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.

Contents

[Uvod 5](#_Toc493427408)

[Državna uprava 6](#_Toc493427409)

[Osnovni činioci javne uprave 6](#_Toc493427410)

[Međusobne veze državnih organa 6](#_Toc493427411)

[Odnos javne uprave sa građanima 7](#_Toc493427412)

[Elektronska javna uprava 7](#_Toc493427413)

[Osnovni procesi javne uprave 8](#_Toc493427414)

[Proces kreiranja usluga 8](#_Toc493427415)

[Definisanje šablona usluga 8](#_Toc493427416)

[Dodeljivanje prava na izvršavanje usluga 9](#_Toc493427417)

[Proces podnošenja zahteva 9](#_Toc493427418)

[Autentikacija korisnika 9](#_Toc493427419)

[Kreiranje poziva eksternih servisa 9](#_Toc493427420)

[Proces obrade zahteva 9](#_Toc493427421)

[Proces administracije institucija i korisnika 9](#_Toc493427422)

[Administracija institucija 9](#_Toc493427423)

[Administracija korisnika 9](#_Toc493427424)

[Administracija uloga i prava korisnika i institucija 9](#_Toc493427425)

[BizTalk server – orkestracija procesa 9](#_Toc493427426)

[Osnovni koncepti XML strukture podataka 9](#_Toc493427427)

[*XML* elementi i atributi 10](#_Toc493427428)

[*XML* šeme 11](#_Toc493427429)

[*xPath –* upitni jezik za kretanje kroz *xml* strukturu 13](#_Toc493427430)

[*SOAP* protokol – osnovni protokol *B2B* komunikacije 14](#_Toc493427431)

[*SOAP* poruke i statusi odgovora 15](#_Toc493427432)

[Definicija *SOA* servisa 17](#_Toc493427433)

[Poređenje *SOAP* i *REST* protokola 19](#_Toc493427434)

[Osnovni koncepti *BizTalk* servera 19](#_Toc493427435)

[Arhitektura BizTalk servera 20](#_Toc493427436)

[Portovi prijema i slanja 22](#_Toc493427437)

[BizTalk orkestracija 24](#_Toc493427438)

[Izlaganje orkestracija kao veb servisa 27](#_Toc493427439)

Sadržaj slika

[Slika 1 - prikaz enkapsulacije soap poruka 16](#_Toc493427440)

[Slika 2 – tok obrade zahteva na *BizTalk* server 21](#_Toc493427441)

[Slika 3 – prikaz structure prijemnog porta 23](#_Toc493427442)

[Slika 4 – prikaz struktura porta slanja 24](#_Toc493427443)

[Slika 5 – BizTalk mape 25](#_Toc493427444)

[Slika 6 – primer BizTalk orkestracije 26](#_Toc493427445)

# Uvod

Javna uprava kao informacioni sistem 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 puno potrošenog vremena čekanja u redovima, papirologije 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 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 uvođenjem 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 koje 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 jako velika i prekompleksna 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 definisanim okvirima po tačno definisanim procedurama. I jedne i druge institucije rade u službi građanja. Oko efikasnosti delovanja javne uprave stara se Ministarstvo državne uprave i lokalne samouprave.

Samo ministarstvo je podeljeno u 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 te ista 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]](#footnote-1) državne uprave ka građanima. Kako su građani 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 n.d.) 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 upravo. 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 olakšava 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 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 skor osvakom savremenom ljudskom biću i u velikoj metri 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 koji će vam stići 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 šelteru, 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, ako ništa 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 instance 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 jako vodi računa 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 jeste, 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 korisnika

### Kreiranje poziva eksternih servisa

## Proces obrade zahteva

## Proces administracije institucija i korisnika

### Administracija institucija

### Administracija korisnika

### Administracija uloga i prava korisnika i institucija

# BizTalk server – orkestracija procesa

## Osnovni koncepti XML strukture podataka

*XML(eXtensible Markup Language)* predstavlja strukturu podataka.Dakle, ukoliko hoćemo da uspostavi 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 Wb Consortium(W3C)* (Ray 2001)*.* Nastao je kao potreba da za standardizacijom. Ono što izdvaja *xml* strukturu od drugih jeste to što u nju mogu biti ugrađeni meta podaci[[2]](#footnote-2).

