UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

# ZAVRŠNI (MASTER) RAD

## Primena BizTalk servera u orkestraciji procesa javne uprave

Ime i prezime studenta:
Srđan Mlađenović

Ime i prezime mentora:
Slađan Babarogić

Beograd
Septembar, 2017.

# Sadržaj

# Sadržaj slika

# Uvod

Javna uprava predstavlja sinergiju nekoliko informacionih sistema državnih organa. Sama činjenica da je sistem sinergija drugih sistema automatski znači kompleksne procedure komunikacije. Funkcionisanje javne uprave koja je na raspolaganju građanima zasniva se na izdavanju dokumenata građanima, prikupljanje istih od strane građana, realizacija kompleksnih procedura izvršavanja određenih upravnih postupaka definisanih zakonom o opštem upravnom postupku. Ovaj mukotrpan proces iziskuje mnogo potrošenog vremena na čekanje u redovima, obimnu papirologiju i veoma puno grešaka koje su neminovno prouzrokovane ljudskim faktorom.

Ono čemu se teži kako bi se eliminisali gorenavedeni problemi jeste razvoj elektronske javne uprave gde će građani korišćenjem računara i interneta moći da završe veliki deo administracije sa državnim organima. Samom upotrebom računara smanjuje se učešće ljudi u celokupnom procesu, što dovodi do umanjenja šanse da dođe do greške prouzrokovane ljudskim faktorom. Kako građani većinu birokratije obavljaju iz fotelje, samim tim izbegava se čekanje u redu koje može biti veoma iscrpljujuće.

Kao jedan od težih poduhvata uvođenja elektronske uprave jeste razvoj pojedinačnih informacionih sistema institucija koje će učestvovati u sinergiji kao i sinhronizacija i održavanje istih. Kako bi svaka institucija trebalo da sama održava svoj sistem koji se sa aspekta elektronske uprave posmatra kao interfejs. Dakle, ono što ostaje kao problem jeste orkestracija procesa koji komuniciraju sa ovim sistemima. Rezultat rada bi trebalo da bude rešenje problema sinhronizacije procesa komunikacije.

# Državna uprava

## Osnovni činioci javne uprave

Državna uprava Republike Srbije predstavlja skup državnih organa koji sprovode birokratske procedure definisane zakonima. Nadležnosti državnih organa su definisane Ustavom, zakonima i drugim propisima (Zakon o državnoj upravi, 79/2005, 101/2007, 95/2010, 99/2014). Jedno moderno društvo je uređeno državnim aparatom koji definiše pravila koja se moraju poštovati prilikom sprovođenja određenih radnji državne uprave. Ovako definisana pravila i procedure najčešće su veoma velika i kompleksna da bi se njima upravljalo sa jednog mesta. Distribucija nadležnosti je jedan od načina lakšeg upravljanja celokupnom birokratskom mašinerijom.

Kao glavni akteri javne uprave javljaju se zakonodavne institucije, izvršne institucije i građanstvo. Zakonodavne institucije su zadužene za donošenje zakona i propisa koji definišu ingerencije izvršnih institucija. Izvršne institucije deluju u okviru zakonom definisanih okvira po tačno definisanim procedurama. I jedne i druge institucije rade u službi građana. O efikasnosti delovanja javne uprave stara se Ministarstvo državne uprave i lokalne samouprave.

Samo ministarstvo je podeljeno na nekoliko sektora koji se staraju o različitim domenima rada javne uprave među kojima su direkcija za elektronsku upravu, sektor za sistem lokalne samouprave, sektor za matične knjige i registre itd (struktura, 2017). Kako je teritorija Republike Srbije prevelika da bi jedna institucija vodila računa o izvršavanju svih procedura, tako se osnivaju upravni okruzi, tj. autonomne jedinice. Ovako podeljenim okruzima se distribuiraju nadležnosti i oni predstavljaju organe za vršenje određenih poslova izvan sedišta organa državne uprave (Zakon o državnoj upravi, 79/2005, 101/2007, 95/2010, 99/2014). Upravni okruzi se najčešće definišu na teritoriji opštine ili okruga.

## Međusobne veze državnih organa

Svaki informacioni sistem je struktura koja gotovo nikada ne funkcioniše sama za sebe, već je u konstantnoj interakciji sa drugim sistemima. U suprotnom, ne bi imao smisla. Državne institucije predstavljaju informacione sisteme. Mnoge od procedura o upravnom postupku su definisane tako da njihovo izvršavanje podrazumeva uključenje više institucija. Na primer, ukoliko želite da podnesete  zahtev za izdavanje uverenja o prosečnom mesečnom prihodu po članu porodice radi ostvarenja prava na učeničke stipendije i studentske kredite neophodno je da pribavite potvrdu o redovnom školovanju(studiranju), dokaz o visini primanja, uverenje poreske uprave o evidenciji poreskog obveznika itd (opis_usluge, 2017). Sva dokumenta koja su neophodna za podnošenje ovog zahteva ne izdaje jedna institucija. Dakle potrebno je otići na nekoliko šaltera u različitim organima, sačekati u redu, podneti zahtev za dokument, sačekati da dokument bude izrađen, pa ponovo doći po dokument i tako u nekoliko iteracija prikupiti dokumenta. Ovde je očigledna interakcija između sistema različitih organa državne uprave.

Komunikacija između dva entiteta se zasniva na razmeni poruka. Kako bi se entiteti razumeli mora biti definisana jasna struktura poruka. Poruke u pomenutom primeru predstavljaju dokumenta(zahtevi) koji imaju definisanu strukturu, tj. skup podataka koji je neophodan. Naime, informacioni sistem da bi mogao da komunicira sa ostalim sistemima mora poštovati strukturu koju je definisao za tu komunikaciju.

## Odnos javne uprave sa građanima

Državna uprava se sastoji iz javne uprave koja je u neposrednoj interakciji sa građanima i dela uprave koji služi internoj administraciji i pružanju podrške javnom delu uprave. Taj administrativni deo uprave je jako kompleksan birokratski proces izvršavanja zakonom definisanih procedura. Pored toga, zadužen je i za dobavljanje sredstava za rad celokupne uprave i svih drugih materijalnih stvari koje su neophodne za nesmetano funkcionisanje državne uprave.

Sa druge strane javna uprava predstavlja interfejs[1] državne uprave ka građanima. Kako je državna uprava u svakodnevnoj interakciji sa građanima ona mora biti sposobna da obradi velike količine zahteva i da izvrši veliki broj birokratskih procedura. Srbija ima oko sedam miliona stanovnika (Republika Srbija) i može se samo zamisliti koliko se transakcija u javnoj upravi obavi na dnevnom nivou. Ipak, skoro svaki građanin je u manjoj ili većoj meri u interakciji sa državnom upravom. Bilo koji zvanični dokument o identitetu ličnosti ili pravnog subjekta je nemoguće izvaditi van kontrole od strane državne uprave, a te dokumente svako mora da poseduje. Zamislite koliki je broj takvih zahteva koji se podnesu preko državne uprave. Kao prvo, ogroman broj zahteva zahteva veliki broj službenika koji će raditi na obradi istih. Takođe javljaju se i veliki materijalni troškovi izdavanja dokumenata. Ovi procesi iziskuju i dosta utrošenog vremena na prikupljanje i obradu zahteva. Građani, podnosioci zahteva, najčešće najviše vremena provode čekajući u redovima, čime ceo proces postaje nepodnošljiv. Pored toga, postoji dosta prostora za kršenje i obilaženje zakonskih pravila i ograničenja. Pored pomenutih javljaju se još mnogobrojni problemi kao što su izdavanje duplih dokumenata, podnošenje duplih zahteva prilikom gubljenja istih, nerazumevanje u komunikaciji itd.

Da bi se izbegli pomenuti problemi ili barem smanjili, uveden je informacioni sistem pod nazivom eUprava. Ovaj sistem predstavlja elektronske servise javne uprave u vidu portala koji je dostupan svim građanima.

## Elektronska javna uprava

Živimo u vremenu informaciono-komunikacionih tehnologija koje u velikoj meri olakšavaju funkcionisanje jednog modernog društva. Ono čemu se u današnje vreme teži jeste da se veliki deo administracije automatizuje po jasno definisanim procedurama kako bi se smanjila mogućnost greške. Ovde se sa tradicionalnih pristupa, gde se celokupno poslovanje privrednog subjekta nalazi u ,,sveskama'', prelazi na digitalizovane podatke koji se čuvaju u velikim bazama

[1] Interfejs u ovom kontekstu predstavlja tačku koja je izložena za treća lica, tj. tačku dodira sistema državne uprave sa građanstvom

podataka. Sam proces ,,digitalizacije'' podataka može biti veoma naporan. Korišćenje digitalizovanih podataka ima dosta prednosti u odnosu na tradicionalni pristup. Prva beneficija jeste brzina obrade podataka koju u ovom slučaju vrše računari. Mogućnost pojave greške se svodi na minimum. I kao treća jako bitna prednost jeste to da je gotovo nemoguće zaobići definisane procedure po kojima je projektovan celokupni sistem.

Kako bi se iskoristile pomenute prednosti informacionih sistema, veliki deo javne uprave je prebačen na digitalnu obradu podataka. Veliki doprinos digitalizaciji obrade podataka daje internet koji obezbeđuje udaljen pristup mašinama na kojima se vrši obrada. U današnje vreme je internet dostupan skoro svakom savremenom ljudskom biću i u velikoj meri olakšava svakodnevni život.

Portal javne uprave predstavlja internet aplikaciju koja nudi građanima informacije, pristup bitnim sadržajima, kao i najnovije vesti vezane za državnu upravu. Portal omogućava građanima da bez odlaska u određene ustanove i čekanja u redu izvrše usluge koje ta ustanova pruža. Na primer, preko portala elektronske uprave možete podneti zahtev za izdavanje izvoda iz matične knjige rođenih koje stiže na kućnu adresu. Podnošenje zahteva se svodi na nekoliko klikova mišem, dok je u pozadini robustna mašinerija koja izvršava gomilu instrukcija. Da biste, na primer, ovaj isti zahtev podneli u opštini potrebno je da odete u opštinu, sačekate dok ne dođete na red na šalteru, odete da uplatite određeni iznos, nakon toga se opet vratite na šalter da biste dobili pečatiran dokument koji je vaš izvod iz matične knjige rođenih. Nakon pomenutog primera, razlika između tradicionalnog i novog pristupa je očigledna, bar u utrošenom vremenu.

## Osnovni procesi javne uprave

Institucije javne uprave pružaju širok dijapazon usluga građanima Republike Srbije. Proces pružanja pojedinačne usluge je definisan zakonom o upravnom postupku u kojem se definišu jasni koraci pružanja usluge, kao i potrebna dokumenta koja je potrebno imati.

Informacioni sistem elektronske uprave je projektovan tako da se prvo definišu šabloni usluga koje će biti na raspolaganju, tj. prvo se definiše specifikacija usluge. Usluge se izvršavaju po jasno utvrđenim šablonima i predstavljaju jednu instancu šablona usluga.

### Proces kreiranja usluga

#### Definisanje šablona usluga

Kao što je već pomenuto, usluge se kreiraju na nivou specifikacije. Dakle, šablon usluge predstavlja jasno definisanu specifikaciju usluge. To znači da je na šablonu usluge definisano sve što je korisniku usluge potrebno da bi on bez ikakvih nedoumica umeo da podnese zahtev koji ta usluga omogućava. Na samom takozvanom ,,generatoru" usluga može se definisati kako će usluga biti prikazana korisniku. Sama usluga se sastoji od tri dela. Prvi deo je informativni deo gde se definišu paragrafi usluga, tj. pasusi koji predstavljaju informacije o tome koja institucija

pruža pomenutu uslugu, nadzorni organ, pravni osnov za pružanje usluge i druge korisne informacije.

Kada su definisane opšte informacije, sledi definisanje dokumenata koje je potrebno priložiti uz izvršavanje usluga. Spisak dokumenata koje je potrebno priložiti je definisan zakonom o upravnom postupku. S obzirom na to da se u ovom koraku definiše šablon izvršavanja usluge tako se definiše i šablon dokumenta koji je potrebno priložiti. Ovde se veoma vodi računa o tome kada će se i kako definisati usluge. Znači da se retko dešavaju izmene ovih šablona, jer su one kreirane u skladu sa zakonom. Izmene su verovatne kada dođe do izmene zakona i kada su te izmene tolikog obima da se zahteva recimo dostavljanje novog dokumenta uz uslugu itd.

### Dodeljivanje prava na izvršavanje usluga

Definisanjem ingerencija institucija, jasno je specificirano koje usluge određena institucija može da pruža. Dakle, potrebno je kreirane usluge dodeliti određenim institucijama na izvršavanje. Znači usluge izvršava institucija, a ne pojedinac. U okviru institucija su zaposleni službenici koji rešavaju zahteve građana.

Drugi vid dodele prava su prava na pozivanje servisa drugih institucija koje ove izlažu. Na primer, u okviru sistema jedne institucije postoji potreba za pozivima servisa drugih institucija. U zavisnosti od ugovora potpisanog između tih institucija servisi mogu biti dodeljeni na korišćenje ili se može ukinuti to pravo. Tako, na primer, isti servis može biti vidljiv iz jednog sistema, dok iz drugog može biti potpuno nevidljiv. O načinima autentikacije i dodeljivanja prava će biti reči kasnije.

## Proces podnošenja zahteva

### Autentikacija i autorizacija korisnika

Sistem koji se bavi manipulacijom ličnim podacima je jako osetljiv i neophodno je da bude veoma dobro zaštićen. Bitno je obezbediti identifikaciju korisnika koji pristupa sistemu. Takođe je od velike važnosti voditi evidenciju o akcijama korisnika koji je prijavljen na sistem.

Autorizacija predstavlja proces utvrđivanja identiteta korisnika, identifikacija uloge korisnika i utvrđivanje dozvoljenih akcija koje korisnik može izvršiti na sistemu. Sa druge strane, autentikacija korisnika predstavlja akciju utvrđivanja identiteta korisnika, tj. da li je korisnik koji želi da izvrši akciju na sistemu registrovan na istom. Postoji nekoliko načina na koji se korisnik može prijaviti na sistem.

Prvi način jeste prijava uz pomoć korisničkog imena i lozinke. Prilikom registracije na sistem korisnik unosi opšte podatke o sebi. Najčešće su to ime i prezime, imejl adresa, broj godina. Pored opštih podataka o sebi, korisnik je u dužnosti da unese korisničko ime, odnosno jedinstveno ime na nivou sistema kojim će se on identifikovati u tom sistemu. Pored korisničkog imena zahteva se i lozinka. Lozinka predstavlja samo korisniku poznat niz karaktera koje unosi prilikom prijave na sistem na kojem se registrovao. Jako je bitno da korisnik čuva u tajnosti svoju

lozinku kako ne bi došlo do zloupotrebe od strane drugih lica. Ovaj način autentikacije pruža visok nivo bezbednosti i najrasprostranjeniji je. Međutim može doći do problema ukoliko korisnik zaboravi svoju lozinku ili ukoliko neko zloupotrebi lozinku.

Prijavljivanje sertifikatom je drugi način autentikacije. Sertifikate izdaju ovlašćene sertifikacione institucije državne uprave (Sl. list SRJ) i mogu biti izdati na usmeni ili pismeni zahtev. Digitalni sertifikat je elektronski dokument koji predstavlja digitalne podatke o građaninu i može se posmatrati kao digitalna lična karta. Da bi se sertifikat očitao na računaru neophodno je imati čitač sertifikata. Međutim, nije samo dovoljno staviti sertifikat učitač da bi se očitali lični podaci o vlasniku sertifikata, već je potrebno i uneti četvorocifreni PIN kod koji je poznat samo vlasniku sertifikata. Autentikacija sertifikatom je verodostojnija i smanjuje šansu da dođe do zloupotrebe. Čak i da neko ko nije vlasnik kartice na neki način dođe do sertifikiata, mala je šansa da će doći i do pin koda bez kojeg je sertifikat beskoristan.

Treći način prijave na sistem jeste anonimna prijava. Dakle, moguće je neke akcije izvršavati bez autentikacije korisnika. Akcije koje dozoljavaju pozive anonimnih korisnika su najčešće informativnog karaktera.

### Kreiranje poziva eksternih servisa

Većina eksternih servisa sa kojima se vrši integracija zahtevaju autorizovane pozive. Što znači da je nemoguće obaviti poziv istih bez nekog vida autorizacije. Kada su u pitanju servisi državne uprave zahteva se najviši nivo autorizacije korisnika. Razlog je taj što servisi komuniciraju sa bazama podataka u kojima se čuvaju lični podaci o građanima, a servisi su ti koji dostavljaju određene skupove podataka.

Da bi se obavio poziv eksternih sistema bitno je da je eksterni sistem registrovao sistem koji ima ulogu klijenta. Pozivi ovog tipa predstavljaju komunikaciju dve kompanije i pozivi su autorizovani na nivou kompanije, a ne na nivou pojedinačnog korisnika. Sistem klijent vodi računa o tome ko će se prijaviti na sistem. Kada se korisnik prijavio na sistem akcenat je na autorizacionim mehanizmima da dozvole ili ukinu korisniku pravo na poziv servisa eksternog sistema.

Postoji više načina na koje je moguće dodeliti prava poziva servisa na nivou sistema. Prvi način jeste omogućavanje VPN[2] konekcije. Drugi način jeste izdavanje sertifikata koji će se slati prilikom obavljanja svakog poziva. Eksterni klijent će na osnovu prikačenog sertifikata dozvoljavati ili odbijati zahtev.

## Proces administracije institucija i korisnika

### Administracija institucija

Državna uprava je sistem koji integriše nekoliko institucija. Gotovo nijedan postupak koji zahteva građanin ne može biti realizovan posredstvom samo jedne institucije, time je prethodna

[2] VPN(*Virtual Private Connection)* je tehnologija koja obezbeđuje sigurnu kriptovanu konekciju između dve mašine

tvrdnja dokazana. Dakle, neophodno je obezbediti mehanizam administriranja državnih ustanova i institucija koje mogu biti učesnici u postupcima koje realizuje sistem.

Jasno je da nisu sve institucije kompetentne da izvršavaju sve definisane postupke. Pored toga ne postoji zakonski osnov po kojem bi svaka institucija sve radila. To bi značilo da ne postoji podela nadležnosti po institucijama. Dakle, kada su definisane usluge(šabloni usluga) potrebno je obezbediti neki vid dodele prava instituciji na izvršavanje određene usluge. Mehanizam kojim se to obezbeđuje je relativno jednostavan i predstavlja agregaciju između entiteta usluge i institucije, gde jedna institucija može izvršavati više usluga i jedna usluga može biti izvršena od strane više institucija. Pored dodele usluga, neophodno je i definisati korisnike sistema.

## Administracija korisnika

Pored dodele usluga, neophodno je i definisati korisnike sistema. Korisnici sistema su službenici u raznim institucijama koje implementiraju sistem. Potrebno je definisati način autorizacije korisnika na sistemu. Kako sistem manipuliše podacima o ličnosti građana, neophodno je obezbediti najviši nivo bezbednosti. Već je pomenuto da najviši nivo autorizacije predstavlja prijava sertifikatom.

Kada su u pitanju uloge korisnika na sistemu postoje uloge izvršivača usluga i administratora. Izvršivači su već pomenuti službenici u institucijama koji obrađuju zahteve. Administratori sistema predstavljaju službenike institucija koja je nadležna svim institucijama koje koriste sistem. Ovlašćenja koja ima administrator su sledeća: dodavanje novog korisnika u sistem, kreiranje novog šablona usluga, ažuriranje korisnika i šablona, pregled izvršenih zahteva itd.

## Administracija uloga i prava korisnika i institucija

Proverom da li je korisnik prijavljen na sistem nije izvršena kompletna autorizacija koja dodatno podrazumeva i proveru uloge korisnika i akcija koje on sme da izvršava. Prilikom kreiranja korisnika, potrebno je pored osnovnih podataka o korisniku izabrati ulogu u sistemu koju će korisnik imati. S obzirom na to da jedan korisnik može imati više uloga, tj. može biti i administrator i izvršivač zahteva, kreiraće se agregacija između korisnika i uloge. Akcije koje korisnik može izvršavati na sistemu se ne vezuju za konkretnog korisnika, već za ulogu u sistemu koju korisnik ima. Primenom iste metode agregacije između akcija na sistemu i uloge omogućava se kontrola akcija za određenu ulogu korisnika.

Sada kada smo definisali koje uloge i akcije na sistemu korisnik može da ima, sledi izbor institucije u kojoj je korisnik registrovan. Dodeljivanjem reference na instituciju lako se obezbeđuje integritet podataka. Ovako kreirana stuktura omogućava automatizovanu kontrolu ko sme koju akciju da izvršava.

# BizTalk server – orkestracija procesa

## Osnovni koncepti XML strukture podataka

*XML(eXtensible Markup Language)* predstavlja strukturu podataka. Dakle, ukoliko hoćemo da uspostavimo jasnu strukturu podataka, tj. da standardizujemo određene podatke sa kojima se manipuliše. Najčešća primena *xml* strukture podataka jeste u definisanju strukture dokumenata, definisanja konfiguracionih podataka, definisanje strukture poruka koje se koriste prilikom komunikacije uz pomoć određenog protokola. Takođe, *xml* se neretko koristi i kao struktura podataka u bazama podataka. *Xml* format je osmislila grupa kompanija koje su prozvane *World Wide Web Consortium(W3C)* (Ray, 2001). Nastao je kao potreba za standardizacijom. Ono što izdvaja *xml* strukturu od drugih jeste to što u nju mogu biti ugrađeni meta podaci[3].

*Xml* fajlovi imaju ekstenziju *.xml*. Ono što je specifično za *xml* strukturu jeste to da je ona hijerarhijski uređena, tj. da postoji osnovni(koreni) element koji je nadređen u hijerarhiji ostalim elementima. Kako je *xml* strogo struktuiran skup podataka koji mogu biti i tipizirani. Tipiziranost podataka znači da je moguće kreirati određene tipove podataka nad određenim domenom. Domen predstavlja skup podataka. Tipovi podataka su brojevi, datumi, tekstualne vrednosti i druge. Za razliku od *JSON*[4] tipa podataka koji nije toliko struktuiran i u kojem postoji mnogo manji broj tipova podataka.

Osnovna sintaksa kreiranja *xml* dokumenata bazira se na tzv. tagovima. Dakle, da bi se određeni podatak mogao pročitati uz pomoć *xml* parsera[5] potrebno je da se nalazi između otvorenog i zatvorenog taga. Dodatno se u okviru *xml* dokumenta može dodati određeni vid dokumentacije, ograničenja nad određeni tip vrednosti itd. *Xml* veoma podseća na *HTML(Hyper Text Markup Language)*[6]. Međutim, *xml* je struktura podataka, a *HTML* predstavlja strukturu stranice koja se prikazuje u internet pretraživaču.

### *XML* elementi i atributi

Kao što je već pomenuto, *xml* predstavlja hijerarhijsku strukturu. Osnovni element, vrhovni element u hijerarhiji, se naziva koreni element(*root node*) (Ray, 2001). Da bi *xml* dokument bio validan, tj. da bi parser umeo da protumači dokument, pored *.xml* ekstenzije fajla, prvi red u dokumentu mora biti tag koji specificira verziju *xml* jezika koji se koristi. U nastavku sledi primer jedne *xml* strukture.

