Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration", Wiley, 2001, ISBN-13: 978-0471200529

[30] John Poole, Dan Chang, Douglas Tolbert, David Mellor, "Common Warehouse Metamodel Developer's guide", Wiley, 2003, ISBN-13: 978-0471202431

[31] Philip A Bernstein, Sergey Melnik, "Model management 2.0: manipulating richer mappings", In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data (2007), pp. 1-12.

[32] M. Alanen, I.Porres "Model interchange using OMG standards", 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005. 30 Aug.-3 Sept. 2005 Page(s): 450 – 458

[33] Java Community Process, "Java Metadata Interface (JMI) Specification – Final Release", Java Technology Specifications

[34] Banek, Marko; Jurić, Damir; Pintar Damir; Skočir, Zoran; Vranić, Mihaela; Vrdoljak, Boris. E-business infrastructure for supporting the integration of tourist services // *Proceedings ELMAR-20080 /* Grgić, Mislav; Grgić, Sonja (ur.)., Zadar: Croatian Society Electronics in Marine – ELMAR, 2008. 289-292

Životopis autora

Damir Pintar rođen je 28. listopada 1978. u Osijeku, gdje je pohađao osnovnu školu te III. gimnaziju. Nakon završene srednje škole 1996. upisuje Fakultet elektrotehnike i računarstva. Tijekom studija nagrađen je priznanjem “Josip Lončar“ za svaku godinu studiranja. Diplomirao je u srpnju 2001. godine s naglaskom na znanstveno-istraživački rad. Po završetku studija nagrađen je i brončanom plaketom“Josip Lončar“ kao jedan od najboljih studenata generacije.

U kolovozu 2001. zapošljava se na Fakultetu elektrotehnike i računarstva kao znanstveni novak na projektu “Umrežena ekonomija“ te upisuje postdiplomski studij. 2005. godine brani magistarski rad pod imena "Razvoj sučelja za elektroničko poslovanje između tvrtki".

Radi na projektima suradnje s gospodarstvom te sudjeluje u nastavi na predmetima "Baze podataka", "Organizacija obrade podataka", "Upravljanje podacima" i "Osnove elektrotehnike".

Dosadašnji interes obuhvaća moderne tehnologije vezane uz Internet, jezik XML, objektno-orijentirano programiranje, automatizaciju poslovnih procesa, uslužno orijentirane arhitekture te skladištenje podataka. Objavio je više znanstvenih radova iz područja svog znanstvenog interesa.

Kratki sažetak

Uslužno orijentirana arhitektura je široko prihvaćena kao učinkovita paradigma za uspostavljanje integracije različitih informacijskih sustava. Aplikacije zasnovane na uslužno orijentiranim principima mogu prelaziti granice platformi, operacijskih sustava i prilagođenih normi podataka, najčešće kroz korištenje tehnologije Usluga weba. S druge strane, metapodaci se također često spominju kao učinkoviti integracijski alat s obzirom na činjenicu da normizirani metapodatkovni objekti mogu sustavima pružati korisne informacije o sustavima s kojima se želi uspostaviti integracija. Ovaj pristup naziva se "integracijom zasnovanom na modelu".

Ovaj rad prikazuje istraživanje moguće sinergije između ovih dvaju principa integracije heterogenih sustava. Ono se izvodi kroz razvoj informacijskih sustava čija uloga je provođenje uslužno orijentiranog poslovnog procesa zasnovanog na metapodacima koji proširuje sustav skladištenja podataka podrškom za stvarnovremensko prikupljanje, preoblikovanje i pohranu podataka. Razvijen je formalizirani model metapodataka zasnovan na proširenju standarda CWM (engl. *Common Warehouse Metamodel*). Nadalje, specificirani su modeli usluga i obrađene moderne tehnologije podesne za implementaciju navedenog sustava (ukratko nazvanog sustav SOA RT-ETL). Konačno, razvijena je implementacija studijskog primjera koji prikazuje iskoristivost ovakvog pristupa razvoja informacijskih sustava te uz pomoć konkretnih testnih podataka dokazuje učinkovitost razvijenog sustava za pružanje stvarnovremenske podrške skladišnim sustavima u realnom poslovnom okruženju.