*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[[3]](#footnote-3)* 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[[4]](#footnote-4) potrebno je da se nalazi između otvorenog i zatvorenog taga. Dodatno se u okviru *xml* dokumenta može dodati određeni vid dokumentacije, ogranučenja nad određeni tip vrednosti itd. *Xml* veoma podseća na *HTML(Hyper Text Markup Language)[[5]](#footnote-5).* 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"?>*

*<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 i meta atributima i verzijama *xml*-a se može pronaći na linku W3C-a([www.w3.org](https://www.w3.org/TR/2008/REC-xml-20081126/)). 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 znati 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 par problema. Prvi je kako validirati primljenu strukturu. Svaki put kada primimo *xml* poruku neophodno je da proverimo da li je zadovoljena dogovrena 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 gore pomenutom 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[[6]](#footnote-6) 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.

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

Dakle, u šemama se definiše jasno definisana, kruta 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 nazvi 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[[7]](#footnote-7) 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* 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[[8]](#footnote-8).

### *SOAP* poruke i statusi odgovora

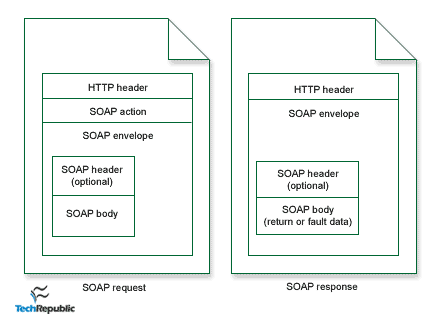
Vec je pomenuto da korisnikovu poruku, soap protokol enkapsulira obvojnicom koja sadrži meta podatke krucijalne za razumevanje između dva soap klijenta. S obzirom na to da soap koristi http protokol aplikativnog sloja OSI arhitekture soap poruka će biti još jednom enkapsulirana obvojnicom koju dodaje 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 enkapsulacije soap obvojnicom gde se dodaju specifični podaci svojstveni soap protokolu. Kada je soap poruka kreirana ona biva prosleđena http protokolu na slanje. Http dalje enkapsulira soap 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 http 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[[9]](#footnote-9) 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.



Slika - 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 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 REQUES – 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 medjuvremenu.

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 n.d.).

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[[10]](#footnote-10)* okruženju i aplikaciju koja poziva dati servis a koja je implementirana u *JAVA[[11]](#footnote-11)* 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 par 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 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[[12]](#footnote-12)* 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[[13]](#footnote-13), meta podaci o servisima, bezbednost aplikacija i podataka, veći broj razlitih tipova razmene poruka i enkodovanja istih, transakcije[[14]](#footnote-14) itd. WCF aplikacije odlikuje i veoma visok stepen konfigurabilnosti bez dodatnog menjanja programskog koda. Wcf u potrpunosti može da zameni klasične veb servise i da šruži 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 a 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 aplikacije 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[[15]](#footnote-15), 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 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 službeni, bilo po 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 tržišzu. 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 pandana. Razlog je taj što su sistemi koji su implementirani preko biztalka toliko kompleksni da bi njihova migracija na drugu tehnologiju bila jednaka izradi identičnog sistema od nule (Brian Loesgen 2012).