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

[3] Meta podaci – podaci o podacima, tj. podaci koji opisuju podatke, daju više informacija o istim
[4] JSON – *Javascript Object Notation*, tip podataka koji je razvijen u okviru programskog jezika *Javascript* i najčešće se koristi kao osnovna struktura podataka prilikom razmene sadržaja korišćenjem *HTTP* mrežnog protokola
[5] Parser predstavlja mehanizam koji je razvijen da bi umeo da tumači određene standardizovane jezike. Parser omogućava i procesiranje podataka koji se nalaze u okviru određenog dokumenta, a koji je kreiran po jasno definisanim pravilima koje parser ume da tumači
[6] *HTML – Hyper Text Markup Language* predstavlja jezik za struktuiranje internet stranica. Takođe je baziran na tagovima i elementima. Poseduje i koreni element u koji su ugrađeni svi ostali podelementi

```
<note >
        <to cc=,,Tom’’>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

U pokaznom primeru se vidi sintaksa *xml* dokumenta. Kao što je već rečeno prvi red dokumenta predstavlja specifikaciju verzije *xml*-a koji se koristi i dodatne meta podatke kao što su standard po kojem se vrši dekodiranje, adresa do online direktorijuma koji se koriste u dokumentu itd. Više informacija o meta atributima i verzijama *xml*-a se može pronaći na linku *W3C*-a(www.w3.org). U primeru se vidi da je koreni element <*note*>. Svi ostali elementi su na nižem nivou hijerarhije u odnosu na njega. U okviru taga *to* možemo uočiti element *cc* koji predstavlja atribut. Atributi obezbeđuju dodatne informacije o elementu, napomene i slično.

## *XML* šeme

Pored pomenutih prednosti *xml* strukture, javljaju se neki nedostaci. Prvi i osnovni, jeste kako utvrditi da li je ispoštovana *xml* struktura? Čak iako je ispoštovana struktura *xml-a,* tj. ako je sintaksa korektna, kako će se evaluirati sam sadržaj dokumenta. Na primer, koristimo *xml* kao strukturu za razmenu poruka. Konvencijom je dogovoreno da poruka mora imati elemente primalac, pošiljalac, sadržaj. Međutim, ovde se javlja nekoliko problema. Prvi je kako validirati primljenu strukturu. Svaki put kada primimo *xml* poruku neophodno je da proverimo da li je zadovoljena dogovorena struktura. Ukoliko nije, takvu poruku ne bi trebalo obrađivati. Međutim, ovo rešenje podrazumeva dodatnu implementaciju logike u samoj aplikaciji. Drugi problem, nastaje prilikom obrade vrednosti. Dodatnu logiku je potrebno implementirati i za validaciju primljenih vrednosti, što predstavlja dodatno mesto na kojem je moguće doći do previda.

Da bi se izbegli pomenuti nedostaci organizacija *W3C* je kreirala nešto što se zove *xml* šema(*xml schema*). *Xml* šema predstavlja jedan oblik šablona po kojem se kreiraju *xml* dokumenti (Ray, 2001). Dakle, u gorepomenutom problemu komunikacije definisali bismo šablon poruke po kojem pošiljalac kreira poruku i po kojem primalac ume da validira poruku koji je pošiljalac kreirao. *Xml* šeme imaju ekstenziju *.xsd.*

Ukoliko napravimo paralelu između *xml* šeme i *xml* dokumenta i objektno orijentisanih programskih jezika i klasa i objekata, pandan *xml* šemi bi bila klasa, dakle definicija strukture, a pandan *xml* dokumenta bi bio objekat. Kako je objekat instanca[7] klase, tako je i *xml* dokument instanca *xml* šeme. U nastavku je dat primer *xml* dokumenta i primer *xml* šeme kojoj odgovara navedeni *xml* dokument.

[7] Instanca – jedno konkretno pojavljivanje klase, tipa, opšte definicije. Na primer, Osoba je klasa koja ima sledeće atribute: ime i prezime, jmbg, adresa, broj telefona. Instanca klase osoba je na primer: Marko Marković, 1236547895632, Beogradska 100, +38166778899

| *Xml šema* | *Xml dokument* |
|---|---|
| *<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">*<br>*<xsd:annotation>*<br>*<xsd:documentation>*<br>*Census form for the Republic of Oz*<br>*Department of Paperwork, Emerald City*<br>*</xsd:documentation>*<br>*</xsd:annotation>*<br>*<xsd:element name="census" type="CensusType"/>*<br>*<xsd:complexType name="CensusType">*<br>*<xsd:element name="censustaker" type="xsd:decimal"*<br>*minoccurs="0"/>*<br>*<xsd:element name="address" type="Address"/>*<br>*<xsd:element name="occupants" type="Occupants"/>*<br>*<xsd:attribute name="date" type="xsd:date"/>*<br>*</xsd:complexType>*<br>*<xsd:complexType name="Address">*<br>*<xsd:element name="number" type="xsd:decimal"/>*<br>*<xsd:element name="street" type="xsd:string"/>*<br>*<xsd:element name="city" type="xsd:string"/>*<br>*<xsd:element name="province" type="xsd:string"/>*<br>*<xsd:attribute name="postalcode" type="PCode"/>*<br>*</xsd:complexType>*<br>*<xsd:simpleType name="PCode" base="xsd:string">*<br>*<xsd:pattern value="[A-Z]-d{3}"/>*<br>*</xsd:simpleType>*<br>*<xsd:complexType name="Occupants">*<br>*<xsd:element name="occupant" minOccurs="1" maxOccurs="50">*<br>*<xsd:complexType>*<br>*<xsd:element name="firstname" type="xsd:string"/>*<br>*<xsd:element name="surname" type="xsd:string"/>*<br>*<xsd:element name="age">*<br>*<xsd:simpleType base="xsd:positive-integer">*<br>*<xsd:maxExclusive value="200"/>*<br>*</xsd:simpleType>*<br>*</xsd:element>*<br>*</xsd:complexType>*<br>*</xsd:element>*<br>*</xsd:complexType>*<br>*</xsd:schema>* | *<census date="**1999-04-29**">*<br>*<censustaker>**738**</censustaker>*<br>*<address>*<br>*<number>**510**</number>*<br>*<street>**Yellowbrick Road**</street>*<br>*<city>**Munchkinville**</city>*<br>*<province>**Negbo**</province>*<br>*</address>*<br>*<occupants>*<br>*<occupant status="**adult**">*<br>*<firstname>**Floyd**</firstname>*<br>*<surname>**Fleegle**</surname>*<br>*<age>**61**</age>*<br>*</occupant>*<br>*<occupant>*<br>*<firstname>**Phylis**</firstname>*<br>*<surname>**Fleegle**</surname>*<br>*<age>**52**</age>*<br>*</occupant>*<br>*<occupant>*<br>*<firstname>**Filbert**</firstname>*<br>*<surname>**Fleegle**</surname>*<br>*<age>**22**</age>*<br>*</occupants>*<br>*</census>* |

Dakle, u šemama se definiše čvrsta struktura, u kojoj se definišu elementi i atributi koje mora da ima *xml* dokument da bi bio validan, dokumentacija, korisnički definisani tipovi podataka itd. Ono što je značajno pomenuti za definisanje šema jeste to da na internetu postoji repozitorijum predefinisanih tipova podataka koji se može referencirati u dokumentu. To je moguće odraditi na sledeći način, *xmlns:xsd="http://www.w3.org/1999/XMLSchema"*. *Xsd* je prefiks kojim se referencira *online xml* repozitorijum, koji se koristi kako bi se obeležile rezervisane reči, a koje su vezane za šemu. Pomenuti, inicijalni element šeme je obavezan i uvek se navodi. Pored pomenutog *online* repozitorijuma moguće je navesti i referencirati druge, korisnički definisane šeme sa tipovima podataka. Naredni element šeme(*annotation*) je opcioni, i koristi se kako bi se obezbedile dodatne informacije o šemi, npr. za kratak opis namene šeme itd (Ray, 2001).

Deklaracija *xml* elemenata se sastoji iz deklaracije dva atributa, a to su *name* koji predstavlja naziv taga elementa i polje *type* koji predstavlja predefinisani ili korisnički definisani tip podataka koji će biti sadržaj elementa. Već je pomenuto da postoje predefinisani tipovi podataka, a neki od njih su *string, integer, boolean, date,* itd. Definisanje korisničkih tipova moguće je odraditi na dva načina. Prvi je da se prilagođeni tipovi nalaze u nekom referenciranom repozitorijumu, drugi je da se definiše korisnički tip u okviru šeme. Specifikacija tipa u šemi podrazumeva specifičnu sintaksu. Postoje dva tipa korisnički definisanih tipova, prosti i kompleksni. Prosti tipovi predstavljaju podskup nekog od predefinisanih tipova. Na primer, prost definisani tip u navedenom primeru jeste skup pozitivnih celih brojeva manjih od dve stotine. Rezervisana reč za deklaraciju prostog tipa jeste *simpleType.* Ono što je potrebno definisati prilikom deklaracije prostog tipa jeste predefinisani tip nad kojim se razvija novi tip(*base*). Kada se definiše tip nad predefinisanim tipom *string* moguće je specificirati regularni izraz[8] koji data vrednost mora zadovoljiti. Složen tip se kreira upotrebom rezervisane reči *complexType.* Potrebno je navesti naziv kompleksnog tipa popunjavanjem atributa *name.* U okviru kompleksnog tipa se može definisati više atributa(elemenata) nad predefinisanim ili korisnički definisanim tipovima. Pored definisanja tipova, lako se mogu definisati ograničenja na broj elemenata u nizu postavljanjem vrednosti atributa *minOccurs* i *maxOccurs.*

Najveći nedostatak *xsd* šema jeste taj što da bi se kreirala jedna šema neophodno je poznavati specifičnu sintaksu za specifikaciju šeme, a koja se dosta razlikuje od *xml* notacije, što iziskuje dodatne napore prilikom razvoja. Sa druge strane, čvrsta struktura šeme ne dozvoljava fleksibilnost. Međutim, ovaj nedostatak se može nadomestiti uvodjenjem opštih tipova(*object type*), koji uvode detipizaciju u šemu, a što dalje implicira odstupanja od strukture i kršenje osnovnih principa. Još jedan nedostatak ovakve strukture jeste to što je ona hijerarhijska struktura koja može imati neograničeno podelemenata što može u velikoj meri otežati parsiranje takvih struktura.

## *xPath* – upitni jezik za kretanje kroz *xml* strukturu

*Xml* struktura je hijerarhijski organizovana koja je implementirana preko stabala podataka. Stabla podataka su dobila ime po tome što podsećaju na stabla biljaka. Osnovni, početni element, je koren stabla iz kojeg proizilaze grane i listovi. Krajnji elementi se zovu listovi stabla. Da bi se doseglo do konkretnog elementa stabla potrebno je od korena preko svih nadređenih elemenata pronaći uočeni element. Ovakva kretanja kroz *xml* strukturu, obezbeđuje specijalna sintaksa pod nazivom *xPath.* Ovaj standard je prihvaćen od strane *W3C* grupe 1999. godine.

Svaki čvor u *xml* dokumentu sadrži informaciju o svojoj lokaciji u stablu. Do lokacije određenog elementa se dolazi preko putanja koje mogu biti apsolutne i relativne. Apsolutna putanja do elementa se pronalazi od početnog, uvek fiksiranog korenog elementa. Primer se može pronaći u sistemu datoteka na operativnom sistemu *Windows* gde je koreni element zapravo *C*

[8] Regularni izrazi(*Regular expression*) – u žargonu poznatiji kao *regex* predstavlja set definisanih literala i operatora koji imaju određeno značenje u prepoznavanju šablona teksta (Microsoft, Regular Expression Language)

particija na kojoj se nalazi sam sistem. Relativna putanja je opozit od apsolutne putanje i ona može da varira u zavisnosti od čvora od kojeg se želi pronaći putanja do željenog čvora. Čvor od kojeg se polazi u potrazi za putanjom do željenog čvora naziva se kontekstni čvor (Ray, 2001).

*xPath* izraz se sastoji od 3 dela. Prvi deo jeste osa po kojoj se krećemo do čvora, a odnosi se na to da li se spuštamo u hijerarhiji ka deci, roditeljima ili vršnjacima kontekstnog čvora. Neki od tipova osa su sledeći:

- *Ancestor* – obuhvata sve čvorove koji se nalaze iznad u hijerarhiji u odnosu na kontekstni čvor
- *Child* – predstavlja reference do prvog deteta kontekstnog čvora
- *Descendant* – obuhvata sve čvorove kojima je kontekstni čvor nadređen
- *Self* – predstavlja kontekstni čvor

Kroz drugi deo izraza se prikazuje tip čvorova koji dolaze u obzir za obradu. Od ose ovaj deo se odvaja sa dve dvotačke(*::*). Neki od opcija koje dolaze u obzir su sledeće:

- / - koreni čvor sa svim svojim atributima
- *node()* – bilo koji element u dokumentu osim korenog čvora i atributa
- *text()* - bilo koji element koji predstavlja tekstualno polje

I poslednji deo izraza predstavlja predikat koji je uslov koji čvor mora da zadovolji da bi se našao u skupu kandidata do kojih se može doći uz pomoć kreiranog *xPah* izraza.

*xPath* sintaksu je jako bitno razumeti jer je neretko potrebno obraditi samo deo dokumenta koji se obrađuje. Neki od razloga zašto se ova sintaksa izbegava jeste taj što na prvi pogled izgleda kompleksno i konfuzno.

## *SOAP* protokol – osnovni protokol *B2B* komunikacije

*SOAP* je skraćenica za *Simple Object Access Protocol* i predstavlja protokol komunikacije preko interneta između distribuiranih, decentralizovanih sistema. *Soap* je osnova *b2b* komunikacije. *B2B* komunikacija(*business to business communication)* je apstrakcija komunikacije između dva ili više privrednih entiteta gde se razmenjuje veliki broj dokumenata, poslovnih informacija koje se neretko razmenjuju dvadeset i četiri časa dnevno.

*Soap* predstavlja jednostavan mehanizam za razmenu struktuiranih podataka između dva entiteta koji je baziran na *xml*-u. Soap kao i drugi protokoli komunikacije definiše redosled razmene poruka, strukturu poruka, kodiranje poruke, a ne obezbeđuje aplikativnu logiku koju je potrebno implementirati u zavisnosti od potreba sistema. Same *soap* poruke su struktuirane kao *xml* dokumenti i sastoje od obvojnice poruke(*message envelope*), zaglavlja poruke(message *header*) i tela poruke(*message body*) (Don Box, 2000). Obvojnica poruke predstavlja sloj koji je relevantan samom protokolu i koji predstavlja meta sloj. U obvojnici se nalaze podaci o načinu transporta, verziji protokola koji se koristi, verziji enkodiranja koja se koristi, itd. Telo poruke je

sam *xml* sadržaj poruke koji je korisnik kreirao i koji želi da razmeni sa primaocem. *Soap* koristi *HTTP* protokol, aplikativni protokol petoslojne ISO/OSI arhitekture[9].

## *SOAP* poruke i statusi odgovora

Već je pomenuto da korisnikovu poruku, *soap* protokol enkapsulira obvojnicom koja sadrži meta podatke krucijalne za razumevanje između dva *soap* klijenta. S obzirom na to da *soap* koristi *http* protokol aplikativnog sloja *OSI* arhitekture *soap* poruka će biti još jednom enkapsulirana obvojnicom koju dodaje *HTTP* protokol a koji je njemu neophodan da bi razmenjivao poruke.

Kada korisnik kreira *xml* poruku koju želi da pošalje, kompletan sadržaj se smešta u telo poruke. Zatim, sledi enkapsulacija *soap* obvojnicom gde se dodaju specifični podaci svojstveni *soap* protokolu. Kada je poruka kreirana ona biva prosleđena *http* protokolu na slanje. *Http* dalje enkapsulira poruku tako što ovako kreiranu poruku smešta u telo *http* poruke i dodaje svoje zaglavlje sa metapodacima. Osnovni atributi koji se nalaze u obvojnici *http* protokola su sledeći:

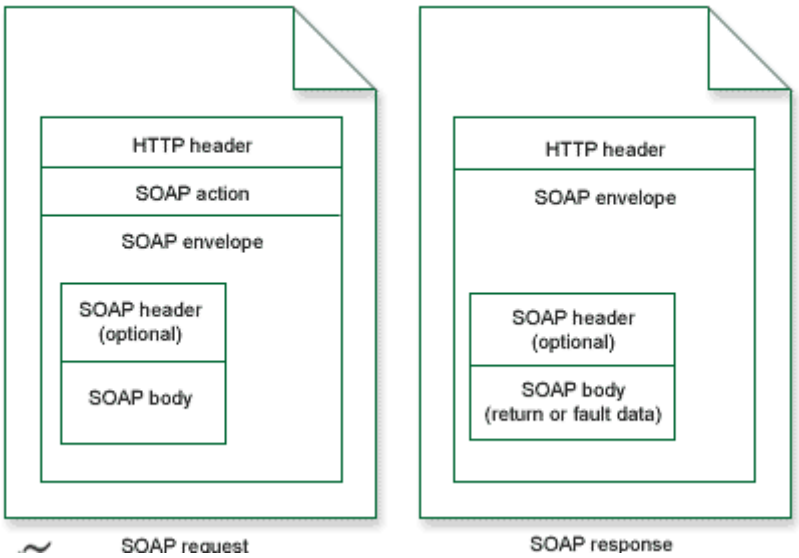
- *Method* – predstavlja tip metode(*GET, POST, PUT, DELETE,...*)
- *Content-Type* – tip sadržaja poruke. Neke od mogućih tipova su tekst, *JSON, xml*, itd
- *Content-Length* – dužina sadržaja tela poruke koji se šalje
- *Authorization* – tip autorizacije koji se koristi
- *Address* – url[10] adresa na koju se šalje zahtev

Pored pomenutih atributa, u zaglavlju http zahteva se nalazi i verzija http protokola koja se koristi(aktuelna verzija je *HTTP 1.1*).

Enkapsulacijom *soap* protokola dodaje se obvojnica. U obvojnici se referenciraju online xml repozitorijumi *"http://schemas.xmlsoap.org/soap/envelope/"* i *"http://schemas.xmlsoap.org/soap/encoding/"*. Prvi repozitorijum je zadužen za definisanje *soap* obvojnice, dok drugi definiše enkodiranje.

[9] Osi model predstavlja internet stek na različitim nivoima apstrakcije. Ovaj model je sačinjen od pet slojeva. Počevši od vrha ka dnu to su redom: aplikativni sloj, transportni sloj, mrežni sloj, sloj veze i fizički sloj. Pored petoslojnog postoji i sedmoslojni refetentni model koji pored pomenutih slojeva između aplikativnog i transportnog sloja sadrži još i sloj prezentacije i sloj sesije (Jim Kurose, 2014).

[10] URL – *Unified resource location* predstavlja jedinstvenu putanju do određenog resursa na internetu



Slika 1 - prikaz enkapsulacije soap poruka

Na slici je prikazan način enkapsulacije *soap* poruka koji je prethodno objašnjen. Takođe, je na intuitivan način prikazan proces enkapsulacije. U nastavku je prikaz jedne soap poruke.

*POST /StockQuote HTTP/1.1*
*Host: www.stockquoteserver.com*
*Content-Type: text/xml; charset="utf-8"*
*Content-Length: nnnn*
*SOAPAction: "Some-URI"*
*<SOAP-ENV:Envelope*
*xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"*
*SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">*
*<SOAP-ENV:Body>*
*<m:GetLastTradePrice xmlns:m="Some-URI">*
*<symbol>DIS</symbol>*
*</m:GetLastTradePrice>*
*</SOAP-ENV:Body>*
*</SOAP-ENV:Envelope>*

Ključna reč *post* predstavlja *http* metod koji označava da se radi u kreiranju novog resursa, tj da se šalju određene informacije na *http* server. Takodje, u *http* zaglavlju su navedeni tip sadržaja, dužina sadržaja, server na koji se šalje zahtev. Počev od linije broj pet, počinje *soap* obvojnica. Kao što je već pomenuto, prvo se referenciraju *online xml* repozitorijumi, a zatim sledi telo poruke koje je kreirano na strani pošiljaoca. Primer odgovora na prikazanu poruku zahteva bi bio na primer sledeći:

*HTTP/1.1 200 OK*
*Content-Type: text/xml; charset="utf-8"*
*Content-Length: nnnn*
*<SOAP-ENV:Envelope*
*xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"*

*SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>*
*<SOAP-ENV:Body>*
*<m:GetLastTradePriceResponse xmlns:m="Some-URI">*
*<Price>34.5</Price>*
*</m:GetLastTradePriceResponse>*
*</SOAP-ENV:Body>*
*</SOAP-ENV:Envelope>*

Struktura poruke je ista kao i poruka zahteva. Prvo sledi *http* zaglavlje sa statusom odgovora. Potencijalni http kodovi odgovora su:

- *200 OK* – indikator da je zahtev uspešno obrađen
- *304 NOT MODIFIED* – pokazatelj da zahtevani resurs nije menjan od prethodnog zahteva
- *400 BAD REQUEST* – znak da je server dobio zahtev koji nije struktuiran u skladu sa konvencijom
- *404 NOT FOUND* – indikator da zahtevani resurs ne postoji na serveru
- *500 INTERNAL SERVER ERROR* – kod koji govori da je došlo do greške na serveru prilikom obrade zahteva

Na osnovu pomenutog se dolazi do zaključka da je zahtev koji je poslat uspešno obrađen i da se u telu poruke nalazi sadržaj odgovora servera. *Soap* obvojnica odgovora je manje-više ista kao i obvojnica zahteva, a telo odgovora sadrži podatke koji su relevantni korisniku. *Http* i *soap* zaglavlja korisnik u principu i ne vidi jer se poruka dekapsulira u međuvremenu.

Ukoliko dođe do greške prilikom procesiranja *soap* poruke, *soap* protokol kreira poruku greške. Greška može nastati ukoliko *xml* poruka nije kreirana u skladu sa standardom, ukoliko dodje do prekida konekcije itd. Poruka greške se sastoji iz dva dela a to su kod greške(*fault code*) i poruka greške(*fault string*). Logika kreiranja poruka grešaka je slična kao kod *http* protokola.

## Definicija *SOA* servisa

Veb servisi predstavljaju skup različitih tipova servisa koji se nalaze na internetu ili nekoj drugoj računarskoj mreži. Sam servis predstavlja jednu modularnu strukturu koja izvršava određenu aplikativnu logiku na zahtev. Akcija se obavlja na poziv servisa, tj. neophodno je da neko inicira početak izvršavanja servisa. Naziv servis potiče od engleske reči *service* koja znači usluga. Servis zapravo i jeste entitet koji opslužuje jednog ili više klijenata, tj. preko servisa se izlaže logika koju će neko ko je izvan sistema moći da koristi. Kombinacija servisa, kako internih tako i eksternih u okviru jednog sistema, naziva se servisno orijentisana arhitektura(*Service-Oriented architecture*) (Barry).

*Soap* servisi su zasnovani na razmeni poruka koje su u *xml* formatu. Kako bi se izvršila integracija dva sistema preko veb servisa koji je jedan izložio drugom na korišćenje, neophodno je poznavati definiciju tog servisa. Definicija servisa se sastoji od definicije ulaznih parametara, izlaznih parametara, naziva metode, kao i adrese servisa. S obzirom da je osnovna jedinica razmene preko *soa* protokola *xml* dokument, tako je, da bi se očuvala konzistentnost, definicija

veb servisa u *xml* formatu. Definicija servisa se iskazuje specifičnom *xml* sintaksom (*Web Service Description Language*), a sam fajl definicije servisa ima ekstenziju *.wsdl.* Jedan veb servise može imati više izloženih metoda. Do definicije veb servisa se dolazi kada se na *url* adresu servisa doda nastavak *?wsdl.* Neke od osnovnih elemenata *.wsdl* fajla su:

- Reference ka *xml* šemama koje predstavljaju definiciju ulaza i izlaza iz metode servisa
- Tip porta koji može da se sastoji iz poziva više metoda
- Definicija metoda sa ulaznim i izlaznim tipovima, kao i definicijom tipa greške koji se može pojaviti kao odgovor usled nepredviđenog izvršavanja logike
- Deo koji se odnosi na dokumentaciju i dodatne informacije o svrsi korišćenja servisa ili njegovih metoda
- Adresa na kojoj se nalazi servis
- Specifikacija korisnički definisanih tipova

Osnovna namena definicije servisa jeste ta da na standardizovani način omogući integraciju servisa sa drugim sistemima. S obzirom na to da se radi o interakciji više sistema koji mogu biti implementirani u različitim tehnologijama, neophodno je obezbediti standradizovanu notaciju koju će svaki sistem umeti da pročita. Upravo to je obezbedilo uvođenje *xml*-a kao standardne stukture za opis servisa. *Xml* je u širokoj primeni i deo je standarda. Na primer, ukoliko imamo veb servise koji su kreiranu u *.NET*[11] okruženju i aplikaciju koja poziva dati servis a koja je implementirana u *JAVA*[12] okruženju. Servis izlaganjem svoje definicije u *wsdl* formatu omogućuje integraciju na lagan način. Svako od razvojnih okruženja ima implementirane parsere *wsdl* fajlova koji u zavisnosti od tehnologije implementiraju servis i generišu pozive servisa kao i tipove zahteva i odgovora. Pošto je glavna tema rada *.NET* implementacija softvera za razvoj *soap* servisa, u nastavku će biti objašnjeno nekoliko implementacija veb servisa u pomenutom okruženju kao i način integracije aplikacija sa *soap* servisima.