Short summary

Metadata-based model of Service Oriented Architecture for real-time data warehousing

Service-oriented architecture (SOA) has already been widely recognized as an effective paradigm for achieving integration of diverse information systems. SOA-based applications can cross boundaries of platforms, operation systems and proprietary data standards, commonly through the usage of Web Services technology. On the other side, metadata is also commonly referred to as a potential integration tool given the fact that standardized metadata objects can provide useful information about specifics of unknown information systems with which one has interest in communicating with, using an approach commonly called "model-based integration".

This thesis presents the result of research regarding possible synergy between those two integration facilitators. This is accomplished by developing an information system whose role is acting as a metadata-driven SOA-based business process that provides ETL (Extraction, Transformation and Loading) and metadata services to a data warehousing system in need of a real-time ETL support. A formalized metadata model is developed based on an extension of a *Common Warehouse Metamodel* standard. Furthermore, service models were specified and available technologies were analyzed which could be used for implementation of such a metadata-driven service-oriented system (named in short SOA RT-ETL system). Finally, a concrete example implementation was made showcasing proof-of-concept as well as providing test data which demonstrates the system's characteristics and shows the feasibility of using such a system for support of real-time data warehousing in a realistic business environment.

Ključne riječi

Integracija poslovnih aplikacija

Uslužno-orijentirana arhitektura

Usluge weba

Stvarnovremensko skladištenje

Integracija pogonjena modelom

Metapodaci

Prikupljanje, preoblikovanje i pohrana

Keywords

Enterprise application integration

Service-oriented architecture

Web services

Real-time data warehousing

Model-driven integration

Metadata

Extraction, transformation and loading

Prilog 1 – formalizacija metamodela kroz XML-ovske sheme

```
<xs:element name="CWMX_SOA_XMLDocument" type="CWMX_SOA_XMLDocument_type"/>

<xs:complexType name="CWMX_SOA_XMLDocument_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="URI" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="schemaURI" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="content" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="type" use="optional"/>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 1 - Shema elementa CWMX_SOA_XMLDocument

```
<xs:element name="CWMX_SOA_XSDDocument" type="CWMX_SOA_XSDDocument_type"/>
<xs:element name="CWMX_SOA_WSDLDocument" type="CWMX_SOA_WSDLDocument_type"/>
<xs:element name="CWMX_SOA_BPELDocument" type="CWMX_SOA_BPELDocument_type"/>
<xs:element name="CWMX_SOA_XSLTDocument" type="CWMX_SOA_XSLTDocument_type"/>

<xs:complexType name="CWMX_SOA_XSDDocument_type">
      <xs:complexContent>
      <xs:restriction base="CWMX_SOA_XMLDocument_type">
      <xs:sequence>
      <xs:element name="URI" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:element name="schemaURI" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="content" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string" fixed="XSD" use="required"/>
      </xs:restriction>
      </xs:complexContent>
</xs:complexType>

<!-- Ostali tipovi deriviraju se na analogni način -->
```

Slika P1- 2 - Shema preostalih XML-ovskih elemenata proširenja CWMX_SOA

```
<xs:element name="CWMX_SOA_XMLSchemaLink" type="CWMX_SOA_XMLSchemaLink_type"
/>

<xs:complexType name="CMWX_SOA_XMLSchemaLink_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="client" type="CWMX_SOA_XMLDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      <xs:element name="supplier" type="CWMX_SOA_XSDDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 3 - Shema elementa CWMX_SOA_XMLSchemaLink

```
<xs:element name="CWMX_SOA_XSLTLink"type="CWMX_SOA_XSLTLink_type" />

<xs:complexType name="CMWX_SOA_XSLTLink_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="XSLT" type="CWMX_SOA_XSLTDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      <xs:element name="sourceXSD" type="CWMX_SOA_XSDDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      <xs:choice>
      <xs:element name="targetXSD" type="CWMX_SOA_XSDDocument_type"
      minOccurs="1‟ maxOccurs="1"/>

<xs:element name="target" type="CWM_ModelElement_type"
      minOccurs="1‟ maxOccurs="1"/>
      </xs:choice>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 4 - Shema elementa CWMX_SOA_XSLTLink