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

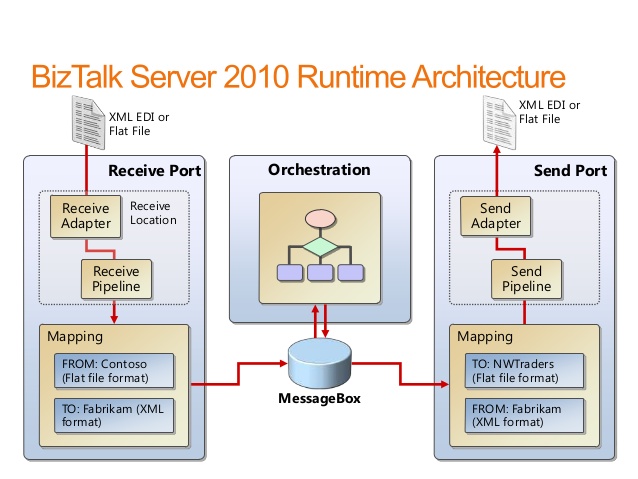
Biztalk kao robustna platforma koja ima veliki broj funkcionalnosti i omogućava raznovrsne implementacije jeste veoma kompleksna. Celokupnu arhitekturu nije moguće u potpunosti sagledati jer veliki deo predstavlja crnu kutiju i potpuno je nedostupan zbog politike *Microsoft-*a jer Biztalk nije softver 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[[16]](#footnote-16)*.

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 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 – 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.

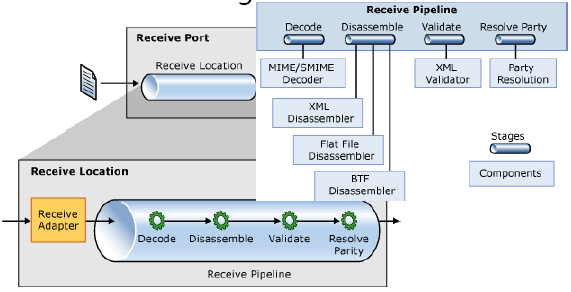
Kada je poruka sačuvana u bazu, može se smatrati da je proces orkestracije pokrenut. Postavljanjem aplikacije na server postavljaju se i šeme poruka koje su specifične za određene orkestracije. Za poruku koja odgovara određenoj šemi, kreiraju se pretplatnici(*subscriber*) na taj tip poruka i njeni izdavači(*publisher*). 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. Poruk 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 isto 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.

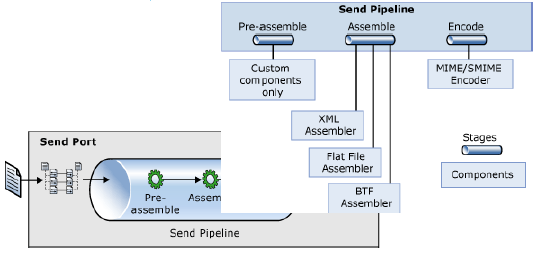
Prijemni port je dodirna tačka kroz koju poruke pristiž u aplikaciju. Prijemni portovi mogu biti dvostrani(*two-way ports*), tj moguće je i primati i slati poruke preko istog porta, tzv. *request-response port*. Ranije je konstatovano da se port sastoji iz dva glavna dela, prijemna lokacija i prijemni 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 *binging,* 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[[17]](#footnote-17). 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 n.d.). 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 – prikaz structure prijemnog porta

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 – prikaz struktura porta slanja

### BizTalk orkestracija

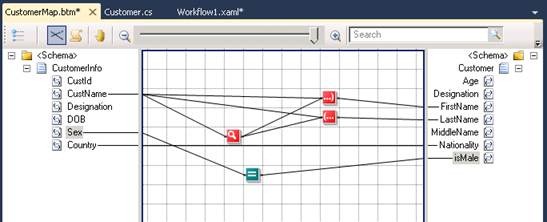
Do sada su objašnjeni mehanizmi prijema i slanja poruka. Prijem poruke je okidač koji pokreće izvršavanje ostatka logike u aplikaciji. Biztalk orkestracija predstavlja srž procesa. U njoj je implementirana kompletna poslovna logika sa pravilima izvršavanja. Orkestracija je poslovni proces i predstavlja redosled izvršavnaja instrukcija. Orkestrircija je dobila ime bas 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[[18]](#footnote-18),* sadrži alat koji omogućava lagan i brz razvoj mapa a koji je baziran na grafičkom interfejsu.

Naredna slika prikazuje alat za razvoj biztalk mapa.