Veb servisi u *.NET* okruženju se zasnivaju na razdvajanju specifikacije od implementacije. Naime, prilikom kreiranja servisa prvo je potrebno kreirati interfejs u kojem će biti navedena specifikacija servisa. Specifikacija servisa podrazumeva popis metoda servisa sa njihovim ulaznim i izlaznim tipovima. Dakle, interfejs sadži samo spisak potpisa metoda. Ova specifikacija predstavlja zapravo definiciju servisa i na osnovu nje se generiše *.wsdl* fajl. Odvajanjem specifikacije od implementacije postiže se mogućnost asinhronog razvoja i integracije. Ako je poznata specifikacija servisa, drugi sistem može početi sa implementacijom poziva istog dok je ovaj još u fazi razvoja. Kada je implementacija gotova potrebno ju je samo izložiti na korišćenje po predefinisanoj specifikaciji. Implementacijom metoda iz specifikacije

[11] *.NET* okruženje predstavlja skup razvojnih okvira koji je razvila kompanija *Microsoft.* Celo okruženje omogućava razvoj raznovrsnih aplikacija za interne, baza podataka, desktop aplikacija i mnogih drugih. Kao celokupna platforma zauzima veliki udeo tržišta

[12] *JAVA* okruženje je razvijeno od strane kompanije *Oracle* i predstavlja najveći platformu otvorenog koda. Veoma je popularna za razvoj robustnih sistema. Takođe je i besplatna što joj dodatno daje na popularnosti

dolazi se do ugradnje poslovne logike u servis. Servisi implementirani u *.NET* okruženju imaju ekstenziju *.asmx*. Veb servisi razvijeni u *.NET* okruženju moraju biti izloženi na *Microsoft-ovom IIS*[13] serveru. Kako je servisno orijentisana arhitektura informacionih sistema i aplikacija veoma popularna, u *.NET* okruženju je razvijen čitav okvir za razvoj servisno orijentisanih sistema pod nazivom *WCF(Windows Communication Foundation).* Neke od funkcionalnosti koje *wcf* obezbeđuje su orijentisanost na servise, interoperabilnost[14], meta podaci o servisima, bezbednost aplikacija i podataka, veći broj razlitih tipova razmene poruka i enkodovanja istih, transakcije[15] itd. *Wcf* aplikacije odlikuje i veoma visok stepen konfigurabilnosti, bez dodatnog menjanja programskog koda. *Wcf* u potpunosti može da zameni klasične veb servise i da pruži dodatne funkcionalnosti. Ono što *wcf* dodatno omogućuje jeste pozivanje servisa uz pomoć dodatnih protokola(*http, tcp/ip, msmq*) dok je klasične veb servise moguće pozivati siključivo koristeći *http* protokol.

### Poređenje *SOAP* i *REST* protokola

*REST (Representational State Transfer)* predstavlja arhitekturalno rešenje koje se koristi za kreiranje servisno orijentisanih aplikacija. *Rest* je mnogo jednostavniji od *soap* protokola. Za razliku od *soap* -a koji za komunikaciju koristi *xml* standard, *rest* koristi *JSON* format koji je znatno lakši za parsiranje. Ono što je zajedničko za oba arhitekturalna pristupa jeste to što koriste *http* protokol za komunikaciju.

*Rest* servisi su servisi bez beleženja stanja (*stateless*), za razliku od *soap* servisa koji se zasnivanju na pamćenju stanja(*statefull*). Rest se najčešće koristi za izradu aplikacija za mobilne uređaje, društvenih mreža i automatizovanih poslovnih procesa. Veoma je pogodan za izradu aplikacija koje opslužuju veliki broj korisnika, jer se za svaki od zahteva kreira posebna instanca servisa koje obrađuje samo taj zahtev. Servisu se pristupa preko njegovog *uri*-ja.

## Osnovni koncepti *BizTalk* servera

*BizTalk* server predstavlja platformu za razvoj robustnih softverskih rešenja namenjenim kompanijama. Veliki sistemi koji se koriste u modernom poslovanju moraju da ispune određene zahteve da bi bili svrsishodni. Neki od osobina modernih sistema koji se koriste u produkciji su robustnost, pouzdanost, skalabilnost[16], efikasnost. *BizTalk* platforma je namenjena razvoju softvera koji će vršiti orkestraciju procesa. Procesi koje je potrebno implementirati su često veoma kompleksni i zahtevaju interakciju više zainteresovanih strana. Neretko postoji više kompanija koje je potrebno integrisati, a svaka od njih izlaže deo logike koju je potrebno ugraditi

[13] *Internet information Services* predstavlja *Microsoft*-ov server na koji se hostuju veb aplikacije razvijene upotrebom .NET razvojnog okruženja

[14] Interoperabilnost je osobina programa ili sistema da vrši interakciju sa drugim programima ili sistemima koji su implementirani u različitoj tehnologiji

[15] Izvršavanje pod transakcijom podrazumeva izvršavanje većeg broja instrukcija po principu „sve ili ništa''. (Branislav Lazarević, 2012)Ukoliko se jedna od instrukcija ne izvrši kako je predviđeno poništava se rezultat izvršavanja ostalih instrukcija

[16] Skalabilnost je osobina sistema da uspešno odgovori na povećanje kapaciteta sistem, tj. da se sa obimnijim ulazom generiše proporcionalno veći izlaz

u sistem. Ovde se javljaju veliki problemi sinhronizacije procesa, kao i automatizacije njihovog izvršavanja. Primer automatizacije procesa se može prikazati na procesu izrade izveštaja. Recimo da se na dnevnom nivou i banci, na kraju radnog vremena izrađuju izveštaji o obavljenim transakcijama na dnevnom nivou. Ukoliko bi taj posao obavljao službenik, bilo bi potrebno da svaki dan u tačno određeno vreme on ručno na računaru počne sa generisanjem izveštaja ili da pokrene mehanizam koji će to uraditi. Sa druge strane automatizacija tog procesa bi mogla da se implementira rešenjem koje se svakog radnog dana u tačno određeno vreme okida i izvršava set instrukcija koji recimo generišu izveštaje i iste šalju trezoru narodne banke na rasknjižavanje.

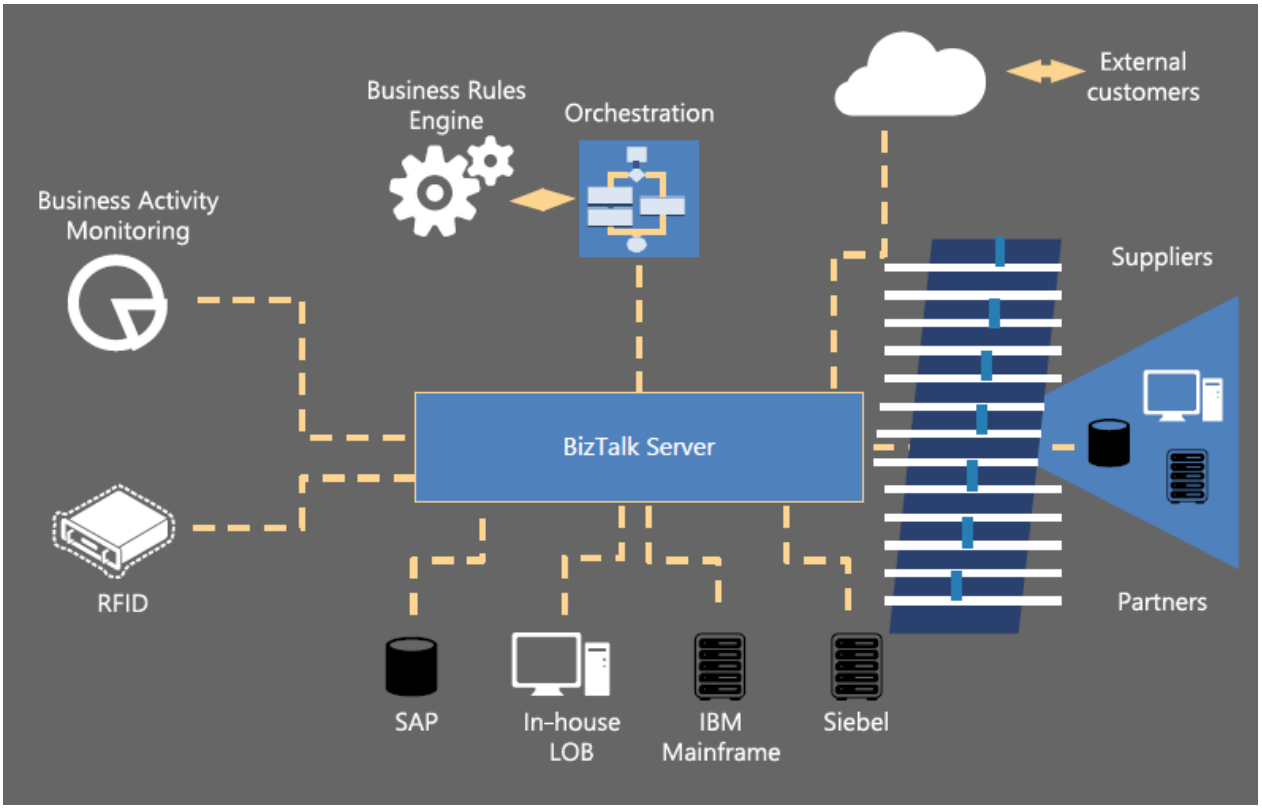
*BizTalk* obezbeđuje već pomenute funkcionalnosti i kao jedna od stabilnih platformi se nameće kao lider na tržištu. Pandan *Microsoft*-ovom rešenju jeste *Jitterbit.* Do sada biztalk platformu koristi preko dvanaest hiljada kompanija i predviđa se da će taj broj još rasti. Ono što je jako bitno jeste to da kompanije koje su svoje sisteme razvile upotrebom Biztalk-a gotovo sigurno neće preći na neki od konkurentskih rešenja. Razlog je taj što su sistemi koji su implementirani preko *BizTalk*-a toliko kompleksni da bi njihova migracija na drugu tehnologiju bila jednaka izradi identičnog sistema od nule (Brian Loesgen, 2012).

*BizTalk* server predstavlja platformu koja je namenjena integraciji više poslovnih podsistema jednog preduzeća. Centralizovana obrada i održavanje su glavni benefiti uvođenja *BizTalk* servera u preduzeće. Glavne integracione komponente koje biztalk integriše su: implementacija poslovnih pravila, praćenje izvršavanja poslovnih aktivnosti, integracija sa *SAP*[17] rešenjima, integracija sa klijentima i dobavljačima, integracija sa eksternim korisnicima, implementacija *RFID*[18] tehnologije u vođenje evidencije o proizvodima i zalihama, povezivanje sa *Siebel*[19] sistemom, itd. Sledeća slika ilustrativno prikazuje šta sve *BizTalk* integriše u kompaniji. Glavni fokus rada jeste analiza komunikacije kompanije sa dobavljačima i korisnicima.

[17] *SAP* predstavlja rešenje za planiranje resursa(*Enterprise resource planning*) u okviru kompanije. Ovo rešenje zauzima najveći procenat tržišta i do sada se pokazao kao najsveobuhvatniji sistem za upravljanje resursima u preduzeću. Pandan SAP-ovom rešenju su *Microsoft Navision, OpenERP i Adempier* sistemi otvorenog koda

[18] *RFID - Radio-Frequency IDentification* je tehnologija bazirana na pincipu mikročipova i antena koji odašilju signal do prijemnika. Ovi uredjaji se ugrađuju u bar kodove i time olakšavaju vođenje evidencije o kretanju proizvoda u okviru skladišta. Široka je primena istih

[19] *Siebel – CRM* rešenje kompanije *Oracle* koje se bavi vođenjeme evidencije o odnosima sa kupcima, dobavljačima i drugim zainteresovanim stranama u kompaniji(*Customer relationship management*). Sistem dodatno omogućuje kopmpaniji da dosegne maksimum benefita na osnovu iskustva sa korisnicima (Oracle)




Slika 2 – integracija komponenti korišćenjem BizTalk servera

U nastavku će biti prikazane osnovne funkcionalnosti *BizTalk* servera kao i arhitektura samog servera. Biće objašnjena primena istog u orkestraciji procesa javne uprave i administracija i postavljanje aplikacija.

## Arhitektura *BizTalk* servera

Biztalk kao robustna platforma koja ima veliki broj funkcionalnosti i omogućava raznovrsne implementacije jeste veoma kompleksna. Celokupnu arhitekturu nije moguće u potpunosti sagledati jer veliki deo predstavlja crnu kutiju i potpuno je nedostupan zbog politike *Microsoft*-a, jer Biztalk nije softver otvorenog koda.

Da bismo na mašini instalirali BizTalk server, neophodno je da na njoj imamo podešen *Microsoft Windows Server*. Pored toga potrebno je imati određenu verziju *Windows*-a koja je kompatibilna sa verzijom biztlk-a koju instaliramo. Takođe je neophodno imati i bazu podataka u koju će biztalk smestiti konfiguracione i transakcione podatke. Neophodno je da baza podataka bude *SQL Server*[20].

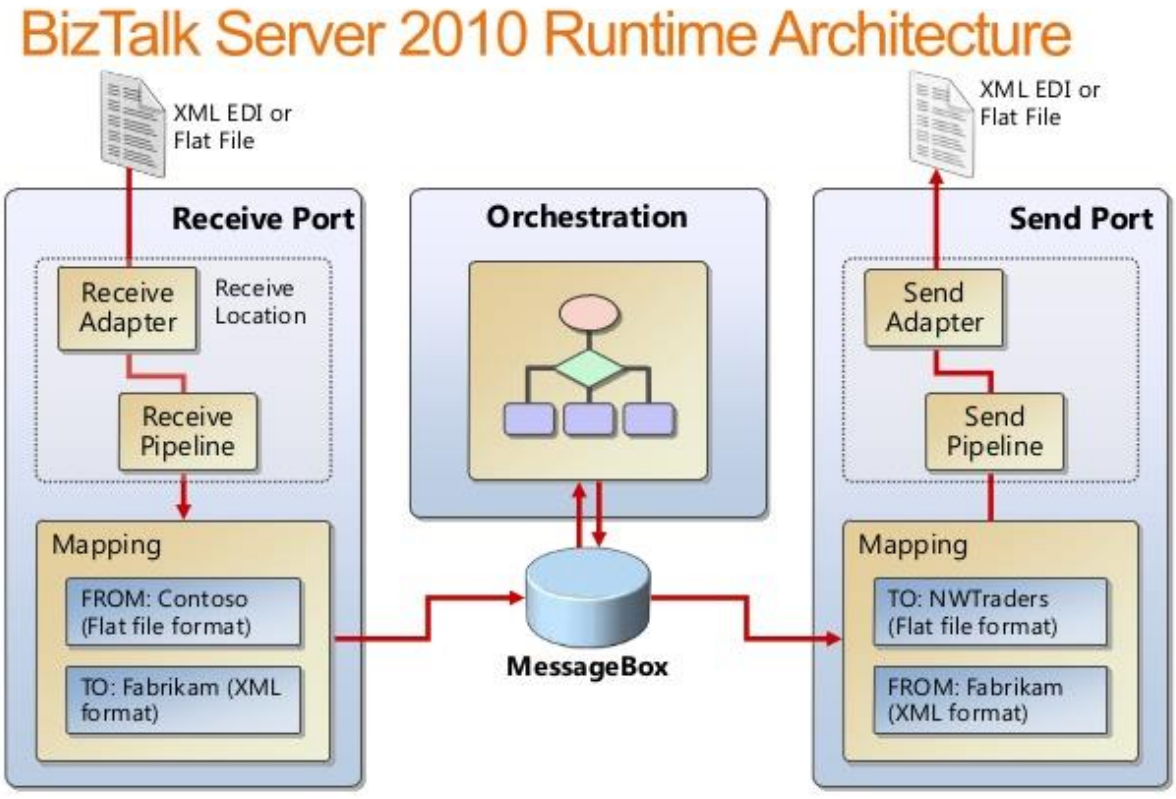
Prilikom instalacije *BizTalk* servera dolazi do instaliranja i dodatnih alata koji su neophodni za konfigurisanje servera i administraciju aplikacija. Sve relevantne konfiguracione podatke *BizTalk* čuva u bazi. Konfiguraciona baza se naziva *BizTalkMgmtDb*. U njoj se nalaze podaci o kredencijalima korisnika koji će se prijavljivati na server, o aplikacijama koje su

[20] SQL Server je *Microsoft*-ov server baze podataka. On implementira SQL bazu podataka.

hostovane na serveru itd. Pored pomenute konfiguracione baze, kreiraju se i sledeće baze podataka koje imaju različite uloge:

- *SSODB* – baza koja je namenjena za čuvanje kredencijala i podataka o korisnicima, podataka o lokacijama prijema, tj. lokacijama koje inicijalizuju pokretanje aplikacije
- *BizTalkRuleEngineDb* – u ovoj bazi se čuvaju podaci o pravilima izvršavnja procesa u okviru aplikacija, tj. orkestracijama
- *BizTalkMsgboxDb* – kako se celokupna komunikacija zasniva na razmeni poruka, ovo je baza u kojoj se čuvaju sadržaji i struktura poruka koje se razmenjuju
- *BizTalk DTADb* – *BizTalk* server je veoma transakcioni. Gotovo svaka akcija koja se izvršava na serveru se izvršava u okviru transakcije. Podaci o transakcijama se čuvaju u ovoj bazi

U nastavku će biti prikazan tok obrade zahteva na *BizTalk* serveru.




Slika 3 – tok obrade zahteva na *BizTalk* server

Dakle, da bi se pokrenuo tok prvo je potrebno da bude primljen zahtev na određenu lokaciju. Osnovna jedinica u celokupnom procesu je poruka koja je u *xml* formatu. Poruka može stići od različitih izvora koji mogu biti lokalni proces, neki od eksternih sistema itd. Tačka lokalnog procesa koja je zadužena za prijem zahteva jeste prijemni port(*receive port*) koji

predstavlja interfejs ka eksternom procesu. Prijemni port se sastoji iz dve komponente, prijemne lokacije(*receive location*) I mapiranja(*mapping*). Prijemna lokacija, kao što samo ime kaže, prvi je entitet na koji pristiže poruka zahteva. Prijemne lokacije mogu biti različiti tipovi. Neki od tipova prijemnih lokacija su:

- *Web service* – klasičan soa servis
- *WCF – wcf* servis
- *SQL* – baza podataka može biti prijemna lokacija
- *File* – poruke mogu biti fizički fajlovi koji se smeštaju u folder na disku

Na prijemnim lokacijama se primenjuje slična logika kao i kod veb servisa jer i one same mogu biti veb servisi, a to podrazumeva odvajanje specifikacije od implementacije. Specifikacija lokacije podrazumeva dodeljivanje *xml* šeme po kojoj će se upoređivati pristigle poruke. Ukoliko poruka pristigla na prijemnu lokaciju ne odgovara nekoj šemi *schema1.xsd,* ta poruka će biti ignorisana i neće ući u dalji proces obrade. Validacija poruke sa ulaznom šemom se vrši u delu prijemne lokacije koja se nazive prijemni tok(*receive pipeline*). Pored standardnih prijemnih tokova, moguće je implementirati prilagođene prijemne tokove gde će se na specifičan način vršiti poređenje i prevođenje. Pošto se poruka prima kao flet struktura neophodno je izvršiti mapiranje iste u hijerarhijsku strukturu. Nakon uspešne validacije poruke ista ulazi u proces obrade. Validna poruka se smešta u *MessageBox,* bazu u kojoj se čuvaju sve pristigle poruke. Ukoliko je poruka prevelika, doći će do podele iste u više delova koji se čuvaju bazi.

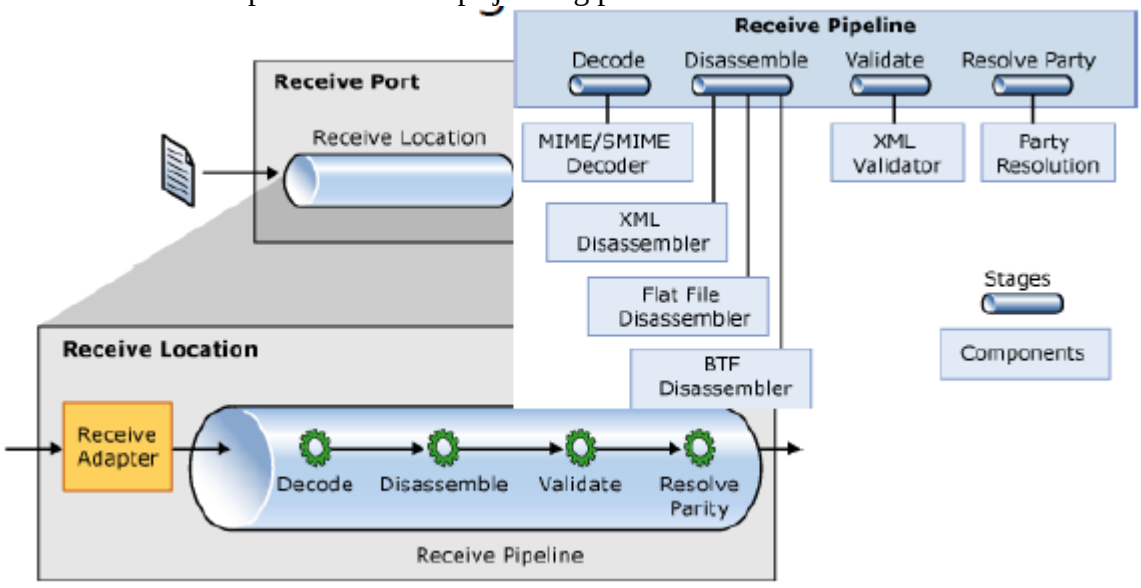
Kada je poruka sačuvana u bazu, može se smatrati da je proces orkestracije pokrenut. Postavljanjem aplikacije na server postavljaju se i šeme poruka koje su specifične za određene orkestracije. Za poruku koja odgovara određenoj šemi, kreiraju se pretplatnici(*subscriber*) na taj tip poruka i njeni izdavači(*publisher*). Izdavači mogu biti prijemni portovi i orkestracije, a pretplatnici na poruku orkestracije i portovi slanja. Čim je poruka određenog tipa smeštena u bazu, pronalaze se pretplatnici na taj tip poruke. Kada su pretplatnici pronađeni poruka im biva prosleđena na obradu. Poruka ja zapravo inicijator orkestracije. Orkestracija je jezgro procesa. U orkestracijama je smeštena kompletna poslovna logika koja je implementirana u aplikaciji i predstavlja niz instrukcija koje se izvršavaju po jasno specificiranom redosledu i uslovima. Ulaz u orkestraciju je poruka, a izlaz iz orkestracije je takođe poruka. Ukoliko orkestracija nema izlaz onda ona nema svoju svrhu, a ceo sistem postoji sam za sebe što dodatno nema smisla. Orkestracija je često i pretplatnik i izdavač poruke. Kada se orkestracija završena, poruka koja je rezultat izvršavanja biva sačuvana u bazu podataka. Izlazne poruke mogu biti različitog tipa od ulaznih. Takva poruka je automatski okidač za pozivanje novog pretplatnika na konkretan tip poruke. Kao što postoji prijemni port koji je zadužen za prijem određenog tipa poruka, tako postoji port za slanje koji je pretplatnik na određeni tip poruka i taj tip je izlaz iz procesa.

## Portovi prijema i slanja

Do sada je objašnjena genereralna priča oko prijema poruka, obrade i slanja istih, kao i opšta arhitektura *BizTalk* aplikacija. Sledi detaljnije objašnjenje oko toga kako su definisani interfejsi *BizTalk* aplikacija ka spoljnjim entitetima.

Prijemni port je dodirna tačka kroz koju poruke pristižu u aplikaciju. Prijemni portovi mogu biti dvostrani(*two-way ports*), tj moguće je i primati i slati poruke preko istog porta, tzv. *request-response port*. Ranije je konstatovano da se port sastoji iz dva glavna dela, prijemna lokacija i prijemni tok. Prijemna lokacija je prva linija prijema i može imati različite implementacije. Može biti veb servis, folder u koji se kopiraju fajlovi, itd. Dodeljivanje prijemne lokacije prijemnom portu se naziva *binding,* tj. dolazi do vezivanja specifične lokacije sa konkretnim portom. Ovaj proces recimo može biti takava da se specifičnom portu dodeli *url* adresa servisa koji će primati poruke.

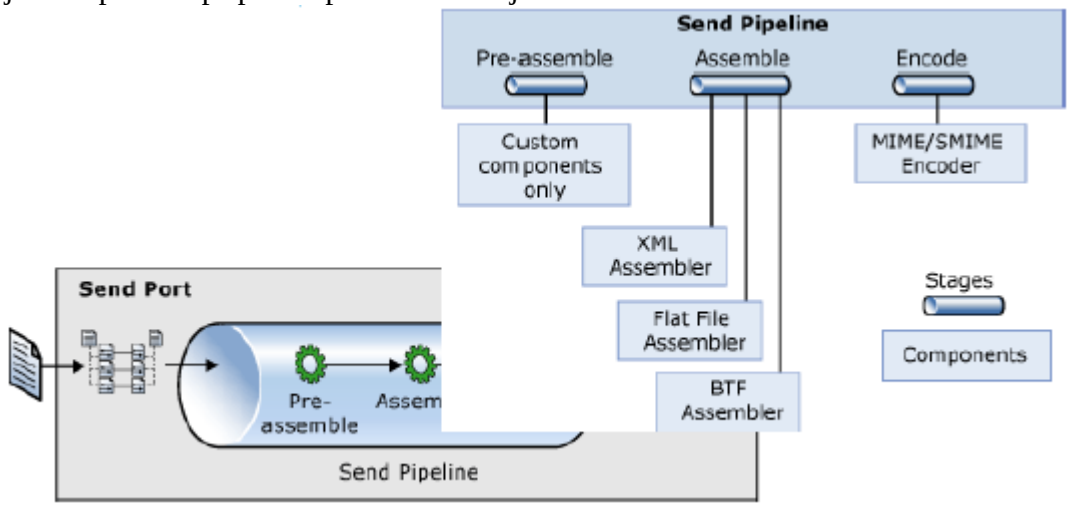
U prijemnom toku(*receive pipeline*) dolazi do prve obrade poruka. Obrada se sastoji od nekoliko faza. Prva faza je dekodiranje poruke koje pristižu u nekom enkodovanom formatu, npr. *MIME*[21]. Nakon što je poruka dekodovana ona prelazi u asembler obradu, tj. deserijalizaciju. Da bi se neki dokument poslao preko veba ili fajl sistema on mora biti serijalizovan u niz bajtova ili neki drugi format. U ovoj etapi dolazi do deserijalizacije odnosno kreiranja dokumenta i prepoznavanja formata poruke (Microsoft, Receive Pipelines). Kada je dokument kreiran sledi validiranje dokumenta. Validacija podrazumeva provere tipa da li je poruka koja je stigla u skladu sa šemom koja je definisana kao ulazni format, da li je *xml* sintaksa ispoštovana itd. Naredna faza podrazumeva rešavanje delova poruke. Ukoliko je poruka koja je pristigla velika, ona će biti podeljena na više delova da ne bi došlo do zagušenja sistema. Ovakvi delovi poruke se čuvaju u bazi i prilikom dolaska na red za obradu sklapaju se u jednu prvobitno primljenu poruku. Kada je poruka prošla kroz prijemni port biva sačuvana u bazu pod nazivom *MsgBoxDb* nakon čega se pokreće već objašnjeni mehanizam obrade poruka i izvršavanja orkestracija. U nastavku sledi vizuelni prikaz struktrue prijemnog porta.




Slika 4 – prikaz structure prijemnog porta

[21] MIME tip označava da je tip koji se razmenjuje zapravo dokument. U biztalk kontekstu, svaka poruka je MIME tip jer je svaka poruka xml

Po završetku izvršavanja orkestracije dolazi do kreiranja novih poruka koje se čuvaju u bazi. Ove poruke moraju biti prosleđene dalje kao odgovor na zahtev. Portovi slanja koji prepoznaju kreiran tip poruke se aktiviraju i poruka iz baze ulazi u proces pripreme za slanje. Proces slanja je obrnut procesu prijema. Dakle, poruka prvo prolazi kroz tok za slanje, pa tek onda se fizički šalje klijentu. Tok za slanje se sastoji od nekoliko etapa koje su obrnutog redosleda u odnosu na prijemni tok. Prva etapa predstavlja pred proces obrade u kojem se mogu kreirati prilagođene komponente sa ugrađenom dodatnom logikom koja će se izvršiti neposredno pred samo slanje. Potom poruka ulazi u asembler fazu u kojoj se poruka serijalizuje kako bi bila kompatibilna za slanje. Izlaz iz ove faze jeste niz bajtova koji je spreman za slanje. I kao poslednja faza slanja sledi kodiranje poruke, tj kreiranje *MIME* tipa. Sledi slika koja vizuelna prikazuje fazu po fazu pripreme poruka za slanje.




Slika 5 – prikaz struktura porta slanja

## BizTalk orkestracija

Do sada su objašnjeni mehanizmi prijema i slanja poruka. Prijem poruke je okidač koji pokreće izvršavanje ostatka logike u aplikaciji. Biztalk orkestracija predstavlja srž procesa. U njoj je implementirana kompletna poslovna logika sa pravilima izvršavanja. Orkestracija je poslovni proces i predstavlja redosled izvršavnaja instrukcija. Orkestracija je dobila ime baš po tome što definiše kada će se nešto izvršiti i kojim redosledom, baš kao što i dirigent u filharmonijskom orkestru komanduje kada će koja grupa instrumenata zasvirati.

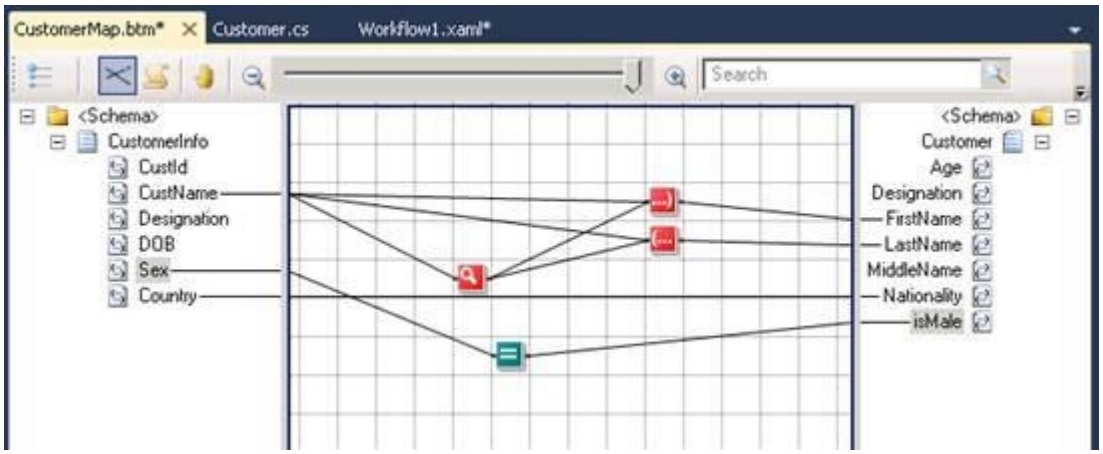
Sve orkestracije na biztalk serveru se čuvaju u bazi pod nazivom *BizTalkRuleEngineDb*. *Rule engine* predstavlja kompletnu mašineriju pravila koja su ugrađena u aplikaciju. Osnovni gradivni elementi orkestracije su šeme, mape(transformacije) i poslovna pravila.

Šeme predstavljaju šablone po kojima će biti struktuirane poruke i to su xml šeme. Xsd šeme su prikaz strukture stanja aplikacije. Već je rečeno da su glavni akteri na biztalk serveru pretplatnici i izdavači poruka. Pretplatnik se pretplaćuje na tačno određeni tip poruka, tj. poruke koje odgovaraju određenoj šemi(strukturi). Kreiranjem virtuelnih prijemnih portova koji obrađuju samo jedan definisani tip poruka dolazi do prijavljivanja za obradu tog tipa poruka. Već je rečeno da je logika kreiranja portova na biztalku slična logici kreiranja veb servisa. Određenim

virtuelnim portovima se dodeljuju fizički porotovi. Naime, logički port je specifikacija ulazne šeme poruke, dok je fizički port zapravo implementacija po definisanoj specifikaciji. Ono što je okidač za pokretanje orkestracije jeste smeštanje poruke onog tipa na koji se orkestracija pretplatila u bazu poruka.

Transformacije, odnosno mape, predstavljaju drugi značajan element orkestracija. One opisuju transformisanje između različitih tipova poruka. Na primer, često da bismo obradili poruku koja je nekog tipa koje je recimo propisan od strane korisnika koji poziva orkestraciju moramo da je transformišemo u drugi tip koji je lakše obraditi u orkestraciji. Logika transformacije se čuva u mapama. Transformacije u sebi sadrže operatore različitog tipa. Neke od njih su transformacije teksturalnih vrednosti, aritmetičke operacije sa brojevnim vrednostima, transformacije datuma itd. Okruženje za razvoj biztalk aplikacija, *Visual studio*[22], sadrži alat koji omogućava lagan i brz razvoj mapa a koji je baziran na grafičkom interfejsu.

Naredna slika prikazuje alat za razvoj biztalk mapa.




Slika 6 – BizTalk mape

Sa leve strane slike se vidi izvorna šema, tj. šema iz koje se vrši transformacija. Metodom prevlačenja se može definisati iz kog u koji element se vrši transformacija. Dakle ulaz u transformaciju je poruka a izlaz je prilagođena, transformisana poruka.

Kada je poruka primljena dolazi do pokretanje orkestracije. Postoji čitav mehanizam koji se bavi administracijom orkestracija, a koji se naziva *BizTalk Orchestration Engine.* Ovaj mehanizam vodi računa o kreiranju instanci orkestracija, očuvnaju stanja orkestracija, optimizaciji iskorišćavanja resursa, kao i obezbeđenju sigurnog gašenja i restartovanja orkestracija. Podešavanjem orkestracija može se definisati koliko procenata resursa orkestracija može maksimalno da koristi. Kada orkestracija iskoristi više resursa nego što joj je dodeljeno ona se polako usporava i gasi. Ova pojava se zove *throtling.*

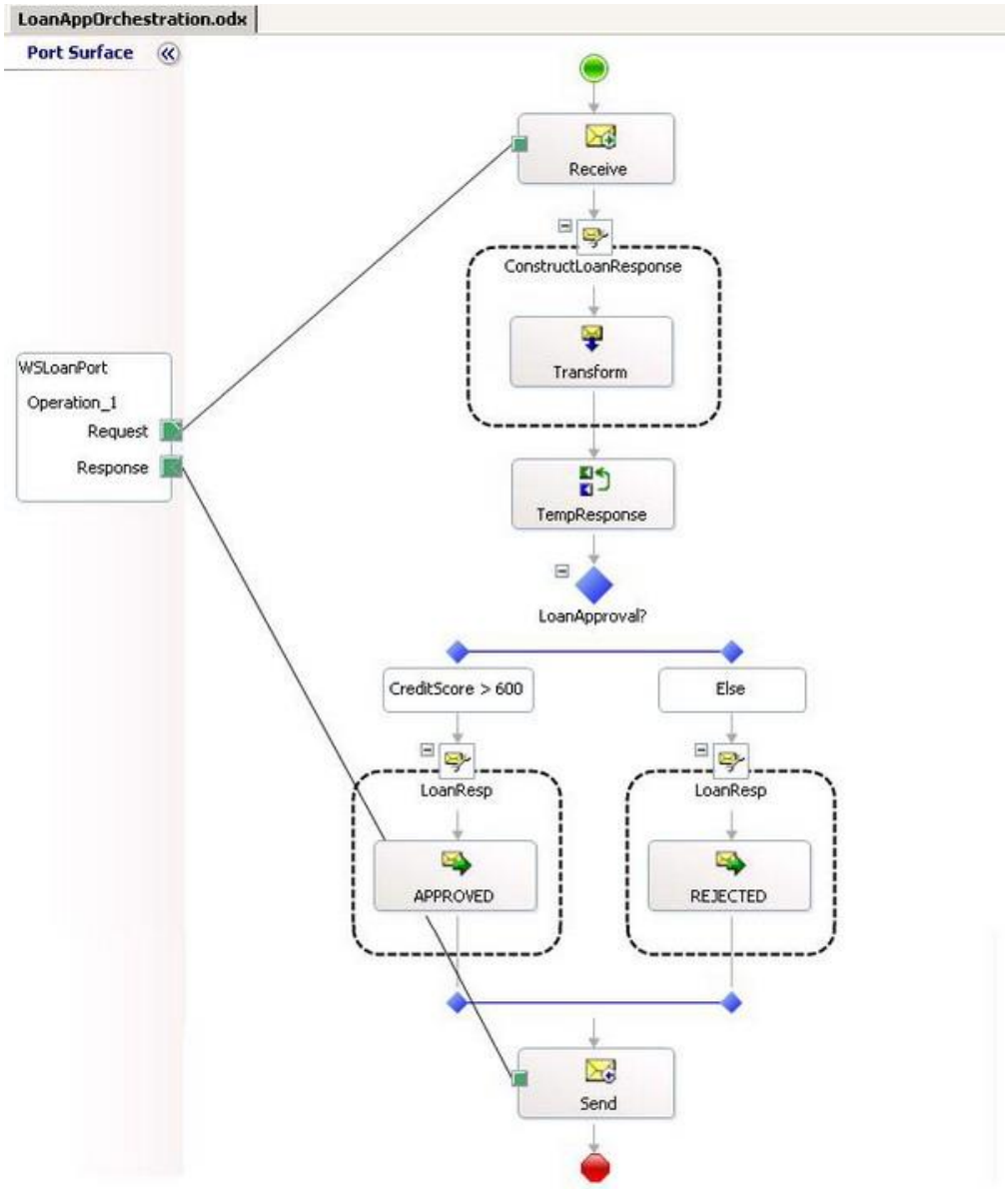
[22] *Visual Studio* predstavlja alat koji je razvio *Microsoft* a koji služi za razvoj aplikacija u *.NET* okruženju. Sadrži veliki broj komponenti za razvoj, testiranje, objavljivanje aplikacija, kao i alate za rad sa bazama podataka, otkrivanje grešaka u programskom kodu aplikacija i još mnogo korisnih dodataka za razvoj i testiranje specifičnih aplikacija. Postoji nekoliko verzija u zavisnosti od namene(*Community, Proffessional, Enterprise,* itd.) i datuma objavljivanja. Najnovija verzija je *Visual Studio 2017.*