```
<xs:element
name="CWMX_SOA_WSDLmessagesLink"type="CWMX_SOA_WSDLmessagesLink_type" />

<xs:complexType name="CMWX_SOA_WSDLmessagesLink_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="WSDLOwner" type="CWMX_SOA_WSDLDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      <xs:element name="XSD" type="CWMX_SOA_XSDDocument_type"
      minOccurs="1‟ maxOccurs="unbounded"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 5 - Shema elementa CWMX_SOA_WSDLDocument

```
<xs:element name="CWMX_SOA_BPELWSDLLink" type="CWMX_SOA_BPELWSDLLink_type" />

<xs:complexType name="CMWX_SOA_BPELWSDLLink_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="BPEL" type="CWMX_SOA_BPELDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      <xs:element name="WSDL" type="CWMX_SOA_WSDLDocument_type"
      minOccurs="1‟ maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 6 - Shema elementa CWMX_SOA_BPELWSDLLink

```
<xs:element name="CWMX_SOA_Expression" type="CWMX_SOA_Expression_type" />

<xs:complexType name="CWMX_SOA_Expression_type">
      <xs:complexContent>
      <xs:extension base="CWM_Expression_type">
      <xs:sequence>
      <xs:element name="params" minOccurs="0" maxOccurs="1">
            <xs:complexType>
            <xs:sequence>
            <xs:element name="param" minOccurs="1" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:attribute name="name" use="required"/>
                        <xs:attribute name="type" use="required"/>
                        <xs:attribute name="value" use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
            </xs:complexType>
      </xs:element>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 7 - Shema elementa CWMX_SOA_Expression

```
<xs:element name="CWMX_SOA_StagingTable" type="CWMX_SOA_StagingTable_type"/>

<xs:complexType name="CWMX_SOA_StagingTable_type">
      <xs:complexContent>
            <xs:extension base="CWM_ModelElement_type">
            <xs:sequence>
            <xs:element name="CWMX_SOA_Expression"
                        type="CWMX_SOA_Expression_type" minOccurs="0"/>
            <xs:element name="ColumnSet">
            <xs:complexType>
                  <xs:sequence>
                  <xs:element name="Column" maxOccurs="unbounded">
                        <xs:complexType>
                        <xs:attribute name="name" type="xs:string"
                        use="required"/>
                        <xs:attribute name="id" type="xs:string"
                        use="optional"/>
                        <xs:attribute name="type" type="xs:string"
                        use="optional"/>
                        <xs:attribute name="idref" type="xs:string"
                        use="optional"/>
                        </xs:complexType>
                  </xs:element>
                  </xs:sequence>
            </xs:complexType>
            </xs:element>
            <xs:element name="RowSet" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                  <xs:sequence>
                  <xs:element name="Row" maxOccurs="unbounded">
                        <xs:complexType>
                        <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="href"
                                    type="xs:string" use="required"/>
                              <xs:attribute name="type"
                                    type="xs:string" use="optional"/>
                        </xs:simpleContent>
                        </xs:complexType>
                  </xs:element>
                  </xs:sequence>
            </xs:complexType> </xs:element>
            </xs:sequence>
      </xs:extension>
</xs:complexContent>
</xs:complexType>
```

Slika P1- 8 - Shema elementa CWMX_SOA_StagingTable

```
<xs:element name="CWMX_SOA_WSOperation"   type="CWMX_SOA_WSOperation_type"/>

<xs:complexType name="CWMX_SOA_WSOperation_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
            <xs:element name="operationName" type="xs:string"/>
            <xs:element name="inputParams" minOccurs="1">
            <xs:complexType>
            <xs:sequence>
                  <xs:element name="param" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:attribute name="name" use="required"/>
                        <xs:attribute name="type" use="required"/>
                  </xs:complexType>
                  </xs:element>
            </xs:sequence>
            </xs:complexType>
            </xs:element>
            <xs:element name="outputParams" minOccurs="1">
            <xs:complexType>
            <xs:sequence>
                  <xs:element name="param" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:attribute name="name" use="required"/>
                        <xs:attribute name="type" use="required"/>
                  </xs:complexType>
                  </xs:element>
            </xs:sequence>
            </xs:complexType>
            </xs:element>
            <xs:element name="inputXSD"
                  type="CWMX_SOA_XSDDocument_type"
                  minOccurs="1" maxOccurs="1"/>
            <xs:element name="outputXSD"
                  type="CWMX_SOA_XSDDocument_type"
                  minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            </xs:complexType>
            </xs:element>
      </xs:sequence>
      </xs:extension> </xs:complexContent>
</xs:complexType>
```