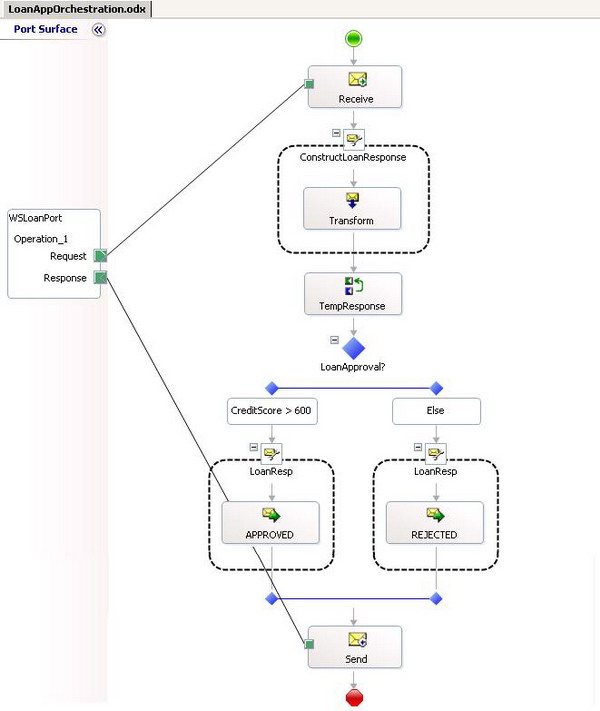
Slika – BizTalk mape

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.*

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[[19]](#footnote-19) fajlovi registruju u GAC[[20]](#footnote-20) operativnog sistema. Nakon instalacije .*msi* jedino još preostaje importovanje aplikacije na server o čemu će biti reči u nastavku.



Slika – 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 odrađea 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.
* *Paralle l 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 apliakcija 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 izvršavnaja.

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 alata 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.

1. Interfejs u ovom kontekstu predstavlja tačku koja je izložena za treća lica, tj. tačku dodira sistema državne uprave sa građanstvom [↑](#footnote-ref-1)
2. Meta podaci – podaci o podacima, tj. podaci koji opisuju podatke, daju više informacija o istim [↑](#footnote-ref-2)
3. 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 [↑](#footnote-ref-3)
4. 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 [↑](#footnote-ref-4)
5. *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 [↑](#footnote-ref-5)
6. 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 [↑](#footnote-ref-6)
7. Regularni izrazi(*Regular expression*) – u žargonu poznatiji kao *regex* predstavlja set definisanih literala i operatora koji imaju određeno značenje u prepoznavanju šablona teksta (Microsoft n.d.) [↑](#footnote-ref-7)
8. 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). [↑](#footnote-ref-8)
9. URL – *Unified resource location* predstavlja jedinstvenu putanju do određenog resursa na internetu [↑](#footnote-ref-9)
10. *.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 [↑](#footnote-ref-10)
11. *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 [↑](#footnote-ref-11)
12. *Internet information Services* predstavlja *Microsoft*-ov server na koji se hostuju veb aplikacije razvijene upotrebom .NET razvojnog okruženja [↑](#footnote-ref-12)
13. Interoperabilnost je osobina programa ili sistema da vrši interakciju sa drugim programima ili sistemima koji su implementirani u različitoj tehnologiji [↑](#footnote-ref-13)
14. Izvršavanje pod transakcijom podrazumeva izvršavanje većeg broja instrukcija po principu ,,sve ili ništa’’. Ukoliko se jedna od instrukcija ne izvrši kako je predviđeno poništava se rezultat izvršavanja ostalih instrukcija [↑](#footnote-ref-14)
15. Skalabilnost je osobina sistema da uspešno odgovori na povećanje kapaciteta sistem, tj. da se sa obimnijim ulazom generiše proporcionalno veći izlaz [↑](#footnote-ref-15)
16. SQL Server je *Microsoft-*ov server baze podataka. On implementira SQL bazu podataka. [↑](#footnote-ref-16)
17. MIME tip označava da je tip koji se razmenjuje zapravo dokument. U biztalk kontekstu, svaka poruka je MIME tip jer je svaka poruka xml [↑](#footnote-ref-17)
18. *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.* [↑](#footnote-ref-18)
19. .*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. [↑](#footnote-ref-19)
20. 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 [↑](#footnote-ref-20)