Na biztalk serveru posotji jedan artifakt koji se zove *host instance* i predstavlja jedan proces operativnog sistema u okviru kojeg se izvršava orkestracija. Dakle orkestracije se dodeljuju jednom proces u okviru kojeg će se izvršavati, a svaka aplikacija na biztalk serveru predstavlja psoeban proces. Tako da ukoliko dođe do problemana jednoj od aplikacija može se restartovati samo ona dok ostale mogu raditi neometano.

Jedna biztalk orkestracija se objavljuje na server kreiranjem *.msi* fajla. Da bi se aplikacija postavila na server neophodno je instalirati .msi fajl na server. Instalacijom .msi fajla se postiže to da se svi .dll[23] fajlovi registruju u GAC[24] operativnog sistema. Nakon instalacije *.msi* jedino još preostaje importovanje aplikacije na server o čemu će biti reči u nastavku.

[23] *.dll* fajlovi su fajlovi specifični za .net okvir i predstavljaju kompajliran kod aplikacije. Ovaj kod nije izvršni pa se ne može pokretati a operativnom sistemu.

[24] GAC(*global assemblz cache*) predstavlja direktorijum operativnog sistema *Windows* u kojem se registruju deljeni *.dll* fajovi koje koristi više aplikacija. Postoje specifične verzije *GAC*-a u zavisnosti od broja instaliranih verzija *.NET* okvira




**Slika 7 – primer BizTalk orkestracije**

Prethodna slika predstavlja primer jedne orkestracije. Orkestracija nije ništa drugo do niz instrukcija prikazanih preko različitih oblika koji se izvršavaju po jasno utvrđenom redosledu. Skroz levo je predstavljen port koji je dvostrani, što znači da se preko istog porta prima zahtev i prosleđuje odgovor. Kasnije će port prijema biti povezan sa fizičkim portom. Prvi oblik u orkestraciji ja znak za prijem zahteva i on je obično oblik koji startuje orkestraciju. Kada je primljen zahtev sledi transformacija istog u neki od internih tipova. Transformacija se nalazi u sklopu okvira kreiranja poruke(*construct message*). Kada je odrađena transformacija, sledi izvršavanje poslovne logike. U konkretnom primeru se radi o odobravanju zajma. Ako je zajam veći od šest stotina kredit će biti odobren, a u suprotnom odbijen. Ishod se šuva u lokalnoj promenljivoj. Poslednji korak orkestracije jeste slanje odgovora klijentu putem porta preko kojeg

je i primljen zahtev. Ovde su prikazani samo osnovni elementi orkestracije. Spisak još nekih značajnih funkcionalnosti koje je moguće implementirati u orkestraciju su i:

- *Call orchestration* – uz pomoć ove funkcionalnosti omogućuje se podela poslovne logike na 2 ili više orkestracija koje će izvršavati jednu celinu instrukcija gde se kreira jedna sveobuhvatna orkestracija koja poziva ostale orkestracije
- *Decide* – oblik koji je pandan *if* klauzuli u programskim jezicima
- *Delay* – predstavlja odlaganje izvršavanja za određeni vremenski interval. Na primer, kada stigne zahtev mora se sačekati pet minuta da bi odgovor bio obrađen
- *Loop* – pandan iteratorima u programskim jezicima i predstavlja petlju koja će se izvršavati dokle god je zadovoljen uslov.
- *Parallel Actions* – omogućuje paralelizaciju izršavanja
- *Terminate* – predstavlja opciju momentalnog prestanka rada orkestracije
- *Throw Exception* – identična svrha kao u programskim jezicima. Funkcionalnost koja prijavljuje da je došlo do greške, nepredviđene ili namerno izazvane od strane korisnika
- *Construct message* – možda najbitnija funkcionalnost koja omogućava da kroz *C#* kod ručno kreiramo poruku van transformacije

Za svaki od navedenih funkcionalnosti(oblika) moguće je izvršiti podešavanje specifičnih opcija(*property*-ja).

Do sada su objašnjeni osnovni koncepti biztalk poslovnih procesa(orkestracija). U nastavku će biti prikazani načini izlaganja orkestracija na korišćenje i administracija aplikacija na serveru.

## Izlaganje orkestracija kao veb servisa

Pošto je glavna namena biztalk servera integracija više sistema u jedni celinu, tako je potrebno obezbediti komunikaciju između sistema. Izlaganjem određenih funkcionalnosti na korišćenje drugim sistemima je jedan od načina. Kreiranjem orkestracije na biztalk serveru nismo je učinili vidljivom. Postavljanjem orkestracija na server orkestracija postaje vidljiva drugima. Međutim, orkestraciju još uvek nije moguće pozivati jer još uvek postoji samo definicija iste. Definicija je sastavljena od ulaznih i izlaznih šema poruka koje orkestracija prihvata i koje vraća kao rezultat izvršavanja.

Implementacija prijemnih portova podrazumeva kreiranje istih po specifikaciji. Ova akcija je olakšana korišćenjem alata koje nam biztalk obezbeđuje. Prilikom instalacije servera na mašinu u pozadini se instalira i alat koji omogućuje da orkestracije izložimo kao veb servis i time je ustupimo eksternim sistemima na korišćenje. Alat se zove *BizTalk Web Services Publishing Wizard.* Prosleđivanjem .dll fajla sa orkestracijama ovaj alat povlači specifikaciju ulaznih i izlaznih portova i kreira web servis od njih. Ovako kreiran servis se objavljuje na lokalnom *IIS* serveru. Kreiranjem servisa na ovaj način postiže se kreiranje prijemne lokacije po specifikaciji. Pošto je reč o veb servisu lako se može doći do definicije servisa, tj. do .wsdl fajla. Sada se orkestracija poziva preko servisa koji predstavlja prijemnu lokaciju. Integracija sa sistemom je

omogućena. Dodatno je potrebno odraditi eventualnu autentikaciju korisnika izdavanjem sertifikata kako aplikaciju ne bi mogao da koristi svako. Pored objavljivanja kao veb servis, orkestraciju je moguće objaviti i kao wcf aplikaciju, zavisno od potreba i korisnčkih zahteva.

Aplikacija ne mora biti objavljena kao veb servis ukoliko je prijemna lokacija fajl sistem jer se u tom slučaju orkestracija može pozivati lokalno.

## Administracija BizTalk aplikacija

Pod administracijom biztalk aplikacija podrazumeva se nekoliko faza. To su: inicjalno postavljanje aplikacije, otkrivanje grešaka i praćenje istorije. Inicijalno postavljanje aplikacije predstavlja prvo objavljivanje aplikacije iz faze razvoja u fazu testiranja, i ono podrazumeva instaliranje aplikacije na server, postavljanje prijemnih portova i portova slanja i obezbeđivanje praćenja istorije poziva. Otkrivanje grešaka(*debugging*) jeste faza postavljanja stabilne verzije aplikacije i otkrivanje grešaka. Praćenje istorije poziva(*tracking*) jeste proces arhiviranja poziva orkestracija, pristiglih i poslatih poruka. U nastavku će svaki od pomenutih celina administracije biti detaljno objašnjena.

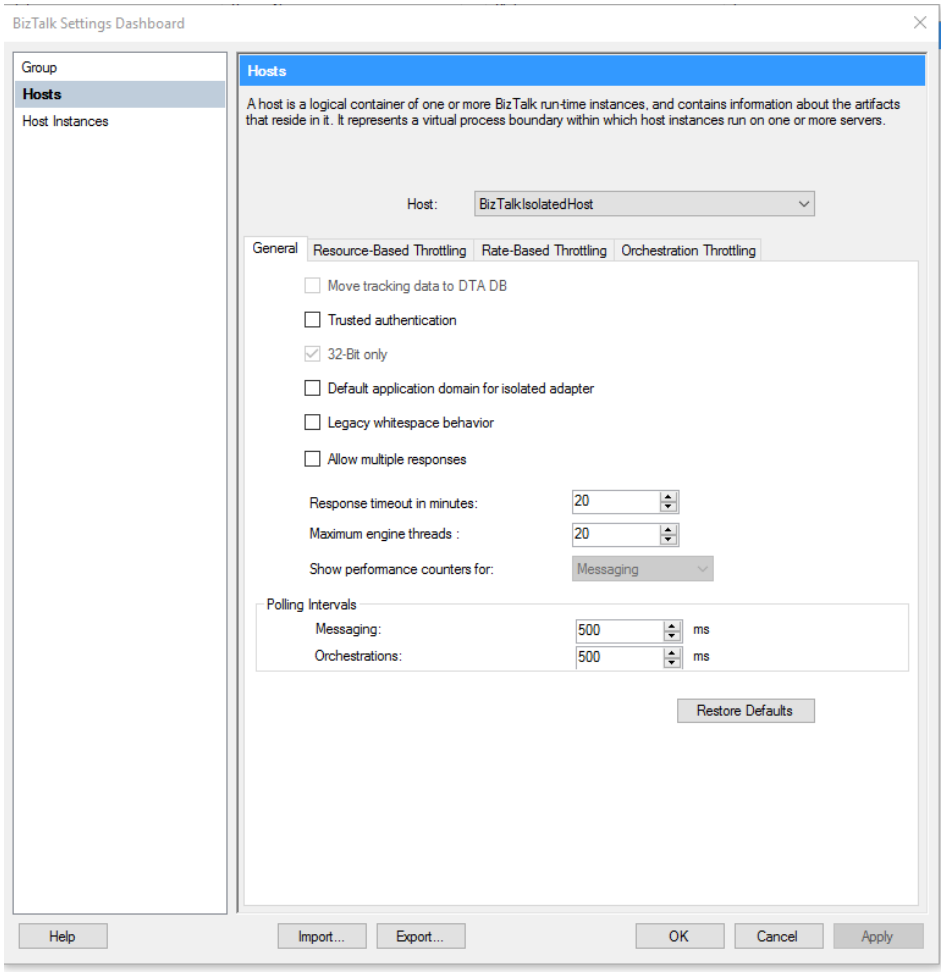
### Inicijalno instaliranje aplikacije

Osnovni tok dostavljanja određenog softverskog rešenja prolazi redom kroz fazu razvoja, fazu testiranja, postavljanje rešenja u produkciju i održavanje. Faza razvoja je početna i predstavlja analizu problema i implementaciju rešenja. Testiranje podrazumeva pronalaske propusta koji su promakli programerima koji su se bavili razvojem. Produkcija je faza puštanja rešenja u rad. I održavanje jeste implementacija dodatnih funkcionalnosti i održavanje rešenja u optimalnom stanju.

U kontekstu *BizTalk*-a razvoj predstavlja implementaciju poslovnih pravila. Poslovna pravila u velikoj meri predstavljaju samo specifikaciju šta se i na koji način radi. A u kontekstu definisanja stanja aplikacije, podrazumeva se kreiranje strukture ulaznih i izlaznih poruka kao i šeme baze podataka. Inicijalno instaliranje aplikacije na server je završna etapa razvoja rešenja i u njoj se obavlja implementacija preostalog dela aplikacije koji nije moguće implementirati u fazi razvoja rešenja.

Redosled koraka po kojem se vrši postavljanje aplikacije na server je sledeći:

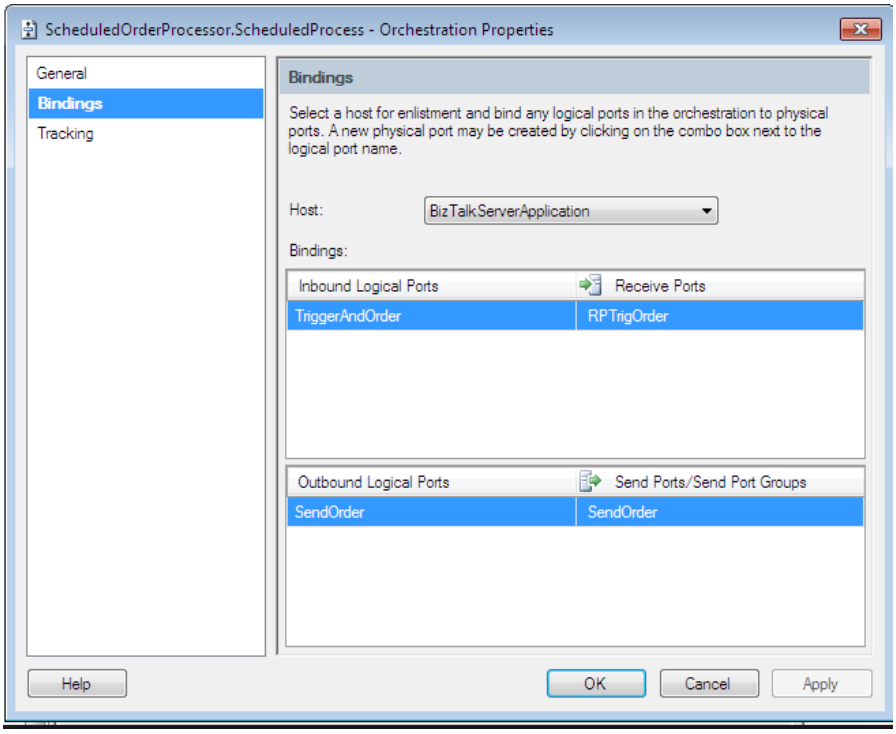
1. Kreiranje aplikacije na serveru – aplikacija se kreira na samom serveru i ono što je bitno jeste da se zove isto kao što je navedeno prilikom kreiranja .msi fajla iz razvojnog okruženja. Prilikom kreiranja aplikacije bira se host instanca(proces) u okviru kojeg će se izvršavati aplikacija na serveru. U okviru podešavanja procesa(*host instance*) moguće je definisati korisnika sa čijim kredencijalima će se izvršavati proces, procenat memorije i procesora koji je dodeljen za izvršavanje aplikacije, itd.




**Slika 8 – podešavanja BizTalk procesa**

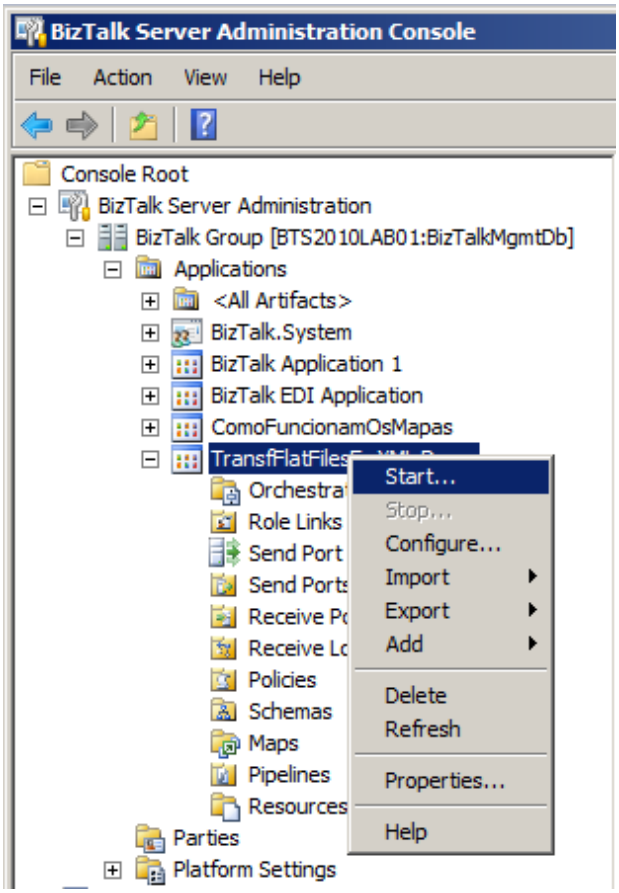
2. Instaliranje i importovanje aplikacije – kao što je već rečeno *BiZtalk* aplikacija je smeštena u .msi fajl. Da bi se ista instalirala na server neophodno je prvo registrovati korišćene .dll fajlove u *GAC*. To se postiže instaliranjem kreiranog .msi fajla. Nakon toga potrebno je te iste .dll fajlove importovati u prethodno kreiranu *BiZtalk* aplikaciju na serveru. Importovanjem aplikacije biće kreirane orkestracije, prijemni portovi i portovi slanja. Takođe, prilikom kreiranja .msi fajla moguće je izabrati koji artifakti[25] će biti eksportovani za instaliranje na serveru. Nakon ovoga koraka aplikacija je instalirana na server. Međutim, još uvek nije u potpunosti funkcionalna. Pošto je specifikacija objavljena ostaje još povezivanje specifikacije sa konkretnom implementacijom, tj. povezivanje logičke specifikacije portova sa fizičkim portovima na serveru.

[25] *Artifacts* – predstavljaju sve entitete na BizTalk serveru, a to su: orkestracije, portovi slanja, grupe portova slanja, prijemni portovi, prijemne lokacije, šeme, mape, korišćeni resursi(.dll fajlovi)




**Slika 9 – Povezivanje orkestracija sa portovima i hostom**

3. Pokretanje aplikacije – nakon svih pomenutih podešavanja potrebno je prvo utvrditi da je pokrenut proces(*host instance*) u okviru kojeg će se izvršavati aplikacija. Nakon toga sledi pokretanje aplikacije, što je prikazano na sledećoj slici.




Slika 10 – Startovanje aplikacije

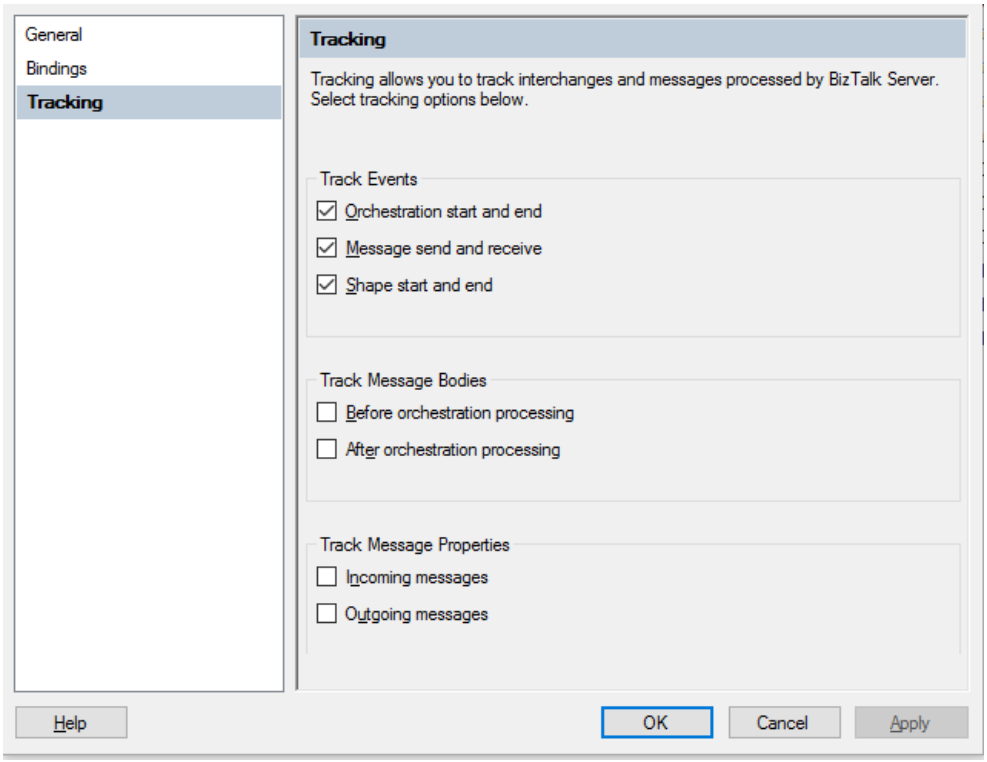
Ono što je značajno jeste to da aplikacija ne mora biti u potpunosti startovana, tj. ne moraju sve orkestracije aplikacije biti startovane. Ovo je značajno u fazi integracije ukoliko eksterni sistem poziva naše servise na *BiZtalk*-u. Znači deo funkcionalnosti može biti u funkciji dok se drugi razvija i testira. Preduslovi koji moraju biti ispunjeni da bi aplikacija mogla da bude parcijalno startovana jeste da host instanca bude pokrenuta i da sve prijemne lokacije koje koristi orkestracija budu aktivne i svi portovi slanja koji su referencirani budu pokrenuti.

## Testiranje aplikacije i otkrivanje grešaka

Kao završna faza pred puštanja rešenja u produkciju jeste testiranje i ispravljanje uočenih grešaka i nedostataka. Ovaj proces bi trebalo iterativno realizovati. Dakle testiranje funkcionalnosti, ispravljanje grešaka, instaliranje nove verzije aplikacije, testiranje i tako dalje u krug dok se ne dođe do stabilne verzije. *BiZtalk* sadrži veliki broj dodataka za praćenje i otkrivanje grešaka, kao i snimanje performansi servera i praćenje kompletnog izvršavanja aplikacije.

Kada je u pitanju testiranje rešenja, potrebno je kreirati niz testnih scenarija. Glavni fokus testiranja su orkestracije jer one predstavljaju samu srž aplikacija. Pored testiranja implementacije aplikacije potrebno je testirati i poslovnu logiku. *BiZtalk* server ima ugrađen alat

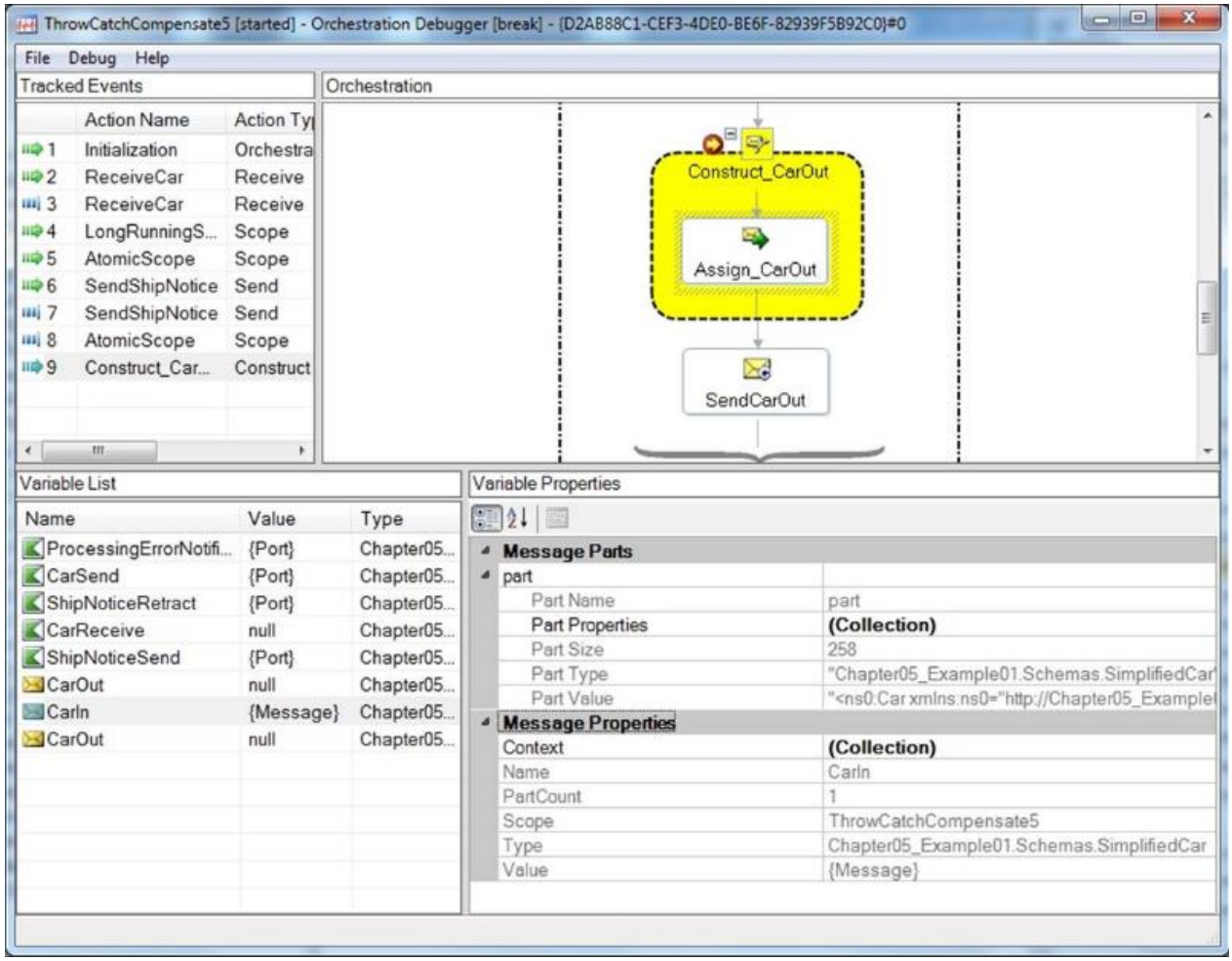
za otkrivanje grešaka(*debugger*). Otkrivanje grešaka se sastoji od prolaska kroz orkestraciju korak po korak i porediti ulaz i izlaz iz svakog od oblika. Da bi se uopšte moglo doći do ove faze, neophodno je obezbediti praćenje poruka i izvršavanja orkestracija na nivou celog procesa. Specifično je moguće ukinuti praćene za odrešene delove aplikacije, npr. može se ukinuti praćenje istestirane orkestracije kako ista akcija ne bi pretrpavala bazu podacima.




**Slika 11 – praćenje izvršavanja orkestracije**

Priložena slika predstavlja opcije koje omogućavaju praćenje tri grupe elemenata orkestracija. Prva grupa su događaji koji se dešavajau prilikom izvršavanja, a to su početak i kraj izvršavanja orkestracije, prijem i slanje poruke i početak i kraj izvršavanja oblika koji je stigao na red. U drugu grupu opcija spadaju praćenje tela poruka, tj. sadržaja a koje se može pratiti pre procesiranja u orkestraciji i nakon procesiranja. I treća grupa opcija spada u praćenje meta podataka o porukama(šeme, vreme prijema i slanja, itd.). Izborom ovih opcija može se početi sa testiranjem i otkrivanjem grešaka.

Kada dođe do poziva orkestracije moguće je postaviti kontrole tačke, tj. tačke orkestracije u kojima će orkestracija biti privremeno zaustavljena kako bi se posmatralo i analizirao izvršavanje orkestracije. Sledi prikaz alata za otkivanje grešaka koji dolazi kao deo *BiZtalk* servera.




Slika 12 – alat za otkrivanje grešaka

Uz pomoć ovog alata je moguće korak po korak prolaziti kroz svaki oblik orkestracije i proveravati ulaz i izlaz iz istog. U sredini prozora se nalazi prikaz implementirane orkestracije. Žutom bojom je obeležen oblik koji se trenutno izvršava. Sa leve strane prozora se nalazi spisak svih oblika koji su izvršeni prilikom izvršavanja trenutne instance orkestracije. Glavni deo prozora jeste donji panel koji omogućava pregled svih poruka i lokalnih promenljivih koje su kreirane u orkestraciji. Klikom na neku od poruka prikazuju se sve osobine te poruke. Ovde se može videti kompletan sadržaj poruke, šema po kojoj je poruka kreirana, dužina poruke itd.

## Bpmn notacija za formalizaciju opisa komunikacije procesa

BPMN(*Business Process Modeling Notation*) je notacija koja formalno i grafički opisuje poslovne procese i veze između njih. Osnovna dva načina na koji se mogu posmatrati poslovni procesi su:

- Orkestracija - predstavlja opis poslovnog procesa iz unutrašnjosti sistema i posmatra proces kao niz aktivnosti koje se odvijaju po predefinisanom redosledu. Aktivnosti su međusobno sinhronizovane preko upravljačkog mehanizma(*Business rule engine*).

- Koreografija – podrazumeva da se proces posmatra izvan sistema i fokusira se na komunikaciju sa poslovnim partnerima gde se posmatra razmena poruka između partnera. Ono što razlikuje koreografiju od orkestracije jeste to da se učesnici sami međusobno sinhronizuju, dok kod orkestracije sinhronizaciju vrši predefinisani mehanizam

Aktuelna verzija *BPMN*-a je verzija 2.0. *BPMN* je razvijen od strane organizacije *OMG*(*Object Management Group*). Jedan od glavnih ciljeva *BPMN*-a jeste da obezbedi da opis izvršavanja određenog procesa koji je definisan u *xml* notaciji može da se vizuelizuje kako bi lakše opisala poslovna pravila (OMG, 2011).

*BPMN* dijagrami predstavljaju niz oblika koji imaju određenu semantiku. *BPMN* dijagrami su slični *BiZtalk* orkestracijama, tj. *BiZtalk* je implementacija, a *BPMN* je specifikacija. U *BPMN* dijagramima postoji nekoliko kategorija elemenata i to su:

- Objekti koji se koriste u tokovima
- Elementi koji predstavljaju rad sa podacima
- Objekti koji se koriste za povezivanje elemenata
- Staze, u žargonu poznatije kao plivačke staze i predstavljaju grupu srodnih elemenata, odnosno elemenata koji se koriste u sklopu istog podprocesa
- Artifakti

Objekti koji se koriste u tokovima predstavljaju elemente koji definišu ponašanje poslovnog procesa. U ovu grupu spadaju

- Događaji – predstavljaju nešto što nastaje tokom izvršavanja procesa. Događaji najčešće nastaju kao rezultat nekog okidača koji može biti na primer istek određenog vremenskog intervala, kraj izvršavanja jedne aktivnosti itd.
- Aktivnosti su generički entiteti. Aktivnosti mogu biti atomske, nedeljive, i kombinovane od više atomskih ili neatomskih aktivnosti
- Prolazi – načini na koji se može izvršiti određeni tok. Oni mogu biti konvergentni(spajaju više grana izvršavanja u jednu), divergentni(kreiraju više izlaznih grana iz ulazne grane). Neki od tipova tokova su grananje, spajanje, odlučivanje itd.

Druga kategorija elemenata predstavlja elemente koji su zaduženi za manipulaciju podacima i to su:

- Objekti podataka – može predstavljati jedan objekat ili kolekciju objekata
- Ulazni podaci – set vrednosti koje predstavljaju ulaz u određenu aktivnost
- Izlazni podaci – set vrednosti koje predstavljaju izlaz iz određene aktivnosti
- Skladišta podataka

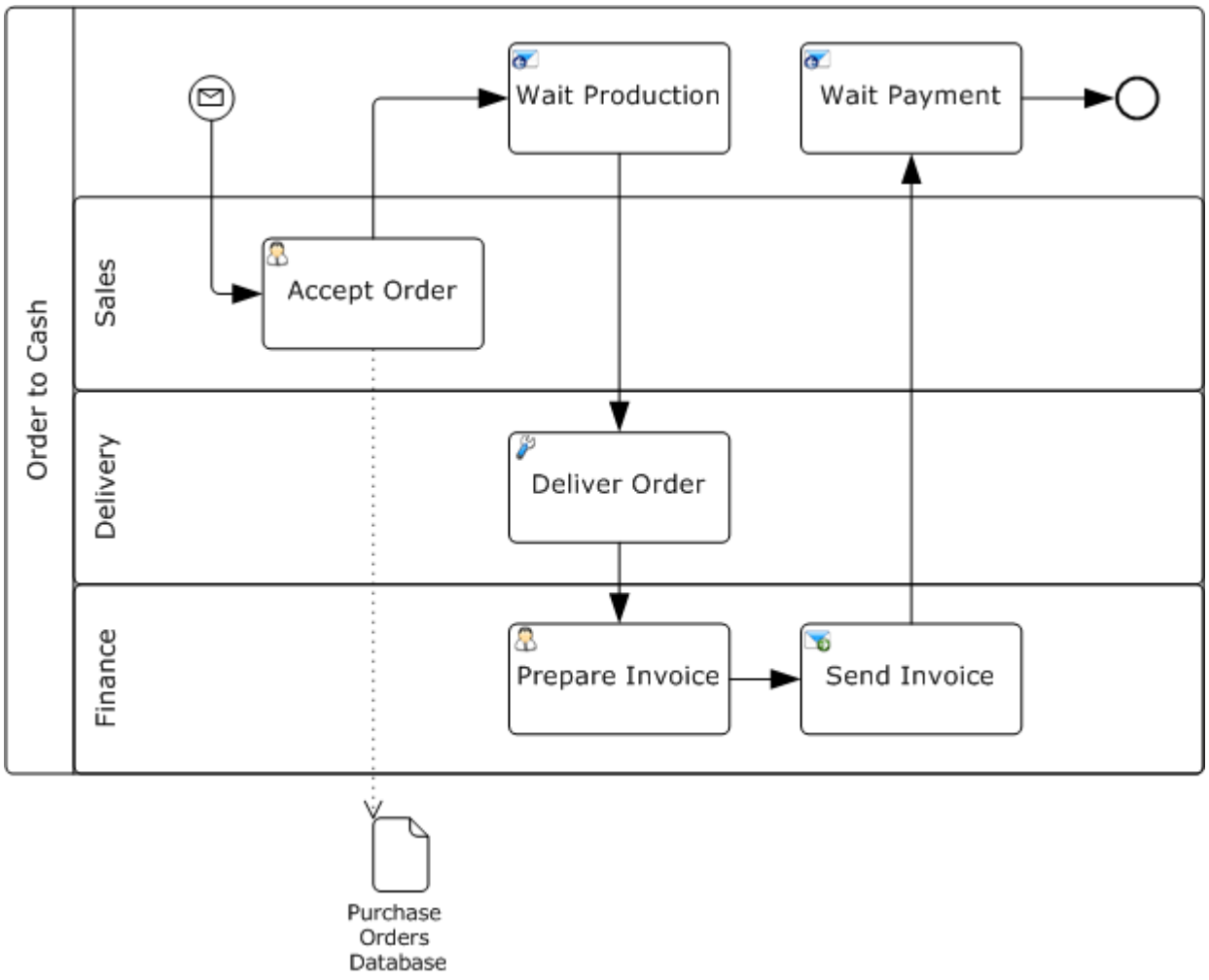
Kada su u pitanju elementi koji služe za povezivanje objekata, moguće je povezivanje objekata međusobom i povezivanje objekata sa nekim drugim elementima. Tipovi koji spadaju u ovu grupu elemenata su:

- Sekvencijalni tok – tok koji služi da prikaže redosled kojim će se aktivnosti izvršavati
- Tok poruka – predstavlja tok koji opisuje kako se razmenjuju poruke između dva učesnika. Učesnici u *BPMN* dijagramu su dva odvojena procesa(bazena)
- Asocijacija – se koristi da poveže informacije i artifakte sa *BPMN* grafičkim elementima. Asocijacija može biti usmerene i bez usmerenja.
- Asocijacija podataka

Staze predstavljaju četvrtu grupu elemenata. One se koriste za grupisanje elemenata i to su:

- Bazen – je grafička predstava učesnika u kolaboraciji i koristi se u *b2b*(*business-to-business*) komunikcaiji. Ovaj oblik takođe može biti i crna kutija, tj. eksterni proces koji ne poznajemo već se koristi samo kada se definišu poruke razmene
- Staza – je deo bazena i predstavlja podproces istog

Artifakti služe da obezbede dodatne informacije o procesu. Postoje dva standardizovana artifakta i to su grupa i tekstualni zapis. Pored pomenutih, moguće je kreirati prilagođene artifakte ukoliko je to potrebno korisniku.




**Slika 13 – *BPMN* dijagram orkestracije**

Prethodna slika predstavlja jednostavan proces prihvatanja porudžbina prikazana u *BPMN* notaciji. Prikazana je komunikacija više podprocesa(staza) jednog preduzeća gde se definiše redosled izvršavanja aktivnosti.

Već je pomenuto da BPMN predstavlja grafičku specifikaciju određenog poslovnog procesa. Na osnovu ovako kreirane standardizovane specifikacije moguće je izvršavati specificirane procese. Izvršavanje ovako definisanih procesa omogućuje *BPEL*(*Business Process Execution Language*). *BPEL* je jezik koji omogućava veb servisima da se integrišu i razmenjuju podatke. Pošto je *xml* osnovni jezik za opis veb servisa kao i za definisanje struktura poruka koje se ovako razmenjuju, tako je i *BPEL* baziran na *xml*-u. Glavne funkcionalnosti koje *BPEL* stavlja na raspolaganje su:

- Praćenje izvršavanja aktivnosti poslovnog procesa, a naročito integracija veb servisa
- Uočavanje korelacija između poruka i poslovnih procesa
- Reanimacija u slučaju da dođe do greške ili otkaza dela sistema
- Bilateralne odnose između veb servisa i uloga istih u celokupnom procesu

## Studijski primer

Do sada su opisani teorijski koncepti na kojima se bazira *BizTalk* server, arhitektura *BizTalk* servera i specifikacija alata za definisanje usluga u javnoj upravi. Nastavak predstavlja praktičnu primenu opisanog na primeru kreiranja orkestracije na *BizTalk* serveru koja vrši integraciju sa servisom Ministarstva unutrašnjih poslova Republike Srbije(u nastavku mup) koji za izabranog građanina vraća informacije o trenutnom prebivalištu. Važno je napomenuti da servisi koji su korišćeni kao primer predstavljaju testne odnosno razvojne servise.

Dakle za implementaciju je korišćena verzija *BizTalk* servera 2010 i razvojno okruženje *visual studio* 2010. Ukratko, arhitektura aplikacije je podeljena u nekoliko projekata i to su:

- Projekat sa šemama u kojem su definisane i specificirane sve šeme koje se koriste u aplikaciji. Šeme predstavljaju strukturu poruka koje su osnovni element komunikacije. Tako da bi ovaj projekat predstavljao zapravo domen aplikacije odnosno modele podataka koji se koriste
- Projekat sa transformacijama sadrži specifikaciju transformacija(mapa) i sadrži referencu na projekat sa šemama. Transformacije predstavljaju osnovnu logiku mapiranja tipova.
- Projekat sa orkestracijama u kojem se nalazi implementacija poslovnih pravila aplikacije(*business process engine*)

Kao što je opisano, šeme predstavljaju definiciju strukture poruka koje se razmenjuju i osnovni su faktor razumevanja komunikacije između dva entiteta. Mup-ov servis za preuzimanje podataka o prebivalištu građanina ima jasno definisanu strukturu koja je opisana preko xml šeme. Servis je specificiran tako da prima tri parametra. Prvi parametar je tekstualnog tipa i u njemu se prosleđuje tekstualna *xml* poruka koja predstavlja definisani set podataka koji će kasnije biti

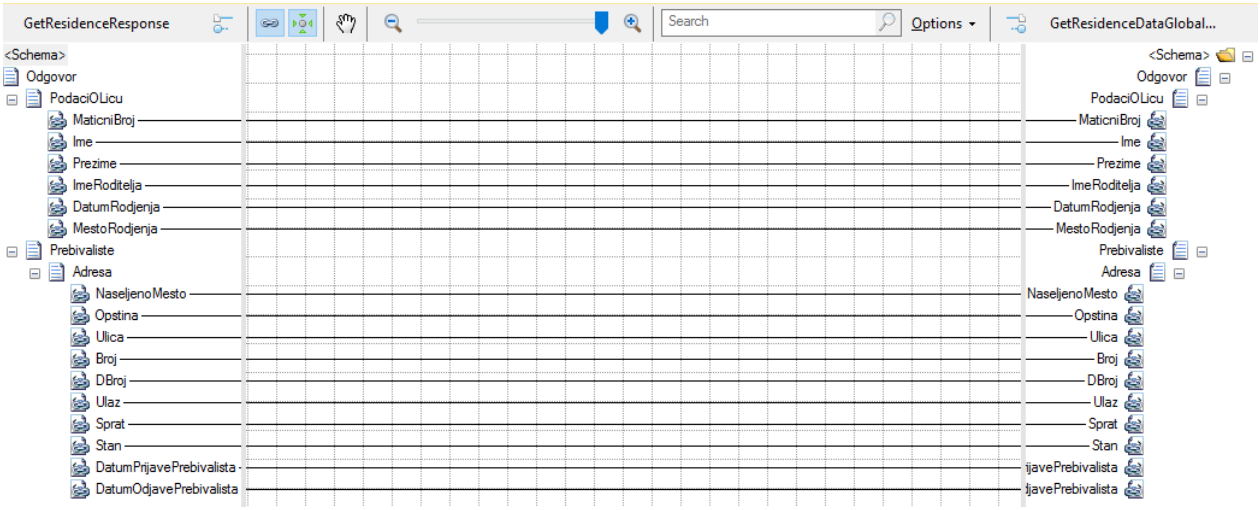
opisan. Drugi parametar servisa je takođe tekstualni tip i predstavlja određenu šifru koja definiše o kojoj metodi servisa se radi a dobijena je u specifikaciji servisa. Treći parametar je opcioni i predstavlja ip adresu sa koje se vrši poziv. Servis nije moguće pozvati anonimno, već je potrebna autorizacija sertifikatom. Pomenuti prvi parametar servisa predstavlja *xml* tekst koji odgovara tačno određenoj strukturi koja je specificirana za pomenutu metodu. Definisani *xml* koji je prvi ulazni parametar mora biti u skladu sa specifikacijom kako bi logika koja obrađuje pomenuti zahtev umela da isparsira zahtev i obradi podatke. Od podataka koje korisnik sistema unosi jeste jedinstveni matični broj građana za kojeg želi da preuzme trenutni prebivalište. Struktura podataka koja predstavlja odgovor sistema ima sledeću strukturu:

```
<?xml version="1.0" encoding="UTF-16"?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://Global.Schemas.GetResidenceDataGlobalResponse"
  xmlns:b="http://schemas.microsoft.com/BizTalk/2003">
    - <xs:element name="Odgovor">
        - <xs:complexType>
            - <xs:sequence>
                - <xs:element name="PodaciOLicu">
                    - <xs:complexType>
                        - <xs:sequence>
                            <xs:element name="MaticniBroj" type="xs:string"/>
                            <xs:element name="Ime" type="xs:string"/>
                            <xs:element name="Prezime" type="xs:string"/>
                            <xs:element name="ImeRoditelja" type="xs:string"/>
                            <xs:element name="DatumRodjenja" type="xs:string"/>
                            <xs:element name="MestoRodjenja" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                - <xs:element name="Prebivaliste">
                    - <xs:complexType>
                        - <xs:sequence>
                            - <xs:element name="Adresa">
                                - <xs:complexType>
                                    - <xs:sequence>
                                        <xs:element name="NaseljenoMesto" type="xs:string"/>
                                        <xs:element name="Opstina" type="xs:string"/>
                                        <xs:element name="Ulica" type="xs:string"/>
                                        <xs:element name="Broj" type="xs:string"/>
                                        <xs:element name="DBroj" type="xs:string"/>
                                        <xs:element name="Ulaz" type="xs:string"/>
                                        <xs:element name="Sprat" type="xs:string"/>
                                        <xs:element name="Stan" type="xs:string"/>
                                        <xs:element name="DatumPrijavePrebivalista"
                                            type="xs:string"/>
                                        <xs:element name="DatumOdjavePrebivalista"
                                            type="xs:string"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Slika 14 – struktura odgovora servisa mup-a