Slika P1- 9 - Shema elementa CWMX_SOA_WSOperation

```
<xs:element name="CWMX_SOA_WebService" type="CWMX_SOA_WebService_type"/>

<xs:complexType name="CWMX_SOA_WebService_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
            <xs:element name="operations" minOccurs="1" maxOccurs="1">
            <xs:complexType>
            <xs:sequence>
                  <xs:element name="WSoperation"
                        type="CWMX_SOA_WSOperation_type"
                        minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            </xs:complexType>
            </xs:element>
            <xs:element name="WSDL"
                  type="CWMX_SOA_WSDLDocument_type"
                  minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="URI" use="optional"/>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 10 - Shema elementa CWMX_SOA_WebService

```
<xs:element name="CWMX_SOA_Extraction" type="CWMX_SOA_Extraction_type"/>

<xs:complexType name="CWMX_SOA_Extraction_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
            <xs:element name="CWMX_SOA_Expression"
                  type="CWMX_SOA_Expression_type"
                  minOccurs="0" maxOccurs="1"/>
            <xs:element name="CWMX_SOA_RDBConnection"
                        type="CWMX_SOA_RDBConnection_type"
                        minOccurs="0" maxOccurs="1"/>
            <xs:element name="input">
                  <xs:complexType>
                  <xs:sequence>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type" minOccurs="0"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                  </xs:complexType>
            </xs:element>
            <xs:element name="output">
                  <xs:complexType>
                  <xs:sequence>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type" minOccurs="0"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                  </xs:complexType>
            </xs:element>
            <xs:element name="extractionService"
                  type="CWMX_SOA_WebService_type"
                  minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="extractionOperation"
                  type="CWMX_SOA_WSOperation_type"
                  minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 11 - Shema elementa CWMX_SOA_Extraction

```
<xs:element name="CWMX_SOA_RDBConnection" type="CWMX_SOA_RDBConnection_type"
/>
<xs:complexType name="CWMX_SOA_RDBConnection_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
      <xs:element name="JNDIName" type="xs:string"    minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="driverName" type="xs:string"  minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="url" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="username" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      <xs:element name="password" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 12 - Shema elementa CWMX_SOA_RDBCConnection

```
<xs:element name="CWMX_SOA_Transformation"
      type="CWMX_SOA_Transformation_type"/>

<xs:complexType name="CWMX_SOA_Transformation_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
            <xs:element name="CWMX_SOA_Expression"
                  type="CWMX_SOA_Expression_type"
                  minOccurs="0" maxOccurs="1"/>
            <xs:element name="XSLTtemplate"
                        type="CWMX_SOA_XSLTDocument_type"
                        minOccurs="0" maxOccurs="1"/>
            <xs:element name="input">
                  <xs:complexType>
                  <xs:sequence>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type" minOccurs="0"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                  </xs:complexType>
            </xs:element>
            <xs:element name="output">
                  <xs:complexType>
                  <xs:sequence>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type" minOccurs="0"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                  </xs:complexType>
            </xs:element>
            <xs:element name="transformationService"
                  type="CWMX_SOA_WebService_type"
                  minOccurs="0" maxOccurs="1"/>
            <xs:element name="transformationOperation"
                  type="CWMX_SOA_WSOperation_type"
                  minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 13 - Shema elementa CWMX_SOA_Transformation

```
<xs:element name="CWMX_SOA_Storage" type="CWMX_SOA_Storage_type"/>

<xs:complexType name="CWMX_SOA_Storage_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
      <xs:sequence>
            <xs:element name="CWMX_SOA_Expression"
                  type="CWMX_SOA_Expression_type"
                  minOccurs="0" maxOccurs="1"/>
            <xs:element name="CWMX_SOA_RDBConnection"
                        type="CWMX_SOA_RDBConnection_type"
                        minOccurs="0" maxOccurs="1"/>
            <xs:element name="input">
                  <xs:complexType>
                  <xs:sequence>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type" minOccurs="0"
                        maxOccurs="unbounded"/>
                  </xs:sequence>
                  </xs:complexType>
            </xs:element>
            <xs:element name="storageService"
                  type="CWMX_SOA_WebService_type"
                  minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 14 - Shema elementa CWMX_SOA_Storage