Iz priložene *xml* šeme se vidi da servis vraća dve grupe podataka. Prva grupa predstavlja opšte podatke o licu za koje se vrši upit i u njih spadaju: matični broj, ime, prezime, ime roditelja, datum rođenja i mesto rođenja. U drugu grupu podataka spadaju podaci o prebivalištu sa podacima o naseljenom mestu, opštini, uluci, broju, ulazu, spratu, stanu, datumu prijave prebivališta i datumu odjave prebivališta.

Kada su definisane šeme prema specifikaciji sledi definisanje transformacija. Ono što je bitno napomenuti jeste to da ulazne šeme aplikacije na *BizTalk*-u nisu jednake šemama koje predstavljaju ulazne šeme servisa koji se poziva. Razlog za ovo je na primer potreba za prosleđivanjem dodanih podataka *BizTalk* aplikaciji. Naime, skup podataka je jednak ili veći skupu podataka koji zahteva stvarni servis ali tip ulazne poruke nije isti tipu koji se zahtva na servisu. Već je rečeno da se svi ulazni parametri u *BizTalk* orkestraciju smeštaju u samo jedan parametar servisa. Kod izlaznog seta podataka priča je nešto drugačija. Dakle, podaci koji se dobijaju od servisa se mapiraju u lokalnu šemu koja ima isti set podataka a kojoj se samo razlikuje naziv repozitorijuma. Mapa koja transformiše primljene podatke data je u nastavku.



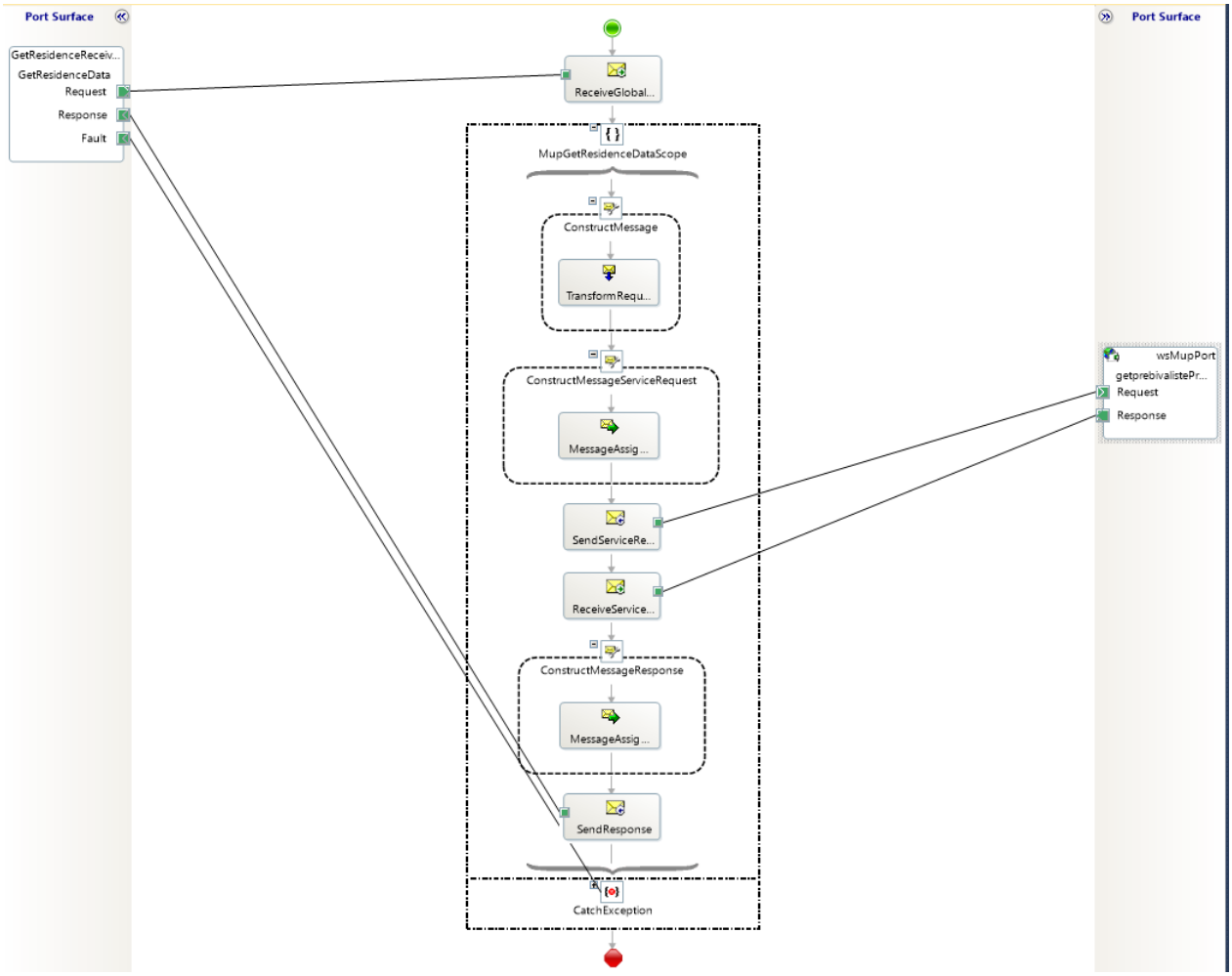

Slika 15 – mapiranje odgovora servisa

Na prvi pogled izgleda da su šeme iste, ali je razlika u nazivu repozitorijuma. Sa leve strane mape se nalazi izvorna šema, šema koja je primljena od strane mup-ovog servisa, a sa desne strane se nalazi odredišna šema, tj. šema koja je izlaz iz orkestracije. Opet je razlog za kreiranje dodatne šeme isti kao i malopređašnji, a to je da se ne želi izložiti šema eksternog sistema kao sopstvena. Generalno u transformacijama su mogle biti primenjene neke od funkcija koje *BizTalk* izlaže, npr. Transformacija tekstualne vrednosti u datum, odsecanje viška praznih mesta na kraju teksta itd.

Definisanjem strukture ulaznih i izlaznih poruka orkestracije i transformacija između poruka može se početi sa ugrađivanjem poslovne logike u aplikaciju. Poslovna pravila se implementiraju u orkestracijama. Orkestracija se sastoji od tri dela. Prvi je obrada ulazne poruke I priprema poruke za slanje servisu, a druga predstavlja prijem odgovora servisa I obrada odgovora. Treći deo predstavlja obrada grešaka nastalih tokom izvršavanja orkestracije.

Prilikom implementacije orkestracije potrebno je definisati početni element odnosno izvršavanje događaja koji inicijalizuje orkestraciju. U konkrentom primeru događaj koji inicijalizuje orkestraciju je prijem zahteva. Znači, orkestracija se pokreće kada na *BizTalk* preko prijemnog porta stigne zahtev. Kada poruka stigne na definisani port, onda biva sačuvana u bazu(*BizTalkMsgboxDb*). Ovaj događaj je indikator da orkestracija treba da počne sa obradom poruke. Specifikacija prijemnog porta se sastoji od tri koraka I to su:

1. Definisanje ulazne šeme
2. Definisanje izlazne šeme
3. Definisanje šeme koja će se prosleđivati ukoliko dođe do greške prilikom izvršavanja orkestracije

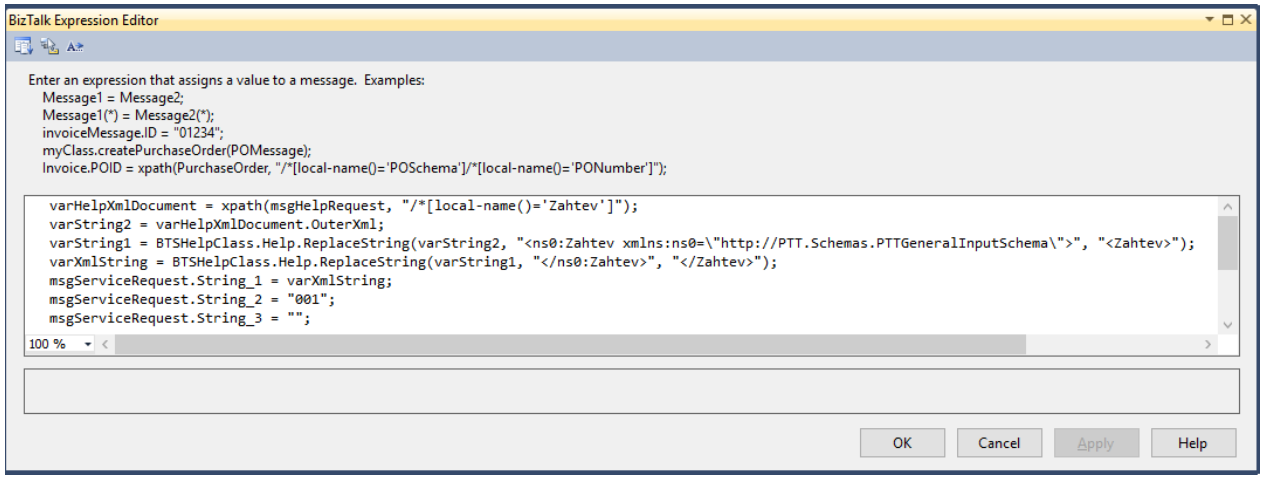



Slika 16 – orkestracija za poziv mup-ovog servisa

Slika šesnaest predstavlja izgled orkestracije za poziv mup-ovog servisa. Sa leve strane slike u osenčenom panelu nalazi se specifikacija prijemnog porta. Port je dvosmerni, što znači da se preko istog interfejsa prima zahtev I vraća odgovor. Od porta se vidi veza asocijacije koja vodi do oblika koji predstavlja prijem zahteva(*ReceiveGlobalRequest*). Prijem zahteva predstavlja akciju koja inicira orkestraciju pa iznad njega postoji zeleni kružić koji je znak za start

orkestracije. Naravno, ne mora uvek prijem poruke da bude inicijator orkestracije, ali pošto će orkestracija biti izložena kao veb servis koji radi po principu zahtev-odgovor neophodno je da inicijator orkestracije bude prijem poruke. Po prijemu poruke sledi kreiranje pomoćnih varijabli koje su neophodne za kreiranje valjanog zahteva koji će biti poslat servisu. Pomoćna varijabla ima strukturu identičnu kao I ulazna poruka ali se naziv repozitorijuma razlikuje. Iako je identična struktura ne radi se o istoj šemi. Oblik pod nazivom *TransformRequest* je zadužen za mapiranje ulazne poruke u pomoćnu varijablu. Mapa u strukturu varijable dodaje naziv repozitorijuma što predstavlja problem. Prvi problem jeste taj što mup-ov servis nije svestan lokalnog repozitorijuma I šeme po kojoj smo kreirali poruku I ovako kreiranu poruku koja sadrži reference do lokalnog repozitorijuma ne može da protumači. Drugi problem do kojeg bi moglo doći ukoliko bismo ulaznu poruku mapirali direktno u poruku za poziv servisa jeste taj što ukoliko bi došlo do promene ulazne šeme bilo bi neophodno menjati I implementaciju orkestracije.

Pošto je kreirana pomoćna varijabla sa podacima koji su mapirani iz ulazne poruke, može se početi sa prilagođavanjem poruke koja će odgovarati specifikaciji servisa.



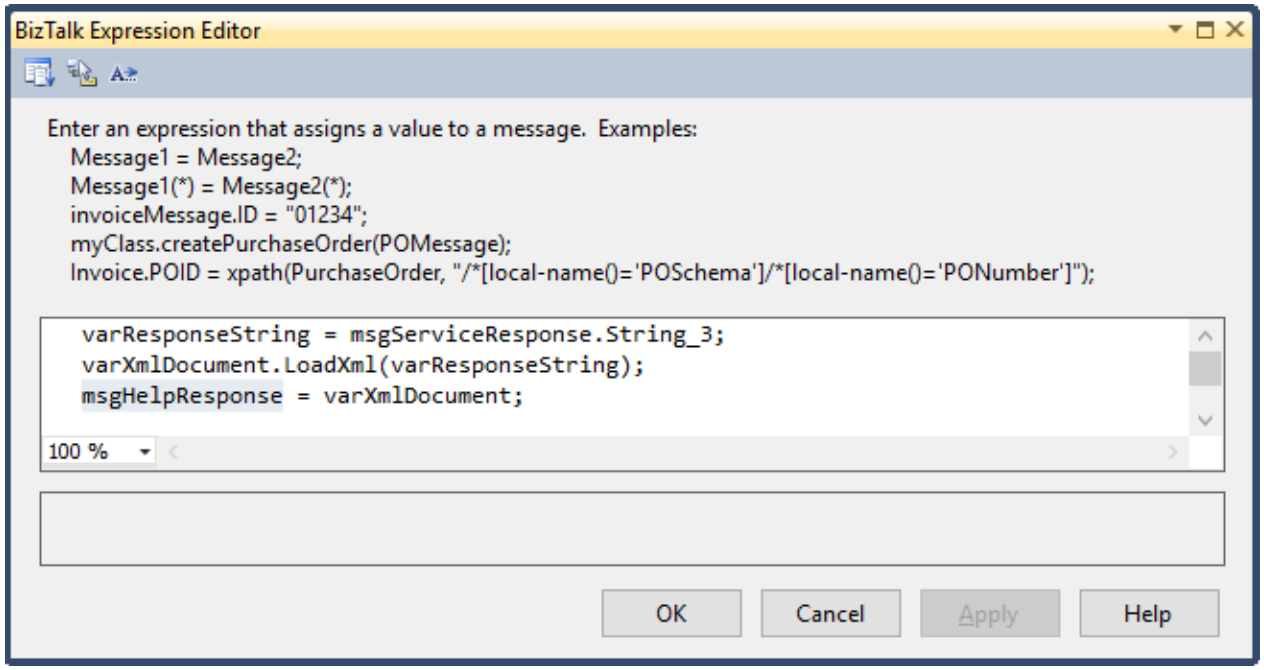

**Slika 17 – konstruisanje poruke uz pomoć biztalk komponente**

*BizTalk* je platforma koja je razvijena korišćenjem *.NET* okvira što omogućava razvoj prilagođenih komponenti u *.NET* okviru koje se mogu koristiti i prilikom razvoja *BizTalk* aplikacija. Ono što je potrebno uraditi jeste kreirati biblioteku u na primer *C#* programskom jeziku i istu referencirati iz *BizTalk* projekta. Sa druge strane postoje komponente u *BizTalk* -u koje nam omogućavaju da direktno u orkestraciji pišemo *C#* programski kod koji će se izvršiti kada ta komponenta dođe na red za izvršavanje. Element koji omogućava pomenutu funkcionalnost naziva se *Message Assignemnt* i predstavljen je na slici broj sedamnaest. Svaka pristigla poruka koja je *xml* formata se transformiše u *C#* objekte. Takođe moguće je kreirati varijable koje mogu biti nekog predefinisanog *.NET* tipa ili korisnički kreiranog tipa(u ovom kontekstu korisnički kreirani tip je *xml* šema koja predstavlja jedan tip poruke). U primeru je ispisan *C#* kod koji obrađuje pomoćnu varijablu i kreira poruku za slanje servisu. Promenljiva *varXmlDocument* je *xml* poruka koja preko *xPath*-a uzima element pod nazivom *Zahtev* i svu

njegovu decu. Dakle, od celokupne primljene poruke sa definisanim repozitorijumima izvlači se samo telo poruke bez deklaracija. Međutim, iako je izvučeno samo telo poruke, *xml* generator je dodao prefikse za repozitorijum na koreni element. Kada bismo ovako definisanu poruku poslali servisu dobli bismo grešku jer u strukturi imamo definisane prefikse koje servis ne ume da obradi. Zbog toga je potrebno odstraniti višak prefiksa. Najlakši način za to je kreirati tekst od *xml* dokumenta, uz pomoć pomoćne biblioteke odstraniti suvišan tekst i tako sređen tekst proslediti servisu. Varijabla *msgServiceRequest* predstavlja instancu poruke koja je kreirana na osnovu specifikacije zahteva koja se nalazi na serveru. Prvi parametar je *xml* tekst koji je prilagođen zahtevu. Drugi parametar po specifikaciji predstavlja identifikator metode. Treći parametar je opcioni pa je u konkretnom primeru izostavljen.

Izvršavanjem operacije za prilagođavanje poruke sledi slanje poruke servisu. Oblik koji se koristi za slanje je sličan obliku prijema i u konkretnoj orkestraciji je nazvan *SendServiceRequest.* Sa desne strane na slici šesnaest na kojoj je prikazana orkestracija se nalaze svi portovi slanja koji su specificirani u orkestraciji. U prikazanom slučaju, port slanja predstavlja mup-ov servis. Kao I na prijemnom portu i na portu slanja se samo specificira šema zahteva i odgovora. Ove šeme su dostavljene u specifikaciji servisa i nalaze se u projektu sa ostalim šemama. *SendServiceRequest* je vezom asocijacije povezan sa portom slanja koji je implementiran kao dvosmerni port. Kako je servis mup-a implementiran kao veb servis on se aktivira na zahtev, pa je s toga potrebno sačekati da on obradi zahtev i pošalje odgovor na zahtev kako bi orkestracija nastavila sa izvršavnjem.

Pošto je servis kreirao odgovor  poslao orkestraciji, taj odgovor je neophodno primiti isa čuvati. U orkestraciji već imamo oblik koji prima poruke i to je oblik koji inicira orkestraciju. Međutim, struktura ove dve poruke nije ista pa se ne mogu dve različite poruke primate preko istog oblika. Potrebno je kreirati novi oblik koji odgovara šemi poruke koju vraća server. Ovaj oblik je nazvan *ReceiveServiceResponse.* Primljen odgovor sadrži podatke o uspešnosti izvršavanja poziva i podatke o prebivalištu lica za koje je izvršen zahtev.

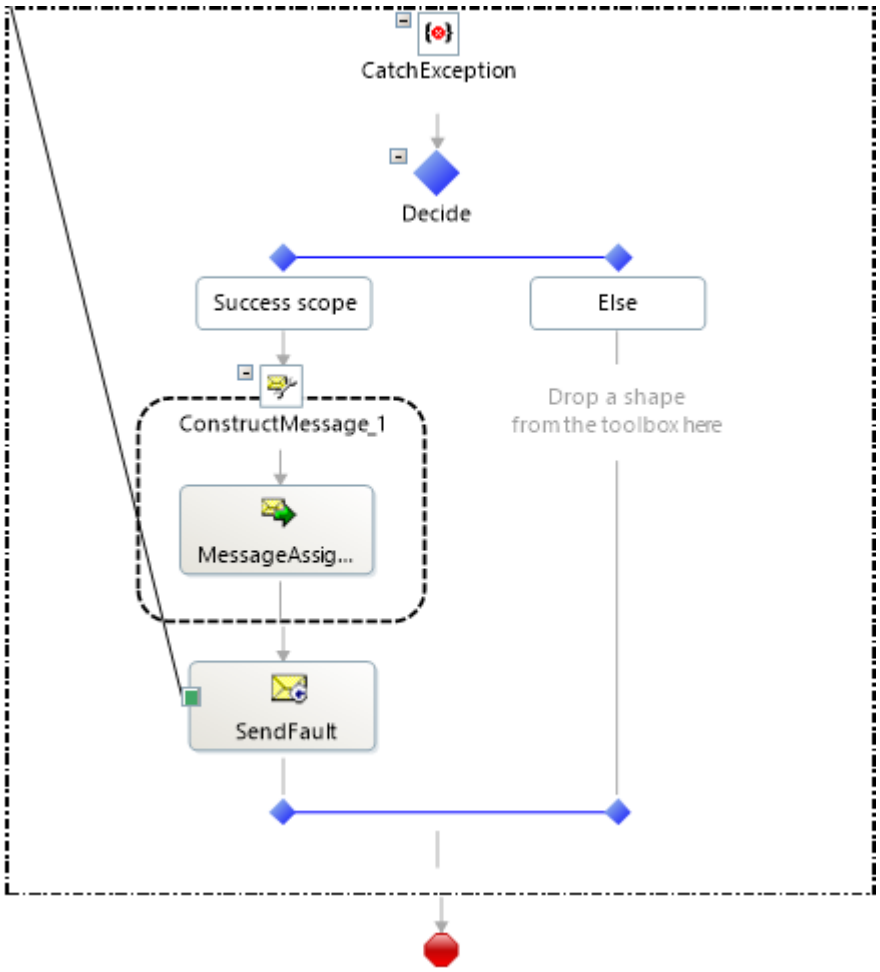



Slika 18 – konstruisanje poruke odgovora

Šema poruke koja će biti izlaz iz orkestracije je kreirana tako da odgovara šemi koja predstavlja podatke o licu tako da nije potrebno dodatno transformisati odgovor već ga samo proslediti na izlazni port. Oblik slanja je već korišćen u aplikaciji ali pošto se ne radi o istim šemama potrebno je napraviti novi oblik za slanje odgovora na portu sa šemom koja je specificirana. Kreirani oblik u orkestraciji koji služi za ovu namenu se zove *SendResponse* i on predstavlja završni korak izvršavnja orkestracije.

Kako se orkestracija sastoji iz više celina, prijem zahteva, poziv eksternog servisa itd., tako postoji veliki broj mesta na kojima može doći do greške prilikom izvršavanja orkestracije. Zbog toga može doći do delimičnog izvršavnaja orkestracije što može dovesti do nekonzistentnog stanja sistema. Na primer, ne može se izvršiti ostatak orkestracije ukoliko nije dobijen odgovor eksternog servisa. Da bi se izbegao navedeni problem, neophodno je obezbediti da se celokupna orkestracija izvršava pod transakcijom. Transakcija garantuje da će se sve u okviru iste izvršiti po principu sve ili ništa (Branislav Lazarević, 2012). Dakle, ako dođe do greške u jednom delu aplikacije, promene koje je aplikacija napravila tokom izvršavanja će biti poništene. Ako se pak sve izvrši bez problema, promene koje je orkestracija napravila na sistemu će biti potvrđene i primenjene na sistem.

Do nepredviđenih okolnosti tokom izvršavanja orkestracije može doći usled uticaja različitih faktora. Na primer, prilikom poziva eksternog servisa, eksterni servis može biti nedostupan, ili može doći do pada sistema na mašini na kojoj se izvršava orkestracija. Kako će korisnik koji je pozvao orkestraciju da zna šta se dogodilo u aplikaciji? Kako sprečiti pojavljivanja greške koju korisnik ne može da razume? Da bi se ovo sprečilo, neophodno je upravljati greškama na nivou orkestracije. Način na koji je to urađeno jeste da se poruka greške šalje ukoliko transakcija nije uspešno izvršena.




Slika 19 – obrada grešaka na nivou orkestracije

Naime, ukoliko izvršavanje orkestracije dovede do toga da dođe do pucanja transakcije, doći će do kreiranja poruke greške koja će biti poslata na interfejs porta koji je zadužen za vraćanje grešaka. Taj interfejs se naziva *fault* i definisan je *SOAP* protokolom. Već je opisano kako se kreira *SOAP* poruka greške. Poruka greške je standardizovana i sadrži dva parametra a to su kod greške i poruka greške. Ovako kreirana poruka se enkapsulira *SOAP* obvojnicom na izlaznom portu i šalje klijentu. Kada je poruka o grešci poslata ukoliko je došlo do pojave greške u sistemu mora se prikazati kraj izvršavanja orkestracije. Indikator za kraj izvršavanja orkestracije je crveni krug. On se nalazi na kraju orkestracije. Kao što je zeleni krug indikator početka izvršavanja orkestracije, tako je crveni krug indikator za kraj izvršavanja orkestracije.

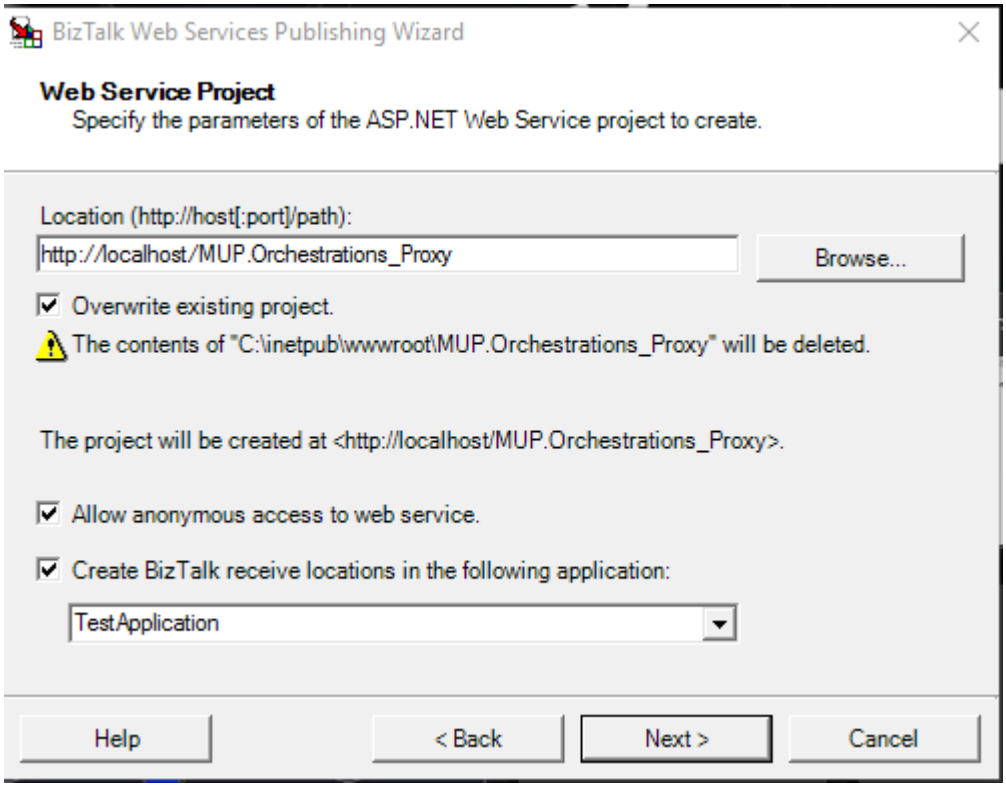
## Postavljanje rešenja na testno okruženje

Kada je implementirano rešenje, sledi kreranje izvršne verzije aplikacije i postavljanje istog na testni sistem. Kreiranje verzije aplikacije se sastoji od nekoliko koraka:

1. *Deploy solution* – prvi korak koji podrazumeva kreiranje izvšne verzije iz razvojnog okruženja. Najbolja praksa je da se iz razvojnog okruženja aplikacija prvo postavi na razvojni server. Da bi aplikacija bila postavljena svaki

projekat u okviru nje mora da bude potpisan, odnosno da naziv svakog *.dll* fajla aplikacije bude jedinstven

2. Objavljivanje orkestracije kao veb servisa – da bi orkestracija bila izložena na korišćenje potrebno je od ulaznih i izlaznih šema kreirati definiciju servisa.

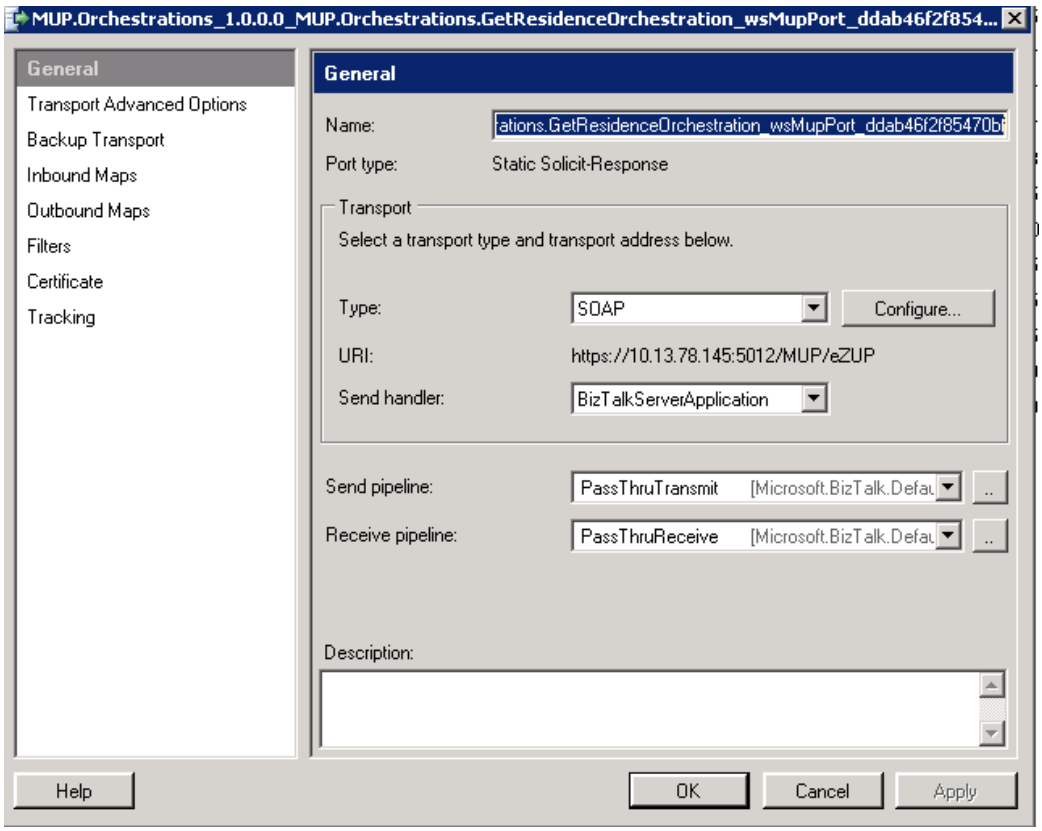



Slika 20 – kreiranje veb servisa od orkestracije

Ovaj proces podrazumeva kreiranje aplikacije na lokalnom *IIS* serveru. Kreirani servis predstavlja prijemnu lokaciju orkestracije. Kada se orkestracija postavi na server port koji je kreiran ovako će automatski biti postavljen kao prijemni port orkestracije

3. Restartovanje procesa pod kojim se izvršava prkestracija je sledeći korak

4. Konfigurisanje orkestracije – pošto je prijemni port već kreiran I povezan sa orkestracijom preostaje kreiranje porta za slanje odnosno poziv eksternog servisa.




Slika 21 – podešavanje porta slanja

Pored podešavanja naziva porta sledi definisanje tipa porta na koji će biti poslat zahtev. U konkretnom slučaju, odredišni servis je implementiran kao *soap* servis pa se za adapter postavlja *soap* adapter. Tokovi slanja i prijema omogućavaju validaciju primljene *xml* poruke. Opcija *PassThru* će eskivirati validaciju *xml* strukture (Rosanova, 2010)

5. Poslednji korak je osvežavanje *BizTalk* grupe kako bi se ubacili najnoviji .dll fajlovi u aplikaciju I pokretanje aplikacije

## Testiranje rešenja

Implementacija jednog informacionog sistema se sastoji od nekoliko faza koje se sastoje od velikog broja aktivnosti. Prva faza razvoja jednog informacionog sistema jeste prikupljanje korisničkih zahteva. U drugoj fazi vrši se analiza i projektovanje sistema. Sledeći korak jeste implementacija rešenja. Kada je kreirana prva funkcionalna verzija aplikacije sledi testiranje. Po uspešnom izvršavanju testova sledi uvođenje sistema u kompaniju i puštanje u rad.