```
<xs:element name="CWMX_SOA_ETL_BPELProcess"
      type="CWMX_SOA_ETL_BPELProcess_type"/>

<xs:complexType name="CWMX_SOA_ETL_BPELProcess_type">
      <xs:complexContent>
      <xs:extension base="CWMX_SOA_ETLProcess_type">
      <xs:sequence>
      <xs:element name="BPEL" type="CWMX_SOA_BPELDocument_type"
            minOccurs="0" maxOccurs="1"/>
      <xs:element name="implementationWS" type="CWMX_SOA_WebService_type"
                  minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 15 - Shema elementa CWMX_SOA_ETL_BPELProcess

```
<xs:element name="CWMX_SOA_ETLProcess" type="CWMX_SOA_ETLProcess_type"/>

<xs:complexType name="CWMX_SOA_ETLProcess_type">
      <xs:complexContent>
      <xs:extension base="CWM_ModelElement_type">
            <xs:choice maxOccurs="unbounded">
                  <xs:element name="CWMX_SOA_Extraction"
                              type="CWMX_SOA_Extraction_type"
                        minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="CWMX_SOA_Transformation"
                        type="CWMX_SOA_Transformation_type"
                        minOccurs ="1" maxOccurs="unbounded"/>
                  <xs:element name="CWMX_SOA_Storage"
                        type="CWMX_SOA_Storage_type"
                        minOccurs="1" maxOccurs="unbounded"/>
                  <xs:element name="CWMX_SOA_StagingTable"
                        type="CWMX_SOA_StagingTable_type"
                        minOccurs="1" maxOccurs="unbounded"/>
            </xs:choice>
      </xs:extension>
      </xs:complexContent>
</xs:complexType>
```

Slika P1- 16 - Shema elementa CWMX_SOA_ETLProcess

Prilog 2 – XML-ovski oblik metapodataka procesa studijskog primjera

```
<?xml version="1.0" encoding="UTF-8"?>

<CWMX_SOA_ETLProcess xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Documents and
Settings\Administrator\Desktop\doktorat\xslltt\doktorat.xsd">

      <!-- Pridruzena inicijalizacijaka vrijednost za identifikaciju
           novih podataka -->
      <taggedValue id="2810">
            <name>racun</name>
            <value>14</value>
      </taggedValue>

      <!—Element opisa prikupljanja podataka iz sustava POS -->

      <CWMX_SOA_Extraction xmlns:xsi=http://www.w3.org/2001/XMLSchema-
            instance" id="1423">

            <!-- SQL-ovski izraz za prikupljanje -->

            <CWMX_SOA_Expression>
                  <body>
                  SELECT racun, proizvod, kupac,  cijena, kolicina
                              FROM RACUN
                              WHERE RACUN=[1213]:
                  </body>
                  <language>SQL99</language>
                  <params>
                        <param id="1213" href="2810"/>
                  </params>
            </CWMX_SOA_Expression>

            <!--  opis pristupa bazi podataka -->

            <CWMX_SOA_RDBConnection>
                  <driverName>oracle.jdbc.driverOracleDriver</driverName>
                  <url>jdbc:oracle:thin:hr/hr@salesserver.sales.hr
                        :1521/ORCL</url>
                  <username>posDataBase</username>
                  <password>********</password>
            </CWMX_SOA_RDBConnection>
```

```
      <!--  izlazni medjuspremnik podataka -->

      <output>
            <CWMX_SOA_StagingTable href="76"/>
      </output>

</CWMX_SOA_Extraction>

<!--  Element opisa XSLT-ovske transformacije -->

<CWMX_SOA_Transformation id="1521">

      <!--  predlozak transformacije -->

      <XSLTTemplate type="XSLT">
      <content xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
            <xsl:stylesheet version="1.0"
                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

            <xsl:output method="xml" indent="yes"/>
            <xsl:template match="/">
            <CWMX_SOA_StagingTable id="77">
            <ColumnSet id="154">
                  <Column type="xs:string" id="177" name="kupac_PK"/>
                  <Column type="xs:string" id="180"
                        name="kategorija_FK"/>
                  <Column type="xs:string" id="181" name="kupac_FK"/>
            </ColumnSet>
            <RowSet>
                  <Row href="177"
                        value="{//RowSet[1]/Row[@href='177']/@value}"/>
                  <Row href="180" value="null"/>
                  <Row href="181" value="null"/>
            </RowSet>
            </CWMX_SOA_StagingTable>
            </xsl:template>
            </xsl:stylesheet>
      </content>
      </XSLTTemplate>

      <!--  ulazne i izlazne tablice -->

      <input><CWMX_SOA_StagingTable href="76"/></input>

      <output><CWMX_SOA_StagingTable href="77"/></output>

</CWMX_SOA_Transformation>
```

```
<CWMX_SOA_Extraction xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" id="1424">

      <CWMX_SOA_Expression>
            <body>SELECT KUPAC_PK, KATEGORIJA_FK
                        FROM KORISNIK
                        WHERE KUPAC_PK=[177];
            </body>
            <language>SQL99</language>
            <params>
                  <param href="77"/>
            </params>
      </CWMX_SOA_Expression>

      <CWMX_SOA_RDBConnection>
            <driverName>oracle.jdbc.driverOracleDriver</driverName>
            <url>jdbc:oracle:thin:hr/hr@salesserver.sales.hr
                  :1521/ORCL</url>
            <username>CRMDataBase</username>
            <password>********</password>
      </CWMX_SOA_RDBConnection>

      <output> <CWMX_SOA_StagingTable href="77"/> </output>

</CWMX_SOA_Extraction>

<CWMX_SOA_Extraction xmlns:xsi=http://www.w3.org/2001/XMLSchema-
      instance" id="1424">

      <CWMX_SOA_Expression>
            <body>SELECT KUPAC_NK, KUPAC_DW_PK
                        FROM KORISNIK_DIM
                        WHERE KUPAC_NK=[177];
            </body>
            <language>SQL99</language>
            <params>
                  <param href="77"/>
            </params>
      </CWMX_SOA_Expression>

      <CWMX_SOA_RDBConnection>
            <driverName>oracle.jdbc.driverOracleDriver</driverName>
            <url>jdbc:oracle:thin:hr/hr@salesserver.sales.hr
                  :1521/ORCL</url>
            <username>DWDataBase</username>
            <password>********</password>
      </CWMX_SOA_RDBConnection>
```

```
      <output> <CWMX_SOA_StagingTable href="77"/> </output>

</CWMX_SOA_Extraction>