Testiranje je jedna od najbitnijih faza razvoja informacionog sistema. U fazi testiranja se proverava da li je aplikacija ispunila funkcionalnu specifikaciju kao i tehničku specifikaciju. Funkcionalnosti sistem su projektovani korisnikovi zahtevi. Testiranje predstavlja utvrđivanje odstupanja implementiranog rešenja od projektovanog. Pored toga testiranje obuhvata i utvrđivanje odstupanja od očekivanog izlaza za određeni ulaz. Jako je bitno razviti odgovarajuću

strategiju testiranja. Idealan scenario podrazumeva da se testiranje vrši na više različitih okruženja. Prvo testiranje se vrši tokom razvoja sistema i njega vrše programeri tokom implementacije. Kada se funkcionalnost inicijalno istestira sledi postavljanje verzije aplikacije na testno okruženje. U ovoj fazi primat u testiranju preuzimaju testeri koji po specifikaciji proveravaju različite testne slučajeve korišćenja. Ukoliko izlaz odstupa od projektovanog znači da se aplikacija ne ponaša kako je očekivano i zahteva doradu. Iterativnim procesom testiranja dolazi se do stabilne verzije aplikacije koja se dalje prosledjuje korisniku. Naredna faza podrazumeva korisnikovo testiranje na sopstvenom testnom okruženju. Ova faza testiranja predstavlja najbitniju u celom procesu jer se tada stiče prvi korisnikov utisak o samom sistemu. I ova faza se iterativno realizuje ukoliko korisnik uoči neke nedostatke ili greške prilikom rada. Poslednja faza jeste postavljanje sistema na produkciono okruženje i puštanja sistema u rad.

Testiranje *BizTalk* aplikacija može da bude mukotrpan proces. Glavni razlog za pomenute poteškoće jeste taj što je veliki deo komponenti predstavlja crnu kutiju. U nastavku će biti opisani načini testiranja *BizTalk* aplikacija u fazi razvoja i u fazi kada se aplikacija nalazi na testnom okruženju.

## Testiranje *BizTalk* aplikacija tokom razvoja

*BizTalk* aplikacije predstavljaju kompleksna rešenja koja služe za integraciju više sistema. Samim tim može postojati veliki broj poteškoća prilikom razvoja. Neki od problema su sledeći:

- Eksterni servisi su nedostupni sa razvojnog okruženja. Najčešći razlozi su bezbednost i zaštita podataka
- Servisi koje je potrebno integrisati nisu implementirani po standardu. Ovo se dešava zbog nekompetentnosti implementatora
- Neažurna dokumentacija. Na primer, ukoliko su oba sistema koja je potrebno integrisati u fazi razvoja, može vrlo lako doći do nekonzistentnosti u specificiranom i implementiranom
- Korišćenje različitih standarda

Pomenuti problemi nisu direktno vezani za kompleksnost *BizTalk* -a, ali se uz kompleksnost *BizTalk* -a veoma teško rešavaju.

U toku procesa implementacije neophodno je izvršiti određena testiranja pre nego aplikacija završi na testnom okruženju. Ono što programeri mogu da istestiraju jeste izvršavanje orkestracija. Ukoliko orkestracija poziva eksterni servis koji nije dostupan sa razvojnog okruženja, poziv servisa je prepreka za dalje testiranje. Neke od taktika koje se mogu primeniti u testiranju aplikacije su testiranje transformacija i testiranje šema. Naime, da bi orkestracija mogla da se razvija neophodno je imati definisane *xml* šeme po kojima se vrši komunikacija. Razvojni alata, *Visual Studio,* sadrži nekoliko vizuelnih alata koji omogućavaju kreiranje *xml* šema bez poznavanja same sintakse. Alat je jako koristan zbog toga što nam omogućava sledeće funkcionalnosti:

- Generisanje *xml* dokumenata koji odgovaraju izabranoj šemi
- Validacija jednog *xml* dokumenta po određenoj šemi

- Validacija šeme koju smo kreirali ručno, ne koristeći vizuelni alat

Pored pomenutih metoda testiranja šema i *xml* poruka, razvojno okruženje nudi i opcije testiranja transformacija. Transformacija sadrži izvornu i odredišnu šemu. U pomenutom alatu, ukoliko hoćemo da izvršimo testiranje transformacija, potrebno je kreirati jedan *xml* dokument koji odgovara ulaznoj šemi i naznačiti da je to ulazna instanca. Klikom na gume *Test map* pokreće se izvršavnaje transformacije sa ulaznim *xml* dokumentom. U prozoru koji se pojavljuje po završetku transformacije ispisuje se ishod transformacije, putanja do *xml* dokumenta koji je rezultat izvršavanja i poruka o eventualnoj grešci.

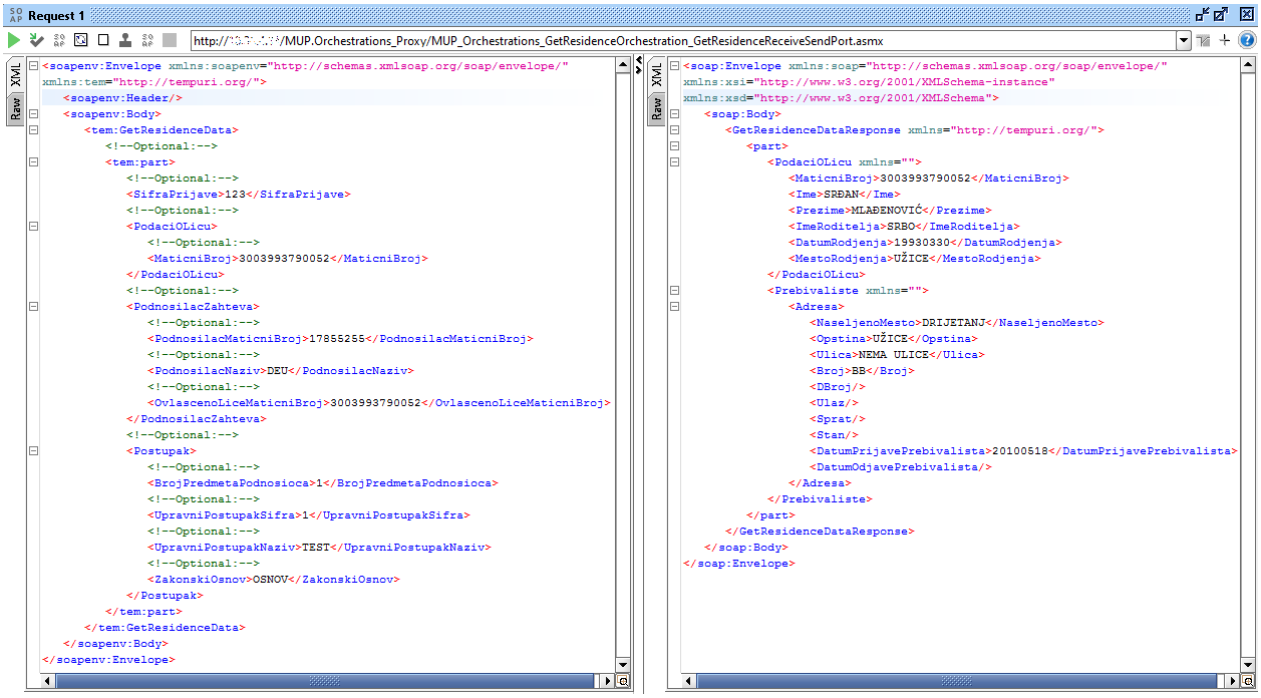
Ovakvim testiranjem se može testirati veliki deo orkestracije. Samim tim što da bismo razvijali aplikaciju moramo znati šeme poruka koje razmenjujemo sa eksternim servisom. Generisanjem instance na pomenuti način možemo testirati transformacije iako nemamo stvarne podatke.

## Testiranje *BizTalk* aplikacijana testnom okruženju

Na testnom okruženju moraju biti otklonjene sve prepreke za nesmetanu komunikaciju. U ovoj fazi je potrebno testirati celokupnu aplikaciju. Iako je do sada testiran veliki deo aplikacije preostaje testiranje glavnog dela aplikacije a to je deo zadužen za komunikaciju sa eksternim sistemima.

Kada je aplikacija postavljena na testno okruženje postaje nemoguće ući u samu implementaciju i proveravati kako se izvršava aplikacija. Sada je potrebno pribeći drugim tehnikama testiranja. Kako je u primeru aplikacija izložena kao veb servis testiranje se može vršiti nekim od alata koji simuliraju *soap* klijenta. Neki od alata koje je moguće korisiti su *soapui, wcfTestClient* i mnogi drugi.

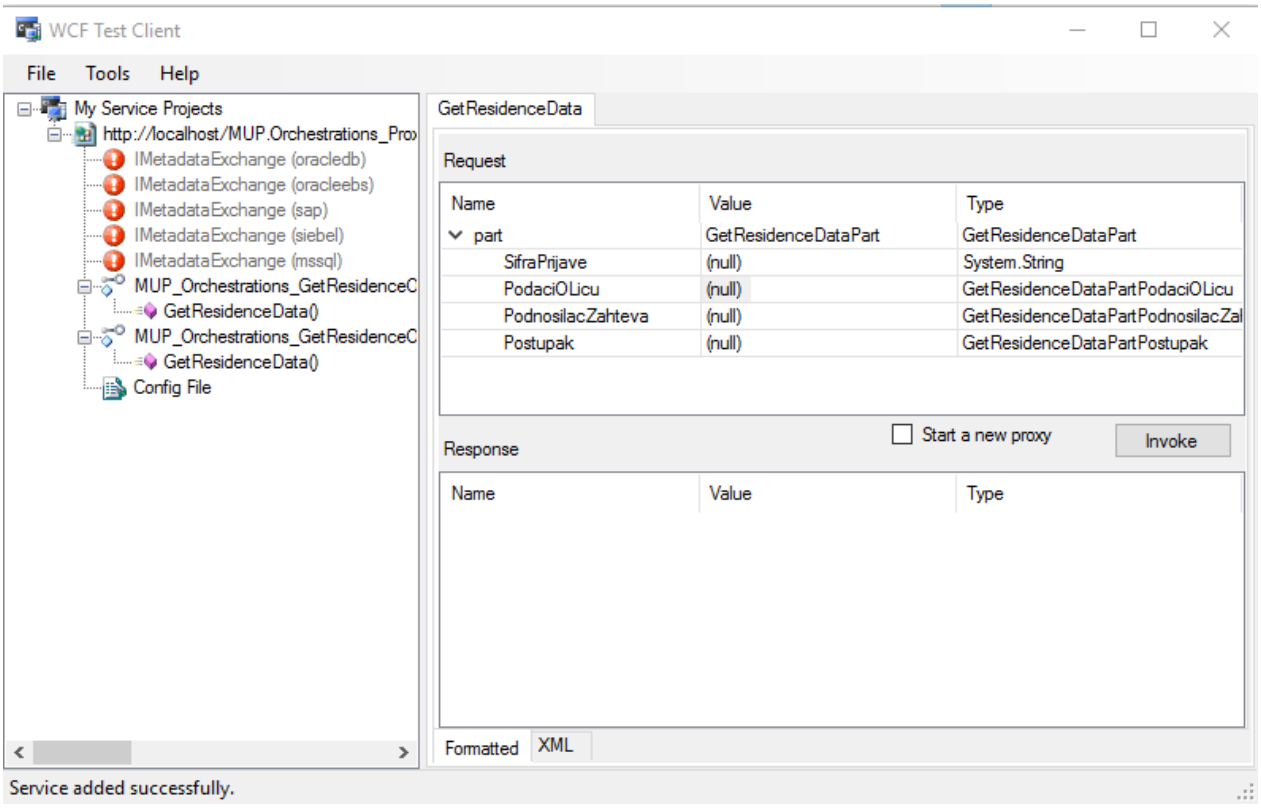
*SoapUI* je aplikacija koja se može preuzeti sa internet i razvijena je u programskom jeziku *JAVA.* Program obezbeđuje kreiranje *soap* zahteva i obradu odgovora. Da bi bilo moguće obaviti poziv servisa neophodno je u *soapui* krerati projekat. Kada je kreiran projekat potrebno je dodati serviskoji će biti testiran. To se radi ubacivanjem definicije servisa. Alat je kreiran tako da prilikom dodavanja servisa generiše poziv sa primerom zahteva sa nasumičnim podacima.




**Slika 22 – prikaz alata za testiranje veb servisa *SoapUI***

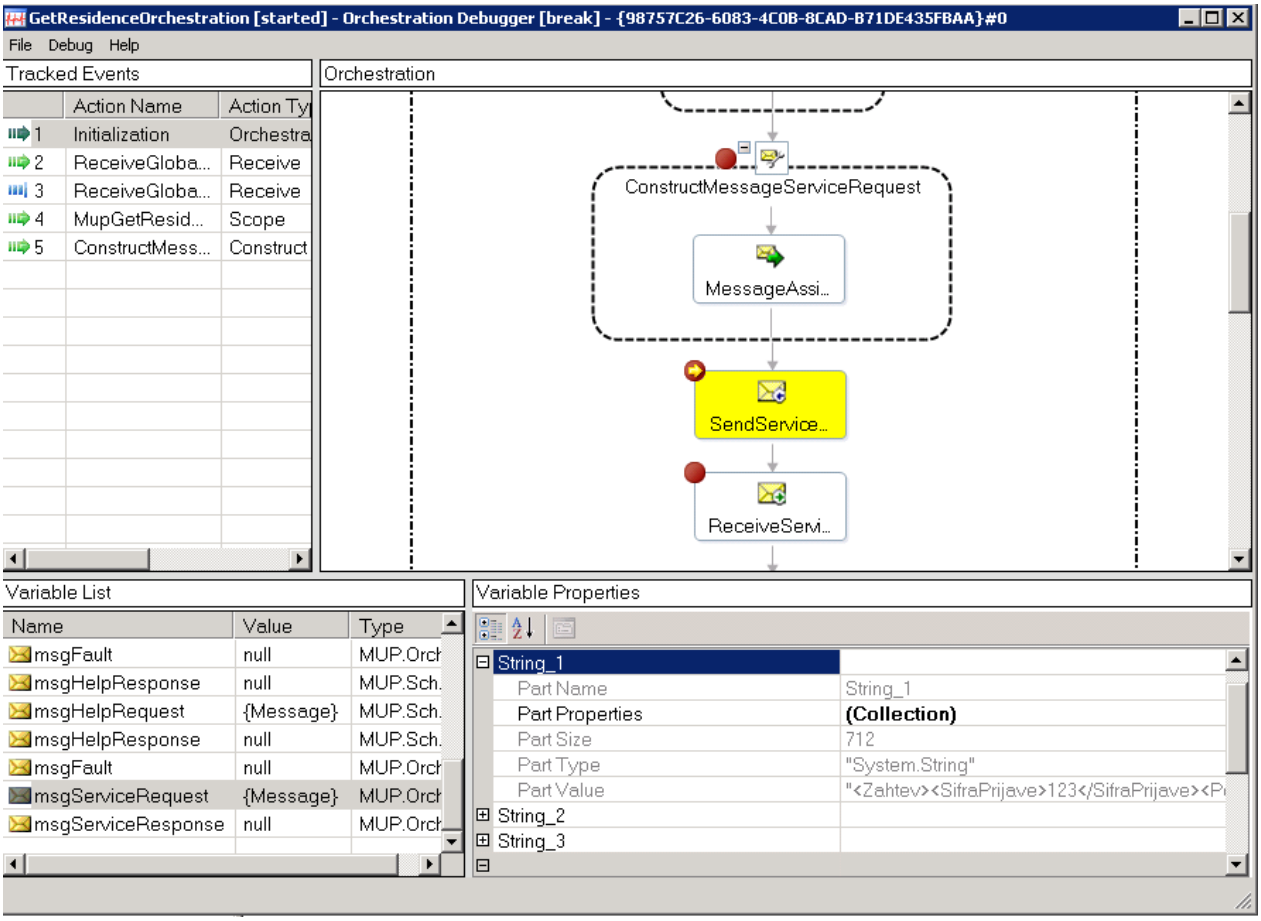
Na slici dvadeset i dva je prikazan prozor testne aplikacije. U gornjem delu slike se vidi *url* adresa servisa koji se testira. U levom panelu je prikazan izgenerisan primer zahteva. Kada je zahtev generisan ostaje još samo da se popune testni podaci kako bi se izvršio poziv servisa. Desni panel je inicijalno prazan i popunjava se po prijemu odgovora.

Drugi pomenuti alat je *WcfTestClient* i predstavlja pandan pomenutom programu. *WcfTestClient* je jednostavniji alat i dolazi kao deo razvojnog alata *Visual Studio*-a. Stoga je ovaj alat pogodan za testiranje servisa iz razvojnog okruženja. Slika dvadeset i tri predstavlja interfejs alata. Logika testiranja je manje-više ista kao i kod *SoapUI*. Razlika je u tome što se u *SoapUI* mogu kreirati kompleksniji projekti i čitavi setovi zahteva. Takođe, za razliku od *WcfTestClient* moguće je dodavati sertifikate u pozive.




Slika 23 – alat za testiranje *WcfTestClient*

Ukoliko dođe do greške prilikom obrade poziva, isti je moguće analizirati u alatu na *BizTalk* serveru koji je služi za otkrivanje grešaka tokom izvršavanja orkestracije(*orchestration debugger*). U narednom primeru je prikazan tok otkrivanja grešaka tokom izvšavanja orkestracije. Na svakom obliku je moguče zaustaviti se i analizirati ulaze i izlaze. Ono što nije moguće ispitati jeste šta se zapravo dešava u okviru oblika. Primer je baziran na testiranju aplikacije koja poziva eksterni servis mup-a. Žuta boja je indikator do kog koraka je orkestracija stigla sa izvršavanjem. Dakle na slici vidimo da je orkestracija stigla do poziva eksternog servisa. Sa donje leve strane se vidi spisak svih varijabli u sistemu, inicijalizovanih i onih koje će tek biti inicijalizovane. Klikom na varijablu u panelu sa desne strane se ispisuju sve osobine varijable. Na primer za poruku *msgServiceRequest* se vidi da ona ima tri dela(*String_1*, *String_2*, *String_3*). U prvom delu poruke se može uočiti osobina *Part Value* koja predstavlja vrednost dela poruke, a u osobini *Part Type* je prikazan tip dela poruke. Prelaskom na sledeći korak doći će do poziva servisa. Po prijemu poruke se može analizirati primljena poruka na isti način. Ako je uočena greška prilikom izvršavanja može se popraviti implementacija orkestracija, kreirati nova verzija aplikacije, spustiti na testno okruženje I isti process testiranja ponoviti sve dok se ne otklone sve uočene greške.




Slika 24 – *orchestration debugger*

Iako opisani proces ne izgleda naročito kompleksan, može postati prava noćna mora. Međutim kada se kreira stabilna verzija sistema proces održavanja istog je prilično jednostavan.

# Zaključak

Ideal moderne elektronske javne uprave predstavlja uvođenje automatizovane obrade podataka i zahteva, što podrazumeva maksimalno smanjenje učešća službenika u celom procesu. Takođe se teži tome da se smanji direktna interakcija između službenika i građana i da se celokupna komunikacija obavlja upotrebom informacionih sistema. Automatizacijom procesa se smanjuje verovatnoća pojave greške usled delovanja ljudskog faktora. Ovo je naročito značajno kada se radi sa osetljivim podacima čije pogrešna obrada dovodi do velikih negativnih konsekvenci.

Korišćenjem stabilnih skalabilnih platformi moguće je u velikoj meri ispuniti pomenute ciljeve. Međutim koliko god automatizovan proces bio i koliko god dobro razvijeni takvi sistemi bili, uvek će postojati deo procesa koji će morati da obavljaju službenici.

Iako su teorijski koncepti na kojima počiva celokupna logika *BizTalk* servera, kao jednog od glavnih platformi za integraciju sistema, neretko se u praksi pokazuju kao prekompleksni i nepraktični za razvoj. Jedan od glavnih razloga neefikasnog uvođenaj istog u poslovanje jeste nepoznavanje arhitekture platforme i nerazumevanje poslovnih rešenja.

Da zaključimo, *BizTalk* platforma je stabilna, skalabilna i sveobuhvatna, ali se do ispoljavanja ovih osobina dolazi samo ozbiljnim pristupom i ulaganjem velikih napora i resursa. Pored toga, korišćenje istog je pogodno za sistem elektronske uprave koje integrišu veliki broj servisa i obrađuju veliki broj zahteva.

# Literatura


- Barry, D. K. (n.d.). *Web Services Definition*. Preuzeto Septembar 15, 2017 sa http://www.service-architecture.com/articles/web-services/web_services_definition.html
- Branislav Lazarević, Z. M. (2012). *Baze podataka.* Beograd: Fakultet organizacionih nauka, Beograd, Jove Ilića 154.
- Brian Loesgen, C. Y. (2012). *Microsoft BizTalk Server 2010 unleashed.* Indianapolis, USA: Pearson Education, Inc.
- Don Box, D. E. (2000, Avgust 22). Simple Object Access Protocol (SOAP) 1.1.
- Jim Kurose, K. R. (2014). *Umrežavanje računara od vrha ka dnu.* Beograd: CET Computer Equipment and Trade.
- Microsoft. (n.d.). *Receive Pipelines*. Preuzeto Septembar 17, 2017 sa https://msdn.microsoft.com/en-us/library/aa561803.aspx
- Microsoft. (n.d.). *Regular Expression Language*. Preuzeto Septembar 14, 2017 sa https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference
- OMG. (2011, Januar). Business Process Model and Notation v2.0.
- *opis_usluge*. (2017, Avgust 23). Preuzeto sa http://www.euprava.gov.rs: http://www.euprava.gov.rs/eusluge/opis_usluge?generatedServiceId=2252&title=Zahtev-za-izdavan%D1%98e-uveren%D1%98a-o-prose%C4%8Dnom-mese%C4%8Dnom-prihodu-po-%C4%8Dlanu-porodice-radi-ostvarivan%D1%98a-prava-na-u%C4%8Deni%C4%8Dke-stipendije-i-studentske-kre
- Oracle. (n.d.). *Siebel* . Preuzeto Septembar 18, 2017 sa http://www.oracle.com/us/products/applications/siebel/overview/index.html
- Ray, E. T. (2001). *Learning XML*.
- Republika Srbija, R. z. (n.d.). *Stanovništvo*. Preuzeto sa http://www.stat.gov.rs: http://www.stat.gov.rs/WebSite/Public/PageView.aspx?pKey=162
- Rosanova, D. (2010). *Microsoft BizTalk Server 2010 Patterns.* Birmingham: Packt Publishing Ltd.
- Sl. list SRJ, b. 3. (n.d.). Zakon o opštem upravnom postupku.
- *struktura*. (2017, Avgust 23). Preuzeto sa http://www.mduls.gov.rs: http://www.mduls.gov.rs/struktura.php
- *Zakon o državnoj upravi.* (79/2005, 101/2007, 95/2010, 99/2014). Beograd: Sl. glasnik RS.