<CWMX_SOA_Extraction xmlns:xsi=http://www.w3.org/2001/XMLSchema-
      instance" id="1425">

      <output> <CWMX_SOA_StagingTable href="78"/> </output>

      <extractionService href="7712"
            URI="http://dinara.tel.fer.hr:8080/
                  NTPServiceWSDL/NTPWSDLPort"/>

      <extractionOperation href="8109" />

</CWMX_SOA_Extraction>

<CWMX_SOA_Transformation id="1522">

      <XSLTTemplate type="XSLT">
      <content xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

      <xsl:stylesheet version="1.0"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output method="xml" indent="yes"/>
      <xsl:template match="/">

      <CWMX_SOA_StagingTable id="76">
      <ColumnSet id="153">
            <Column type="xs:string" id="175" name="racun"/>
            <Column type="xs:string" id="176" name="proizvod"/>
            <Column type="xs:string" id="177" name="kupac"/>
            <Column type="xs:float" id="178" name="cijena"/>
            <Column type="xs:int" id="179" name="kolicina"/>
            <Column type="xs:string" id="180" name="kategorija_FK"/>
            <Column type="xs:string" id="181" name="kupac_PK"/>
            <Column type="xs:date" id="182" name="timestamp"/>
      </ColumnSet>

      <xsl:for-each select="//RowSet[@href=153]">
      <RowSet>
            <Row href="175" value="{./Row[@href=175]/@value}"/>
            <Row href="176" value="{./Row[@href=176]/@value}"/>
            <Row href="177" value="{./Row[@href=177]/@value}"/>
            <Row href="178" value="{./Row[@href=178]/@value}"/>
            <Row href="179" value="{./Row[@href=179]/@value}"/>
```

```
                <Row href="180"
                      value="{/input/CWMX_SOA_StagingTable[@id=77]
                            /RowSet/Row[@href=180]/@value}"/>
                <Row href="181"
                      value="{/input/CWMX_SOA_StagingTable[@id=77]
                            /RowSet/Row[@href=181]/@value}"/>
                <Row href="182"
                      value="{/input/CWMX_SOA_StagingTable[@id=78]
                            /RowSet/Row[@href=182]/@value}"/>
          </RowSet>
          </xsl:for-each>
          </CWMX_SOA_StagingTable>
          </xsl:template>
          </xsl:stylesheet>
          </content>
          </XSLTTemplate>

          <input>
                <CWMX_SOA_StagingTable href="76"/>
                <CWMX_SOA_StagingTable href="77"/>
                <CWMX_SOA_StagingTable href="78"/>
          </input>

          <output><CWMX_SOA_StagingTable href="76"/></output>

    </CWMX_SOA_Transformation>

    <CWMX_SOA_Storage id="1631">

          <CWMX_SOA_Expression>
                <body>INSERT INTO RACUN_FACT VALUES
                      [175],[176],[177],[178],[179],[180],[181],[182];
                </body>
                <language>SQL99</language>
                <params>
                      <param href="76"/>
                </params>
          </CWMX_SOA_Expression>

          <CWMX_SOA_RDBConnection>
                <driverName>oracle.jdbc.driverOracleDriver</driverName>
                <url>jdbc:oracle:thin:hr/hr@salesserver.
                      sales.hr:1521/ORCL</url>
                <username>RTDWDataBase</username>
                <password>********</password>
          </CWMX_SOA_RDBConnection>
```

```
            <input><CWMX_SOA_StagingTable href="76"/></input>

      </CWMX_SOA_Storage>

      <CWMX_SOA_StagingTable id="76">
            <ColumnSet id="153">
                  <Column type="xs:string" id="175" name="racun"/>
                  <Column type="xs:string" id="176" name="proizvod"/>
                  <Column type="xs:string" id="177" name="kupac"/>
                  <Column type="xs:float" id="178" name="cijena"/>
                  <Column type="xs:int" id="179" name="kolicina"/>
                  <Column type="xs:string" id="180" name="kupac_kat_FK"/>
                  <Column type="xs:string" id="181" name="kupac_PK"/>
                  <Column type="xs:string" id="182" name="timestamp"/>
            </ColumnSet>
      </CWMX_SOA_StagingTable>

      <CWMX_SOA_StagingTable id="77">
            <ColumnSet id="154">
                  <Column type="xs:string" id="177" name="kupac_PK"/>
                  <Column type="xs:string" id="180" name="kategorija_FK"/>
                  <Column type="xs:string" id="181" name="kupac_FK"/>
            </ColumnSet>
      </CWMX_SOA_StagingTable>

      <CWMX_SOA_StagingTable id="78">
            <ColumnSet id="155">
                  <Column type="xs:date" id="182" name="timestamp"/>
            </ColumnSet>
      </CWMX_SOA_StagingTable>

</CWMX_SOA_ETLProcess>
```

Slika P2 - 1 – XML-ovski oblik metapodataka ETL-ovskog procesa iz studijskog primjera